Хмельницький національний університет

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Вебзастосунок для купівлі та продажу книг

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 2201104.01.09.ПЗ

Виконав студент IV курсу, група ІПЗ-22-1 _____ Вікторія МЕЛЬНИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз-мат.наук, проф. _____ Леонід БЕДРАТЮК
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 грудня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мельник Вікторії Миколаївній

Прізвище, ім'я, по батькові студента

1. Тема роботи Вебзастосунок для купівлі та продажу книг

Керівник роботи Бедратюк Леонід Пертович, д-р фіз-мат. наук, проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Строк подання студентом роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4.Зміст пояснювальної записки (перелік питань, які потрібно розробити) дослідження предметної області, проєктування програмного забезпечення, програмна реалізація та тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Три креслення:

1. Конктекстна IDEF-0 діаграма
2. UML-Діаграма варіантів використання
3. Діаграма міжмодульних зв'зків

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н. Ї., доцент		
Антиплагіат	Форкун Ю. В., доцент	05.05.2026	05.05.2026

7. Дата видачі завдання «_02_» __січня_ 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2026	
3 Проектування програмного забезпечення	21.02 – 20.03 2026	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6 Попередній захист КвР	травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.06.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з **01.06.2026**	

Студент _____ Вікторія МЕЛЬНИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Вебзастосунок для купівлі та продажу книг».

Автор роботи: Мельник Вікторія Миколаївна.

Керівник роботи: Бедратюк Леонід Петрович.

Пояснювальна записка: 75 с., 27 рис., 3 табл., 4 дод., 30 джерел.

Графічна частина: 3 креслення.

ІНТЕРНЕТ-МАГАЗИН, КНИГА, ЕЛЕКТРОННА КОМЕРЦІЯ, ASP.NET CORE MVC, C#, MySQL Server.

Метою роботи є розроблення вебзастосунку для купівлі та продажу книг, який буде відповідати сучасним технічним вимогам та принципам побудови систем електронної комерції.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначено вимоги до вебзастосунків електронної комерції та, зокрема, до інтернет-магазинів книжкової продукції, розроблено детальну архітектуру застосунку, спроєктовано структуру модулів та інтерфейс користувача.

Для реалізації вебзастосунку використано мову програмування C#, архітектурний шаблон MVC, платформу ASP.NET Core MVC, сервер бази даних MySQL, а також технології HTML і CSS для створення користувацького інтерфейсу.

У результаті проєктування здійснена програмна реалізація вебзастосунку інтернет-магазину книг, що забезпечує зручний пошук і перегляд книжкової продукції, роботу з кошиком, оформлення замовлень та адміністрування каталогу товарів. Розроблений застосунок може бути використаний як базова платформа для створення сучасних систем електронної комерції у сфері продажу книжкової продукції.

27.05.2026
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ. 2201104.01.09.ПЗ	Пояснювальна записка	85		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ. 2201104.01.09.ПЗ	Контекстна IDEF-0	1		
			діагарама			
5	А3	КвРІПЗ. 2201104.01.09.ПЗ	UML-діагарама варіантів	1		
			використання			
6	А3	КвРІПЗ. 2201104.01.09.ПЗ	Діагарама міжмодульних	1		
			звязків			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.00308080.01.09.ВД		
Виконав		Мельник В.М.		28.05	Вебзастосунок для купівлі та продажу книг	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Л.П.		28.05			
Н. контр.		Праворська Н.І.		28.05	Відомість документів	ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л.П.		28.05			

ЗМІСТ

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.00308080.01.09.ВД		
Виконав		Мельник В.М.		28.05	Вебзастосунок для купівлі та продажу книг	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Л.П.		28.05			
Н. контр.		Праворська Н.І.		28.05	Відомість документів	ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л.П.		28.05			

					КвРІПЗ.00308080.01.09.ВД	
Змн.	Арк.	№ докум.	Підпис	Дата		
Виконав		Мельник В.М.		28.05	Вебзастосунок для купівлі та продажу книг	Літ. / Арк. 1 / Аркушів 1
Керівник		Бедратюк Л.П.		28.05		
Н. контр.		Праворська Н.І.		28.05	Відомість документів	ХНУ, ІПЗ-22-1
Зав. каф.		Бедратюк Л.П.		28.05		

ВСТУП

Сучасний розвиток інформаційних технологій та процеси цифрової трансформації зумовлюють активне перенесення традиційних бізнес-процесів у вебсередовище. Однією з найдинамічніших сфер у цьому контексті є електронна комерція, яка охоплює різноманітні онлайн-платформи для реалізації товарів і послуг. Використання вебзастосунків у сфері електронної торгівлі забезпечує зручний доступ користувачів до асортименту продукції, автоматизує процеси оформлення замовлень, здійснення платежів і обробки даних, що сприяє підвищенню ефективності діяльності підприємств та покращенню якості обслуговування споживачів.

Особливої актуальності набуває розроблення вебзастосунків для продажу та купівлі книг, оскільки книжковий ринок активно переходить в онлайн-формат. Такі системи дозволяють користувачам швидко знаходити необхідну літературу, переглядати детальну інформацію про книги, формувати кошик покупок та оформлювати замовлення без необхідності фізичного відвідування магазину. Крім того, сучасні вебплатформи забезпечують персоналізацію контенту, систему рекомендацій та інтеграцію з платіжними сервісами, що робить процес купівлі більш зручним та ефективним.

З технічної точки зору створення вебзастосунку для продажу книг є важливим напрямом, який дозволяє поєднати знання з розробки клієнт-серверних систем, проєктування баз даних та реалізації бізнес-логіки. Використання сучасних технологій, таких як ASP.NET Core, Entity Framework та архітектурні підходи на основі шаблону MVC, дає змогу створювати масштабовані, безпечні та продуктивні вебсистеми.

Важливим фактором є також зниження порогу входу у веброзробку завдяки сучасним фреймворкам і бібліотекам, які значно спрощують процес створення складних систем. Наявність готових рішень для аутентифікації, роботи з базами даних

та побудови інтерфейсу користувача дозволяє розробникам зосередитися на реалізації бізнес-логіки та покращенні користувацького досвіду.

Саме тому розроблення вебзастосунку для продажу та купівлі книг є актуальною задачею, оскільки дозволяє поглибити знання в області вебпрограмування, проєктування інформаційних систем та роботи з базами даних. Крім того, реалізація подібного проєкту може бути корисною з практичної точки зору та мати потенціал для подальшого комерційного використання.

Метою кваліфікаційної роботи є розроблення вебзастосунку для продажу та купівлі книг, який забезпечує зручну взаємодію користувача з системою та відповідає сучасним вимогам до вебплатформ електронної комерції.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати предметну область вебзастосунків електронної комерції;
- дослідити існуючі аналоги систем продажу книг та їх функціональні можливості;
- сформувати вимоги до функціоналу та архітектури системи;
- визначити архітектурний підхід та основні програмні патерни;
- спроєктувати структуру бази даних та основні модулі системи;
- обрати технології реалізації вебзастосунку;
- реалізувати основні функціональні модулі системи;
- виконати тестування розробленого програмного продукту;
- проаналізувати отримані результати та зробити висновки.

1.ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Ринок електронної комерції в Україні є одним із найбільш динамічних секторів економіки, який демонструє стабільне зростання протягом останніх років. Розвиток цифрових технологій, постійний доступ до Інтернету та зміна поведінки споживачів сприяли активному переходу торгівлі у онлайн-середовище. Особливе місце у структурі електронної комерції займає ринок книжкової продукції, який поєднує культурну, освітню та комерційну складові.

Згідно зі статистичними даними щодо активності українських інтернет-користувачів, близько 58% опитаних здійснюють покупки в інтернеті частіше ніж один раз на місяць. При цьому категорія «Книги» займає вагому частку ринку - 24% користувачів купують товари цієї групи онлайн. Загальна динаміка електронної комерції показує зростання кількості замовлень на 13%, а середній чек збільшився на 18% порівняно з попередніми періодами, що підтверджує фінансову перспективність розробки рішень у цій ніші. Статистичні дані зображені на Рисунку 1.1 [].

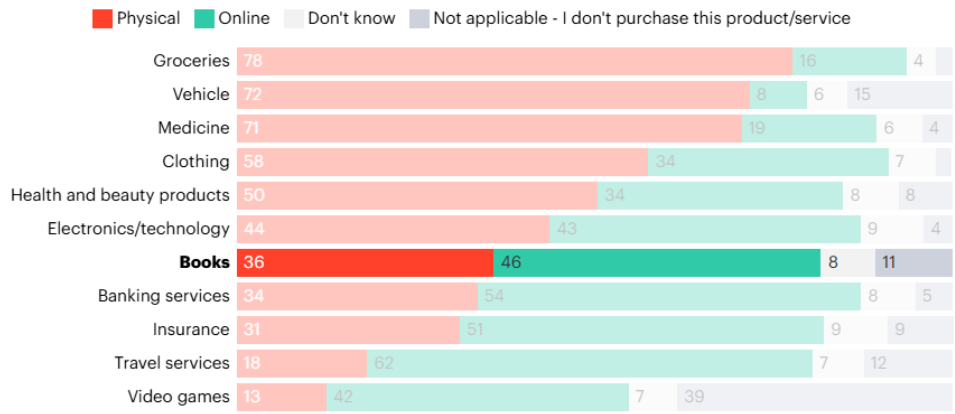


Рисунок 1.1 - Статистика розподілу споживчих переваг між фізичними та онлайн-покупками за категоріями товарів та послуг.

У сучасних умовах книжкова індустрія зазнала суттєвих змін. Якщо раніше вебресурси виконували переважно інформаційну функцію (онлайн-каталоги), то сьогодні вони трансформувалися у повноцінні інтернет-магазини, які забезпечують повний цикл взаємодії з користувачем: від пошуку бажаної книги до оформлення замовлення та його доставки. Онлайн-продаж книг сприяє популяризації читання, оскільки забезпечує швидкий доступ до широкого асортименту літератури незалежно від географічного розташування користувача.

На українському ринку функціонують провідні вебзастосунки для продажу та купівлі книжкової продукції, такі як Yakaboo, Книгарня «Є» та Vivat, які формують основні стандарти якості обслуговування, швидкості обробки замовлень та зручності користувацького інтерфейсу. Дані платформи пропонують широкий функціонал, включаючи детальну каталогізацію товарів, гнучкі системи пошуку та персоналізовані рекомендації.

Незважаючи на наявність потужних магазинів, актуальним залишається створення нових вебзастосунків, орієнтованих на конкретні сегменти ринку або з покращеним користувацьким досвідом. До таких сегментів можна віднести студентів та школярів, які потребують навчальної літератури; поціновувачів художньої літератури за окремими жанрами (фентезі, детективи, наукова фантастика); спеціалістів, зацікавлених у професійній літературі (IT, медицина, бізнес); батьків, які шукають дитячі книги; користувачів, що читають іноземними мовами; колекціонерів рідкісних та антикварних видань; прихильників електронних та аудіокниг; а також аудиторію, зацікавлену у вживаних книгах або продукції локальних авторів. Це обумовлює необхідність детального аналізу предметної області з метою визначення основних вимог до програмного забезпечення.

Вебзастосунок інтернет-магазину книг є складною інформаційною системою, яка забезпечує автоматизацію процесів продажу книжкової продукції. Основною

метою такого застосунку є надання користувачам зручного інструменту для перегляду каталогу, вибору товарів та оформлення замовлень.

Предметна область даного проєкту включає наступні основні компоненти:

- користувачі системи (покупці та адміністратори);
- каталог книг;
- процес оформлення замовлення;
- система управління замовленнями;
- механізми пошуку та фільтрації;
- система безпеки та авторизації.

Користувачі системи поділяються на дві основні категорії. Покупці здійснюють пошук товарів, перегляд інформації про книги, формують замовлення та взаємодіють із системою через інтерфейс. Адміністратори відповідають за наповнення каталогу, оновлення інформації, контроль замовлень та загальне управління системою.

Каталог книг є ключовим елементом системи та містить інформацію про товари. До основних характеристик книг належать: назва, автор, жанр, ціна, опис та наявність на складі. Ефективна організація каталогу дозволяє забезпечити швидкий доступ до інформації та зручність пошуку.

Важливою складовою є система пошуку та фільтрації, яка повинна забезпечувати можливість швидкого знаходження необхідних товарів за різними параметрами. Це особливо актуально для великих каталогів, що містять тисячі позицій.

Процес оформлення замовлення включає декілька етапів: додавання товарів до кошика, введення контактних даних, вибір способу доставки та підтвердження покупки. Від зручності та швидкості цього процесу значною мірою залежить рівень конверсії вебзастосунку.

З технічної точки зору вебзастосунок реалізується за клієнт-серверною архітектурою. Клієнтська частина відповідає за відображення інформації та взаємодію з користувачем, тоді як серверна частина забезпечує обробку запитів, реалізацію бізнес-логіки та роботу з базою даних.

Структура сучасного вебзастосунку інтернет-магазину включає декілька основних рівнів:

- рівень представлення (інтерфейс користувача), який відповідає за відображення сторінок;
- рівень бізнес-логіки, який реалізує основні функції системи;
- рівень доступу до даних, який забезпечує взаємодію з базою даних.

Для реалізації вебзастосунку доцільно використовувати сучасні технології, зокрема платформу ASP.NET Core, яка забезпечує високу продуктивність, масштабованість та безпеку. Для роботи з базою даних може використовуватися система управління базами даних MS SQL Server, а взаємодія з нею реалізується за допомогою технології Entity Framework Core.

Важливим аспектом є забезпечення безпеки системи. Вебзастосунок повинен гарантувати захист персональних даних користувачів, безпечне зберігання паролів та захист від несанкціонованого доступу. Для цього використовуються механізми аутентифікації та авторизації.

Аналіз предметної області також передбачає детальне визначення характеристик кінцевих користувачів, їх поведінки та очікувань від вебзастосунку. Основну частину аудиторії інтернет-магазинів книг складають користувачі віком орієнтовно від 18 до 40 років, серед яких студенти, молоді спеціалісти та активні читачі, що регулярно користуються цифровими сервісами. Вони мають достатній рівень цифрової грамотності та очікують інтуїтивно зрозумілого інтерфейсу, який не потребує додаткового навчання. Важливими є зручна структура сайту, швидкий доступ до основних функцій , таких як пошук, фільтрація, кошик, оформлення замовлення, а

також ефективна система пошуку з можливістю сортування за різними параметрами, такими як автор, жанр, ціна.

Користувачі також висувають високі вимоги до продуктивності та зручності використання вебзастосунку. Зокрема, вони очікують швидкого завантаження сторінок, адаптивного дизайну для коректної роботи на мобільних пристроях, а також наявності персоналізованих функцій, таких як рекомендації, списки бажаного та історія переглядів. Важливим фактором є довіра до сервісу, що включає безпечні способи оплати, прозору інформацію про доставку та повернення товарів, а також наявність служби підтримки. У сукупності ці характеристики визначають основні вимоги до розробки сучасного інтернет-магазину книг, орієнтованого на потреби користувачів.

У результаті проведеного аналізу можна зробити висновок, що розробка вебзастосунку інтернет-магазину книг є актуальною задачею. Такий застосунок повинен забезпечувати ефективну взаємодію користувача із системою, мати зручний інтерфейс, високу продуктивність та відповідати сучасним вимогам електронної комерції.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для проведення якісного аналізу існуючого програмного забезпечення у сфері електронної комерції книжкового напрямку необхідно врахувати специфічні особливості даної галузі. Вебзастосунки для продажу книг характеризуються складною структурою каталогів, наявністю розвинених механізмів пошуку та фільтрації, а також необхідністю забезпечення швидкої та надійної обробки замовлень.

При аналізі сучасних рішень доцільно звернути увагу на такі ключові аспекти:

- ручність та інтуїтивність інтерфейсу користувача (UI/UX);
- ефективність пошукової системи;
- функціональність кошика та процесу оформлення замовлення;
- можливості особистого кабінету користувача;
- продуктивність та швидкість роботи системи.

Для аналізу було обрано найбільш популярні вебзастосунки українського ринку книжкової продукції.

Першим розглянемо Yakaboo [1] (Рисунок 1.2). Це один із найбільших інтернет-магазинів книг в Україні, який пропонує широкий асортимент товарів, а саме друковані, електронні та аудіокниги, що задовільняє потреби різних користувачів.

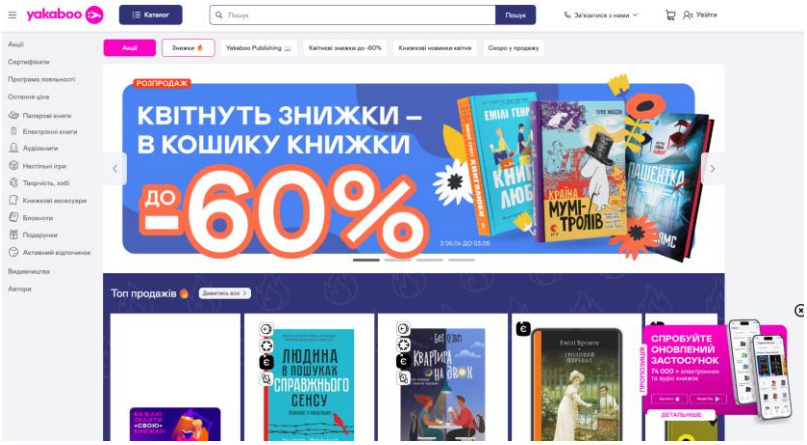


Рисунок 1.2- Головна сторінка вебзастосунку Yakaboo

Вебзастосунок відзначається масштабною базою даних та розвиненою системами пошуку та фільтрації. Користувач має можливість здійснювати пошук за різними параметрами, зокрема за автором, назвою, видавництвом та іншими характеристиками. Також реалізована багаторівнева система фільтрації, що дозволяє швидко знаходити необхідні товари, вона зображена на Рисунку 1.3. Особистий

кабінет користувача забезпечує перегляд історії замовлень, збереження обраних товарів та отримання персоналізованих рекомендацій. Якщо говорити про недоліки то можна зазначити що в даного застосунку перевантажений інтерфейс через велику кількість функцій, можливість затримки завантаження сторінок, надлишкова кількість інформації на сторінках, а також обмежена зручність мобільної версії.

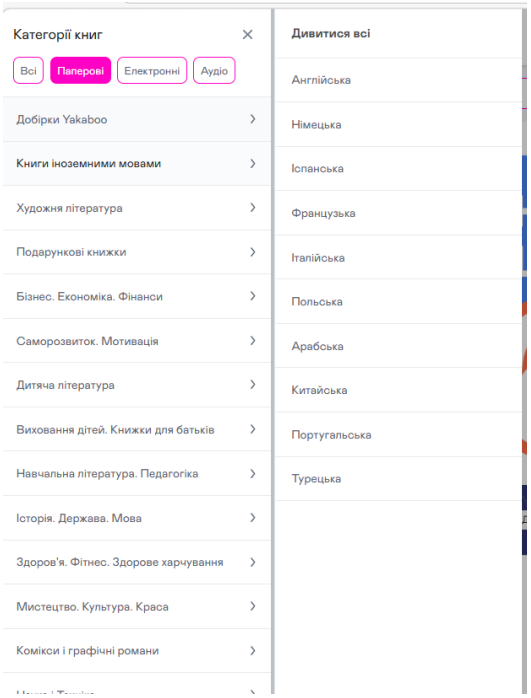


Рисунок 1.3- Система фільтрації Yakaboo

Наступним об’єктом аналізу є Книгарня «Є»[2] (Рисунок 1.4). Даний вебзастосунок відрізняється мінімалістичним дизайном та зручністю навігації. Особливу увагу приділено процесу оформлення замовлення, який є простим та зрозумілим для користувача. Система дозволяє швидко додавати товари до кошика та оформлювати покупку з використанням різних способів доставки. Крім того, сайт містить додатковий контент у вигляді статей та рецензій, що підвищує залученість користувачів. Водночас, незважаючи на зручність та простоту використання, вебзастосунок має і певні недоліки. Зокрема, обмежений асортимент товарів, також

інформація про окремі книги не є достатньо детальною, що може впливати на прийняття рішення про покупку.

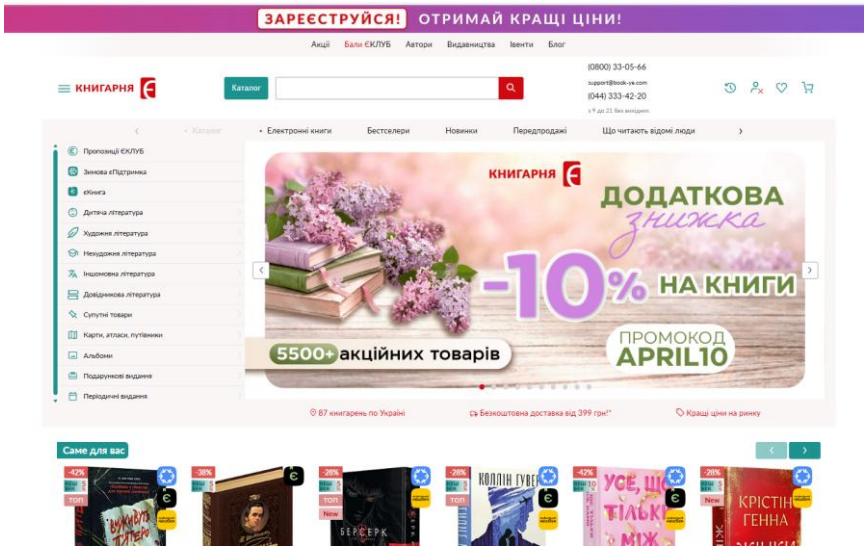


Рисунок 1.4- Головна сторінка вебзастосунку Книгарня «Є»

Далі розглянемо Vivat [3]. Вебзастосунок даного видавництва є прикладом брендового інтернет-магазину. Основною його особливістю є акцент на власній продукції та можливість оформлення передзамовлень. Це дозволяє користувачам замовляти книги ще до їх офіційного виходу. Приклад асортименту товарів зображено на Рисунку 1.5. Однією з його переваг є простий, зрозумілий та неперезагружений інтерфейс. Також система містить модуль відгуків, що дає змогу користувачам оцінювати продукцію та залишати коментарі.

Ще одним прикладом є Наш Формат [4]. Цей вебзастосунок спеціалізується не лише на друкованій літературі, але й на цифровому контенті. Серед його особливостей варто виділити можливість читання електронних книг та прослуховування аудіокниг безпосередньо через браузер (Рисунок 1.6). Водночас такі можливості потребують складнішої технічної реалізації, що впливає на продуктивність системи та швидкість

завантаження сторінок, особливо на слабших пристроях або при нестабільному інтернет-з’єднанні.

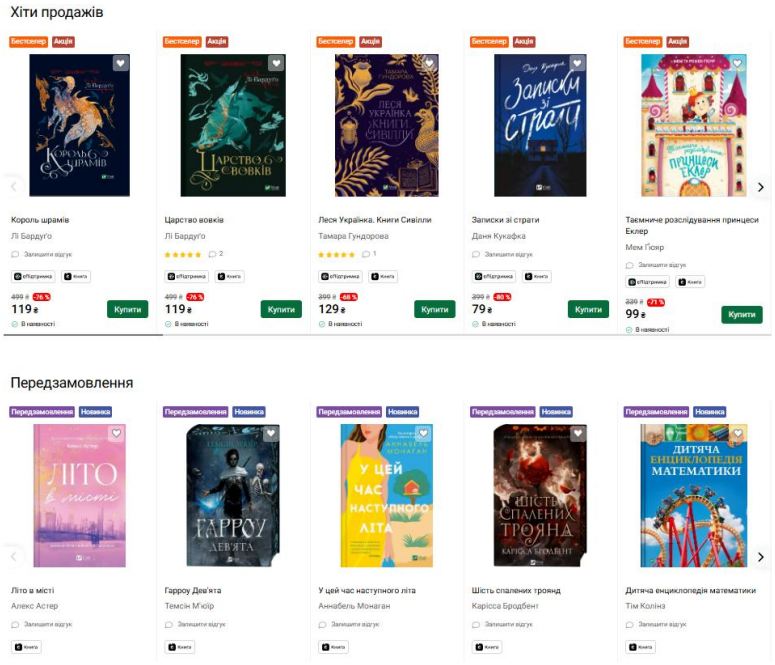


Рисунок 1.5- Асортимент товарів у магазині Vivat

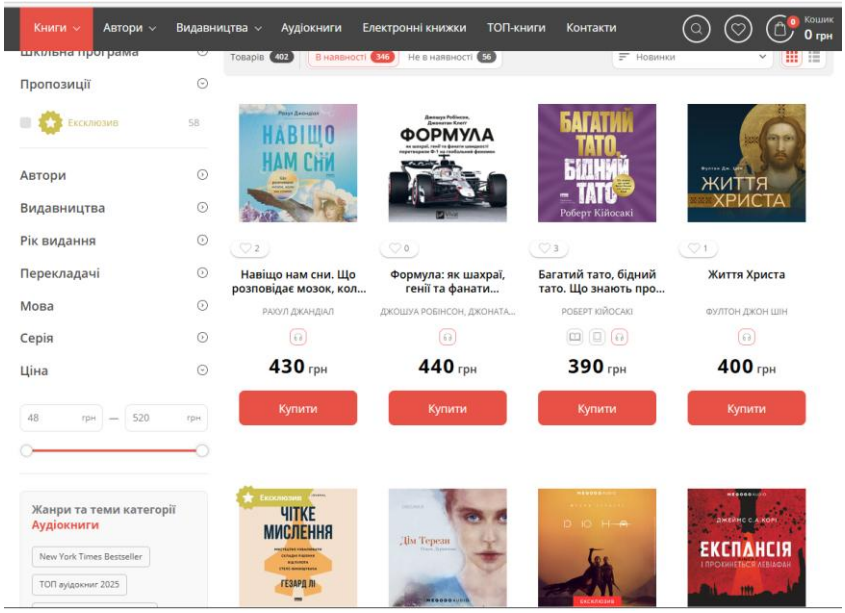


Рисунок 1.6- Модуль прослуховування аудіокниг у «Наш Формат»

Останнім у даному аналізі є Bookling [5], який спеціалізується на продажі іноземної літератури та навчальних матеріалів для вивчення мов. Вебзастосунок підтримує багатомовність інтерфейсу та має розширену систему фільтрації, зокрема за мовою, жанром і рівнем складності (A1–C2), що дозволяє користувачам швидко знаходити відповідні книги. Це робить платформу зручною для навчання та поступового підвищення рівня володіння мовою. Організація каталогу іноземної літератури зображена на Рисунку 1.7. Водночас така функціональність потребує якісної організації даних і гнучкої архітектури, оскільки обробляється велика кількість параметрів, а також може ускладнювати реалізацію інтерфейсу та підтримку багатомовності на всіх рівнях системи.

