Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, цикловой комісії))

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна система обліку виробів з пластику»

08-54.БКР.010.00.000 ПЗ

Виконала: студентка 2 курсу, групи КІ-23мсз
спеціальності 123 — Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
_____ Турко Г. А.
(підпис)

Керівник: ст. викл. каф. ОТ
_____ Кисюк Д. В.
(підпис)
«_____» _____ 2025 р.

Рецензент: д. т. н., профю, зав. каф. ОТ
_____ Романюк О. Н.
(підпис)
«_____» _____ 2025 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«17» 06 2025 р.

Вінниця ВНТУ — 2025 рік

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Галузь знань — 12 «Інформаційні технології»

Спеціальність — 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Турко Ганні Андріївні

1 Тема роботи: «Інформаційна система обліку виробів з пластику», керівник роботи Кисюк Д.В. ст. викл. каф. ОТ затверджені наказом вищого навчального закладу від «20» березня 2025 року № 97.

2 Термін подання студентом роботи: 13 травня 2025 р.

3 Вихідні дані до роботи: Клієнська та серверна частина інформаційної системи обліку виробів з пластику. Інтерфейс користувача має бути інтуїтивно зрозумілий.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Обґрунтування доцільності розробки інформаційної системи обліку виробів з пластику, аналіз існуючих програмних рішень, проєктування програмних модулів інформаційної системи та їх програмна реалізація.

5 Перелік графічного матеріалу: ER-діаграма бази даних системи обліку, діаграма прецедентів інформаційної системи обліку виробів з пластику,

UML-діаграма послідовностей для авторизації, веб-сторінки інформаційної системи обліку виробів з пластику

6 Консультанти розділів роботи роботи приведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	ст. викл. каф. ОТ Кисюк Д.В.	21.03.25	13.05.25
Нормоконтроль	ас. каф. ОТ Швець С. І.	13.05.25	

7 Дата видачі завдання: 21 березня 2025 р.

8 Календарний план виконання БКР приведений в таблиці 2.

Таблиця 2 — Календарний план виконання БКР

№	Назва етапу	Термін виконання	Підпис
1	Вибір, узгодження та затвердження теми БКР	03.09.2024	
2	Аналіз існуючих програмних рішень	16.03.2025 — 18.03.2025	
3	Проєктування інформаціної системи обліку виробів з пластику	19.03.2025 — 27.03.2025	
4	Програмна реалізація інформаціної системи обліку виробів з пластику	28.03.2025 — 26.04.2025	
5	Тестування інформаціної системи обліку виробів з пластику	27.04.2025 — 05.05.2025	
6	Оформлення додатків	06.05.2025 — 09.05.2025	
7	Попередній захист БКР, доопрацювання, рецензування БКР	13.05.2025	

Студентка Ганна ТУРКО

Керівник роботи ст. викл. каф. ОТ Дмитро КИСЮК

АНОТАЦІЯ

УДК 004.72

Турко Г. А. Інформаційна система обліку виробів з пластику. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця: ВНТУ, 2025. 126 с.

На укр. мові. Бібліогр.: 34 назв, рис.: 41, табл. 2.

Основною метою дослідження є побудова інформаційної системи для обліку пластикової продукції.

У рамках роботи проведено аналіз підходів до розробки облікових систем, здійснено огляд актуального програмного забезпечення, що використовується в подібних задачах. На основі зібраної інформації розроблено архітектуру та реалізовано власне прикладне рішення.

Програмну реалізацію було виконано з використанням сучасного технологічного стеку — мови програмування C# у середовищі .NET, а також JavaScript з бібліотекою React для розробки інтерфейсу користувача. Для побудови UI застосовано компонентну бібліотеку Ant Design. У якості системи керування базами даних використано PostgreSQL, а для контейнеризації та розгортання — платформу Docker.

У результаті розроблено повноцінну інформаційну систему для обліку виробів з пластику, яку було протестовано на відповідність функціональним вимогам. Проведене тестування підтвердило працездатність і придатність системи до практичного використання.

Ключові слова: інформаційна система, облік, ASP.NET Core, Swagger, PostgreSQL, Docker, React, LiqPay, пластик.

ABSTRACT

UDC 004.72

Turko H. A. Information System for Accounting Plastic Products. Bachelor’s qualification thesis in the specialty 123 "Computer Engineering", educational program "Computer Engineering". Vinnytsia: VNTU, 2025. 126 pages.