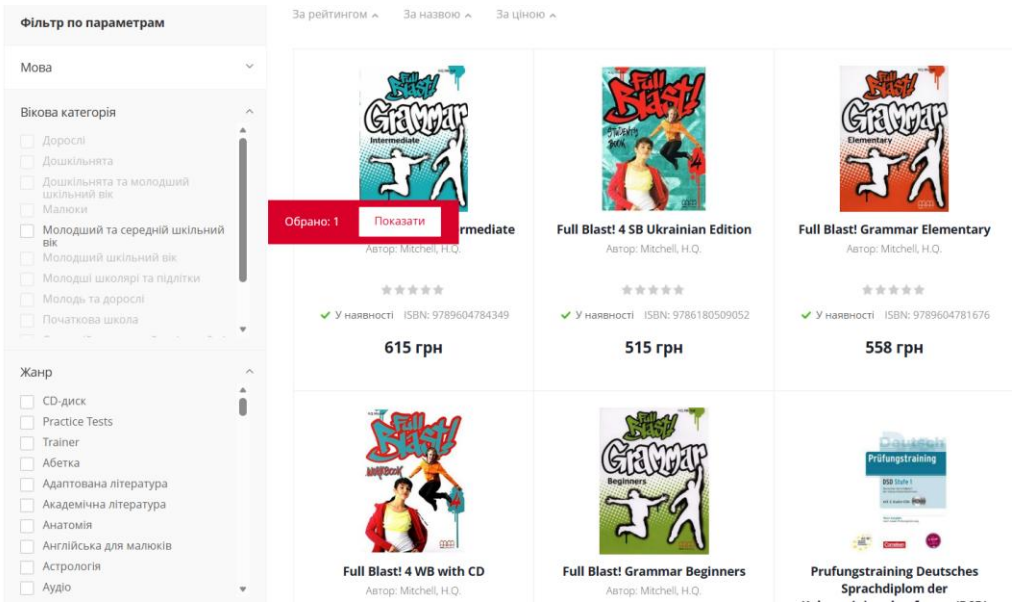


Рисунок 1.7 - Організація каталогу іноземної літератури на платформі Bookling

Отже, було створено порівняльну Таблицю 1, у якій наведено аналіз основних сучасних вебзастосунків у сфері електронної комерції книжкової продукції. У таблиці узагальнено ключові характеристики досліджених платформ, зокрема асортимент товарів, функціональність пошукових механізмів, зручність інтерфейсу користувача,

особливості оформлення замовлення, можливості особистого кабінету, а також додаткові функції та рівень продуктивності систем.

Критерій	Yakaboo	Книгарня «Є»	Vivat	Наш Формат	Bookling
Асортимент	Дуже широкий (друковані, електронні, аудіокниги)	Середній	Обмежений (власна продукція)	Середній (друк + цифровий контент)	Спеціалізований (іноземна література)
Пошук та фільтрація	Дуже розвинена, багаторівнева	Базова/середня	Базова	Середня	Розширена (мова, рівень A1–C2)
Інтерфейс користувача	Перевантажений	Мінімалістичний і зручний	Простий і зрозумілий	Сучасний, функціональний	Функціональний, але складний
Оформлення замовлення	Стандартне	Дуже просте та швидке	Стандартне	Стандартне	Стандартне
Особистий кабінет	Розширений (історія, рекомендації)	Базовий	Базовий	Базовий	Середній
Додаткові функції	Рекомендації, велика база	Статті, рецензії	Передзамовлення	Читання/прослуховування онлайн	Багатомовність, рівні складності
Продуктивність	Середня (інколи повільна)	Висока	Висока	Середня	Середня/нижча через складність
Недоліки	Перевантажений інтерфейс, повільність	Обмежений асортимент, мало деталей	Обмежений асортимент	Високе навантаження при мультимедіа	Складність інтерфейсу та підтримки

Таблиця 1 - Порівняння вебзастосунків книжкової електронної комерції

На основі проведеного аналізу можна виділити спільні характеристики, притаманні сучасним вебзастосункам у сфері продажу книг:

- наявність складної системи каталогізації товарів;
- використання розширених механізмів пошуку та фільтрації;
- підтримка системи відгуків та рейтингів;
- оптимізований процес оформлення замовлення;
- використання сучасних технологій для забезпечення швидкої роботи інтерфейсу.

Таким чином, проведений аналіз показує, що сучасні інтернет-магазини книг мають високий рівень функціональності та орієнтовані на забезпечення максимально зручного користування. Отримані результати будуть враховані при розробці власного вебзастосунку, що дозволить створити ефективну, зручну та конкурентоспроможну систему на основі платформи ASP.NET Core.

1.3 Визначення вимог до програмного забезпечення та розробка технічного завдання

На основі проведеного аналізу предметної області та існуючих рішень у сфері електронної комерції можна зробити висновок про доцільність розробки власного вебзастосунку для продажу та купівлі книг. Основною метою розробки є створення зручної, функціональної та надійної системи для продажу книжкової продукції, яка відповідатиме сучасним вимогам користувачів.

Розроблюваний вебзастосунок буде орієнтований на платформу веб-браузерів та реалізований з використанням сучасних технологій, зокрема платформи ASP.NET Core. Система повинна забезпечувати стабільну роботу, зручний інтерфейс та швидку обробку запитів користувачів.

Функціональні вимоги визначають перелік можливостей, які повинна забезпечувати система для задоволення потреб користувачів.

З функціональних можливостей для користувача можна виділити, що **с**истема повинна забезпечувати виконання наступних функцій:

- можливість реєстрації нового користувача шляхом введення персональних даних, таких як email або номер телефону;
- можливість авторизації користувача за допомогою логіна та пароля;
- перегляд загального каталогу книг;
- пошук книг за ключовими словами, такими як назва книги або ім’я автора;
- фільтрація книг за заданими параметрами, такими як жанр, ціна, видавництво;
- перегляд детальної інформації про книгу, а саме опис, ціна, наявність також відгуки попередніх покупців;
- додавання товарів до кошика;
- редагування вмісту кошика: зміна кількості, видалення товарів;
- оформлення замовлення із зазначенням контактних даних;
- вибір міста та відділення пошти для доставки.

З фунціональних можливоштей адміністративна частина системи повинна забезпечувати:

- додавання нових товарів до каталогу;
- редагування існуючих записів про книги;
- видалення товарів;
- управління категоріями товарів;
- перегляд списку замовлень;
- зміна статусу замовлень: нове, обробляється, виконано;
- управління обліковими записами користувачів.

Нефункціональні вимоги визначають якісні характеристики програмного забезпечення.

- Продуктивність. Система повинна забезпечувати швидке завантаження сторінок (не більше 2–3 секунд) та оперативну обробку запитів.

- Надійність. Вебзастосунок повинен працювати стабільно без критичних помилок та збоїв.
- Зручність використання (Usability). Інтерфейс повинен бути інтуїтивно зрозумілим для користувача та не вимагати додаткового навчання.

- Масштабованість. Архітектура системи повинна дозволяти розширення функціональності без значних змін у структурі.
- Сумісність. Вебзастосунок повинен коректно працювати у сучасних браузерах (Google Chrome, Microsoft Edge, Mozilla Firefox).

Безпека є важливою складовою вебзастосунку. Система повинна забезпечувати:

- аутентифікацію користувачів;
- авторизацію доступу до функцій системи;
- шифрування паролів;
- захист від SQL-ін’єкцій;
- захист від XSS-атак;
- захист персональних даних користувачів.

У системі визначено два основні типи користувачів: покупець та адміністратор**.**

Варіанти використання для покупця:

1. Робота з каталогом:
    - перегляд товарів;
    - пошук та фільтрація;
    - перегляд деталей товару.
2. Оформлення замовлення:
    - додавання до кошика;
    - редагування кошика;

– підтвердження замовлення.

3. Робота з профілем:

– реєстрація;
– авторизація;
– можливість оформити замовлення.

Варіанти використання для адміністратора:

– управління каталогом;
– управління замовленнями;
– управління користувачами.

На основі цих варіантів використання будується UML-діаграма (Use Case Diagram), яка відображає взаємодію користувачів із системою.

1.4 Висновки. Постановка задачі

Отже,у цьому розділі було проведено комплексний аналіз предметної області електронної комерції у сфері продажу та купівлі книг. Визначено основні процеси, що відбуваються в інтернет-магазинах, а також досліджено функціональні можливості сучасних вебзастосунків.

На основі проведеного аналізу було сформовано функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення. Визначено основних користувачів системи та їхні потреби.

Для реалізації поставленої мети необхідно виконати наступні задачі:

– дослідити сучасні підходи до проєктування вебзастосунків;

- розробити архітектуру системи;
- спроєктувати базу даних;
- створити інтерфейс користувача;
- реалізувати функціональні можливості системи;
- забезпечити безпеку даних;
- провести тестування програмного забезпечення;
- проаналізувати результати тестування;
- сформулювати висновки.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір типу архітектури та шаблонів проєктування

Спершу необхідно розглянути, які типи архітектур використовуються для розробки вебзастосунків, зокрема інтернет-магазинів, адже вони мають забезпечувати надійність, масштабованість та зручність підтримки. Архітектура вебзастосунків може значно відрізнятися залежно від вимог до функціональності, навантаження та способу взаємодії з користувачем. Нижче наведено основні підходи, що використовуються при створенні сучасних вебсистем.

Монолітна архітектура [6] — це підхід, за якого весь вебзастосунок реалізується як єдина система, що об’єднує інтерфейс користувача, бізнес-логіку та доступ до даних в одному проєкті або сервері. Така архітектура є простою у реалізації, забезпечує швидку розробку та спрощене розгортання завдяки єдиній кодовій базі. Водночас зі збільшенням функціональності ускладнюється підтримка системи, оскільки зміни в одному модулі можуть впливати на інші, що негативно позначається на масштабованості та командній розробці. Для кращого розуміння даної архітектури нижче на Рисунку 2.1 наведена її структурна схема.

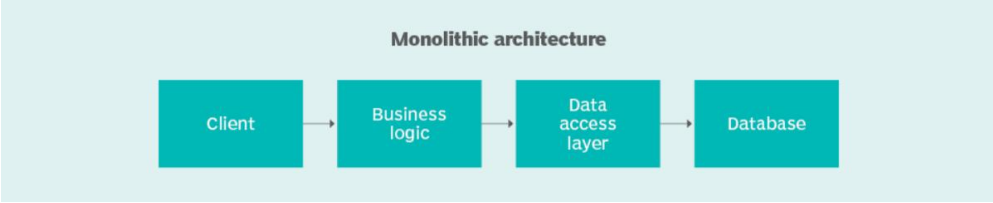


Рисунок 2.1 - Структурна схема монолітної архітектури програмного забезпечення

Мікросервісна архітектура [7] – це модель, яка розбиває одну систему на набір незалежних сервісів, кожен з яких відповідає за окрему функцію. Наприклад для

вебзастосунку для продажу та купівлі книг :каталог книг, обробка замовлень, управління користувачами та платежі. Кожен сервіс може розроблятися, тестуватися та масштабуватися незалежно. Такий підхід забезпечує високу гнучкість і масштабованість, але ускладнює розгортання та потребує додаткових механізмів для комунікації між сервісами. Структурна схема взаємодії мікросерверів зображена на Рисунку 2.2.

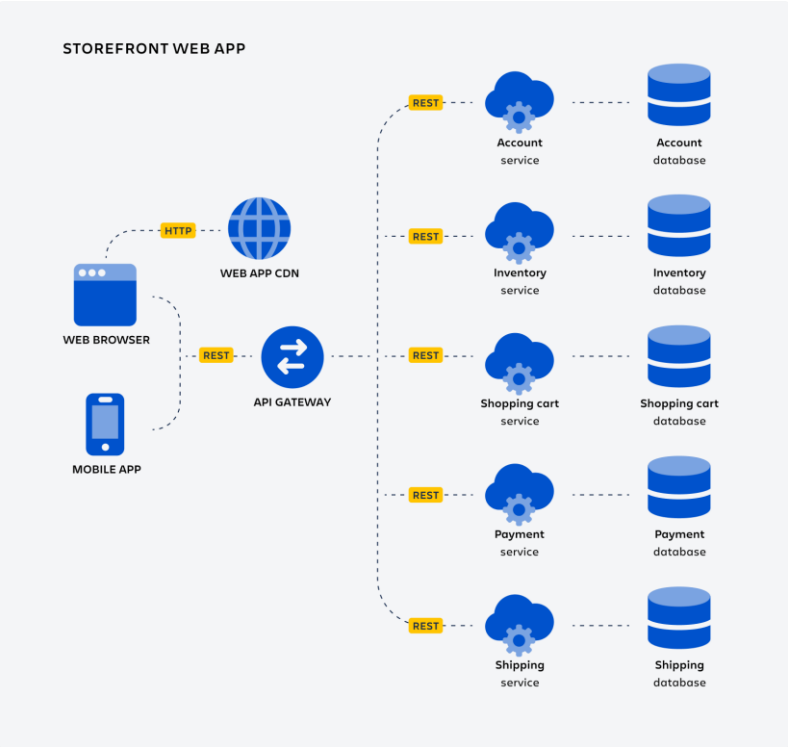


Рисунок 2.2 - Структурна схема взаємодії мікросервісів

Шарова архітектура [8] — це підхід до розробки вебзастосунків, який передбачає поділ системи на окремі логічні шари: представлення, бізнес-логіки, доступу до даних та бази даних. Кожен шар виконує визначені функції та взаємодіє лише із суміжними рівнями, що забезпечує чітке розділення відповідальностей і зменшує складність системи. Такий підхід спрощує розробку, тестування, підтримку та масштабування застосунку, оскільки окремі компоненти можуть змінюватися

незалежно. Водночас надмірна кількість шарів може негативно впливати на продуктивність, тому на практиці зазвичай використовують 3–4 рівні, приклад розподілу шарів зображено на Рисунку 2.3.

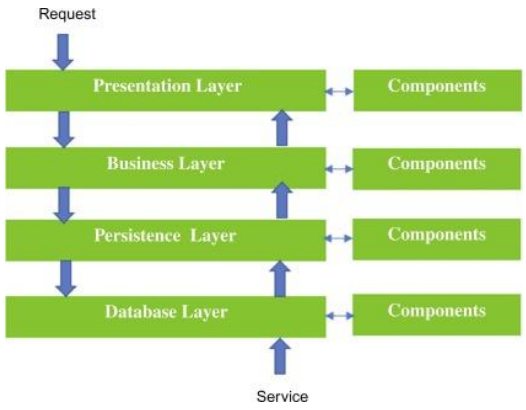


Рисунок 2.3 – Розподілення шарів в шаровій архітектурі

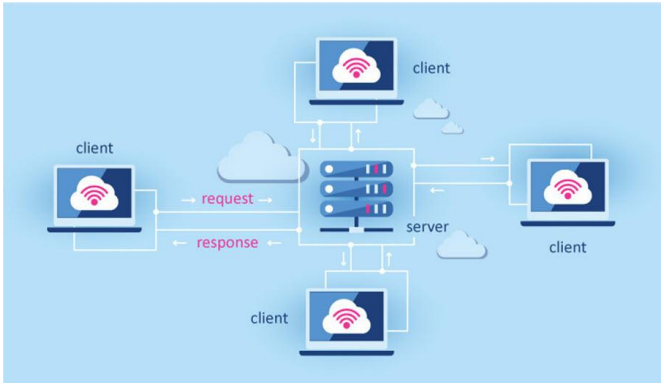


Рисунок 2.4- Принцип роботи клієнт-серверної архітектури

Клієнт-серверна архітектура [9] [10] — це модель побудови програмного забезпечення, у якій система поділяється на клієнтську та серверну частини. Клієнт відповідає за взаємодію з користувачем і формування запитів, а сервер — за обробку запитів, зберігання даних та надання ресурсів. Взаємодія між ними здійснюється за принципом «запит–відповідь» через мережу. Такий підхід забезпечує централізоване

керування даними, підтримку одночасної роботи багатьох користувачів, ефективне використання ресурсів і можливість масштабування системи, що робить клієнт-серверну архітектуру основою більшості сучасних вебзастосунків. Принцип роботи клієнт-серверної архітектури зображено на Рисунку 2.4.

MVC (Model-View-Controller) [11] - це архітектурний шаблон, який розділяє застосунок на три основні компоненти: модель, представлення та контролер. Модель – відповідає за дані застосунку та бізнес-логіку, зберігає стан системи і виконує всі необхідні обчислення та операції. Представлення – відповідає за відображення даних користувачу. Контролер – відповідає за обробку запитів користувача, взаємодіє з моделлю та обирає, яке представлення треба відобразити користувачу для показу результатів обробки даних**.** Такий підхід забезпечує розподіл відповідальностей, що спрощує розробку, тестування та супровід системи, оскільки зміни в інтерфейсі не зачіпають бізнес-логіку, а модель може функціонувати незалежно від інших частин системи, що є особливо важливим для побудови масштабованих і гнучких вебзастосунків.

MVVM (Model-View-ViewModel) [12] — це архітектурний шаблон побудови інтерфейсу користувача, який передбачає поділ застосунку на три компоненти: модель, представлення та модель представлення. Модель відповідає за дані та бізнес-логіку системи, представлення — за відображення інтерфейсу користувача, а модель представлення забезпечує взаємодію між ними, обробляючи дії користувача та синхронізуючи дані. Використання MVVM забезпечує чіткий розподіл відповідальностей, що спрощує розробку, тестування та супровід вебзастосунку, а також підвищує його масштабованість і гнучкість.

Після аналізу розглянутих архітектурних підходів і шаблонів проєктування можна зробити висновок, що для розробки вебзастосунку для продажу та купівлі книг найбільш доцільним є поєднання кількох підходів, а не використання лише одного. Монолітна архітектура підходить лише на початковому етапі або для невеликих проєктів, оскільки забезпечує простоту розробки, однак у випадку інтернет-магазину

вона швидко стає складною для масштабування та підтримки через зростання кількості функцій, таких як каталог, замовлення, оплати і користувачі.

Клієнт-серверна архітектура є базовою та обов'язковою, оскільки саме вона забезпечує взаємодію користувача з системою через браузер і серверну частину, де обробляються запити, зберігаються дані та виконується бізнес-логіка. Вона добре підходить для вебзастосунків і є основою більшості сучасних систем.

Шарова архітектура є найбільш практичною для реалізації такого проєкту, оскільки дозволяє чітко розділити систему на рівні, а саме UI, бізнес-логіка, доступ до даних. Саме це значно спрощує розробку, тестування та подальше масштабування. Даний підхід найчастіше використовується у вебзастосунках на базі MVC та .NET-платформ.

MVC та MVVM виступають як шаблони проєктування, які забезпечують структуровану організацію коду на рівні інтерфейсу. Для вебзастосунку MVC є більш природним вибором, оскільки він добре інтегрується з серверною логікою, дозволяє розділити дані, логіку та представлення, а також широко підтримується у вебфреймворках. MVVM частіше використовується у настільних та клієнтських застосунках, але може застосовуватись у сучасних frontend-рішеннях.

Розподілена архітектура є перспективною для великих інтернет-магазинів із високим навантаженням,однак для навчального або середнього за розміром проєкту її використання може бути надлишковим.

Таким чином, оптимальним рішенням для розробки вебзастосунку є комбінація клієнт-серверної та шарової архітектури з використанням MVC як основного шаблону проєктування. Це дозволяє досягти балансу між простотою реалізації, продуктивністю, масштабованістю та зручністю подальшого супроводу системи.

2.2 Опис декомпозиції

2.2.1 Загальна декомпозиція

Після того, як архітектура була обрана необхідно провести декомпозицію проєкту за основними напрямками. Декомпозиція архітектури [13] є одним з ключових принципів проєктування ПЗ, вона являє собою процес поділу складної системи на окремі більш керовані компоненти, що взаємодіють між собою. Такий підхід дозволяє суттєво зменшити складність системи, спростити розробку, підвищити зрозумілість структури, а також забезпечити зручність її подальшого тестування, супроводу та масштабування.

Загальна декомпозиція є початковим рівнем деталізації системи та має на меті формування цілісного уявлення про її структуру. На даному етапі було виділено три основні рівні системи: рівень представлення та інтерфейсів, рівень бізнес-логіки та рівень даних і зовнішніх інтеграцій.

У результаті проведеного аналізу було виділено три основні рівні системи: рівень представлення та інтерфейсів, рівень бізнес-логіки та рівень даних і зовнішніх інтеграцій.

Рівень представлення та інтерфейсів забезпечує безпосередню взаємодію користувача із системою та відповідає за відображення інформації у зручному для сприйняття вигляді. Основним компонентом цього рівня є клієнтський веб-інтерфейс, який реалізує функціональні можливості взаємодії користувача із системою. До його складу входять сторінки каталогу книг, сторінка перегляду окремого товару, кошик покупця, сторінка оформлення замовлення, а також особистий кабінет користувача. Даний інтерфейс забезпечує навігацію між розділами, обробку дій користувача та передачу відповідних запитів до серверної частини. Важливим аспектом є також забезпечення зручності використання, адаптивності та швидкодії інтерфейсу.

Окрім користувацького інтерфейсу, до рівня представлення належить інтерфейс адміністратора, який реалізується у вигляді адміністративної панелі. Цей компонент призначений для керування вмістом системи та її функціонуванням. За допомогою адміністративної панелі здійснюється додавання, редагування та видалення книг, управління товарами, користувачами та замовленнями, а також перегляд статистичної інформації щодо роботи системи. Таким чином, даний інтерфейс забезпечує ефективне адміністрування вебзастосунку.

Рівень бізнес-логіки виконує центральну роль у роботі системи, оскільки саме на цьому рівні реалізується основна функціональність застосунку та обробляються запити, що надходять від клієнтської частини. Він забезпечує координацію роботи всіх компонентів системи, перевірку даних, виконання обчислень та прийняття рішень відповідно до заданих правил.

До складу цього рівня входять декілька модулів, а саме модуль управління користувачами, каталогу та пошуку, обробки замовлень та кошика, управління замовленнями, маркетингу та лояльності, повідомлень та інтеграції платіжних сервісів.

Модуль управління користувачами забезпечує реєстрацію нових користувачів, їх аутентифікацію та авторизацію, а також управління власними профілями. Наступний важливий компонент - модуль каталогу та пошуку, який відповідає за обробку інформації про книги, реалізацію механізмів пошуку, фільтрації та сортування за різними параметрами, такими як назва, автор, жанр або ціна.

Модуль обробки замовлень та кошика реалізує функціональність додавання товарів до кошика, зміни їх кількості, розрахунку загальної вартості замовлення та підготовки даних для його оформлення. У свою чергу, модуль управління замовленнями відповідає за обробку життєвого циклу замовлення, включаючи зміну його статусів, таких як створено, оплачено, відправлено, доставлено або скасовано.

Також слід виділити модуль взаємодії з покупцями, який включає систему відгуків та рейтингів. Цей модуль дозволяє користувачам залишати оцінки та коментарі щодо книг, що сприяє формуванню довіри до товарів та покращенню користувацького досвіду. Додатково реалізується модуль повідомлень, який забезпечує інформування користувачів про важливі події, такі як зміна статусу замовлення або проведення акцій.

Ще одною важливою складовою рівня бізнес-логіки є модуль інтеграції платіжних сервісів, який забезпечує взаємодію із зовнішніми системами для здійснення онлайн-оплати. Він відповідає за обробку транзакцій, перевірку їх статусу та передачу відповідної інформації до інших компонентів системи.

Рівень даних та зовнішніх інтеграцій відповідає за збереження, обробку та надання доступу до інформаційних ресурсів системи, а також за взаємодію із зовнішніми сервісами. Основним елементом цього рівня є центральна база даних, яка реалізована у вигляді реляційної системи керування базами даних та містить структуровану інформацію про користувачів, книги, категорії, замовлення, відгуки та історію покупок.

Для зберігання додаткових ресурсів, таких як зображення обкладинок книг або інші медіа-файли, використовується окреме сховище статичних даних. Це дозволяє оптимізувати роботу системи та зменшити навантаження на базу даних.

Крім того, на даному рівні реалізується інтеграція із зовнішніми API. До них належать платіжні сервіси та сервіси електронної пошти. Використання таких інтеграцій дозволяє розширити функціональні можливості вебзастосунку, автоматизувати окремі процеси та підвищити зручність для кінцевого користувача.

Отже, проведена загальна декомпозиція дозволила виділити основні рівні та компоненти вебзастосунку для продажу та купівлі книг, визначити їх призначення та роль у загальній архітектурі системи. Запропонована структура забезпечує чітке розмежування відповідальності між складовими, сприяє підвищенню гнучкості та

масштабованості системи, а також створює надійну основу для подальшої деталізації та реалізації програмного продукту.

2.2.2 Модульна декомпозиція

Після проведення загальної декомпозиції доцільно виконати більш детальну модульну декомпозицію системи. Першочергово необхідно виділити основні керуючі модулі, які забезпечують роботу всього вебзастосунку. Одним із таких є модуль ядра системи, який відповідає за загальну конфігурацію, ініціалізацію застосунку та взаємодію між іншими модулями.

Важливим модулем є модуль користувачів, який забезпечує реєстрацію, авторизацію, управління профілем користувача та збереження його персональних даних. Цей модуль відповідає за безпеку доступу до системи та взаємодію користувача із застосунком.

Для реалізації основного функціоналу інтернет-магазину необхідно виділити модуль каталогу книг, який відповідає за відображення списку товарів, пошук, фільтрацію та перегляд детальної інформації про книги. Він працює з базою даних та надає інформацію користувачу у зручному вигляді.

Окремим модулем є модуль кошика, який дозволяє користувачу додавати товари, змінювати їх кількість, видаляти позиції та формувати замовлення. Цей модуль забезпечує тимчасове збереження вибраних товарів до моменту оформлення покупки.

Модуль замовлень відповідає за створення, обробку та збереження замовлень користувача. Він взаємодіє з модулем кошика, отримує дані про обрані товари та формує повноцінне замовлення.

Для реалізації фінансових операцій необхідно виділити модуль оплати, який забезпечує обробку платежів, взаємодію з платіжними системами та перевірку статусу транзакцій.

Модуль доставки відповідає за обробку інформації про місто та відділення пошти, розрахунок вартості та збереження даних користувача для відправлення товару.

Також важливим є модуль управління товарами (адміністративний модуль), який дозволяє додавати, редагувати та видаляти книги, керувати категоріями та оновлювати інформацію про товари.

Для взаємодії з користувачем необхідно виділити модуль інтерфейсу користувача, який відповідає за відображення даних, обробку дій користувача та забезпечення зручного UX/UI.

Окремо можна виділити модуль повідомлень, який відповідає за надсилання сповіщень користувачу (наприклад, підтвердження замовлення або зміна його статусу).

Для збереження та отримання даних необхідно реалізувати модуль доступу до даних, який забезпечує взаємодію з базою даних, виконання CRUD-операцій та ізоляцію бізнес-логіки від роботи з даними.

У результаті модульної декомпозиції було виділено такі основні модулі системи:

- модуль ядра системи;
- модуль користувачів;
- модуль каталогу книг;
- модуль кошика;
- модуль замовлень;
- модуль оплати;

- модуль доставки;
- модуль управління товарами;
- модуль інтерфейсу користувача;
- модуль повідомлень;
- модуль доступу до даних.

Таким чином, система розбита на незалежні модулі, кожен з яких відповідає за окремий функціонал, що спрощує розробку, тестування та подальше масштабування вебзастосунку.

2.2.3 Декомпозиція моделі переходів станів

Декомпозиція моделі переходів станів є етапом проєктування, що передбачає формалізацію поведінки системи через виділення окремих станів та умов переходів між ними. Це забезпечує структурованість логіки, спрощує реалізацію, тестування та супровід вебзастосунку.

Для вебзастосунку з продажу та купівлі книг основним об’єктом моделі переходів станів є замовлення, оскільки воно проходить кілька етапів обробки. Основними станами замовлення є: створене, оформлене, очікує оплати, оплачено, обробляється, відправлено, доставлено та скасовано. Використання таких станів дозволяє контролювати процес виконання замовлення та забезпечує коректну взаємодію між компонентами системи.

Переходи між цими станами відбуваються внаслідок дій користувача (оформлення, оплата, скасування) або дій системи (підтвердження оплати, зміна статусу доставки). Така модель дозволяє чітко контролювати життєвий цикл замовлення та уникати некоректних ситуацій, наприклад повторної обробки або втрати даних.

Наступним важливим об’єктом системи є користувач, який взаємодіє з вебзастосунком. Для нього визначено такі стани: неавторизований, у процесі

реєстрації, авторизований, редагування профілю та вихід із системи. Переходи між станами відбуваються під час виконання відповідних дій користувача та забезпечують контроль доступу до функціоналу системи і безпеку даних.

Також важливим елементом є кошик користувача, який виступає проміжною ланкою між переглядом товарів та оформленням замовлення. Основними станами кошика є: порожній, містить товари, оновлюється, готовий до оформлення та перетворений у замовлення. Це дозволяє коректно відстежувати дії користувача та передавати дані до модуля замовлень.

Окремо визначено стани процесу оплати: ініційована, в обробці, успішна, неуспішна та скасована. Їх використання забезпечує коректну обробку фінансових операцій, синхронізацію з платіжними системами та зменшує ймовірність помилок.

Модель станів також застосовується до інтерфейсу користувача. Для цього передбачено стани: активний, неактивний, стан завантаження, стан помилки та оновлення даних. Це забезпечує коректне відображення інформації та покращує взаємодію користувача із системою.

Таким чином, модель станів формалізує життєвий цикл ключових сутностей системи та забезпечує коректність і передбачуваність її роботи.

2.3 Опис залежностей

Опис залежностей є важливим етапом при проєктуванні вебзастосунку інтернет-магазину книг, оскільки дозволяє визначити, які модулі системи взаємодіють між собою, яким чином відбувається обмін даними та які компоненти залежать один від одного. Це дає змогу забезпечити правильну організацію системи, підвищити її масштабованість, гнучкість і спростити подальшу підтримку та розширення

функціоналу. Опис залежностей може бути представлений у вигляді списків або діаграм, що відображають напрямки взаємодії між модулями та їх ролі у системі.

Першим етапом є визначення міжмодульних залежностей, які формують основу архітектури системи. Для інтернет-магазину книг було виділено основні модулі та встановлено зв’язки між ними:

- модуль користувача взаємодіє з модулем авторизації та реєстрації для забезпечення доступу до персонального кабінету;
- модуль авторизації звертається до бази даних користувачів для перевірки облікових даних;
- модуль каталогу книг взаємодіє з базою даних для отримання інформації про книги, їх опис, авторів, жанри та наявність;
- модуль пошуку та фільтрації використовує дані каталогу книг для формування результатів відповідно до запиту користувача;
- модуль кошика взаємодіє з модулем каталогу для додавання обраних книг та з модулем замовлень для подальшого оформлення покупки;
- модуль замовлень звертається до модуля користувача, кошика та платіжного модуля для обробки покупки;
- платіжний модуль взаємодіє із зовнішніми платіжними сервісами та повертає результат обробки платежу в модуль замовлень;
- модуль доставки отримує дані із модуля замовлень та забезпечує розрахунок і вибір способу доставки;
- модуль відгуків і рейтингів взаємодіє з модулем користувача та каталогом книг для збереження і відображення оцінок і коментарів;
- модуль адміністратора взаємодіє з каталогом книг, замовленнями та користувачами для управління контентом, обробки замовлень та контролю системи;
- модуль інтерфейсу користувача (UI) взаємодіє з усіма основними модулями, забезпечуючи відображення даних і обробку дій користувача.

На основі визначених залежностей формується структура взаємодії модулів, яка дозволяє чітко зрозуміти потоки даних у системі та визначити ключові точки інтеграції, а також була побудована діаграма міжмодульних зв’язків, яка зображена на Рисунку А.1 у додатку А. Також варто зазначити, що такий підхід дає можливість оптимізувати процес розробки, уникнути надмірної зв’язаності між компонентами та забезпечити можливість незалежного вдосконалення окремих частин системи без впливу на інші. Це особливо важливо для інтернет-магазину книг, який повинен ефективно обробляти велику кількість користувачів, замовлень і даних у реальному часі.

2.4 Опис інтерфейсу модулів

Інтерфейси модулів інтернет-магазину книг забезпечують взаємодію між компонентами системи, приховуючи внутрішню реалізацію та захищаючи дані від некоректного використання. Такий підхід дозволяє дотримуватись принципу інкапсуляції, спрощує тестування та забезпечує можливість незалежного розвитку окремих модулів.

Кожен модуль надає чітко визначений набір публічних методів, через які інші частини системи можуть отримувати або змінювати дані. На основі цього було визначено основні інтерфейси модулів та сформовано Таблицю 2.1. Вони забезпечують узгоджену взаємодію всіх компонентів системи та дозволяють реалізувати гнучку, масштабовану та зручну у підтримці архітектуру інтернет-магазину книг.

Модуль	Призначення	Основні інтерфейси
Модуль користувача	Керування даними	GetUser(), UpdateUser(), IsAdmin()

	користувача	
Модуль авторизації	Реєстрація та доступ до системи	Login(), Register(), Logout()
Модуль каталогу книг	Управління книгами та їх даними	GetBooks(), GetBookById(), AddBook(), UpdateBook(), DeleteBook()
Модуль пошуку та фільтрації	Пошук і відбір книг	SearchBooks(query), FilterBooks(params)
Модуль кошика	Керування вибраними товарами	AddToCart(bookId), RemoveFromCart(bookId), GetCart(), ClearCart()
Модуль замовлень	Обробка та управління замовленнями	CreateOrder(), GetOrders(), GetOrderById(), UpdateOrderStatus()
Модуль оплати	Обробка платежів	ProcessPayment(orderId, paymentData), CheckPaymentStatus()
Модуль доставки	Управління доставкою	CalculateDelivery(), SetDeliveryMethod(), GetDeliveryInfo()
Модуль відгуків і рейтингів	Робота з відгуками користувачів	AddReview(), GetReviews(bookId), DeleteReview()
Модуль адміністратора	Адміністративне управління системою	ManageUsers(), ManageBooks(), ManageOrders()
UI-модуль	Відображення даних та взаємодія з користувачем	RenderPage(), ShowMessage(), UpdateView()

Таблиця 2.1 - Модулі системи та їх інтерфейси

2.5 Проєктування інтерфейсу користувача

На цьому етапі необхідно визначити, які функції повинен підтримувати інтерфейс користувача інтернет-магазину книг, сформувати перелік основних екранів та створити їх макети, що відображають структуру та розміщення елементів.

Інтерфейс має бути зручним, зрозумілим та забезпечувати швидкий доступ до основного функціоналу системи.

У результаті аналізу функціональних вимог визначено дві основні групи екранів вебзастосунку: основні сторінки магазину та допоміжні службові екрани. Основні сторінки забезпечують навігацію користувача та взаємодію з каталогом книг, тоді як допоміжні екрани підтримують процес оформлення замовлення та адміністрування системи.

До основних сторінок належать головна сторінка, каталог книг, сторінка книги та сторінка кошика. Головна сторінка містить популярні книги, категорії та засоби пошуку. Каталог відображає перелік доступних книг. Сторінка книги надає детальну інформацію про видання, зокрема опис, автора, ціну, рейтинг і відгуки, а також функцію додавання товару до кошика. Сторінка кошика забезпечує перегляд обраних товарів, зміну їх кількості та перехід до оформлення замовлення.

До допоміжних екранів віднесено сторінки авторизації та реєстрації, сторінку оформлення замовлення та адміністративну панель. Вони забезпечують автентифікацію користувачів, вибір способів доставки й оплати, підтвердження покупки, а також керування книгами, замовленнями та користувачами.

Крім окремих сторінок, інтерфейс містить спільні елементи, які використовуються в більшості екранів: навігаційне меню, інформаційні повідомлення та футер. Їх використання забезпечує цілісність інтерфейсу та зручність взаємодії користувача із системою.

При проєктуванні макетів важливо визначити розміщення елементів. Наприклад, панель навігації зазвичай розміщується у верхній частині сторінки, пошук - у центрі, список товарів - у центральній частині, а кошик- у правому верхньому куті. Сторінка книги повинна містити зображення, опис і кнопку покупки у видимій зоні, щоб користувач міг швидко прийняти рішення.

Отже, під час проєкутвання ікористувацького інтерфейсу були розроблені такі макети : головна сторінка (Рисунок А.2), каталог книг(Рисунок А.3), кошик (Рисунок А.4), сторінка авторизації та реєстрації (Рисунок А.5) та панель адміністратора (Рисунки А.6-А.9). Їх можна переглянути в Додатку А.

Розуміння структури інтерфейсу та створення макетів дозволяє ще на етапі проєктування оцінити зручність використання системи, логіку навігації та взаємодію користувача з основними функціями. Це значно спрощує подальшу реалізацію інтернет-магазину книг та забезпечує якісний користувацький досвід.

2.6Детальне проєктування модулів

Визначення основних модулів інтернет-магазину книг було виконано в попередніх розділах, де було описано їх функціональне призначення та загальні зв’язки між ними. Наступним етапом є детальне проєктування модулів, що передбачає уточнення їх структури, взаємозв’язків та поведінки, що дозволяє сформувати цілісне уявлення про архітектуру системи та забезпечити ефективну реалізацію програмного продукту.

Найвищим рівнем ієрархії системи є вебзастосунок (Application), який об’єднує всі модулі та забезпечує їх взаємодію. Центральним елементом є модуль керування запитами (ApplicationController), який приймає запити від користувача через інтерфейс (UI) та перенаправляє їх до відповідних модулів. Він взаємодіє з модулями користувачів, каталогу книг, кошика, замовлень та інших підсистем.

Модуль каталогу книг (CatalogModule) містить у собі колекцію об’єктів Book, що представляють окремі книги. Зв’язок між CatalogModule та Book визначається як один до багатьох. Кожен об’єкт Book містить інформацію про назву, автора, ціну,

опис, рейтинг та інші характеристики. Каталог взаємодіє з модулем пошуку (SearchModule), який здійснює фільтрацію та пошук книг за заданими параметрами.

Модуль користувача (UserModule) містить об’єкти User, що зберігають інформацію про зареєстрованих користувачів. Цей модуль пов’язаний з модулем авторизації (AuthModule), який відповідає за перевірку облікових даних та керування сесіями. Один користувач може мати декілька замовлень, що визначає зв’язок один до багатьох між User та Order.

Модуль кошика (CartModule) агрегує об’єкти CartItem, кожен з яких містить інформацію про обрану книгу та її кількість. CartModule пов’язаний з User (один до одного) та CatalogModule (багато до одного), оскільки користувач може додавати до кошика книги з каталогу.

Модуль замовлень (OrderModule) містить об’єкти Order, які формуються на основі даних кошика. Кожне замовлення пов’язане з користувачем, платіжним модулем (PaymentModule) та модулем доставки (DeliveryModule). PaymentModule відповідає за обробку платежів і взаємодію із зовнішніми платіжними системами, а DeliveryModule — за розрахунок і організацію доставки.

Модуль відгуків (ReviewModule) пов’язаний як з користувачами, так і з книгами, що визначає зв’язок багато до одного для обох сутностей. Користувач може залишати відгуки для книг, а система зберігає та відображає їх у каталозі.

На основі деталізації зв’язків між модулями формується діаграма класів (Рисунок А.10 у додатку А), яка відображає структуру системи, зв’язки між об’єктами та їх інтерфейси. Така діаграма дозволяє краще зрозуміти внутрішню організацію застосунку та є основою для реалізації.

Для опису взаємодії між модулями доцільно використовувати UML-діаграми послідовності [14] , які відображають сценарії роботи системи. Першим важливим сценарієм є процес авторизації користувача (Рисунок 2.5). У цьому випадку

користувач через UI вводить свої дані, після чого ApplicationController передає їх до AuthModule. Модуль авторизації перевіряє дані через UserModule та базу даних, і у разі успіху повертає результат, що дозволяє користувачу отримати доступ до системи.

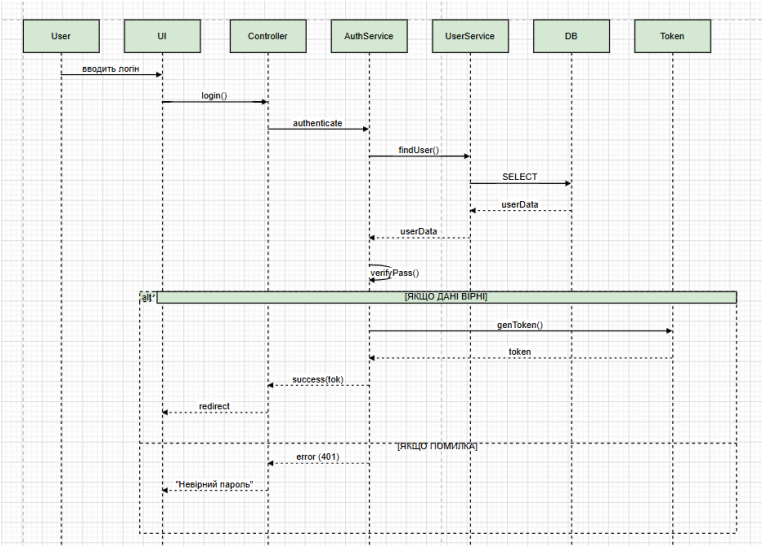


Рисунок 2.5- Діаграма послідовності авторизації користувача

Ще один сценарій — пошук книг (Рисунок 2.6). Користувач вводить запит у UI, який передається до SearchModule. Модуль обробляє запит, звертається до CatalogModule та повертає відфільтрований список книг, який відображається користувачу.

Іншим важливим сценарієм є оформлення замовлення. Користувач додає книги до кошика через CartModule, після чого ініціює створення замовлення. ApplicationController передає дані до OrderModule, який формує об’єкт замовлення, звертається до PaymentModule для обробки платежу та до DeliveryModule для визначення умов доставки. Після успішного виконання всіх операцій користувач отримує підтвердження замовлення. Діаграма послідовності оформлення замовлення зображена на Рисунку А.11 у додатку А.

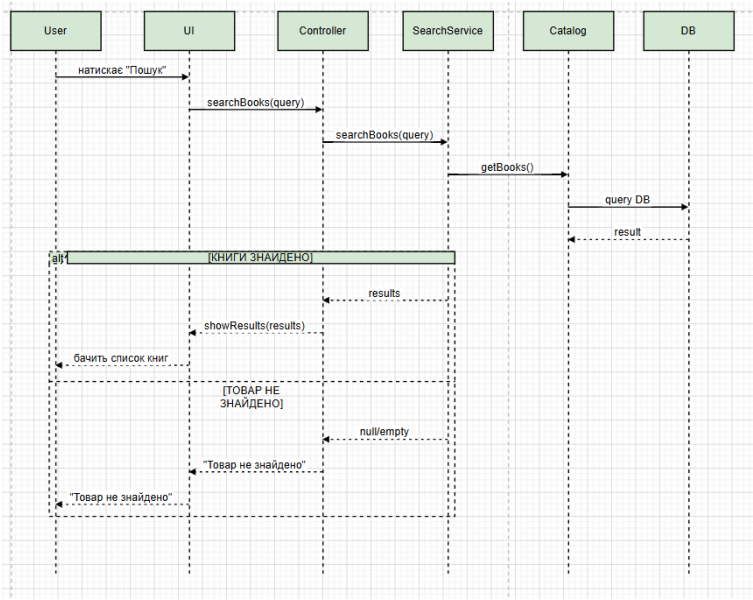


Рисунок 2.6 – Діаграма послідовності пошуку книг

Для кращого розуміння поведінки системи також доцільно використовувати діаграми станів. Наприклад, об’єкт Order може перебувати у таких станах: створено, очікує оплати, оплачено, відправлено, доставлено, скасовано. Перехід між станами відбувається в залежності від дій користувача та результатів роботи платіжної та логістичної систем.

Об’єкт User може перебувати у станах: неавторизований, авторизований, заблокований. Це дозволяє контролювати доступ до функціоналу системи. Діаграма станів користувача зображена на Рисунку 2.7.

Завершальним етапом є опис алгоритмів ключових методів. Наприклад, алгоритм додавання товару до кошика передбачає перевірку наявності книги, додавання її до списку CartItem або збільшення кількості, після чого оновлюється загальна вартість. Даний алгоритм зображений на Рисунку 2.8.

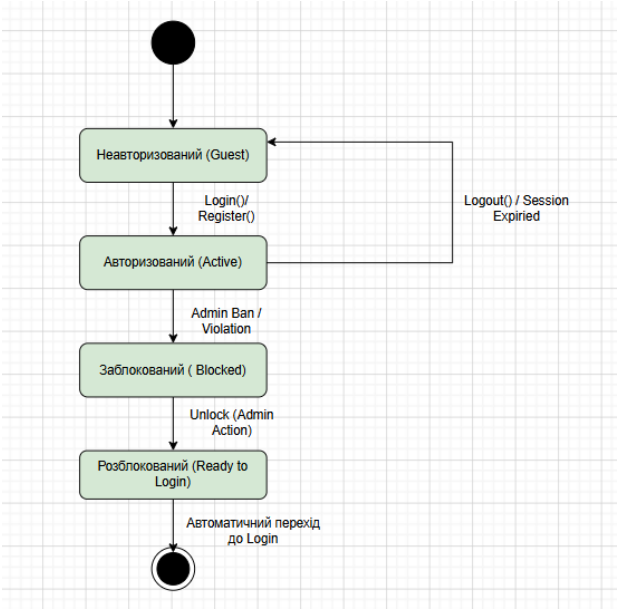


Рисунок 2.7 – Діаграма станів користувача

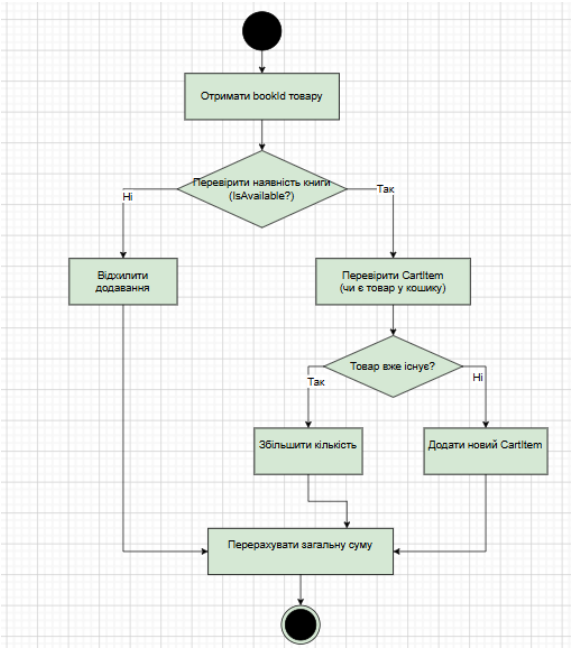


Рисунок 2.8 – Алгоритм додавання книг у кошик

Таким чином, детальне проєктування модулів дозволяє сформувати чітку структуру системи, визначити взаємодію між її компонентами та підготувати основу для подальшої реалізації інтернет-магазину книг, забезпечуючи його надійність, масштабованість та зручність підтримки.

2.7 Аналіз та вибір технологій і методів реалізації вебзастосунку

Створення вебзастосунку є досить складним процесом, що потребує детального аналізу та обґрунтованого вибору технологій і методів реалізації. Від правильного вибору залежить продуктивність системи, її масштабованість, безпека, а також зручність супроводу та подальшого розвитку. У цьому підрозділі проведено аналіз сучасних підходів до розроблення вебзастосунків та обрано найбільш доцільні технології для реалізації поставлених завдань. Основною метою є формування стабільної технічної бази, яка забезпечить ефективне функціонування застосунку відповідно до вимог і очікувань користувачів.

Одним із ключових етапів проєктування вебзастосунку є вибір серверної технології, яка відповідає за обробку HTTP-запитів, реалізацію бізнес-логіки, управління даними та взаємодію з базою даних. У процесі аналізу сучасних серверних рішень було розглянуто такі технології, як Node.js (з фреймворком Express), Java Spring Boot, PHP Laravel та ASP.NET Core MVC.

Node.js [15] є асинхронним середовищем виконання JavaScript, яке забезпечує високу швидкість обробки запитів завдяки неблокуючій моделі вводу-виводу. Воно добре підходить для створення легких та високонавантажених сервісів, особливо в реальному часі. Проте при розробці великих корпоративних систем може виникати складність у структуризації коду та підтримці архітектури через динамічну типізацію та відсутність суворих обмежень.

Java Spring Boot [16] є потужним фреймворком для створення корпоративних застосунків. Він забезпечує високу надійність, масштабованість та розвинену екосистему. Водночас його недоліком є складність початкового налаштування та

висока крива навчання, що може уповільнювати процес розробки, особливо у невеликих або середніх проєктах.

PHP Laravel [17] є популярним фреймворком, який відзначається простотою використання та зручним синтаксисом. Він добре підходить для швидкої розробки вебзастосунків середнього рівня складності. Водночас у масштабних системах можуть виникати обмеження продуктивності та залежність від інтерпретованої природи PHP, що впливає на швидкодію у порівнянні з компільованими мовами.

ASP.NET Core MVC [18][19] є сучасним кросплатформним фреймворком від Microsoft, який призначений для створення високопродуктивних вебзастосунків та API. Він використовує мову програмування C#, яка є строго типізованою, що зменшує кількість помилок на етапі компіляції та підвищує надійність програмного забезпечення. Архітектурний патерн MVC забезпечує чітке розділення логіки застосунку, що позитивно впливає на підтримку та масштабування системи. Крім того, ASP.NET Core відзначається високою продуктивністю завдяки оптимізованому runtime, підтримці асинхронного програмування та можливості роботи на різних платформах (Windows, Linux, macOS). Важливою перевагою також є вбудовані механізми безпеки, інтеграція з сучасними системами автентифікації та підтримка корпоративних стандартів розробки.

У результаті проведеного порівняльного аналізу було встановлено, що ASP.NET Core MVC є найбільш доцільним вибором для реалізації серверної частини вебзастосунку. Це зумовлено його високою продуктивністю, строгістю типізації, розвиненою архітектурною моделлю, а також можливістю ефективної роботи з великими обсягами даних та складною бізнес-логікою. Додатковою перевагою є стабільність платформи та її орієнтація на розробку масштабованих і безпечних корпоративних систем.

Наступним важливим етапом проєктування є вибір технології доступу до даних, яка забезпечує взаємодію серверної частини застосунку з базою даних. Від обраного

підходу залежить продуктивність роботи системи, зручність реалізації бізнес-логіки, а також рівень абстракції при роботі з даними.

У процесі аналізу було розглянуто три основні підходи: ADO.NET та Entity Framework Core. Кожен із них має свої особливості, переваги та обмеження, що впливають на доцільність їх використання в межах конкретного проєкту.

ADO.NET [20] є базовим інструментом для роботи з базами даних у середовищі .NET. Він надає повний контроль над SQL-запитами та підключенням до бази даних, що дозволяє максимально точно налаштовувати виконання операцій. Водночас використання ADO.NET вимагає написання значного обсягу низькорівневого коду, що ускладнює розробку та збільшує час реалізації функціоналу.

Entity Framework Core [21] є повноцінним ORM-фреймворком, який забезпечує високий рівень абстракції при роботі з базою даних. Він дозволяє працювати з даними на рівні об’єктів доменної моделі, що значно спрощує розробку та підвищує читабельність коду. Підтримка підходу Code First дозволяє формувати структуру бази даних на основі моделей застосунку, що спрощує процес її проєктування та супроводу. Крім того, Entity Framework Core інтегрується з LINQ, що забезпечує зручний та типобезпечний спосіб формування запитів до бази даних.

З огляду на баланс між продуктивністю, зручністю розробки та рівнем абстракції, а також враховуючи інтеграцію з обраною платформою ASP.NET Core MVC, було прийнято рішення використовувати Entity Framework Core як основну технологію доступу до даних.

При виборі технології для реалізації клієнтської частини вебзастосунку було проведено аналіз сучасних підходів до побудови користувацьких інтерфейсів. Зокрема, було розглянуто класичний стек вебтехнологій, що включає HTML[22], CSS[23] та JavaScript, а також сучасні JavaScript-фреймворки, такі як React[24] та Angular[25], які широко використовуються для створення складних інтерактивних односторінкових застосунків (SPA).

Класичний підхід із використанням HTML, CSS та JavaScript забезпечує достатній рівень гнучкості для створення вебінтерфейсів та не потребує додаткових залежностей. Однак при збільшенні складності проєкту значно ускладнюється підтримка коду, а також організація взаємодії між клієнтською та серверною частинами.

Використання сучасних JavaScript-фреймворків, таких як React або Angular, дозволяє будувати високодинамічні односторінкові застосунки з розвиненою клієнтською логікою. Такі рішення забезпечують високу інтерактивність інтерфейсу та ефективне оновлення даних без перезавантаження сторінки. Водночас їх використання передбачає розділення системи на окремий фронтенд та бекенд через API, що ускладнює архітектуру застосунку, збільшує обсяг розробки та потребує додаткової інтеграції між компонентами системи.

З урахуванням мети даного проєкту та необхідності забезпечення простоти реалізації, підтримки та інтеграції, було обрано підхід із використанням ASP.NET MVC із технологією Razor Views. Даний підхід дозволяє реалізувати серверно-орієнтовану генерацію вебсторінок, поєднуючи логіку відображення та серверну обробку даних у межах єдиної архітектури. Це значно спрощує розробку, зменшує складність взаємодії між клієнтською та серверною частинами, а також скорочує час розроблення проєкту.

Важливим етапом при розробці програмного забезпечення є вибір середовища розроблення (IDE), оскільки воно безпосередньо впливає на продуктивність розробника, швидкість створення програмного коду, зручність налагодження та загальну ефективність процесу розробки.

У процесі аналізу було розглянуто Visual Studio Code та Visual Studio 2022, які широко використовуються для створення застосунків на платформі .NET.

Visual Studio Code[26] є легким та універсальним редактором коду, який підтримує велику кількість мов програмування завдяки розширенням. Його основною

перевагою є швидкодія та гнучкість налаштування. Водночас для повноцінної розробки ASP.NET Core застосунків він потребує додаткової конфігурації, встановлення розширень та ручного налаштування середовища, що може ускладнювати процес розроблення, особливо для великих проєктів.