In Ukrainian. Bibliography: 34 titles, pictures: 41, tables: 2.

The main objective of the research is to develop an information system for tracking plastic products. The thesis includes an analysis of approaches to building accounting systems and a review of current software solutions used in similar tasks. Based on the collected data, the system architecture was designed and a custom application was implemented.

The software was developed using a modern technology stack — the C# programming language within the .NET environment, and JavaScript with the React library for the user interface. The Ant Design component library was used to build the UI. PostgreSQL was employed as the database management system, and Docker was used for containerization and deployment.

As a result, a complete information system for accounting plastic products was developed and tested for compliance with functional requirements. The conducted testing confirmed the system's operability and its suitability for practical use.

Keywords: information system, accounting, ASP.NET Core, Swagger, PostgreSQL, Docker, React, LiqPay, plastic.

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації наявність веб-сайту стала критично важливою для будь-якого бізнесу, незалежно від його розміру чи сфери діяльності. Веб-ресурс виконує роль онлайн-представництва компанії, сприяє формуванню довіри серед клієнтів, забезпечує постійний доступ до інформації про товари й послуги, а також відкриває можливості для взаємодії з цільовою аудиторією. Через веб-платформу підприємство може не лише демонструвати свій асортимент, але й отримувати замовлення, налагоджувати комунікацію з клієнтами та просувати продукцію у пошукових системах. Відсутність такого каналу комунікації значно звужує охоплення ринку та зменшує шанси на залучення нових споживачів.

Формочки для печива та відповідні трафарети мають широкий спектр застосування як у побуті, так і в комерційній діяльності. Вони дозволяють виготовляти печиво з оригінальним дизайном, що особливо актуально до святкових подій, таких як Новий рік, Великдень, День святого Валентина, Хелловін тощо. Завдяки великій різноманітності форм і можливості створення індивідуальних трафаретів, такі вироби стають популярними як подарунки або прикраси для святкового столу. Це відкриває перспективи для започаткування власного бізнесу з виготовлення та продажу тематичного печива. Підприємці можуть пропонувати клієнтам продукцію під замовлення, а також брати участь в ярмарках, співпрацювати з кав’ярнями чи пекарнями, що значно розширює можливості монетизації цього напрямку, а отже і популярність галузі і відповідних матеріалів в ній.

Актуальність розробки інформаційної системи обліку виробів з пластику зумовлена потребою в оптимізації бізнес-процесів, підвищенні ефективності управління ресурсами та створенні умов для подальшого масштабування підприємницької діяльності.

У даній роботі буде розглянуто програмну реалізацію інформаційної системи обліку виробів з пластику, що передбачає впровадження функціоналу авторизації користувачів а також забезпечення можливості формування та

обробки замовлень. Зазначена система спрямована на підвищення ефективності управління товарообігом і покращення взаємодії з клієнтами.

Метою дослідження є створення інформаційної системи для обліку пластикових виробів, що дозволить автоматизувати ключові бізнес-процеси підприємства.

Для досягнення мети необхідно вирішити такі завдання:

— вивчити предметну галузь і сформулювати вимоги до системи;

— проаналізувати існуючі програмні рішення, що вирішують подібні завдання;

— розробити архітектуру майбутньої системи та описати її логічну структуру;

— реалізувати функціональні модулі для обліку товарів, авторизації користувачів і роботи з замовленнями;

— провести повноцінне тестування і оцінити результати роботи розробленого ПЗ.

Об'єкт дослідження — процес розробки інформаційної системи для автоматизації діяльності підприємства.

Предмет дослідження — методи та інструменти створення веб-орієнтованих програмних рішень для обліку продукції.

Практичне значення полягає в створенні прикладного інструменту, який може бути ефективно використаний малими й середніми компаніями для оптимізації обліку продукції з пластику та поліпшення управлінських процесів.

Результати проведених досліджень були **апробовані** на наукових конференціях, зокрема:

— конференція «Молодь в науці: дослідження, проблеми, перспективи» (МН-2025) [1].

Матеріали дослідження опубліковані у збірнику тез конференції «Молодь в науці: дослідження, проблеми, перспективи» (МН-2024) [1].