Найбільш доцільним вибором для реалізації даного вебзастосунку є Visual Studio 2022 [27], яка є офіційним середовищем розроблення від Microsoft для платформи .NET. Дане середовище забезпечує повну та глибоку інтеграцію з ASP.NET Core, підтримує роботу з Entity Framework Core, а також надає вбудовані інструменти для створення, тестування та налагодження вебзастосунків.

Таким чином, використання Visual Studio 2022 є обґрунтованим рішенням, оскільки воно забезпечує повний набір інструментів для ефективної розробки, тестування та супроводу вебзастосунку на базі ASP.NET Core.

Для забезпечення зберігання та обробки даних у вебзастосунку було проведено аналіз сучасних систем управління базами даних (СУБД), серед яких розглянуто MySQL та Microsoft SQL Server. Вибір СУБД є одним із ключових етапів проєктування, оскільки він безпосередньо впливає на продуктивність системи, надійність збереження даних, масштабованість та зручність інтеграції з серверною частиною застосунку.

MySQL [28] є однією з найпоширеніших реляційних СУБД, яка характеризується простотою використання, високою швидкістю виконання базових операцій та широкою підтримкою у веброзробці. Водночас її інтеграція з платформою .NET є менш тісною порівняно з рішеннями від Microsoft, що може вимагати додаткових налаштувань та адаптації при використанні в екосистемі ASP.NET Core.

Microsoft SQL Server [29] є реляційною системою управління базами даних, яка розроблена компанією Microsoft та забезпечує повну інтеграцію з екосистемою .NET. Дана СУБД відзначається високою продуктивністю, надійністю та розширеними можливостями керування даними. Особливо важливою є її повна сумісність з Entity

Framework Core, що дозволяє ефективно реалізовувати підхід ORM та спрощує взаємодію між прикладним рівнем та базою даних.

Таким чином, з урахуванням вимог до проєкту, обраної серверної платформи ASP.NET Core та необхідності глибокої інтеграції з ORM Entity Framework Core, найбільш доцільним рішенням було визначено використання Microsoft SQL Server як основної системи управління базою даних.

Для реалізації користувацького інтерфейсу вебзастосунку та створення графічних ресурсів було проаналізовано сучасні інструменти та технології, що використовуються у веброзробці. Формування якісного інтерфейсу є важливим аспектом загальної розробки системи, оскільки безпосередньо впливає на зручність використання, сприйняття функціоналу та загальний користувацький досвід.

Для побудови адаптивного та структурованого інтерфейсу було обрано CSS-фреймворк Bootstrap. Його використання дозволяє значно прискорити процес розробки завдяки наявності готових компонентів, таких як форми, кнопки, навігаційні панелі та сіткова система верстки. Це забезпечує коректне відображення інтерфейсу на різних типах пристроїв та екранів, що є важливою вимогою сучасних вебзастосунків.

Для забезпечення інтерактивності користувацького інтерфейсу використовується мова програмування JavaScript, яка дозволяє реалізовувати динамічну поведінку елементів сторінки, обробку подій користувача та взаємодію з серверною частиною без необхідності повного перезавантаження сторінки.

Останнім етапом є вибір методології розроблення. Було розглянуто каскадну (Waterfall), ітераційну та гнучку (Agile) моделі. Каскадна модель є занадто жорсткою для вебпроєктів, оскільки не дозволяє вносити зміни у функціонал до завершення всіх етапів, що критично для динамічного ринку електронної комерції. Натомість Agile забезпечує максимальну гнучкість, проте часто потребує розмитих часових меж, що може бути незручним при суворому плануванні ресурсів. Найкращим вибором для даного проєкту є ітераційна модель, оскільки вона дозволяє поєднати чітку структуру

планування з можливістю поступового нарощування функціоналу. Це ідеально узгоджується з проведеною модульною декомпозицією: розробку можна розпочати з базового інкременту (ядро та каталог), поступово додаючи складніші модулі (оплата, система відгуків), що мінімізує ризики та дозволяє тестувати окремі частини системи ще до повного завершення проєкту.

Отже, у результаті проведеного аналізу було сформовано оптимальний стек технологій: ASP.NET Core MVC для серверної частини, Entity Framework Core для роботи з базою даних, Razor Views та Bootstrap для клієнтської частини, Microsoft SQL Server як систему управління базами даних та Visual Studio 2022 як середовище розроблення. Обрані технології забезпечують високу продуктивність, зручність розроблення та відповідність сучасним стандартам веброзробки.

2.8 Висновки

У процесі виконання розділу було здійснено комплексне проєктування програмного забезпечення інтернет-магазину книг. Проведено аналіз основних архітектурних підходів та обґрунтовано вибір клієнт-серверної та шарової архітектури з використанням патерну MVC, що забезпечує баланс між простотою реалізації, продуктивністю та масштабованістю системи.

На основі обраної архітектури виконано поетапну декомпозицію системи: визначено загальні рівні, виділено основні модулі та описано їх функціональне призначення. Деталізовано взаємодію між модулями, їх залежності та інтерфейси, що дозволило сформувати чітку структуру програмного забезпечення.

Також було розроблено моделі поведінки системи, включаючи діаграми станів, сценарії взаємодії та алгоритми ключових процесів, що забезпечує повніше розуміння логіки роботи застосунку.

Завершальним етапом став аналіз та вибір технологій реалізації, у результаті якого обрано стек ASP.NET Core MVC, Entity Framework Core та Microsoft SQL Server, а також визначено середовище розробки Visual Studio 2022.

Отже, отримана проєктна документація формує цілісне уявлення про архітектуру системи, її модульну структуру та принципи взаємодії компонентів, що створює надійну основу для подальшої реалізації вебзастосунку.

3.ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Особливості програмної реалізації на ASP.NET MVC (C#)

Для початку програмної реалізації модулів вебзастосунку інтернет-магазину книг доцільно детально розглянути архітектурні особливості платформи ASP.NET MVC та принципи її функціонування. Застосування даної технології дозволяє реалізувати гнучку, масштабовану та структуровану систему, що відповідає сучасним вимогам до веброзробки. В основі підходу лежить чітке розмежування компонентів застосунку, що забезпечує підвищення якості коду, спрощення тестування та подальшої підтримки програмного продукту.

Програмна реалізація здійснюється з використанням мови програмування C#, яка підтримує об’єктно-орієнтовану парадигму. Це передбачає використання класів, інтерфейсів, методів, властивостей і наслідування для побудови логіки застосунку. Кожен функціональний елемент системи, зокрема модулі керування товарами, користувачами, замовленнями та кошиком, реалізується у вигляді окремих класів, що взаємодіють між собою через чітко визначені інтерфейси. Такий підхід сприяє повторному використанню коду та зменшенню залежностей між компонентами.

Ключовою особливістю ASP.NET MVC є реалізація шаблону проєктування Model-View-Controller, який забезпечує логічний поділ застосунку на три взаємопов’язані складові. Модель відповідає за представлення даних та бізнес-логіку. У межах інтернет-магазину книг моделі описують такі сутності, як книга, автор, категорія, користувач, замовлення та елементи кошика. Вони містять властивості, що відповідають полям бази даних, а також можуть включати методи для обробки даних і реалізації бізнес-правил.

Представлення (View) відповідає за формування користувацького інтерфейсу. У ASP.NET MVC представлення створюються за допомогою Razor-синтаксису, який дозволяє інтегрувати C# код безпосередньо в HTML-розмітку. Це забезпечує динамічне відображення інформації, наприклад списків книг, деталей товару чи результатів пошуку. Використання шаблонів представлення сприяє повторному використанню інтерфейсних компонентів і підвищує узгодженість дизайну.

Контролери виступають центральним елементом взаємодії між моделлю та представленням. Вони обробляють HTTP-запити користувача, виконують відповідні дії (actions), звертаються до моделей для отримання або зміни даних і повертають результати у вигляді представлень або інших типів відповідей. Кожен контролер відповідає за окрему функціональну область, наприклад, керування каталогом книг, обробку замовлень або авторизацію користувачів.

Важливим механізмом у ASP.NET MVC є маршрутизація, яка визначає відповідність між URL-адресами та методами контролерів. Це дозволяє створювати зрозумілі та логічні адреси сторінок, що покращує навігацію користувача та сприяє оптимізації для пошукових систем. Маршрути можуть налаштовуватися як централізовано, так і за допомогою атрибутів безпосередньо в коді контролерів.

Для доступу до даних у застосунку використовується технологія Entity Framework, що реалізує підхід об’єктно-реляційного відображення. Вона дозволяє працювати з базою даних через об’єкти C#, мінімізуючи необхідність написання SQL-запитів. Основним компонентом є контекст даних (DbContext), який забезпечує взаємодію з таблицями бази даних через колекції типу DbSet. Це значно спрощує операції створення, читання, оновлення та видалення даних.

З метою оптимізації передачі даних між контролерами та представленнями використовуються моделі представлення (ViewModel). Вони дозволяють агрегувати дані з різних джерел і подавати їх у зручному для відображення форматі. Це сприяє

зменшенню залежності між доменною моделлю та інтерфейсом користувача, що є важливим для забезпечення гнучкості системи.

Для забезпечення коректності введених користувачем даних застосовуються механізми валідації, зокрема атрибути Data Annotations. Вони дозволяють задавати обмеження для властивостей моделей, такі як обов'язковість заповнення, довжина рядків або формат даних. Валідація виконується як на стороні сервера, так і на стороні клієнта, що підвищує надійність і зручність використання застосунку.

Окрему роль у структурі застосунку відіграє сервісний рівень (Service Layer), який інкапсулює бізнес-логіку та забезпечує взаємодію між контролерами і моделями. Наприклад, обробка замовлення може включати перевірку наявності товарів, розрахунок загальної вартості, застосування знижок і взаємодію з платіжними системами. Винесення такої логіки в окремі сервіси дозволяє уникнути перевантаження контролерів і підвищує модульність системи.

Для реалізації автентифікації та авторизації користувачів використовується ASP.NET Identity, що надає готові механізми для керування обліковими записами, ролями та правами доступу. Це дозволяє забезпечити безпеку застосунку та обмежити доступ до певних функцій залежно від ролі користувача, наприклад адміністратора або покупця.

Таким чином, використання ASP.NET MVC у поєднанні з мовою програмування C# дозволяє реалізувати інтернет-магазин книг як структурований, масштабований і надійний вебзастосунок. Чітке розмежування компонентів, використання сучасних підходів до роботи з даними та організації бізнес-логіки забезпечують ефективність розробки та зручність подальшого супроводу системи.

3.2 Програмна реалізація модулів

Програмна реалізація інформаційної системи інтернет-магазину книг виконана із використанням платформи .NET та фреймворку ASP.NET Core MVC, що забезпечує побудову сучасного вебзастосунку. В основу розробки покладено принцип розділення відповідальності, відповідно до якого логіка доступу до даних, бізнес-логіка та рівень представлення реалізуються як незалежні компоненти. Такий підхід дозволяє підвищити гнучкість системи, спростити її тестування та забезпечити можливість подальшого масштабування без суттєвих змін у вже реалізованих модулях. Загальна структура програмного проєкту, що включає контролери, моделі, представлення та допоміжні сервіси, наведена на Рисунку 3.1.

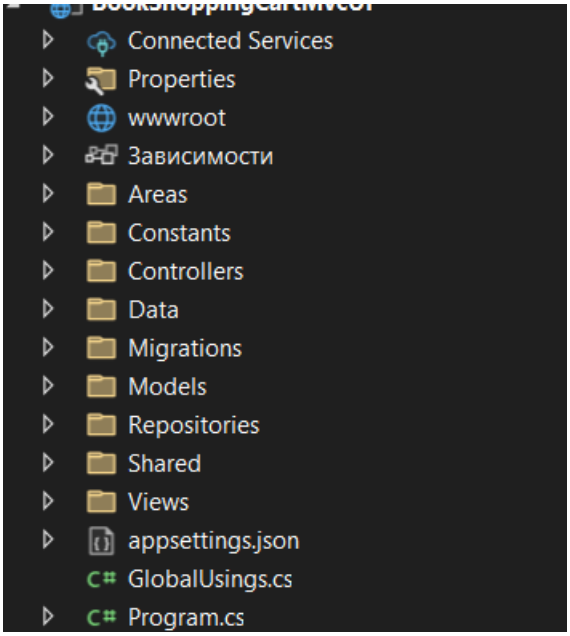


Рисунок 3.1- Структура проєкту у середовищі розробки

Для організації взаємодії з базою даних використано технологію Entity Framework Core із підходом Code First, який передбачає опис структури бази даних у

вигляді класів предметної області. Центральним елементом цього рівня є клас ApplicationDbContext, який наслідується від IdentityDbContext та забезпечує інтеграцію з підсистемою автентифікації користувачів. У межах даного класу визначено набори сутностей (DbSet), що відповідають таблицям бази даних, зокрема книги, категорії, замовлення та деталі замовлень. Контекст бази даних виконує функції координування транзакцій та відстеження змін у сутностях, що дозволяє реалізувати шаблон Unit of Work без додаткового програмного коду. Приклад реалізації класу контексту бази даних представлено на Рисунку 3.2.

```
public class ApplicationDbContext : IdentityDbContext
{
    Ссылок: 0
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    Ссылок: 8
    public DbSet<Genre> Genres { get; set; }
    Ссылок: 9
    public DbSet<Book> Books { get; set; }
    Ссылок: 4
    public DbSet<ShoppingCart> ShoppingCarts { get; set; }
    Ссылок: 7
    public DbSet<CartDetail> CartDetails { get; set; }
    Ссылок: 5
    public DbSet<Order> Orders { get; set; }
    Ссылок: 1
    public DbSet<OrderDetail> OrderDetails { get; set; }

    Ссылок: 2
    public DbSet<OrderStatus> orderStatuses { get; set; }
    Ссылок: 6
    public DbSet<Stock> Stocks { get; set; }
    Ссылок: 4
    public DbSet<Comment> Comments { get; set; }
```

Рисунок 3.2 - Реалізація класу ApplicationDbContext

Моделі даних реалізовані у вигляді класів мови C#, які відображають сутності предметної області. Кожна модель містить набір властивостей, що відповідають полям таблиць бази даних, а також навігаційні властивості для опису зв’язків між сутностями. Для забезпечення коректності введених даних використано механізм Data Annotations, який дозволяє визначати обмеження на значення полів безпосередньо у

коді моделей. Зокрема, для полів визначено обов'язковість заповнення, обмеження довжини рядків та допустимі діапазони числових значень. Це забезпечує валідацію даних як на серверному рівні, так і на рівні користувацького інтерфейсу. Приклад реалізації моделі книги з використанням атрибутів валідації наведено на Рисунку 3.3.

Рисунок 3.3 - Модель Book із використанням Data Annotations

Зв'язки між сутностями реалізуються за допомогою зовнішніх ключів та навігаційних властивостей. Наприклад, між сутностями категорії та книги реалізовано зв'язок типу один-до-багатьох, що дозволяє кожній книзі належати до певної категорії. Аналогічно, між замовленням та його деталями встановлено зв'язок, який дозволяє зберігати інформацію про склад кожного замовлення. Така структура забезпечує нормалізацію даних та спрощує виконання запитів.

Формування структури бази даних здійснюється за допомогою механізму міграцій. Кожна зміна у моделях супроводжується створенням нової міграції, яка містить інструкції для оновлення схеми бази даних. Застосування міграцій дозволяє автоматизувати процес синхронізації структури бази даних із програмною моделлю та забезпечує контроль версій. Приклад створення та застосування міграцій наведено на Рисунку 3.4.

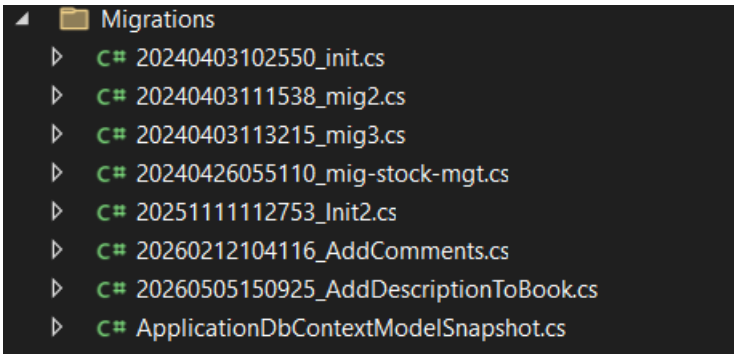


Рисунок 3.4 - Створення та застосування міграцій

Логіка обробки запитів користувача у системі реалізована через контролер BookController, який виконує роль посередника між інтерфейсом та рівнем даних. Контролер приймає HTTP-запити та передає операції роботи з даними до класу BookRepository, який інкапсулює доступ до бази даних. Такий підхід забезпечує розділення відповідальності та підвищує підтримуваність системи.

Для отримання списку книг у методі GetBooks() застосовується «жадібне завантаження» через метод Include(), що дозволяє одразу отримати пов’язані сутності (наприклад, жанри), зменшуючи кількість запитів до бази даних. Також використовується метод AsNoTracking(), який вимикає відстеження змін об’єктів Entity Framework Core, що підвищує продуктивність при операціях лише читання.

Таким чином, контролер відповідає за обробку запитів і формування відповіді, а репозиторій — за ефективне отримання даних. Приклад реалізації наведено на Рисунках 3.5-3.6.

```
Ссылок: 3
public async Task<IActionResult> Index()
{
    var books = await _bookRepo.GetBooks();
    return View(books);
}
```

Рисунок 3.5 - Реалізація методу Index у BooksController

```
Ссылок: 3
public async Task<Book?> GetBookById(int id) => await _context.Books.FindAsync(id);

Ссылок: 2
public async Task<IEnumerable<Book>> GetBooks() => await _context.Books.Include(a=>a.Genre).ToListAsync();
```

Рисунок 3.6 - Реалізація логіки отримання даних у класі BookRepository

Окрему увагу приділено реалізації модуля кошика, який забезпечує збереження вибраних товарів користувача у базі даних, що дозволяє підтримувати його стан між різними сесіями. Вся логіка роботи інкапсульована у класі CartRepository, який взаємодіє з базою даних через ApplicationDbContext.

Додавання товару до кошика (метод AddItem) реалізовано з перевіркою користувача через UserManager та перевіркою наявності активного кошика. У разі повторного додавання товару його кількість оновлюється, а при першому додаванні створюється новий запис у деталях кошика.

Оформлення замовлення (метод DoCheckout) реалізовано з використанням транзакцій, що забезпечує цілісність даних. У межах транзакції виконується створення замовлення, перенесення товарів із кошика до його деталей та оновлення залишків на

складі. Після успішного завершення операцій транзакція фіксується, а кошик очищується. Приклад реалізації наведено на Рисунку 3.7.

```
public async Task<bool> DoCheckout(CheckoutModel model)
{
    using var transaction = _db.Database.BeginTransaction();
    try
    {
        var userId = GetUserId();
        if (string.IsNullOrEmpty(userId))
            throw new UnauthorizedAccessException("User is not logged-in");
        var cart = await GetCart(userId);
        if (cart is null)
            throw new InvalidOperationException("Invalid cart");
        var cartDetail = _db.CartDetails
                            .Where(a => a.ShoppingCartId == cart.Id).ToList();
        if (cartDetail.Count == 0)
            throw new InvalidOperationException("Cart is empty");
        var pendingRecord = _db.orderStatuses.FirstOrDefault(s => s.StatusName == "Pending");
        if (pendingRecord is null)
            throw new InvalidOperationException("Order status does not have Pending status");
        var order = new Order
        {
            UserId = userId,
            CreateDate = DateTime.UtcNow,
            Name=model.Name,
            Email=model.Email,
            MobileNumber=model.MobileNumber,
            PaymentMethod=model.PaymentMethod,
            Address=model.Address,
            IsPaid=false,
            OrderStatusId = pendingRecord.Id
        };
        _db.Orders.Add(order);
        _db.SaveChanges();
        foreach(var item in cartDetail)
        {
            var orderDetail = new OrderDetail
            {
                BookId = item.BookId,
                OrderId = order.Id,
                Quantity = item.Quantity,
                UnitPrice = item.UnitPrice
            };
            _db.OrderDetails.Add(orderDetail);
```

Рисунок 3.7- Метод DoCheckout: логіка перевірки залишків на складі та фіналізації замовлення

Адміністративна частина системи реалізує функції створення, редагування та видалення записів. При цьому особливу увагу приділено обробці введених даних, зокрема перевірці ModelState, яка дозволяє запобігти збереженню некоректної

інформації. Додатково реалізовано механізм завантаження файлів, що використовується для додавання зображень книг. Файли зберігаються на сервері, а у базі даних зберігається шлях до них, що забезпечує ефективне використання ресурсів.

Рівень представлення реалізовано із використанням Razor, що дозволяє поєднувати HTML-розмітку з серверною логікою. Для передачі даних у представлення використовуються ViewModel-класи, які формуються у контролерах та містять лише необхідні для відображення дані. Це дозволяє зменшити зв’язаність компонентів та підвищити зрозумілість коду. Приклад реалізації сторінки відображення списку книг наведено на рисунку 3.8.

```
<h2>Книги</h2>

<a asp-action="AddBook" asp-controller="Book" class="btn btn-primary">Додати книгу</a>

@if (Model?.Count() > 0)
{
    <table class="table table-striped my-2">
        <tr>
            <th>Зображення</th>
            <th>Назва книги</th>
            <th>Автор</th>
            <th>Жанр</th>
            <th>Ціна</th>
            <th>Дії</th>
        </tr>

        @foreach (var book in Model)
        {
            <tr>
                <td>
                    @if (string.IsNullOrWhiteSpace(book.Image))
                    {
                        <img src="/Images/NoImage.png" style="width:80px;height:100px" alt="book image" />
                    }
                    else
                    {
                        <img src="/images/@book.Image" style="width:80px;height:100px" alt="book image" />
                    }
                </td>
                <td>@book.BookName</td>
                <td>@book.AuthorName</td>
                <td>@book.Genre.GenreName</td>
                <td>@book.Price</td>
                <td>
                    <a asp-action="UpdateBook" asp-controller="Book" asp-route-id="@book.Id" class="btn btn-success">Редагувати</a>
                    <a asp-action="DeleteBook" asp-controller="Book" asp-route-id="@book.Id" class="btn btn-danger" onclick="return window.confirm('Ви впевнені?')">Видалити</a>
                </td>
            </tr>
        }
    </table>
}
else
{
    <h3>Немає записів</h3>
}
```

Рисунок 3.8 - Представлення списку книг із використанням Razor

Для спрощення розробки інтерфейсу використано Tag Helpers, які дозволяють генерувати HTML-елементи з урахуванням серверної логіки. Повторювані елементи інтерфейсу, такі як навігаційне меню або картки товарів, винесено у Partial Views, що забезпечує повторне використання коду. Загальна структура сторінок визначається у файлі _Layout.cshtml, який містить базовий шаблон оформлення.

Безпека системи забезпечується за рахунок використання ASP.NET Core Identity, який надає готові механізми для автентифікації та авторизації користувачів. Автентифікація передбачає перевірку облікових даних та створення захищеного сеансу користувача, тоді як авторизація реалізується на основі ролей. Це дозволяє обмежити доступ до окремих функцій системи, зокрема адміністративних операцій. Для цього використовується атрибут Authorize, який визначає права доступу до методів контролерів. Приклад використання даного механізму наведено на рисунку 3.9.

```
namespace BookShoppingCartMvcUI.Controllers;

[Authorize(Roles = nameof(Roles.Admin))]
Ссылок: 1
public class AdminOperationsController : Controller
{
    private readonly IUserOrderRepository _userOrderRepository;
    Ссылок: 0
    public AdminOperationsController(IUserOrderRepository userOrderRepository)
    {
        _userOrderRepository = userOrderRepository;
    }

    Ссылок: 1
    public async Task<IActionResult> AllOrders()
    {
        var orders = await _userOrderRepository.UserOrders(true);
        return View(orders);
    }
```

Рисунок 3.9- Реалізація обмеження доступу до адміністративних операцій на основі ролей користувачів

Додатково у системі реалізовано захист від типових веб-загроз. Зокрема, для запобігання CSRF-атакам використовується механізм антифоржері-токенів, який автоматично перевіряється при надсиланні форм. Захист від XSS-атак забезпечується за рахунок автоматичного кодування виводу у Razor-представленнях.

Функціональність пошуку та фільтрації реалізована із використанням LINQ, що дозволяє формувати запити до бази даних у вигляді виразів мови програмування. Запит будується динамічно залежно від параметрів, введених користувачем, що забезпечує гнучкість та ефективність обробки даних. Фільтрація виконується

безпосередньо на рівні бази даних, що зменшує обсяг переданої інформації та підвищує продуктивність системи. Приклад реалізації пошуку наведено на рисунку 3.10.

```
Ссылок: 0
public async Task<IEnumerable<Book>> GetBooks(string sTerm = "")
{
    sTerm = sTerm.ToLower();
    var books = await (from book in _db.Books
                       join genre in _db.Genres
                       on book.GenreId equals genre.Id
                       where string.IsNullOrWhiteSpace(sTerm) ||
                             book.BookName.ToLower().StartsWith(sTerm)
                       select new Book
                       {
                           Id = book.Id,
                           Image = book.Image,
                           AuthorName = book.AuthorName,
                           BookName = book.BookName,
                           Price = book.Price,
                           GenreId = book.GenreId,
                           GenreName = genre.GenreName
                       }
                ).ToListAsync();
    return books;
}
```

Рисунок 3.10- Реалізація пошуку та фільтрації книг

Таким чином, програмна реалізація модулів інформаційної системи забезпечує повний функціональний цикл роботи інтернет-магазину — від збереження даних і обробки запитів до формування користувацького інтерфейсу та забезпечення безпеки. Використання сучасних технологій і підходів дозволило створити ефективну, надійну та масштабовану систему, придатну для подальшого розвитку та вдосконалення.

3.3 Вимоги до технічних та програмних засобів

Визначення системних вимог є важливим етапом розроблення програмного забезпечення, оскільки воно дозволяє забезпечити стабільну та коректну роботу вебзастосунку інтернет-магазину книг на цільових пристроях користувачів. Чітко сформульовані вимоги також дають змогу оцінити сумісність програмного продукту з апаратним та програмним середовищем, а також сприяють правильному плануванню розгортання та подальшої оптимізації системи.

Основними характеристиками, до яких висуваються вимоги, є тип і продуктивність процесора, розрядність операційної системи, обсяг оперативної пам’яті, доступний дисковий простір, а також характеристики програмного середовища, зокрема підтримка сучасних вебтехнологій і систем керування базами даних. Оскільки інтернет-магазин реалізований на платформі ASP.NET MVC, важливим також є наявність сумісного середовища виконання .NET та підтримка вебсерверних технологій.

Аналізуючи розроблений вебзастосунок, було визначено мінімальні та рекомендовані системні вимоги, які наведено в таблиці 3.1.