Також результати бакалаврської роботи планується впровадити в діяльність підприємства «В.Д.Т», що підтверджується довідкою, наведеною в додатку Г.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ВИРОБІВ З ПЛАСТИКУ

1.1 Обгрунтування доцільності розробки

У сучасних умовах цифровізації бізнес-процесів усе більше підприємств прагнуть автоматизувати облік продукції, оптимізувати управління замовленнями та забезпечити ефективну взаємодію з клієнтами. Підприємства, що займаються виробництвом та реалізацією виробів із пластику, не є винятком. Враховуючи зростаючий попит на пластикову продукцію, виникає потреба у впровадженні зручного інструменту для систематизації даних про товари, контролю складських залишків, швидкого оформлення замовлень та збереження історії взаємодії з клієнтами.

Розробка спеціалізованої інформаційної системи обліку виробів з пластику дозволить підвищити точність обліку, зменшити ризики помилок, пов’язаних з ручним введенням даних, а також забезпечити прозорість бізнес-процесів. Крім того, наявність авторизації користувачів і можливість фільтрації замовлень відкриває додаткові можливості для аналітики та прийняття управлінських рішень. Таким чином, створення подібної системи є доцільним і обґрунтованим кроком у напрямку цифрової трансформації підприємства та підвищення його конкурентоспроможності.

В умовах воєнного стану малий та середній бізнес (МСБ) відіграє критично важливу роль у підтримці економічної стабільності країни. На фоні зниження обсягів великого промислового виробництва саме МСБ демонструє гнучкість, адаптивність до змін та здатність оперативно реагувати на виклики. Такі підприємства забезпечують робочими місцями значну частину населення, зменшують рівень безробіття, стимулюють внутрішнє споживання та сприяють збереженню економічної активності в регіонах [2].

Крім того, малі та середні підприємства активно залучаються до волонтерської, гуманітарної та відновлювальної діяльності, підтримуючи як армію, так і цивільне населення. Їхній розвиток є важливою умовою для післявоєнного відновлення економіки, нарощування внутрішнього виробництва

та зменшення залежності від імпорту. Тому створення зручних цифрових інструментів для автоматизації обліку, управління ресурсами та просування продукції — є актуальним завданням, що сприяє підвищенню ефективності та життєздатності таких бізнесів у кризових умовах.

Отже, використання сучасних цифрових технологій має значний потенціал для підвищення ефективності ведення малого та середнього бізнесу, зокрема у сфері виробництва виробів з пластику. Впровадження інформаційних систем обліку дозволяє автоматизувати ключові процеси, оптимізувати використання ресурсів, зменшити операційні витрати та підвищити якість обслуговування клієнтів. Більше того, такі рішення сприяють зручності в управлінні підприємницькою діяльністю, забезпечуючи доступ до даних у будь-який час і з будь-якого пристрою.

1.2 Огляд існуючих рішень

Сайти компаній, що займаються реалізацією формочок для випічки, виконують передусім функцію зручного інструменту для ознайомлення споживачів з товарним асортиментом. Вони дозволяють легко оформити замовлення, отримати консультацію та дізнатися детальну технічну інформацію про продукцію. Такі веб-ресурси суттєво сприяють розвитку домашнього та малого підприємництва, популяризації кондитерського мистецтва й хобі-діяльності.

На ринку вже представлено чимало онлайн-платформ, які пропонують формочки для печива з пластику. До найбільш відомих серед них, як в Україні, так і за кордоном, належать ресурси на кшталт Formochki.com.ua, CakeShop і Sweet4uCutters.

CakeShop — це український інтернет-магазин, орієнтований на професійних кондитерів та аматорів. Він пропонує широкий асортимент товарів, серед яких — формочки для печива, шоколаду, мастики, желе, леденців та інших видів випічки [3]. Вся продукція виготовляється з безпечного для харчових продуктів пластику, а також силікону й металу. Магазин має зручну навігацію,

добре структурований каталог і фільтри, що дозволяє швидко знайти потрібний товар.

Серед переваг CakeShop варто відзначити різноманітність формочок для випічки, доступність додаткових товарів — таких як харчові барвники, декор та пакування — а також можливість доставки по всій Україні та за її межі. Це робить магазин універсальним майданчиком для забезпечення потреб як приватних майстрів, так і малого бізнесу.

Попри численні переваги, ресурс має і певні недоліки. Зокрема, відсутня можливість створення індивідуальних формочок за дизайном клієнта, що обмежує персоналізацію продукції. Також деякі позиції іноді тимчасово недоступні в наявності, а ціни на продукцію не завжди є найнижчими в порівнянні з конкурентами. Головна сторінка сайту наведена на рисунку 1.1.

Рисунок 1.1 — Головна сторінка сайту CakeShop

Sweet4uCutters — це зарубіжний інтернет-магазин зі США, що спеціалізується на виготовленні формочок для печива за допомогою технології 3D-друку [4]. Вироби створюються з екологічно безпечного PLA-пластику, що

підходить для харчових продуктів. Асортимент магазину включає як стандартні формочки, так і індивідуальні дизайни, які виготовляються на замовлення. Платформа орієнтована на хобі-кондитерів і власників невеликих бізнесів, які займаються створенням тематичного печива до свят, подій чи під конкретний запит.

Серед основних переваг Sweet4uCutters варто виділити можливість кастомізації — покупець може замовити унікальний дизайн, що вирізняє магазин серед конкурентів. Крім того, платформа пропонує широкий вибір нестандартних форм на різні свята й події, а висока якість друку гарантує точність та довговічність формочок.

Разом із тим, ресурс має і певні недоліки. Зокрема, для українських користувачів однією з головних проблем є висока вартість доставки, а також тривалий час очікування через міжнародну логістику. Ще одним обмеженням є англомовний інтерфейс сайту, що може ускладнювати навігацію для користувачів без знання мови. Головна сторінка сайту наведена на рисунку 1.2.