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Розрядність процесора і ОС	x64	x64
Операційна система	Windows 10	Windows 11
Процесор	AMD Ryzen 5 2400G або Intel Core i3-10105F	Intel Core i5-8400 або AMD Ryzen 3 3300X
Оперативна пам’ять	6 ГБ	8 ГБ і більше
Відеоадаптер	NVIDIA GeForce GTX 1060 3 GB або AMD Radeon RX 580 4 GB	NVIDIA GeForce GTX 1070 8 GB або AMD Radeon RX Vega 56 8 GB

Підтримка DirectX	DirectX 12	DirectX 12
Місце на диску	1 ГБ вільного простору	2 ГБ і більше

Таблиця 3.1 – Системні вимоги

Мінімальні вимоги забезпечують базову працездатність вебзастосунку, включаючи можливість відкриття інтерфейсу, виконання основних операцій пошуку, перегляду каталогу книг та оформлення замовлень. При цьому можливі обмеження у швидкодії системи при високому навантаженні або одночасній роботі великої кількості користувачів.

Рекомендовані вимоги орієнтовані на забезпечення більш стабільної та продуктивної роботи системи, особливо у випадках інтенсивного використання функціоналу, зокрема при роботі з великими обсягами даних, складними запитами до бази даних та одночасній взаємодії багатьох користувачів.

Таким чином, дотримання визначених системних вимог є необхідною умовою для коректного функціонування інтернет-магазину книг, забезпечення високої продуктивності та комфортного користувацького досвіду.

3.4 Тестування вебзастосунку

3.4.1 Аналіз методів тестування вебзастосунків

Тестування вебзастосунків є ключовим етапом життєвого циклу розробки програмного забезпечення, спрямованим на забезпечення відповідності системи встановленим вимогам, виявлення дефектів та підвищення загальної якості

програмного продукту [30]. У сучасних умовах стрімкого розвитку інформаційних технологій вебзастосунки характеризуються високим рівнем складності, інтеграцією з різноманітними сервісами та необхідністю підтримки великої кількості користувачів, що обумовлює необхідність застосування комплексного підходу до їх тестування.

Одним із базових підходів до класифікації методів тестування є їх поділ за рівнем доступу до внутрішньої структури програмного коду. Зокрема, тестування за принципом «чорного ящика» передбачає оцінювання функціональності системи без урахування її внутрішньої реалізації, що дозволяє перевірити відповідність поведінки системи функціональним вимогам. Натомість тестування «білого ящика» базується на аналізі внутрішньої логіки програмного коду, включаючи перевірку умовних операторів, циклів та структур даних. Комбінованим підходом є тестування «сірого ящика», яке поєднує елементи обох зазначених методів і передбачає часткове знання внутрішньої архітектури системи.

Важливим аспектом є класифікація тестування за рівнями його проведення. Модульне тестування орієнтоване на перевірку окремих компонентів програмного забезпечення, що дозволяє виявляти помилки на ранніх етапах розробки. Інтеграційне тестування спрямоване на перевірку взаємодії між окремими модулями, що є критичним для вебзастосунків, які часто мають багаторівневу архітектуру. Системне тестування передбачає оцінювання всієї системи як єдиного цілого, тоді як приймальне тестування виконується з метою підтвердження відповідності продукту вимогам замовника та готовності до експлуатації.

Функціональне тестування відіграє центральну роль у забезпеченні коректності роботи вебзастосунку. Воно передбачає перевірку реалізації бізнес-логіки, коректності обробки користувацьких даних, функціонування механізмів аутентифікації та авторизації, а також взаємодії клієнтської та серверної частин. При цьому широко застосовуються такі методи, як еквівалентне розбиття, аналіз граничних значень та тестування сценаріїв використання, що дозволяють оптимізувати процес формування тестових випадків.

Поряд із функціональним тестуванням важливе значення має нефункціональне тестування, яке оцінює характеристики системи, що не пов’язані безпосередньо з її функціональністю. Зокрема, тестування продуктивності дозволяє визначити швидкодію системи та її стабільність при різних рівнях навантаження. Навантажувальне та стрес-тестування дають змогу оцінити поведінку системи в умовах нормального та пікового використання. Тестування безпеки спрямоване на виявлення вразливостей, таких як SQL-ін’єкції чи міжсайтовий скриптинг, тоді як тестування зручності використання (usability testing) дозволяє оцінити ефективність взаємодії користувача з інтерфейсом системи.

Суттєвим напрямом є також тестування інтерфейсу користувача, яке забезпечує перевірку коректності відображення елементів, адаптивності дизайну та відповідності сучасним вимогам до доступності. У зв’язку з різноманіттям браузерів та пристроїв особливої актуальності набуває кросбраузерне тестування, що дозволяє гарантувати однакову роботу вебзастосунку в різних середовищах.

Окрему роль відіграє тестування прикладних програмних інтерфейсів (API), яке забезпечує перевірку обміну даними між компонентами системи. Воно включає аналіз HTTP-запитів, статус-кодів відповідей, форматів переданих даних та обробки помилок, що є критично важливим для сучасних вебзастосунків, побудованих за принципами клієнт-серверної архітектури.

Залежно від способу виконання тестування поділяється на ручне та автоматизоване. Ручне тестування дозволяє виявляти нетипові помилки та оцінювати поведінку системи з точки зору користувача, проте є трудомістким і залежить від людського фактору. Автоматизоване тестування, навпаки, забезпечує швидке та багаторазове виконання тестів, що особливо важливо для регресійного тестування, однак потребує додаткових ресурсів для розробки та підтримки тестових сценаріїв.

Таким чином, ефективне тестування вебзастосунків передбачає комплексне використання різних методів і підходів, що дозволяє забезпечити високу якість

програмного продукту, знизити ризики виникнення помилок та підвищити рівень задоволеності кінцевих користувачів.

3.4.2 Аналіз результатів тестування

Першим етапом тестування вебзастосунку інтернет-магазину книг було створення та виконання модульних (unit) тестів для ключових компонентів бізнес-логіки системи. Основною метою цього етапу було забезпечення коректності роботи базових операцій із сутностями предметної області, зокрема моделі книги та модуля кошика покупця. Особливу увагу приділено перевірці цілісності даних, арифметичних обчислень та поведінки системи у граничних випадках.

Для реалізації модульного тестування було використано фреймворк xUnit, який забезпечує зручні засоби для автоматизованої перевірки логіки застосунку. У межах тестування було сформовано два основних тестових класи: ShoppingCart_Module_Validation та ShoppingCart_Module_Calculation, які відповідають за перевірку коректності роботи моделі книги та логіки обчислення кошика.

Перший набір тестів був спрямований на перевірку моделі Book. Зокрема, тест «Перевірка_наявності_деталей» підтвердив, що основні властивості книги, такі як назва та опис, коректно ініціалізуються та не містять порожніх значень. Це є важливим аспектом, оскільки відсутність обов’язкових даних може призвести до некоректної роботи інтерфейсу користувача та помилок при збереженні даних у базі. Результати виконання даного тесту наведено на рисунку 3.11.

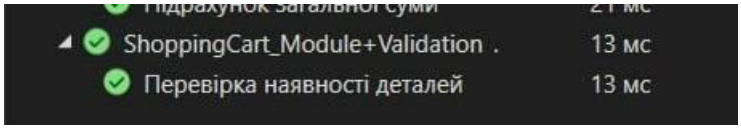


Рисунок 3.11 - Результат виконання unit-тесту перевірки наявності деталей книги

Другий тест цього блоку — «Перевірка_оновлення_ціни» — був призначений для перевірки механізму зміни вартості книги. Отримані результати підтвердили, що значення ціни коректно оновлюється та зберігається без спотворень. Це є критично важливим для забезпечення точності фінансових операцій у системі.

Другий блок тестів був присвячений перевірці логіки роботи кошика покупця. У тесті «Підрахунок_загальної_суми» було змодельовано список товарів із різною кількістю та ціною, після чого виконано обчислення загальної вартості замовлення. Результат тестування підтвердив правильність реалізації алгоритму підрахунку, оскільки отримане значення повністю відповідало очікуваному. Результати виконання тесту наведено на рисунку 3.12.

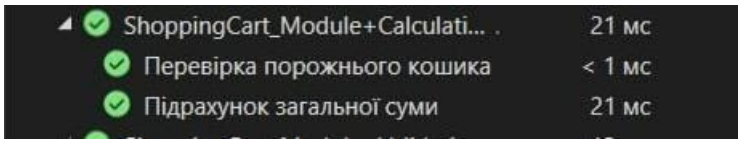


Рисунок 3.12 - Результат unit-тесту підрахунку загальної суми кошика

Додатково було проведено тест «Перевірка_порожнього_кошика», який підтвердив коректну поведінку системи у випадку відсутності товарів у кошику. У такій ситуації система повертає нульове значення загальної суми, що відповідає очікуваній логіці роботи та запобігає виникненню помилок при обробці даних.

Аналіз результатів модульного тестування показав, що всі реалізовані тести були успішно виконані без виявлення критичних помилок. Це свідчить про правильність реалізації основних алгоритмів бізнес-логіки та стабільність роботи ключових компонентів системи. Окрім цього, використання модульного тестування дозволило виявити потенційні проблеми ще на етапі розробки та забезпечити високу якість програмного коду.

Після завершення модульного тестування було проведено перевірку коректності інтеграції компонентів системи, що підтвердило правильну взаємодію між модулями вебзастосунку. Загалом результати тестування свідчать про те, що розроблений програмний продукт функціонує відповідно до заданих вимог та готовий до подальшого використання.

3.6 Висновки

У процесі реалізації вебзастосунку інтернет-магазину книг було виконано комплекс етапів розробки, що базувалися на попередньо визначених вимогах та архітектурі системи. На початковому етапі було реалізовано основні модулі системи, включаючи моделі даних, бізнес-логіку та механізми обробки замовлень.

Окрему увагу приділено реалізації модульного тестування, яке дозволило перевірити коректність роботи ключових компонентів системи. Додатково було проведено тестування логіки кошика, що підтвердило правильність виконання фінансових розрахунків.

Результати тестування свідчать про стабільність та коректність роботи вебзастосунку, а також про відповідність реалізованого функціоналу вимогам технічного завдання. Усі тестові сценарії були успішно виконані, що підтверджує готовність системи до використання.

ВИСНОВКИ

Завершивши виконання кваліфікаційної роботи на тему «Вебзастосунок для продажу та купівлі книг», можна зробити такі висновки.

На початковому етапі було проведено аналіз предметної області електронної комерції та визначено актуальність розроблення вебзастосунків для онлайн-продажу книг. У процесі дослідження було розглянуто особливості функціонування подібних систем, проаналізовано існуючі рішення та визначено основні вимоги до функціоналу майбутнього застосунку. Це дозволило сформувати чітке уявлення про структуру системи та ключові задачі, які необхідно реалізувати.

У результаті аналізу предметної області було сформовано функціональні та нефункціональні вимоги до системи, побудовано діаграми варіантів використання, що дало змогу деталізувати сценарії взаємодії користувача із системою та визначити основні функції застосунку.

Наступним етапом стало проєктування вебзастосунку. Було розглянуто сучасні архітектурні підходи до побудови вебсистем, у результаті чого обрано архітектуру на основі шаблону MVC із використанням патерну репозиторію. Такий підхід забезпечує розподіл відповідальності між компонентами системи, підвищує її масштабованість та спрощує супровід. На основі обраної архітектури було спроєктовано структуру бази даних і визначено основні модулі системи.

У процесі проєктування було побудовано відповідні діаграми:

- діаграма варіантів використання;
- діаграма класів;
- діаграми послідовності.

Також було визначено вимоги до інтерфейсу користувача та створено макети основних сторінок вебзастосунку, що дозволило краще уявити логіку роботи системи та спростило процес подальшої реалізації.

На етапі реалізації було обрано сучасні технології розробки, зокрема платформу ASP.NET Core та ORM Entity Framework Core для роботи з базою даних. Як середовище розробки використовувалося Microsoft Visual Studio. Реалізація застосунку здійснювалася з використанням ітеративного підходу, що дозволило поступово впроваджувати функціонал і вчасно вносити необхідні зміни.

Було реалізовано основні модулі системи, включаючи каталог книг, механізм пошуку, кошик покупця та оформлення замовлення. Окрему увагу приділено реалізації модуля кошика із використанням транзакцій для забезпечення цілісності даних та патерну репозиторію для ефективної роботи з базою даних.

Завершальним етапом стало тестування вебзастосунку. Було застосовано модульне тестування для перевірки коректності роботи окремих компонентів, зокрема логіки кошика та обробки даних книг. Результати тестування показали стабільну роботу системи та відповідність реалізованого функціоналу поставленим вимогам.

Отже, можна стверджувати, що мета кваліфікаційної роботи, яка полягала у розробленні вебзастосунку для продажу та купівлі книг, була досягнута. Розроблений застосунок успішно реалізований, протестований та відповідає сучасним вимогам до систем електронної комерції. Він забезпечує зручний інтерфейс користувача, коректну обробку даних і стабільну роботу.

Таким чином, результати виконаної роботи підтверджують її успішне завершення, а створений вебзастосунок може бути використаний як основа для подальшого розвитку та вдосконалення, а також має потенціал для практичного застосування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Yakaboo. URL: https://www.yakaboo.ua/ (дата звернення: 12.01.2026).
2. Книгарня «Є». URL: https://book-ye.com.ua/ (дата звернення: 15.01.2026).
3. Vivat. URL: https://vivat.com.ua/ (дата звернення: 05.02.2026).
4. Наш Формат. URL: https://nashformat.ua/ (дата звернення: 10.02.2026).
5. Bookling. URL: https://bookling.ua/ (дата звернення: 14.02.2026).
6. Monolithic Architecture. TechTarget. URL: https://www.techtarget.com/whatis/definition/monolithic-architecture (дата звернення: 22.02.2026).
7. Microservices Architecture. Atlassian. URL: https://www.atlassian.com/microservices/microservices-architecture (дата звернення: 25.02.2026).
8. Layered Architecture. ScienceDirect Topics. URL: https://www.sciencedirect.com/topics/computer-science/layered-architecture (дата звернення: 02.03.2026).
9. Oracle7 Server Concepts Manual. Distributed Systems. Oracle Documentation. URL: https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch20.htm (дата звернення: 05.03.2026).
10. Client-Server Architecture. QATestLab Training Center. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 10.03.2026).
11. Microsoft Learn. Overview of ASP.NET Core MVC. URL: https://learn.microsoft.com/aspnet/core/mvc/overview (дата звернення: 12.03.2026).

12. Data Binding and MVVM. Microsoft Learn. URL: https://learn.microsoft.com/en-us/windows/apps/develop/data-binding/data-binding-and-mvvm (дата звернення: 15.03.2026).

13. Indeed Editorial Team. Decomposition in Project Management. Indeed Career Guide. URL: https://www.indeed.com/career-advice/career-development/decomposition-project-management (дата звернення: 18.03.2026).

14. What is Sequence Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (дата звернення: 20.03.2026).

15. About Node.js. Node.js Official Website. URL: https://nodejs.org/en/about (дата звернення: 22.03.2026).

16. Spring Boot Documentation. Spring Documentation. URL: https://docs.spring.io/spring-boot/ (дата звернення: 25.03.2026).

17. Laravel Documentation. Laravel Documentation. URL: https://laravel.com/docs/13.x (дата звернення: 28.03.2026).

18. Adam Freeman. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. 9th ed. New York: Apress, 2021 (дата звернення: 01.04.2026).

19. ASP.NET MVC Overview. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview (дата звернення: 05.04.2026).

20. ADO.NET Overview. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/ (дата звернення: 10.04.2026).

21. Microsoft Docs. Create your first app with Entity Framework Core // Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli (дата звернення: 12.04.2026).

22. HTML: HyperText Markup Language. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 15.04.2026).

23. CSS: Cascading Style Sheets. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 17.04.2026).

24. React Documentation. React. URL: https://react.dev/learn (дата звернення: 20.04.2026).

25. Angular Documentation. Angular. URL: https://angular.dev/ (дата звернення: 22.04.2026).

26. Visual Studio Code Documentation. Microsoft. URL: https://code.visualstudio.com/docs (дата звернення: 25.04.2026).

27. Visual Studio 2022 Release Notes. Microsoft Learn. URL: https://learn.microsoft.com/en-us/visualstudio/releases/2022/release-notes (дата звернення: 27.04.2026).

28. MySQL Reference Manual. Introduction (MySQL 9.7 Documentation). URL: https://dev.mysql.com/doc/refman/9.7/en/introduction.html (дата звернення: 28.04.2026).

29. SQL Server Provider for Entity Framework Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/providers/sql-server/?tabs=dotnet-core-cli%2Csqlserver (дата звернення: 30.04.2026).

30. ISTQB Foundation Level Syllabus. Software Testing Fundamentals. URL: https://www.istqb.org/certifications/certified-tester-foundation-level (дата звернення: 05.05.2026).

Додаток А

(обов’язковий)

ГРАФІЧНІ МАТЕРІАЛИ

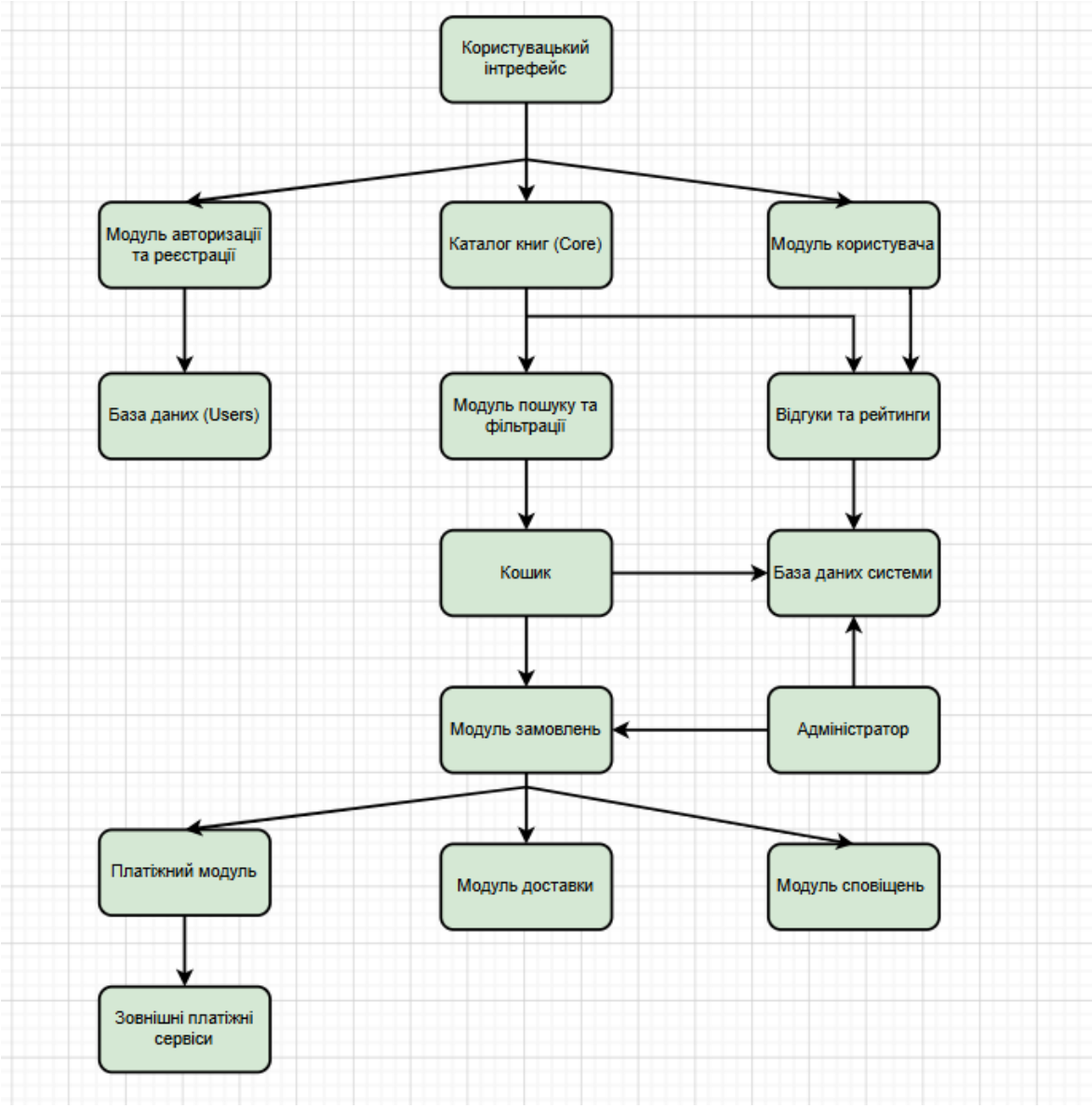


Рисунок А.1 – Діаграма міжмодульних зв’язків

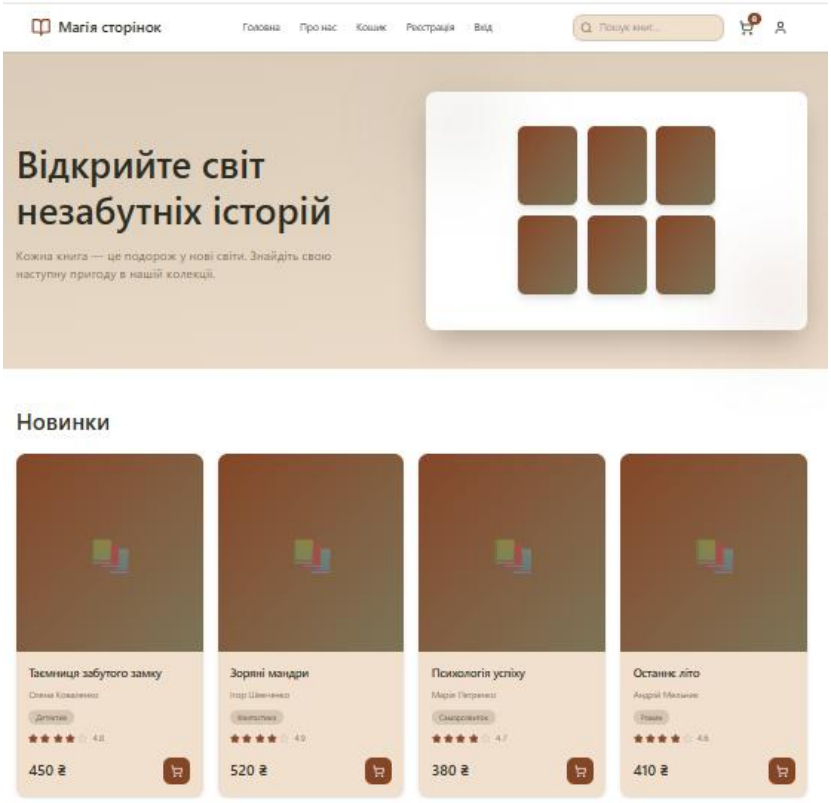


Рисунок А.2 – Макет головної сторінки застосунку

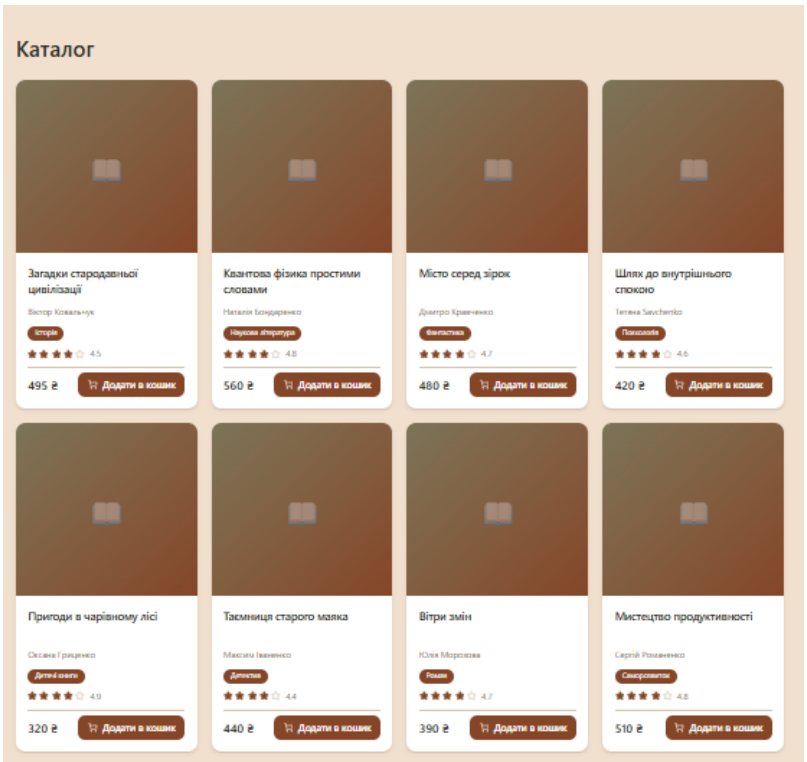


Рисунок А.3- Макет каталогу книг

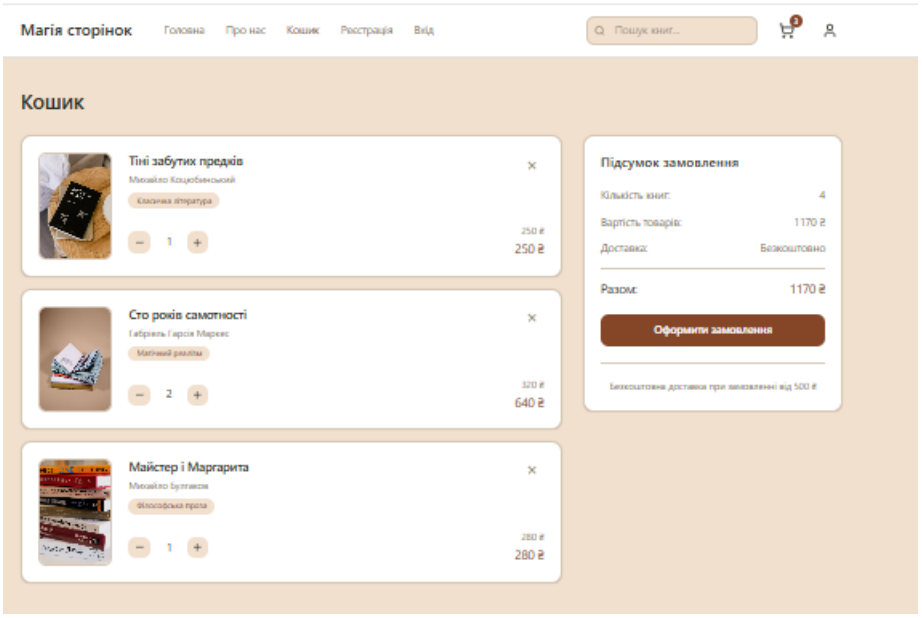


Рисунок А.4- Макет кошику покупок користувача

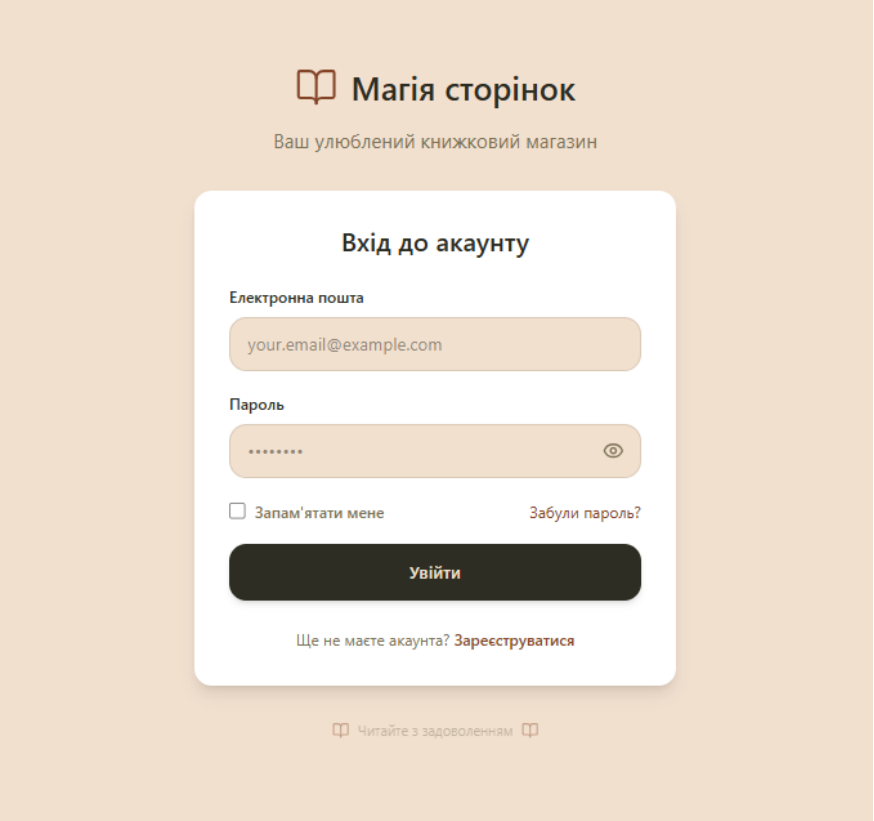


Рисунок А.5 – Макет сторінки реєстрації та авторизації

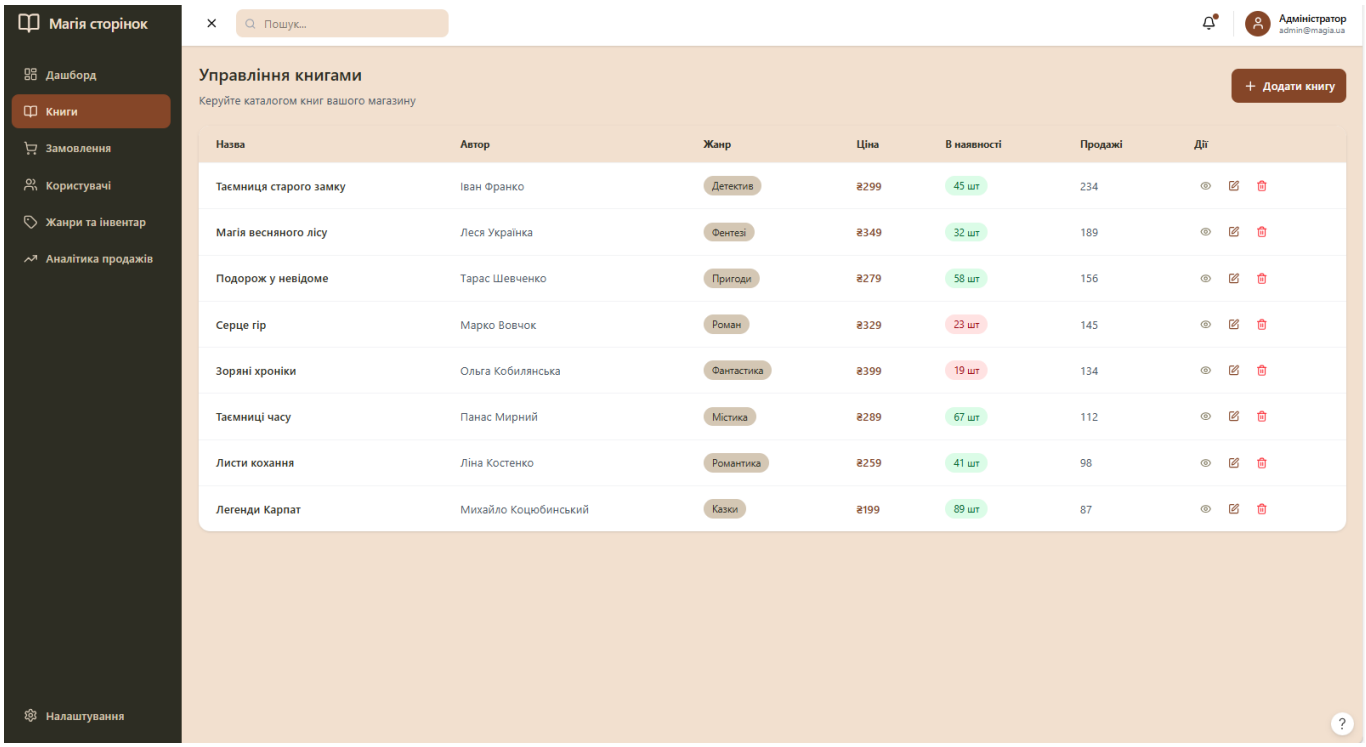


Рисунок А.6 – Макет сторінки адміністративної панелі управління книгами

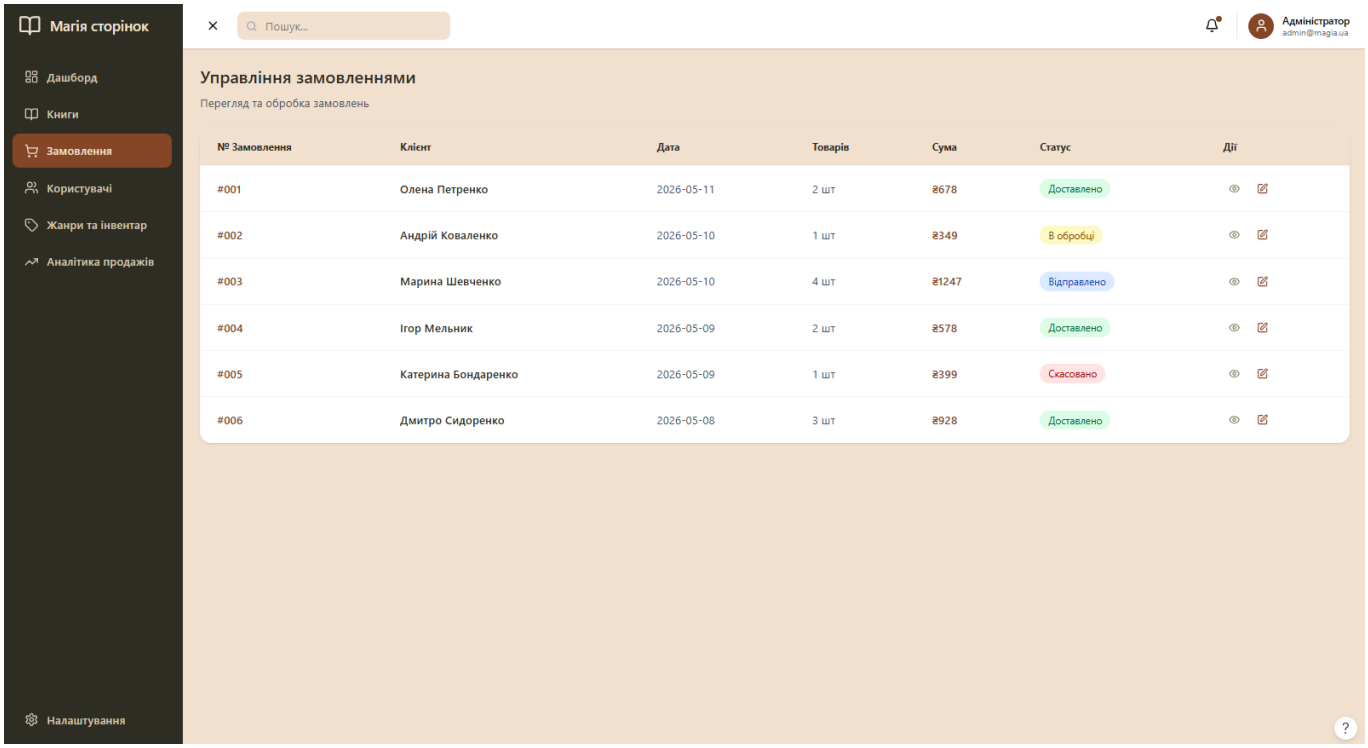


Рисунок А.7- Макет сторінки адміністративної панелі замовлення

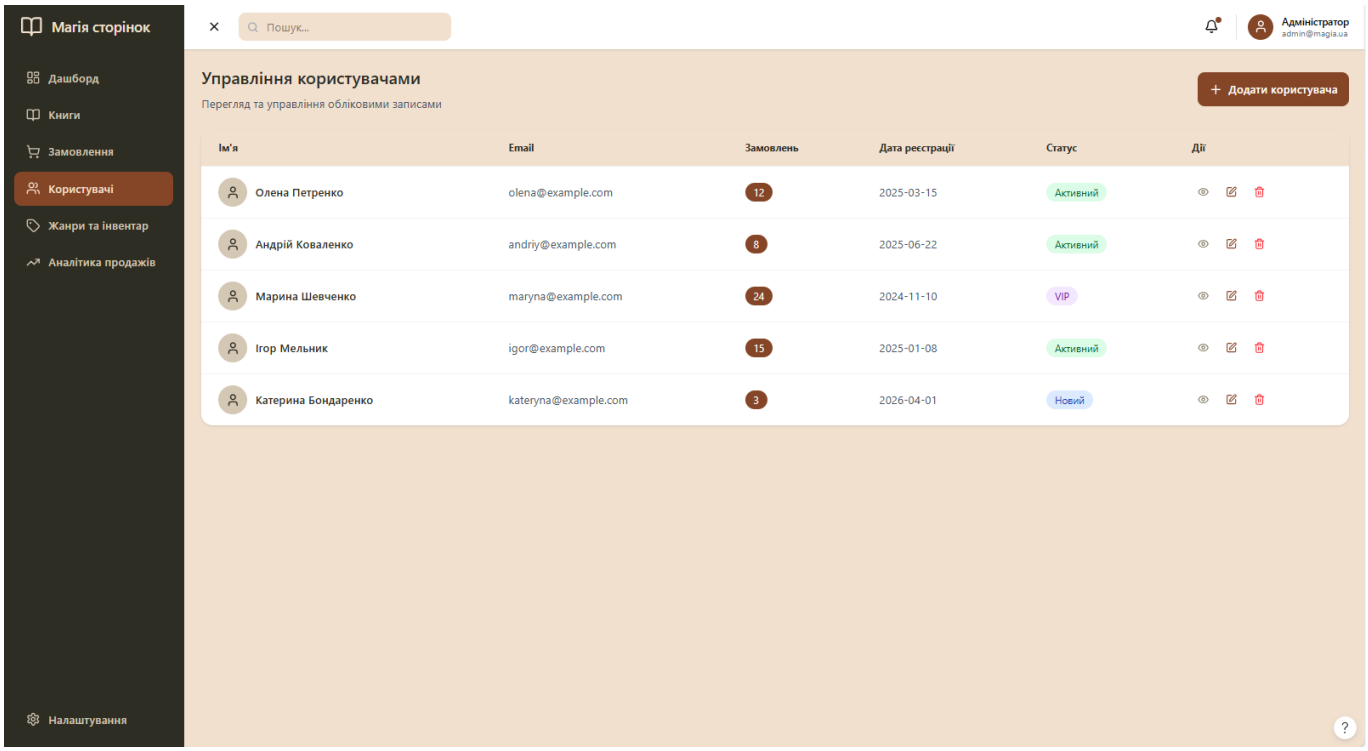


Рисунок А.8 - Макет сторінки адміністративної панелі управління користувачами

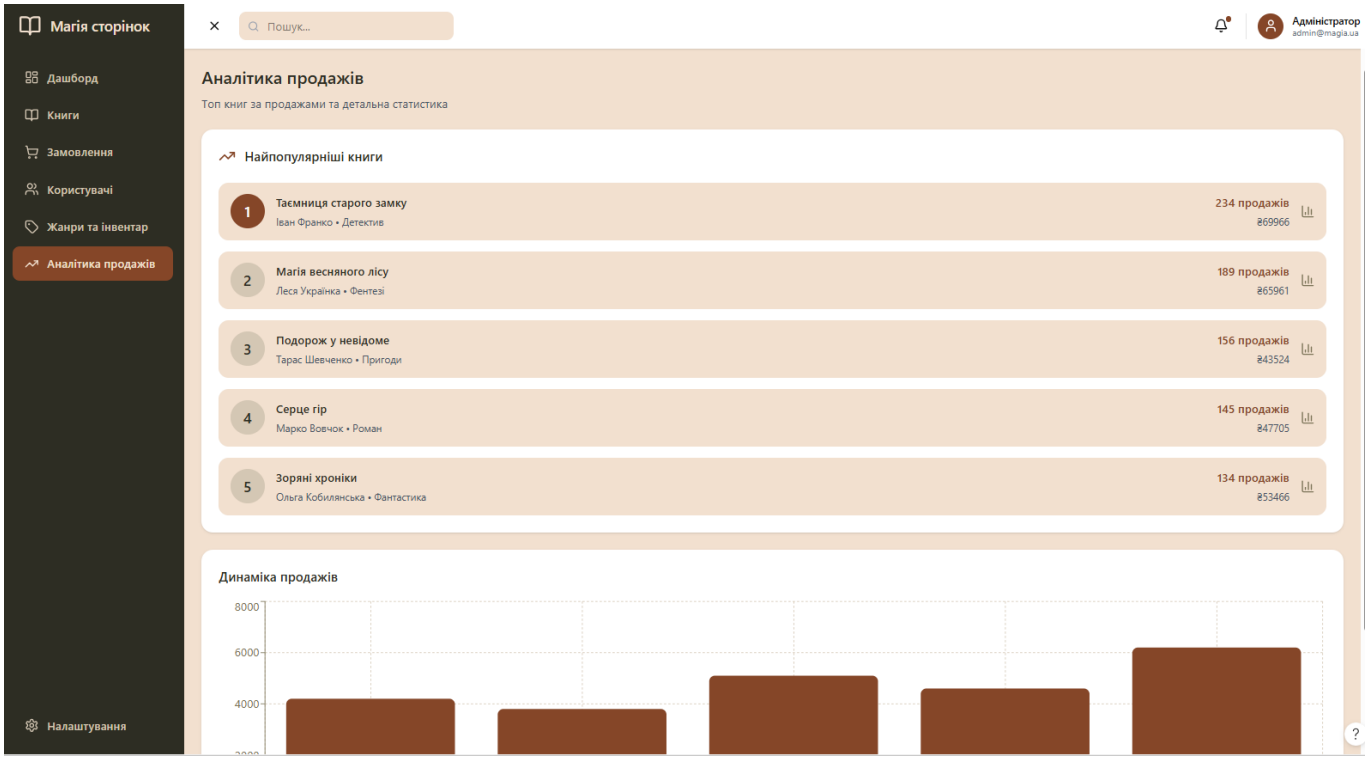


Рисунок А.9 - Макет сторінки адміністративної панелі аналітика продажів

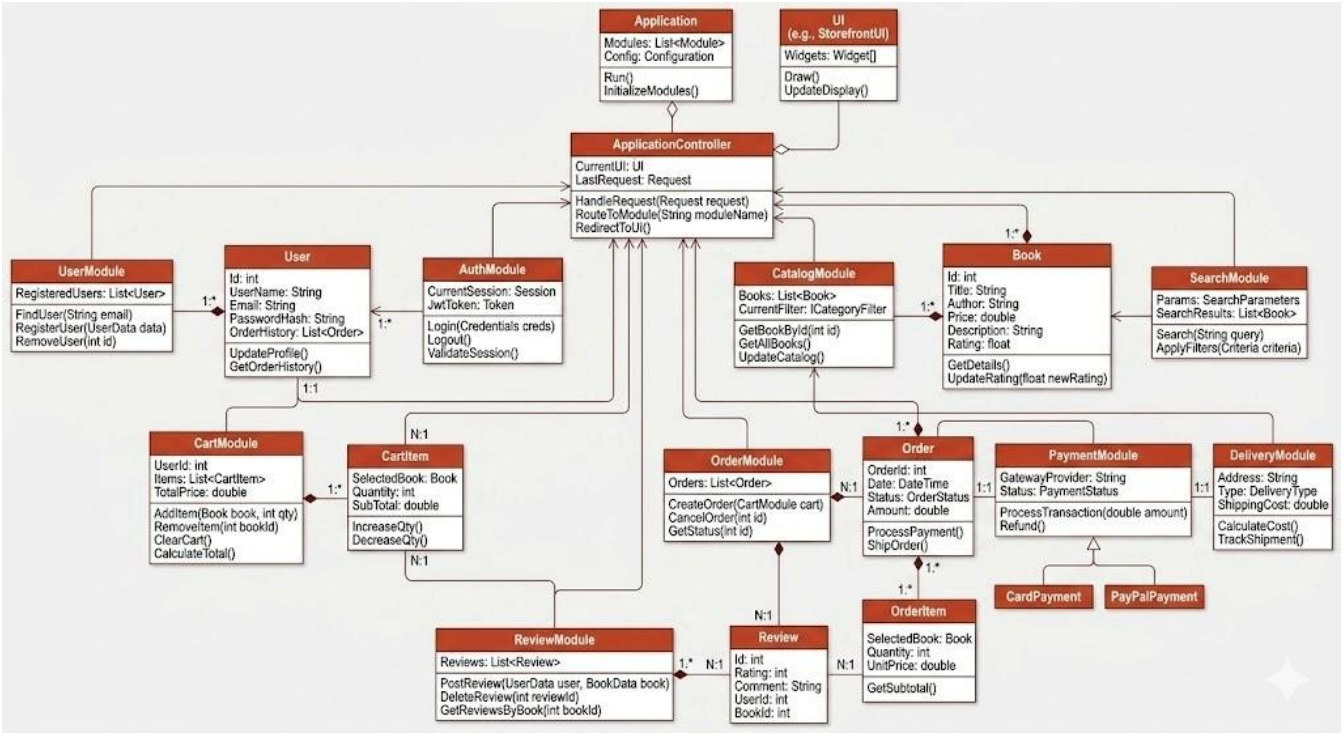
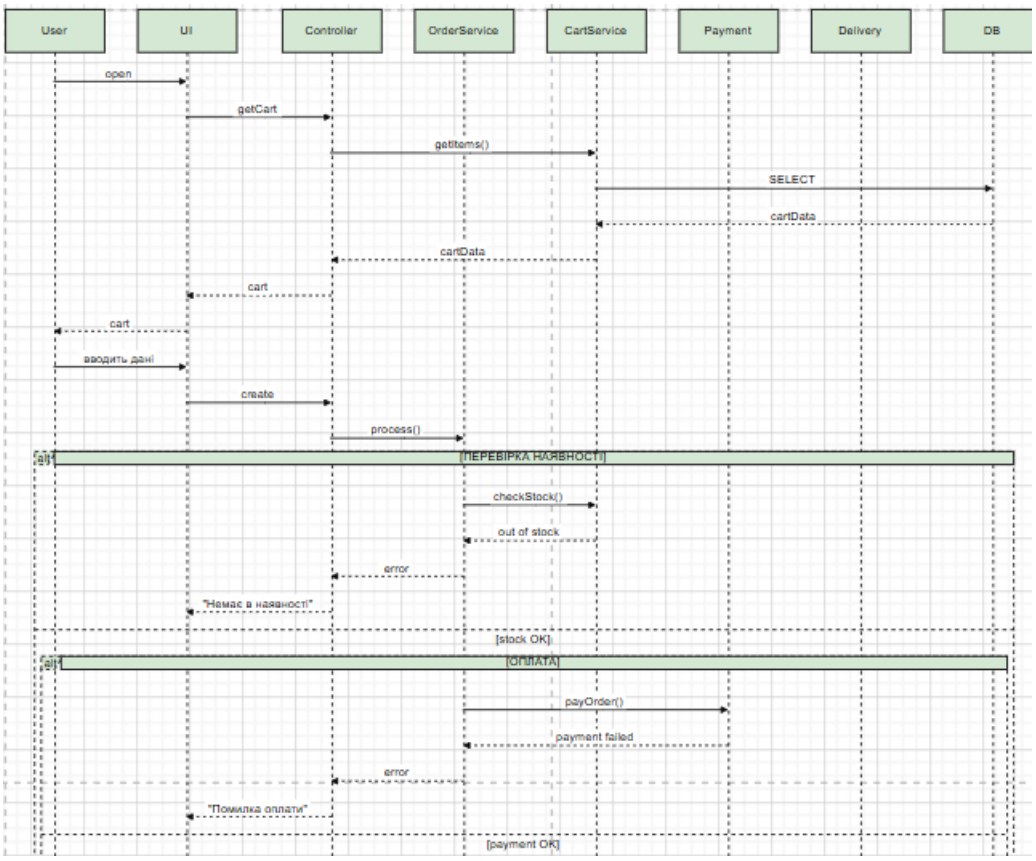


Рисунок А.10 – Діаграма класів



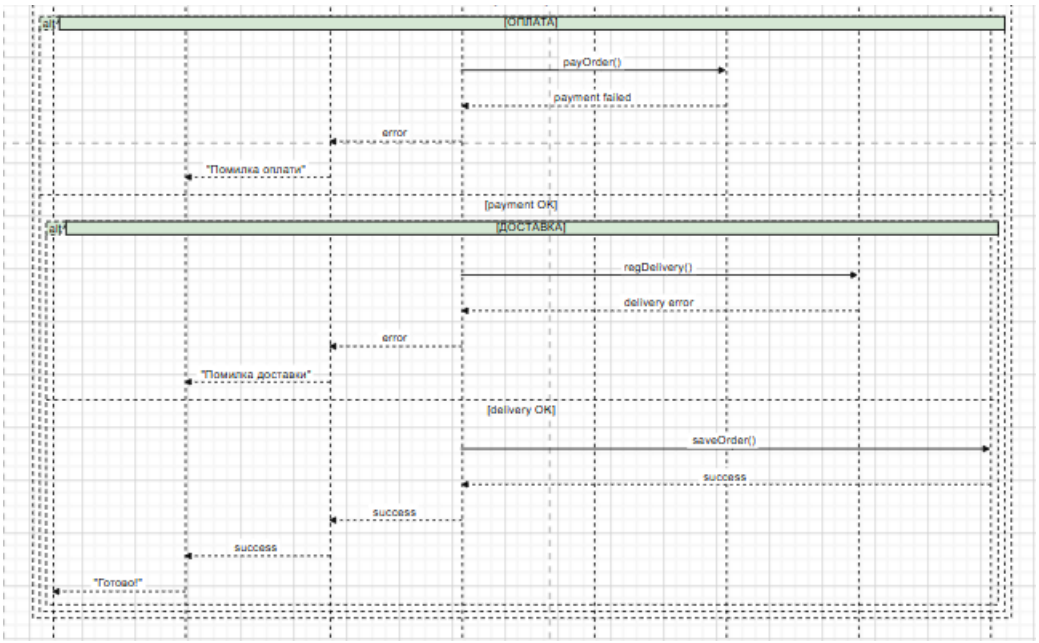


Рисунок А.11 – Діаграма послідовності оформлення замовлення

Додаток Б

(обов'язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

Код контролера BookController:

```
using BookShoppingCartMvcUI.Shared;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace BookShoppingCartMvcUI.Controllers;

[Authorize(Roles = nameof(Roles.Admin))]
public class BookController : Controller
{
    private readonly IBookRepository _bookRepo;
    private readonly IGenreRepository _genreRepo;
    private readonly IFileService _fileService;

    public BookController(IBookRepository bookRepo, IGenreRepository
genreRepo, IFileService fileService)
    {
        _bookRepo = bookRepo;
        _genreRepo = genreRepo;
        _fileService = fileService;
    }

    public async Task<IActionResult> Index()
    {
        var books = await _bookRepo.GetBooks();
        return View(books);
    }

    public async Task<IActionResult> AddBook()
    {
        var genreSelectList = (await _genreRepo.GetGenres()).Select(genre
=> new SelectListItem
        {
            Text = genre.GenreName,
            Value = genre.Id.ToString(),
        });
        BookDTO bookToAdd = new() { GenreList = genreSelectList };
        return View(bookToAdd);
    }

    [HttpPost]
    public async Task<IActionResult> AddBook(BookDTO bookToAdd)
    {
        var genreSelectList = (await _genreRepo.GetGenres()).Select(genre
=> new SelectListItem
        {
            Text = genre.GenreName,
            Value = genre.Id.ToString(),
        });
        bookToAdd.GenreList = genreSelectList;
```

```
            if (!ModelState.IsValid)
                return View(bookToAdd);

            try
            {
                if (bookToAdd.ImageFile != null)
                {
                    if(bookToAdd.ImageFile.Length> 1 * 1024 * 1024)
                    {
                        throw new InvalidOperationException("Image file can
not exceed 1 MB");
                    }
                    string[] allowedExtensions = { ".jpeg", ".jpg", ".png" };
                    string                                          imageName=await
_fileService.SaveFile(bookToAdd.ImageFile, allowedExtensions);
                    bookToAdd.Image = imageName;
                }
                // manual mapping of BookDTO -> Book
                Book book = new()
                {
                    Id = bookToAdd.Id,
                    BookName = bookToAdd.BookName,
                    AuthorName = bookToAdd.AuthorName,
                    Image = bookToAdd.Image,
                    GenreId = bookToAdd.GenreId,
                    Price = bookToAdd.Price,
                    Annotation = bookToAdd.Annotation  // Ensure Annotation is
being saved
                };
                await _bookRepo.AddBook(book);


                TempData["successMessage"] = "Book is added successfully";
                return RedirectToAction(nameof(AddBook));
            }
            catch (InvalidOperationException ex)
            {
                TempData["errorMessage"]= ex.Message;
                return View(bookToAdd);
            }
            catch (FileNotFoundException ex)
            {
                TempData["errorMessage"] = ex.Message;
                return View(bookToAdd);
            }
            catch (Exception ex)
            {
                TempData["errorMessage"] = "Error on saving data";
                return View(bookToAdd);
            }
        }

        public async Task<IActionResult> UpdateBook(int id)
        {
            var book = await _bookRepo.GetBookById(id);
            if(book==null)
            {
                TempData["errorMessage"] = $"Book with the id: {id} does not
found";
                return RedirectToAction(nameof(Index));
            }
```

```
            var genreSelectList = (await _genreRepo.GetGenres()).Select(genre
=> new SelectListItem
            {
                Text = genre.GenreName,
                Value = genre.Id.ToString(),
                Selected=genre.Id==book.GenreId
            });
            BookDTO bookToUpdate = new()
            {
                GenreList = genreSelectList,
                BookName=book.BookName,
                AuthorName=book.AuthorName,
                GenreId=book.GenreId,
                Price=book.Price,
                Image=book.Image
            };
            return View(bookToUpdate);
        }

        [HttpPost]
        public async Task<IActionResult> UpdateBook(BookDTO bookToUpdate)
        {
            var genreSelectList = (await _genreRepo.GetGenres()).Select(genre
=> new SelectListItem
            {
                Text = genre.GenreName,
                Value = genre.Id.ToString(),
                Selected=genre.Id==bookToUpdate.GenreId
            });
            bookToUpdate.GenreList = genreSelectList;

            if (!ModelState.IsValid)
                return View(bookToUpdate);

            try
            {
                string oldImage = "";
                if (bookToUpdate.ImageFile != null)
                {
                    if (bookToUpdate.ImageFile.Length > 1 * 1024 * 1024)
                    {
                        throw new InvalidOperationException("Image file can
not exceed 1 MB");
                    }
                    string[] allowedExtensions = { ".jpeg", ".jpg", ".png" };
                    string imageName = await
_fileService.SaveFile(bookToUpdate.ImageFile, allowedExtensions);
                    // hold the old image name. Because we will delete this
image after updating the new
                    oldImage = bookToUpdate.Image;
                    bookToUpdate.Image = imageName;
                }
                // manual mapping of BookDTO -> Book
                Book book = new()
                {
                    Id = bookToUpdate.Id,
                    BookName = bookToUpdate.BookName,
                    AuthorName = bookToUpdate.AuthorName,
                    GenreId = bookToUpdate.GenreId,
                    Price = bookToUpdate.Price,
                    Image = bookToUpdate.Image,
                    Annotation = bookToUpdate.Annotation  // Ensure Annotation
is updated
```

```
                };
                await _bookRepo.UpdateBook(book);


                // if image is updated, then delete it from the folder too
                if (!string.IsNullOrWhiteSpace(oldImage))
                {
                    _fileService.DeleteFile(oldImage);
                }
                TempData["successMessage"] = "Book is updated successfully";
                return RedirectToAction(nameof(Index));
            }
            catch (InvalidOperationException ex)
            {
                TempData["errorMessage"] = ex.Message;
                return View(bookToUpdate);
            }
            catch (FileNotFoundException ex)
            {
                TempData["errorMessage"] = ex.Message;
                return View(bookToUpdate);
            }
            catch (Exception ex)
            {
                TempData["errorMessage"] = "Error on saving data";
                return View(bookToUpdate);
            }
        }

        public async Task<IActionResult> DeleteBook(int id)
        {
            try
            {
                var book = await _bookRepo.GetBookById(id);
                if (book == null)
                {
                    TempData["errorMessage"] = $"Book with the id: {id} does
not found";
                }
                else
                {
                    await _bookRepo.DeleteBook(book);
                    if (!string.IsNullOrWhiteSpace(book.Image))
                    {
                        _fileService.DeleteFile(book.Image);
                    }
                }
            }
            catch (InvalidOperationException ex)
            {
                TempData["errorMessage"] = ex.Message;
            }
            catch (FileNotFoundException ex)
            {
                TempData["errorMessage"] = ex.Message;
            }
            catch (Exception ex)
            {
                TempData["errorMessage"] = "Error on deleting the data";
            }
            return RedirectToAction(nameof(Index));
        }
    }
```