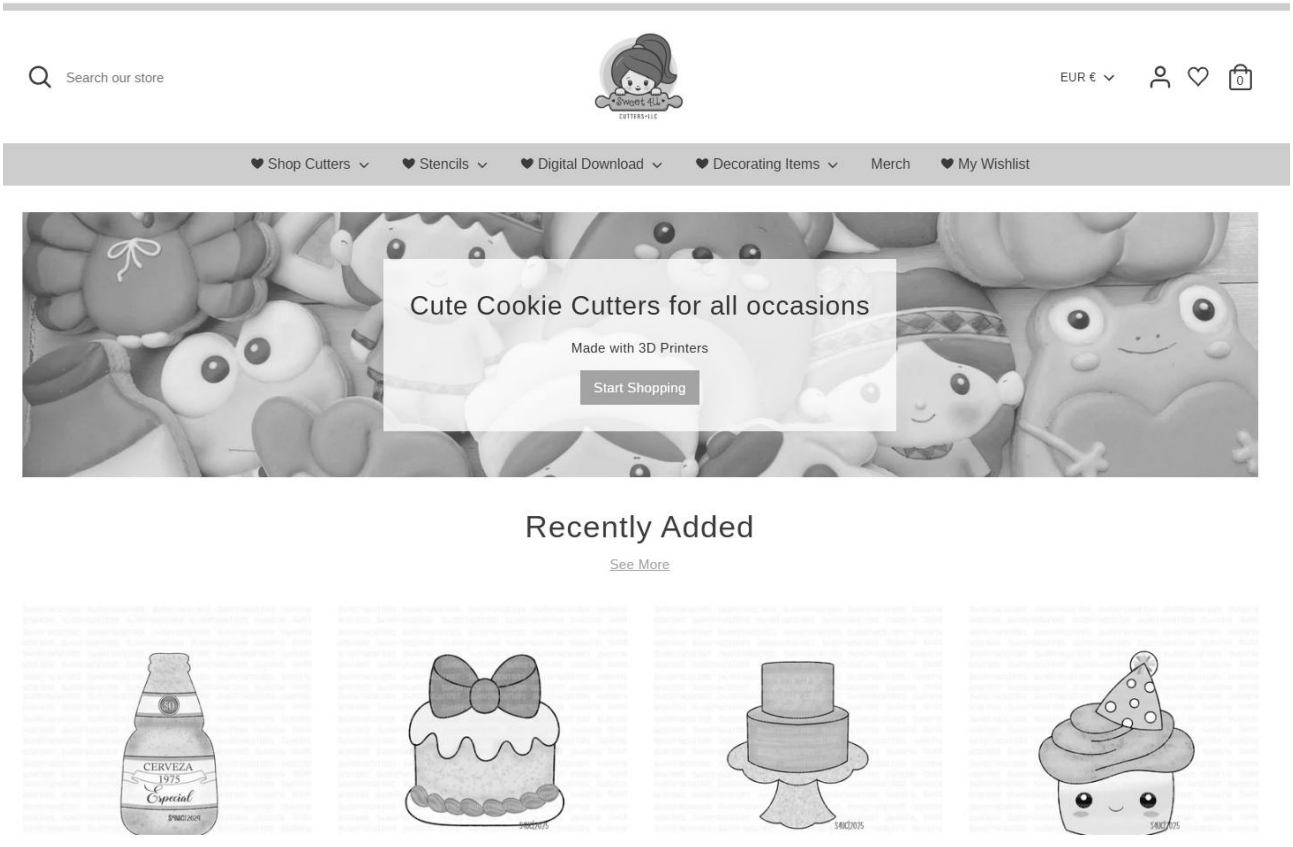

Рисунок 1.2 — Головна сторінка сайту Sweet4uCutters

Formochki.com.ua — це український онлайн-магазин, орієнтований як на домогосподарок, так і на професійних кондитерів [5]. Платформа спеціалізується на продажу формочок для печива з різноманітних матеріалів, зокрема пластику, силікону та нержавіючої сталі. Магазин дозволяє придбати продукцію як у роздріб, так і оптом, що робить його привабливим для різних категорій клієнтів — від поодиноких замовлень до дрібного бізнесу.

Серед переваг ресурсу — великий вибір матеріалів, з яких виготовлені формочки, а також значне різноманіття тематичних дизайнів: дитячі, святкові, абетки, цифри тощо. Важливою особливістю є швидка доставка по Україні, що дозволяє оперативно отримати замовлення. Інтерфейс сайту повністю україномовний, а для зручності клієнтів передбачена підтримка телефоном.

Водночас, платформа має і деякі обмеження. Зокрема, кастомізація продукції — тобто створення індивідуального дизайну формочок — практично не реалізована. Крім того, сайт має досить простий візуальний вигляд, без сучасних елементів інтерфейсу, що може впливати на користувацький досвід. Ще однією проблемою є те, що каталог не завжди оновлюється вчасно — іноді товари залишаються на сайті навіть тоді, коли їх немає в наявності. Головна сторінка сайту наведена на рисунку 1.3.