Код контролера GenreController:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace BookShoppingCartMvcUI.Controllers
{
    [Authorize(Roles = nameof(Roles.Admin))]
    public class GenreController : Controller
    {
        private readonly IGenreRepository _genreRepo;

        public GenreController(IGenreRepository genreRepo)
        {
            _genreRepo = genreRepo;
        }

        public async Task<IActionResult> Index()
        {
            var genres = await _genreRepo.GetGenres();
            return View(genres);
        }

        public IActionResult AddGenre()
        {
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> AddGenre(GenreDTO genre)
        {
            if(!ModelState.IsValid)
            {
                return View(genre);
            }
            try
            {
                var genreToAdd = new Genre { GenreName = genre.GenreName,
Id = genre.Id };
                await _genreRepo.AddGenre(genreToAdd);
                TempData["successMessage"] = "Genre added successfully";
                return RedirectToAction(nameof(AddGenre));
            }
            catch(Exception ex)
            {
                TempData["errorMessage"] = "Genre could not added!";
                return View(genre);
            }

        }

        public async Task<IActionResult> UpdateGenre(int id)
        {
            var genre = await _genreRepo.GetGenreById(id);
            if (genre is null)
                throw new InvalidOperationException($"Genre with id: {id}
does not found");
            var genreToUpdate = new GenreDTO
            {
                Id = genre.Id,
                GenreName = genre.GenreName
            };
```

```
                return View(genreToUpdate);
            }

            [HttpPost]
            public async Task<IActionResult> UpdateGenre(GenreDTO
genreToUpdate)
            {
                if (!ModelState.IsValid)
                {
                    return View(genreToUpdate);
                }
                try
                {
                    var genre = new Genre { GenreName =
genreToUpdate.GenreName, Id = genreToUpdate.Id };
                    await _genreRepo.UpdateGenre(genre);
                    TempData["successMessage"] = "Genre is updated
successfully";
                    return RedirectToAction(nameof(Index));
                }
                catch (Exception ex)
                {
                    TempData["errorMessage"] = "Genre could not updated!";
                    return View(genreToUpdate);
                }

            }

            public async Task<IActionResult> DeleteGenre(int id)
            {
                var genre = await _genreRepo.GetGenreById(id);
                if (genre is null)
                    throw new InvalidOperationException($"Genre with id: {id}
does not found");
                await _genreRepo.DeleteGenre(genre);
                return RedirectToAction(nameof(Index));

            }

        }
    }
```

Код контролера StockController:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace BookShoppingCartMvcUI.Controllers
{
    [Authorize(Roles=nameof(Roles.Admin))]
    public class StockController : Controller
    {
        private readonly IStockRepository _stockRepo;

        public StockController(IStockRepository stockRepo)
        {
            _stockRepo = stockRepo;
        }

        public async Task<IActionResult> Index(string sTerm="")
        {
            var stocks=await _stockRepo.GetStocks(sTerm);
            return View(stocks);
```

```
            }

            public async Task<IActionResult> ManangeStock(int bookId)
            {
                var existingStock = await _stockRepo.GetStockByBookId(bookId);
                var stock = new StockDTO
                {
                    BookId = bookId,
                    Quantity = existingStock != null
                ? existingStock.Quantity : 0
                };
                return View(stock);
            }

            [HttpPost]
            public async Task<IActionResult> ManangeStock(StockDTO stock)
            {
                if (!ModelState.IsValid)
                    return View(stock);
                try
                {
                    await _stockRepo.ManageStock(stock);
                    TempData["successMessage"]    =    "Stock    is    updated
successfully.";
                }
                catch (Exception ex)
                {
                    TempData["errorMessage"] = "Something went wrong!!";
                }

                return RedirectToAction(nameof(Index));
            }
        }
    }
```

Код контролера CartController:

```
using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;


namespace BookShoppingCartMvcUI.Controllers

{

    [Authorize]

    public class CartController : Controller

    {

        private readonly ICartRepository _cartRepo;


        public CartController(ICartRepository cartRepo)

        {

            _cartRepo = cartRepo;

        }

        public async Task<IActionResult> AddItem(int bookId, int qty = 1,
int redirect = 0)
```

```
{
    var cartCount = await _cartRepo.AddItem(bookId, qty);
    if (redirect == 0)
        return Ok(cartCount);
    return RedirectToAction("GetUserCart");
}

public async Task<IActionResult> RemoveItem(int bookId)
{
    var cartCount = await _cartRepo.RemoveItem(bookId);
    return RedirectToAction("GetUserCart");
}
public async Task<IActionResult> GetUserCart()
{
    var cart = await _cartRepo.GetUserCart();
    return View(cart);
}

public async Task<IActionResult> GetTotalItemInCart()
{
    int cartItem = await _cartRepo.GetCartItemCount();
    return Ok(cartItem);
}

public IActionResult Checkout()
{
    return View();
}

[HttpPost]
public async Task<IActionResult> Checkout(CheckoutModel model)
{
    if (!ModelState.IsValid)
        return View(model);
    bool isCheckedOut = await _cartRepo.DoCheckout(model);
    if (!isCheckedOut)
        return RedirectToAction(nameof(OrderFailure));
```

```
                return RedirectToAction(nameof(OrderSuccess));
            }

            public IActionResult OrderSuccess()
            {
                return View();
            }

            public IActionResult OrderFailure()
            {
                return View();
            }

        }
    }
```

Код контролера AdminOperationsController:

```
    using BookShoppingCartMvcUI.Constants;
    using Microsoft.AspNetCore.Authorization;
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.AspNetCore.Mvc.Rendering;

    namespace BookShoppingCartMvcUI.Controllers;

    [Authorize(Roles = nameof(Roles.Admin))]
    public class AdminOperationsController : Controller
    {
        private readonly IUserOrderRepository _userOrderRepository;
        public                  AdminOperationsController(IUserOrderRepository
userOrderRepository)
        {
            _userOrderRepository = userOrderRepository;
        }

        public async Task<IActionResult> AllOrders()
        {
            var orders = await _userOrderRepository.UserOrders(true);
            return View(orders);
```

```
        }

        public async Task<IActionResult> TogglePaymentStatus(int orderId)
        {
            try
            {
                await _userOrderRepository.TogglePaymentStatus(orderId);
            }
            catch (Exception ex)
            {
                // log exception here
            }
            return RedirectToAction(nameof(AllOrders));
        }

        public async Task<IActionResult> UpdateOrderStatus(int orderId)
        {
            var order = await _userOrderRepository.GetOrderById(orderId);
            if (order == null)
            {
                throw new InvalidOperationException($"Order with id:{orderId}
does not found.");
            }
            var              orderStatusList              =              (await
_userOrderRepository.GetOrderStatuses()).Select(orderStatus =>
            {
                return new SelectListItem { Value = orderStatus.Id.ToString(),
Text = orderStatus.StatusName, Selected = order.OrderStatusId == orderStatus.Id
};
            });
            var data = new UpdateOrderStatusModel
            {
                OrderId = orderId,
                OrderStatusId = order.OrderStatusId,
                OrderStatusList = orderStatusList
            };
            return View(data);
        }
```

```
        [HttpPost]
        public async Task<IActionResult> UpdateOrderStatus(UpdateOrderStatusModel data)
        {
            try
            {
                if (!ModelState.IsValid)
                {
                    data.OrderStatusList = (await _userOrderRepository.GetOrderStatuses()).Select(orderStatus =>
                    {
                        return new SelectListItem { Value = orderStatus.Id.ToString(), Text = orderStatus.StatusName, Selected = orderStatus.Id == data.OrderStatusId };
                    });

                    return View(data);
                }
                await _userOrderRepository.ChangeOrderStatus(data);
                TempData["msg"] = "Updated successfully";
            }
            catch (Exception ex)
            {
                // catch exception here
                TempData["msg"] = "Something went wrong";
            }
            return RedirectToAction(nameof(UpdateOrderStatus), new { orderId = data.OrderId });
        }


        public IActionResult Dashboard()
        {
            return View();
        }

    }
```

Код контролера UserOrderController:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace BookShoppingCartMvcUI.Controllers
{
    [Authorize]
    public class UserOrderController : Controller
    {
        private readonly IUserOrderRepository _userOrderRepo;

        public UserOrderController(IUserOrderRepository userOrderRepo)
        {
            _userOrderRepo = userOrderRepo;
        }
        public async Task<IActionResult> UserOrders()
        {
            var orders = await _userOrderRepo.UserOrders();
            return View(orders);
        }
    }
}
```

Код контролера HomeController:

```
using BookShoppingCartMvcUI.Models;
using BookShoppingCartMvcUI.Models.DTOs;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace BookShoppingCartMvcUI.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly IHomeRepository _homeRepository;

        public HomeController(ILogger<HomeController> logger,
IHomeRepository homeRepository)
        {
```

```
                _homeRepository = homeRepository;
                _logger = logger;
            }

            public async Task<IActionResult> Index(string sterm = "", int
genreId = 0)
            {
                IEnumerable<Book> books = await
_homeRepository.GetBooks(sterm, genreId);
                IEnumerable<Genre> genres = await _homeRepository.Genres();
                BookDisplayModel bookModel = new BookDisplayModel
                {
                    Books = books,
                    Genres = genres,
                    STerm = sterm,
                    GenreId = genreId
                };
                return View(bookModel);
            }

            public IActionResult Privacy()
            {
                return View();
            }

            [ResponseCache(Duration = 0, Location =
ResponseCacheLocation.None, NoStore = true)]
            public IActionResult Error()
            {
                return View(new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
            }
        }
    }
```

Код контролера ReportsController:

```
using Microsoft.AspNetCore.Mvc;

namespace BookShoppingCartMvcUI.Controllers;
```

```
public class ReportsController : Controller
{
    private readonly IReportRepository _reportRepository;
    public ReportsController(IReportRepository reportRepository)
    {
        _reportRepository = reportRepository;
    }
    // GET: ReportsController
    public async Task<ActionResult> TopFiveSellingBooks(DateTime? sDate = null, DateTime? eDate = null)
    {
        try
        {
            // by default, get last 7 days record
            DateTime startDate = sDate ?? DateTime.UtcNow.AddDays(-7);
            DateTime endDate = eDate ?? DateTime.UtcNow;
            var topFiveSellingBooks = await _reportRepository.GetTopNSellingBooksByDate(startDate, endDate);
            var vm = new TopNSoldBooksVm(startDate, endDate, topFiveSellingBooks);
            return View(vm);
        }
        catch (Exception ex)
        {
            TempData["errorMessage"] = "Something went wrong";
            return RedirectToAction("Index", "Home");
        }
}
}
```

Додаток В

(обов’язковий)

КЕРІВНИЦТВО КОРИСТУВАЧА

1. Загальна інформація про систему

Вебзастосунок для купівлі та продажу книг призначений для забезпечення зручного доступу користувачів до каталогу книжкової продукції, оформлення замовлень та адміністрування системи через вебінтерфейс. Система дозволяє переглядати книги, здійснювати пошук за різними параметрами, додавати товари до кошика та оформлювати замовлення онлайн.

Основними категоріями користувачів системи є:

- покупець;
- адміністратор.

Для роботи із системою необхідно:

- комп’ютер, ноутбук;
- доступ до мережі Інтернет;
- сучасний веббраузер (Google Chrome, Mozilla Firefox, Microsoft Edge).

Для початку роботи необхідно відкрити головну сторінку вебзастосунку у браузері.

2.Функції користувача

Після відкриття головної сторінки користувач отримує доступ до каталогу книг. Для зручності передбачено систему пошуку та фільтрації товарів за назвою, автором, жанром або ціною.

Для перегляду інформації про книгу необхідно:

1. Вибрати книгу із каталогу.
2. Перейти на сторінку товару.
3. Ознайомитися з описом, ціною, автором та наявністю книги.

Для придбання книги користувач виконує такі дії:

1. Натискає кнопку «Додати до кошика».
2. Переходить до сторінки кошика.
3. Перевіряє список вибраних товарів.
4. Натискає кнопку «Оформити замовлення».
5. Вводить контактні дані, місто і відділення пошти.
6. Підтверджує оформлення замовлення.

Після успішного оформлення замовлення система автоматично зберігає інформацію про покупку та відображає повідомлення про успішне завершення операції.

Для роботи із персональним обліковим записом користувач може:

- зареєструвати новий акаунт;
- авторизуватися у системі;
- переглядати історію замовлень;
- редагувати особисті дані;
- виходити із системи.

3. Функції адміністратора

Адміністратор має розширені права доступу до системи та відповідає за управління контентом вебзастосунку.

Для входу в адміністративну панель необхідно:

1. Авторизуватися у системі під обліковим записом адміністратора.
2. Перейти до розділу «Панель адміністратора».

Адміністратор може виконувати такі дії:

- додавати нові книги до каталогу;
- редагувати інформацію про книги;
- видаляти книги із системи;
- переглядати список користувачів;
- керувати замовленнями;
- змінювати статус замовлень;
- контролювати актуальність інформації у каталозі.

Для додавання нової книги необхідно:

1. Перейти у розділ «Книги».
2. Натиснути кнопку «Додати книгу».
3. Ввести назву, автора, жанр, опис та ціну.
4. Завантажити зображення книги.
5. Підтвердити збереження даних.

Для редагування або видалення книги адміністратор обирає відповідний товар зі списку та використовує кнопки «Редагувати» або «Видалити».

4. Призначення основних елементів інтерфейсу

Основні елементи інтерфейсу вебзастосунку:

- «Головна сторінка» – відображення каталогу книг;
- «Каталог» – список доступних товарів;
- «Кошик» – перегляд вибраних книг;
- «Профіль» – персональні дані користувача;
- «Замовлення» – інформація про покупки;

- «Панель адміністратора» – керування системою.

В.5 Можливі помилки та способи їх усунення

Під час роботи із системою можуть виникати такі помилки:

- неправильне введення логіна або пароля;
- відсутність підключення до мережі Інтернет;
- неможливість оформлення замовлення через незаповнені поля.

Для усунення помилок необхідно:

- перевірити правильність введених даних;
- оновити сторінку;
- повторити авторизацію;
- перевірити підключення до Інтернету.

У разі виникнення критичних помилок необхідно звернутися до адміністратора системи.

Додаток Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Кафедра інженерії програмного забезпечення

Тема кваліфікаційної роботи:

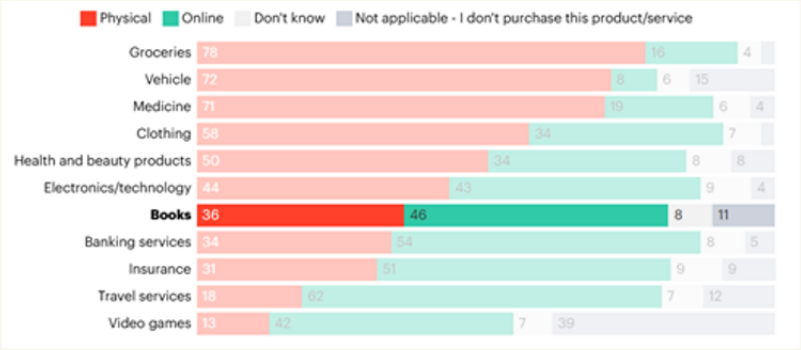
ВЕБЗАСТОСНОК ДЛЯ ПРОДАЖУ ТА КУПІВЛІ КНИГ

Виконала: студентка 4 курсу, ІПЗ-22-1, Мельник Вікторія
Керівник: д-р фіз-мат. наук, проф. Бедратюк Леонід Петрович

Рисунок Г.1 – Слайд 1

Актуальність теми

Розвиток електронної комерції та зростання онлайн-покупок в Україні зумовлюють підвищений попит на інтернет-магазини книг. Сучасні користувачі очікують швидкий, зручний та персоналізований доступ до літератури. Тому розробка вебзастосунку інтернет-магазину книг є актуальною та практично значущою, оскільки спрямована на покращення користувацького досвіду та автоматизацію процесів продажу.



Статистика розподілу споживчих переваг між фізичними та онлайн-покупками за категоріями товарів та послуг.

Рисунок Г.2 – Слайд 2

МЕТА І ЗАВДАННЯ

Мета:

Розроблення вебзастосунку для купівлі та продажу книг, який забезпечує зручну взаємодію користувачів і відповідає сучасним вимогам електронної комерції.

Завдання:

- проаналізувати предметну область електронної комерції;
- дослідити існуючі системи продажу книг;
- сформувати вимоги до функціоналу та архітектури;
- обрати архітектурний підхід і програмні патерни;
- спроєктувати базу даних та структуру модулів;
- визначити технології реалізації;
- реалізувати основні функції системи;
- протестувати розроблений вебзастосунок;
- проаналізувати результати та сформулювати висновки.

Рисунок Г.3 – Слайд3

АНАЛІЗ НАЯВНОГО ПРОГРАМНО-ТЕХНІЧНОГО забезпечення

Критерій	*Yakaboo*	*Книгарня «Є»*	*Vivat*	*Наш Формат*	*Bookling*
Асортимент	Дуже широкий	Середній	Обмежений	Середній	Спеціалізований
Пошук і фільтри	Розвинені	Базові	Базові	Середні	Розширені
Інтерфейс	Перевантажений	Зручний	Простий	Сучасний	Складний
Оформлення	Стандартне	Дуже просте	Стандартне	Стандартне	Стандартне
Особистий кабінет	Розширений	Базовий	Базовий	Базовий	Середній
Додаткові функції	Рекомендації	Рецензії	Передзамовлення	Мультимедіа	Багатомовність
Продуктивність	Середня	Висока	Висока	Середня	Середня/нижча
Недоліки	Перевантаження	Обмеженість	Обмеженість	Навантаження	Складність

Рисунок Г.4 – Слайд 4

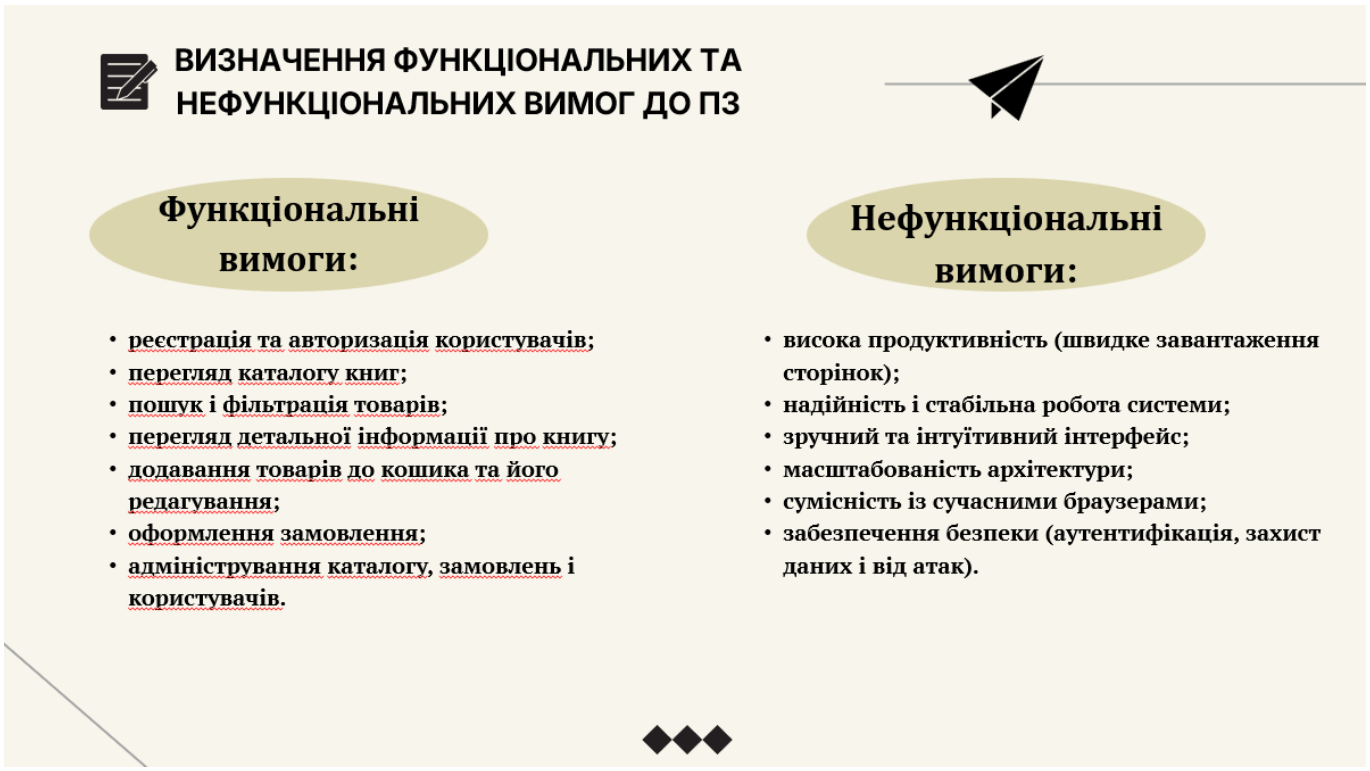
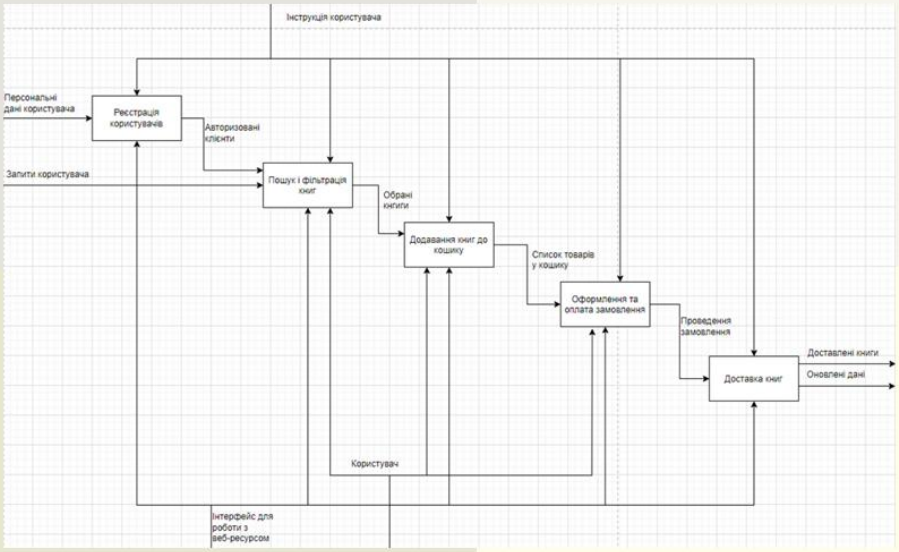


Рисунок Г.5– Слайд 5

Рисунок Г.6 – Слайд 6

ОПИС ДЕКОМПОЗИЦІЇ, ЗАЛЕЖНОСТЕЙ, ІНТЕРФЕЙСІВ



Діаграма декомпозиції

Рисунок Г.7 – Слайд 7

ПРОЄКТУВАННЯ МОДУЛІВ І ДАНИХ

Діаграма міжмодульних зв'язків

Рисунок Г.8 – Слайд 8

АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ

У процесі розробки вебзастосунку було проаналізовано сучасні серверні, клієнтські та базові технології. Для серверної частини обрано ASP.NET Core MVC завдяки високій продуктивності, безпеці та масштабованості. Для доступу до даних використано Entity Framework Core, що спрощує роботу з базою даних та підвищує ефективність розробки.

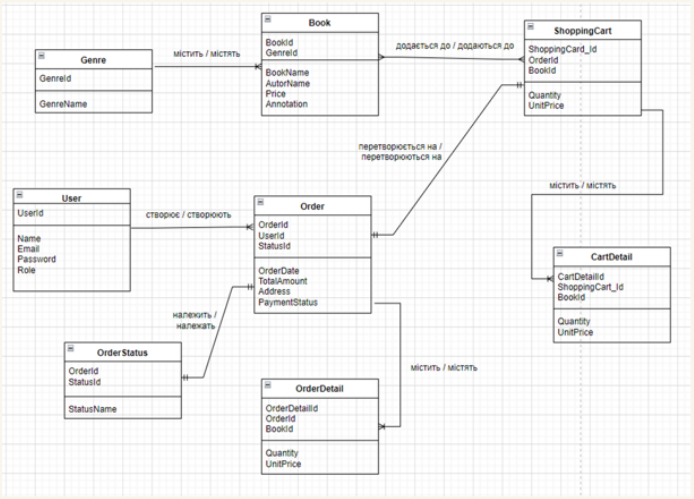
Клієнтську частину реалізовано за допомогою Razor Views, що забезпечує зручний, адаптивний інтерфейс. Як систему управління базою даних обрано Microsoft SQL Server через її надійну інтеграцію з .NET.

Таким чином, сформовано оптимальний технологічний стек для створення продуктивного та масштабованого вебзастосунку інтернет-магазину книг.

Рисунок Г.9 – Слайд 9

РЕАЛІЗАЦІЯ МОДУЛІВ І БАЗА ДАНИХ

- Використано ASP.NET Core MVC (.NET);
- Архітектура: MVC ;
- База даних: Entity Framework Core ;
- Реалізовано модулі: користувачі, каталог, кошик, замовлення, адмін-панель;
- Інтерфейс: Razor + Bootstrap;
- Додано пошук, фільтрацію та систему безпеки.



ER-діаграма

Рисунок Г.10 – Слайд 10

ТЕСТУВАННЯ ПЗ

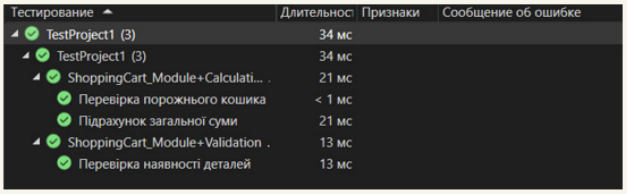
Проведені типи тестів:

- Тестування моделі Book;
- Тестування оновлення та валідації ціни книги;
- Тестування логіки роботи кошика покупця;
- Тестування розрахунку загальної суми замовлення;
- Тестування поведінки системи при порожньому кошику;
- Інтеграційне тестування взаємодії модулів (кошик і замовлення)

Отримані результати:

- Усі модульні тести виконані успішно;
- Помилок у бізнес-логіці не виявлено;
- Алгоритм розрахунку вартості працює коректно;
- Система правильно обробляє граничні випадки (порожній кошик);
- Взаємодія між модулями працює стабільно.

В процесі тестування було проведено модульне (unit) та інтеграційне тестування ключових компонентів системи з використанням фреймворку xUnit.



Результати тестування підтвердили правильність реалізації основних функцій системи та її готовність до використання.

Рисунок Г.11 – Слайд 11

ВИСНОВКИ

Завдання	*Результат виконання*
Проаналізувати предметну область електронної комерції	Проведено аналіз ринку електронної комерції та онлайн-продажу книг
Дослідити існуючі системи продажу книг	Проаналізовано функціонал сучасних книжкових вебзастосунків
Сформувати вимоги до функціоналу та архітектури	Визначено функціональні та нефункціональні вимоги системи
Обрати архітектурний підхід і програмні патерни	Обрано поєднання клієнт-серверної та шарової архітектури з використанням шаблону MVC
Спроєктувати базу даних та структуру модулів	Розроблено структуру БД та модульну організацію системи
Визначити технології реалізації	Обрано ASP.NET Core MVC, EF Core, SQL Server, Razor, Bootstrap
Реалізувати основні функції системи	Реалізовано каталог, кошик, замовлення та адмін-панель
Протестувати розроблений вебзастосунок	Проведено модульне та інтеграційне тестування
Проаналізувати результати та сформулювати висновки	Підтверджено коректність роботи та готовність системи

Усі поставлені завдання були успішно виконані. Розроблений вебзастосунок відповідає сучасним вимогам електронної комерції та забезпечує зручний функціонал для користувачів і адміністраторів.

Рисунок Г.12 – Слайд 12

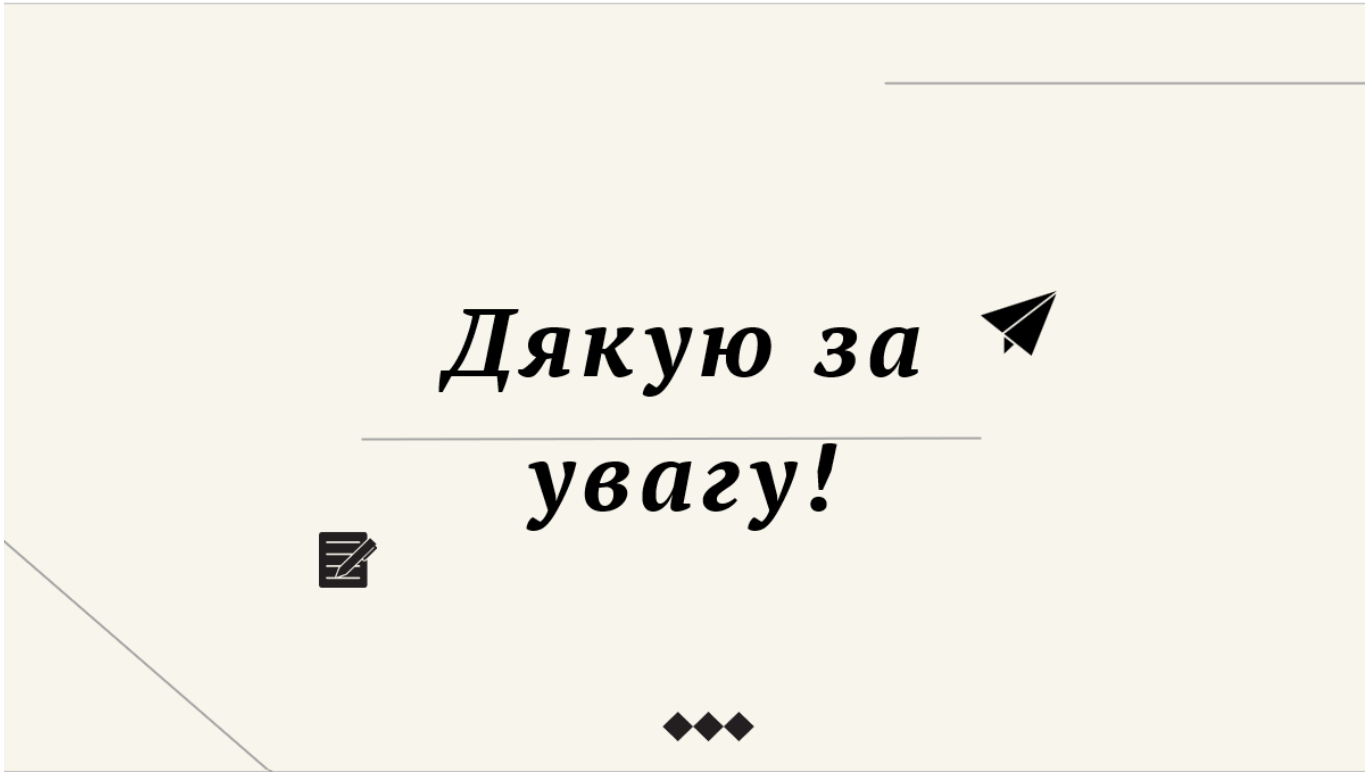
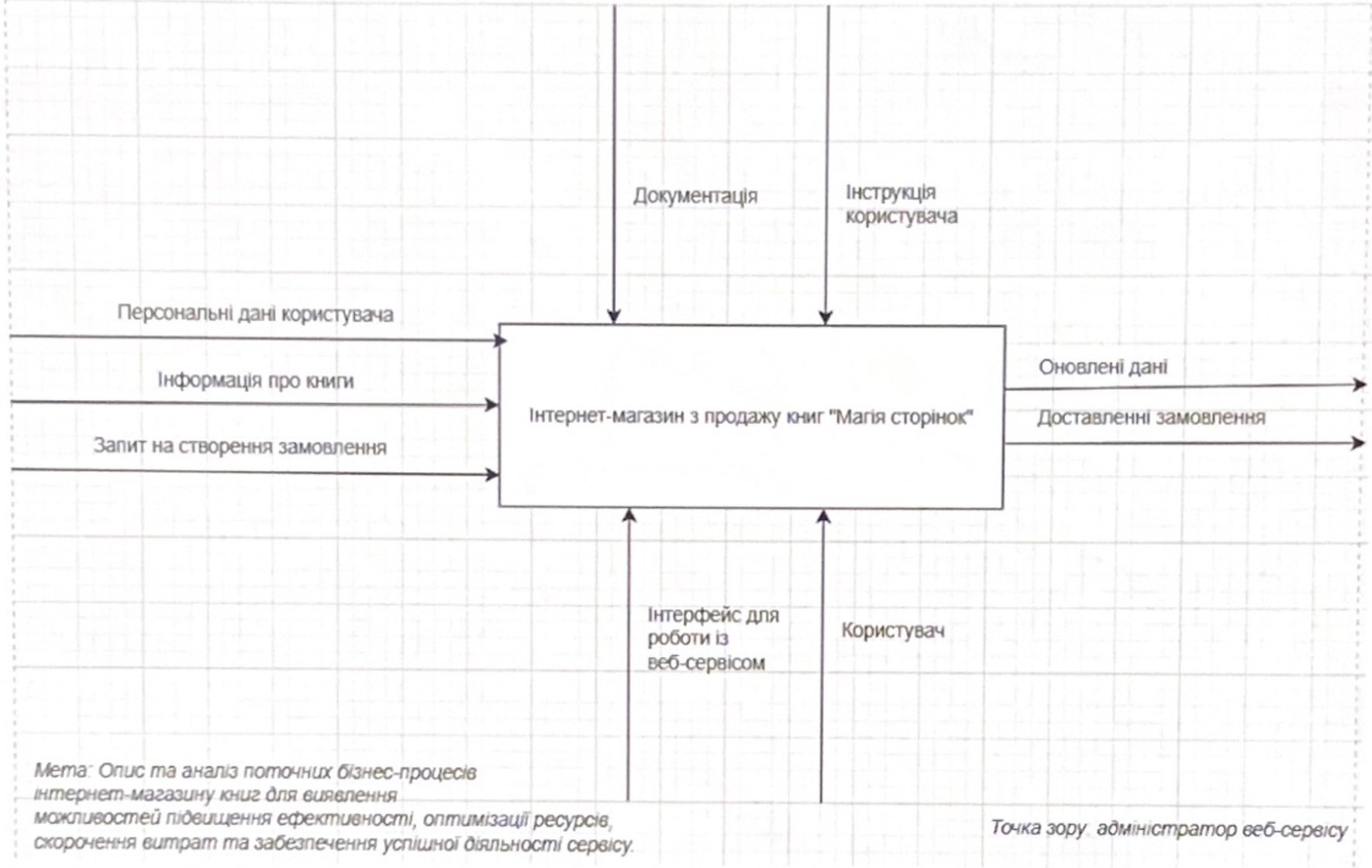


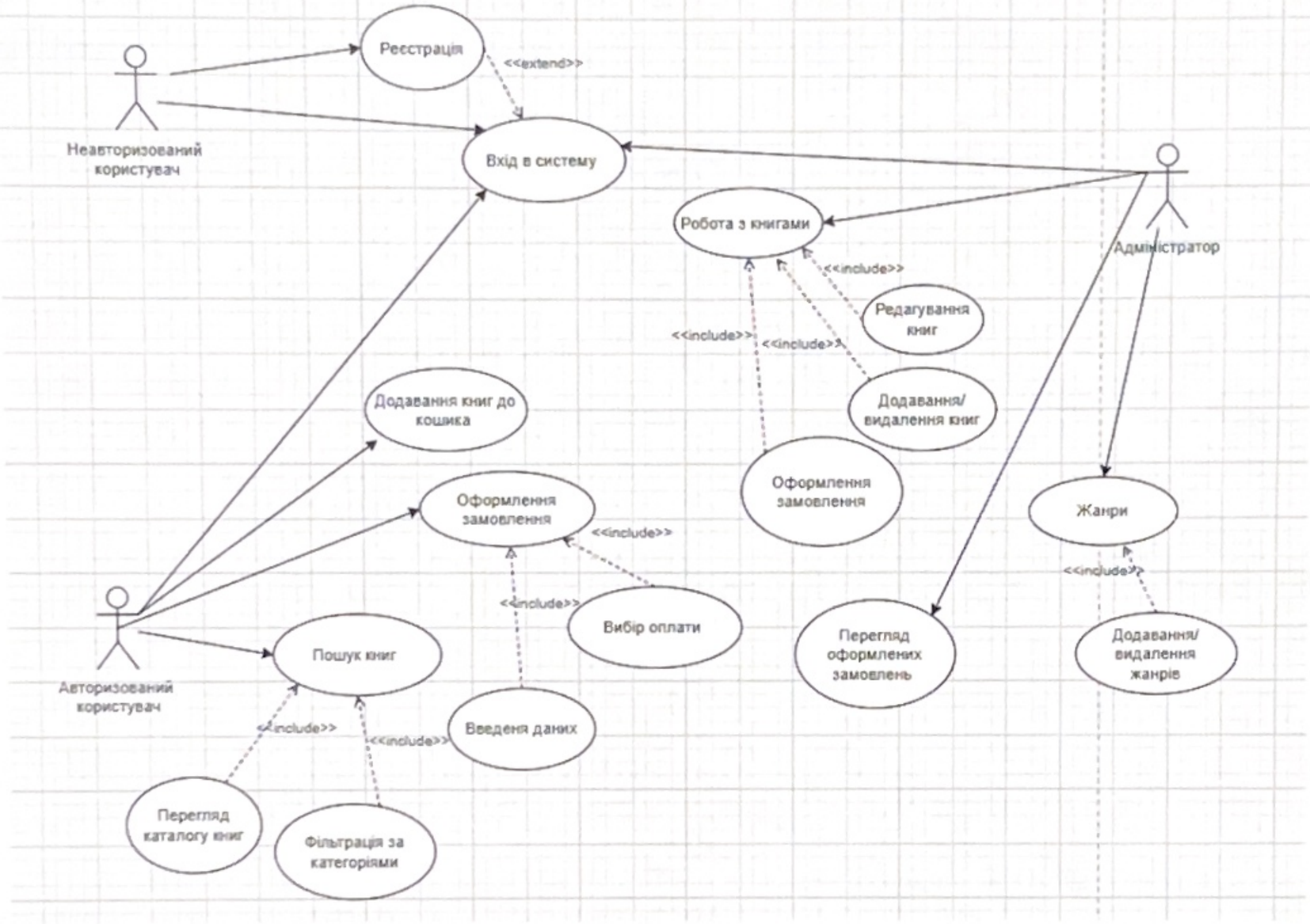
Рисунок Г.13 – Слайд 13

ГРАФІЧНІ МАТЕРІАЛИ

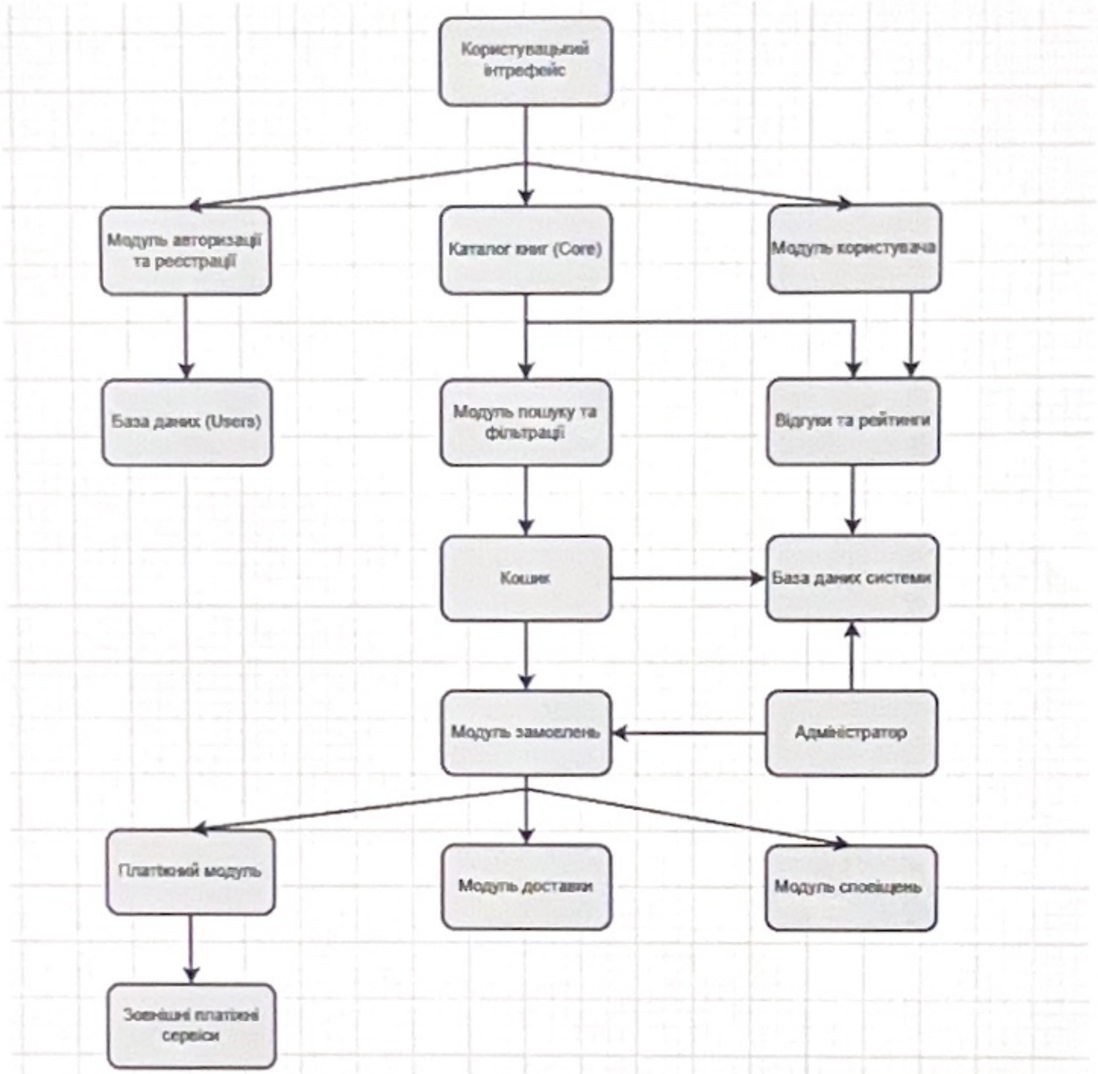


					КвРІПЗ.2201104.01.09.ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Мельник В.М.			Контекстна IDEF-0 діагарама	Літера	Маса / Масштаб
Керівник		Бедратюк Л. П.					
Консульт.						Аркуш 1	Аркушів 3
Н. Контр.		Праворська Н.І.				ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л. П.					

Керівник



					КвРІПЗ.2201104.01.09.ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	UML-Діаграма варіантів використання	Літера	Маса	Масштаб
Розробив		Мельник В.М.		11.05				
Керівник		Бедратюк Л. П.		11.05		Аркуш 2	Аркушів 3	
Консульт.								
Н. Контр.		Праворська Н.І.		11.05		ХНУ, ІПЗ-22-1		
Зав. каф.		Бедратюк Л. П.		11.05				



					КвРІПЗ.2201104.01.09.ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Мельник В.М.		28.05	Діаграма міжмодульних зв'язків	Літера	Маса	Масштаб
Керівник		Бедратюк Л. П.		28.05				
Консульт.						Аркуш 3	Аркушів 3	
Н. Контр.		Праворська Н.І.		28.05		ХНУ, ІПЗ-22-1		
Зав. каф.		Бедратюк Л. П.		28.05				

Керівник

СУПРОВІДНІ ДОКУМЕНТИ

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Мельник Вікторія Миколаївна
студент IV курсу спеціальності 121 – Інженерія програмного забезпечення,
група ІПЗ-22-1
здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився (-лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

02 вересня 2025 р. _____

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи Мельник В.М.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: «Вебзастосунок для купівлі та продажу книг »

(керівник роботи – Бедратюк Леонід Петрович)
Прізвище, ім'я, по батькові

01.01.2026
Дата

Підпис студента

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Мельник Вікторії Миколаївної
факультет ІТ, IVкурс, група ІПЗ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщена та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

21.05.2026
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Вебзастосунок для купівлі та продажу книг»
Автор: Мельник Вікторія Миколаївна
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Леонід Петрович д-р. фіз.-мат. наук, професор.
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	**відповідає**
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	**відповідає**
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	**відповідає**
5	Інше:	

Підтвердження:
Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:
1)У тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту назвах розділів/підрозділів, рамках форм у назвах та URL-адресах публікацій переліку джерел посилання;
2)Запозичення, виявлені у тексті є фрагментарними. Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1,29% з одного джерела. Загальна сумарна подібність у базі даних складає 4,78%. Крім того, за результатами додаткового аналізу не виявлено заміни букв, зайвих білих знаків або маніпуляцій з інтервалами. Наявність 38 мікропробілів не впливає на загальний характер роботи. З урахуванням наведених обґрунтувань, робота відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 25.05.26

Завідувач кафедри _____ Леонід БЕДРАТЮК
Гарант освітньої програми _____ Леонід БЕДРАТЮК
Керівник кваліфікаційної роботи _____ Леонід БЕДРАТЮК

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Вікторія МЕЛЬНИК

Співавтор:

Назва: Вебзастосунок для купівлі та продажу книг

Науковий керівник: д-р фіз-мат.наук, проф. Леонід БЕДРАТЮК

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:4.78%

Коефіцієнт подібності 2:1.29%

Мікропробіли: 38

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-21 10:10:56.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 15-05-2026

експерт

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 271870 Назва: БКР Вебзастосунок для купівлі та продажу книг Додано в БД: 2026-05-21 Автора: Вікторія МЕЛЬНІК Керівники: д-р фіз-мат.наук, проф. Леонід БЕДРАТЮК Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	101304	761	3034 (3%)	43 (6%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Мельник Вікторія Миколаївна

Тема Вебзастосунок для купівлі та продажу книг

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 75
1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати шаровій та клієнт-серверній архітектурам. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. Також у цьому розділі виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки на сьогодні в Україні вебзастосунки для купівлі та продажу книг потребують сучасних функціональних можливостей та зручного інтерфейсу. Також для розробки програмного продукту було використано новітні технології для побудови та актуальні архітектурні рішеня.

5. Негативні сторони роботи У роботі функціонал рекомендацій був обмежений та відсутність мобільного застосунку.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ
К.Т.Н., доцент кафедри КЖ ХНУ Жашуєвич Марія Вікторівна

"28" 05 2026 р. _____
(підпис)

SemanticAI for Education

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

першого освітнього рівня «Бакалавр»

Студента: Мельник Вікторії
Група: ІПЗ-22-1

Тема: *«Вебзастосунок для купівлі та продажу книг»*

Спеціальність: 121 – Інженерія програмного забезпечення

📘 Рецензія на кваліфікаційну роботу

✉️ Короткий зміст пояснювальної записки

У вступі роботи окреслено, що сучасний розвиток інформаційних технологій та цифрова трансформація зумовлюють активне перенесення бізнес-процесів у вебсередовище, зокрема в електронну комерцію. Актуальність розроблення вебзастосунків для продажу книг підкреслюється зростанням книжкового ринку в онлайн-форматі. Метою роботи є створення вебзастосунку для продажу книг, що забезпечує зручну взаємодію користувача з системою. Для досягнення цієї мети визначено завдання, серед яких аналіз предметної області, дослідження аналогів, формування вимог до системи, проєктування бази даних та реалізація функціональних модулів.

✉️ Відповідність отриманих результатів роботи поставленим завданням

Завдання, сформульовані у вступі, включають аналіз предметної області, дослідження аналогів, формування вимог, проєктування та реалізацію. Висновки роботи підтверджують, що результати в цілому відповідають поставленим завданням,

оскільки проведено детальний аналіз ринку, визначено функціональні та нефункціональні вимоги, реалізовано основні модулі системи. Оцінка відповідності: **в цілому відповідають**.

✉️ Оцінка розділів

Розділ 1: Змістовний аналіз предметної області, її структурних та функціональних особливостей

У цьому розділі представлено детальний опис предметної області, зокрема ринку електронної комерції в Україні, з акцентом на книжкову продукцію. Окреслено проблеми, які вирішує програмне забезпечення, а також визначено структуру та функції предметної області. Визначено категорії користувачів та їх інформаційні потреби, а також вхідні та вихідні дані.

Позитивні сторони: логічна структура викладу, наявність статистичних даних, чітке визначення користувачів та їх потреб. Негативні сторони: поверхневий опис прикладів вебзастосунків, недостатня конкретизація проблем, відсутність графічних діаграм для ілюстрації структури.

Розділ 2: Аналіз наявного програмно-технічного забезпечення предметної області

У цьому розділі наведено перелік аналогів, їх характеристики та проведено порівняльний аналіз. Визначено доцільність нового програмного забезпечення на основі проведеного аналізу.

Позитивні сторони: актуальні та релевантні аналоги, логічно структуроване порівняння. Негативні сторони: недостатня деталізація характеристик, відсутність повного обґрунтування доцільності нового рішення.

Розділ 3: Реалізація основних функціональних модулів

У розділі описано реалізацію логіки роботи з товарами, кошиком, замовленнями, а також систему ролей і автентифікацію користувачів. Однак, відсутні конкретні приклади коду та опис інтеграції з платіжними системами.

Позитивні сторони: чітке розмежування відповідальностей між компонентами, акцент на безпеку. Негативні сторони: відсутність прикладів коду, недостатня деталізація інтеграції з платіжними системами.

✉️ Позитивні сторони

Кваліфікаційна робота демонструє оригінальність у виборі теми, що є актуальною в умовах сучасного розвитку електронної комерції. Якість рішень, представлених у роботі, свідчить про глибоке розуміння предметної області. Зручність для користувача підкреслюється в описі інтерфейсу та функціональності системи.

✉️ Недоліки

Деякі розділи роботи потребують більшої деталізації, зокрема в частині опису прикладів вебзастосунків та механізмів інтеграції з платіжними системами. Відсутність графічних діаграм та прикладів коду ускладнює сприйняття інформації. Також, деякі висновки можуть бути сформульовані більш чітко.

📨 Відгук в цілому

Кваліфікаційна робота є актуальною та має практичну значущість, оскільки відповідає сучасним вимогам до вебплатформ електронної комерції. Зміст роботи відповідає темі та завданню, а новизна ідей та рішень підкреслює її цінність. Об'єктивність та обґрунтованість викладених матеріалів свідчить про високий рівень підготовки здобувача.

📨 Оцінка кваліфікаційної роботи

Кваліфікаційна робота заслуговує оцінки **добре**. Вона виконана в повному обсязі з дотриманням основних вимог, хоча є деякі недоліки, які потребують доопрацювання.

✍️ Рекомендації

Рекомендується доопрацювати роботу, зокрема додати більше прикладів коду, графічні діаграми для ілюстрації структури системи, а також уточнити висновки та обґрунтування доцільності нового програмного забезпечення.

OpenAI API-асистент
Session ID: 214d9761-822f-4651-a93d-2eec1f300423
Підписано автоматично, модель gpt-4o-mini
Дата: 27.05.2026