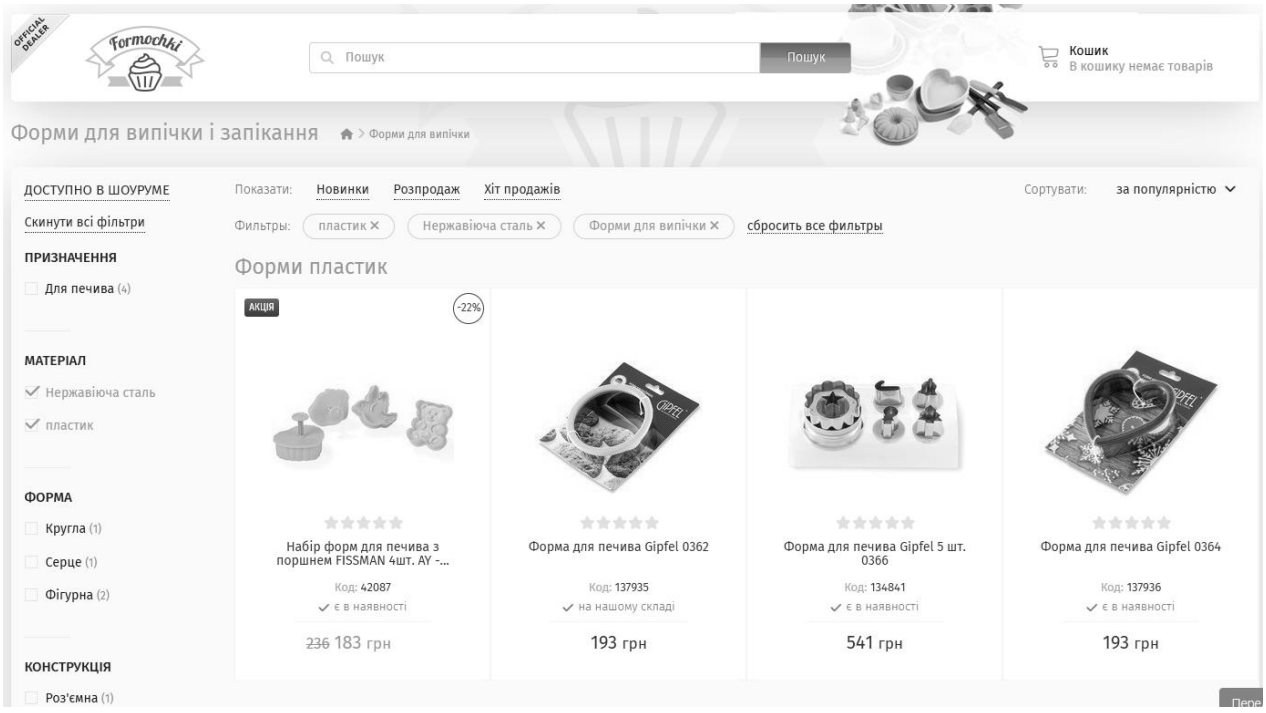

Рисунок 1.3 — Головна сторінка сайту Formochki.com.ua

Узагальнюючи результати аналізу трьох веб-ресурсів — CakeShop, Sweet4uCutters та Formochki.com.ua, можна виділити низку спільних недоліків, які не дозволяють жодному з них повністю замінити функціональність та ідею авторської інформаційної системи обліку формочок для печива.

По-перше, лише один із проєктів (Sweet4uCutters) надає можливість кастомізації формочок, проте цей функціонал обмежений, недоступний в Україні без значних витрат на доставку, і не адаптований до локального ринку. Українські сайти взагалі не передбачають створення індивідуальних форм або інтерактивної взаємодії з клієнтом під час оформлення унікального замовлення.

По- друге, у жодному з сайтів не реалізовано персоналізованого кабінету клієнта з відображенням попередніх замовлень.

Результат порівняння сайтів з продажу виробів з пластику наведений у таблиці 1.1

Таблиця 1.1 — Порівняльна Таблиця сайтів з продажу виробів з пластику

Характеристика	CakeShop	Sweet4uCutters	Formochki.com.ua
Країна	Україна	США	Україна
Мова інтерфейсу	Українська, російська	Англійська	Українська
Доставка по Україні	Так	Ні (лише міжнародна)	Так
Матеріал формочок	Пластик, силікон, метал	PLA-пластик	Пластик, нержавіюча сталь
Тематичні набори	Так	Так	Так
Можливість кастомного дизайну	Ні	Так	Обмежено
Ціновий діапазон	Від 15 до 550 грн	Від $3.50	Від 74 до 610 грн
Додаткові товари	Так (барвники, декор, інструменти)	Так (трафарети)	Так (коврики, упаковка)

Продовження таблиці 1.1.

Підходить для бізнесу	Так	Так	Так

Таким чином, жоден із наявних веб-ресурсів не поєднує функцій онлайн-магазину та системи кастомізації. Саме це робить розробку власної інформаційної системи обліку формочок актуальною, конкурентоспроможною та такою, що заповнює помітну нішу на ринку.

1.3 Постановка задачі та формулювання вимог

У межах розробки автоматизованої системи обліку виробів з пластику важливо забезпечити стабільну та гнучку технічну архітектуру, яка відповідатиме сучасним вимогам до зручності використання, розширюваності та ефективності обробки даних. В основі реалізації лежить комбінація інструментів: клієнтська частина побудована з використанням React і бібліотеки компонентів Ant Design, серверна логіка реалізована на ASP.NET Core Web API, як система управління базами даних використовується PostgreSQL, а для розгортання та підтримки середовища — платформа Docker.

React виступає як основа для побудови інтерфейсу користувача завдяки своїй модульній структурі та широким можливостям адаптації [6]. Він був обраний саме тому, що він є одним із найпопулярніших і найактивніше підтримуваних інструментів для розробки інтерфейсів користувача. React забезпечує компонентний підхід до побудови UI, що значно спрощує повторне використання коду, покращує структуру проєкту та полегшує його масштабування. Завдяки віртуальному DOM React дозволяє створювати швидкі та динамічні веб-застосунки з плавною взаємодією без перезавантаження сторінок. Крім того, React має велику екосистему, багату документацію та потужну підтримку спільноти, що робить його зручним для впровадження як у невеликих, так і у великих проєктах.

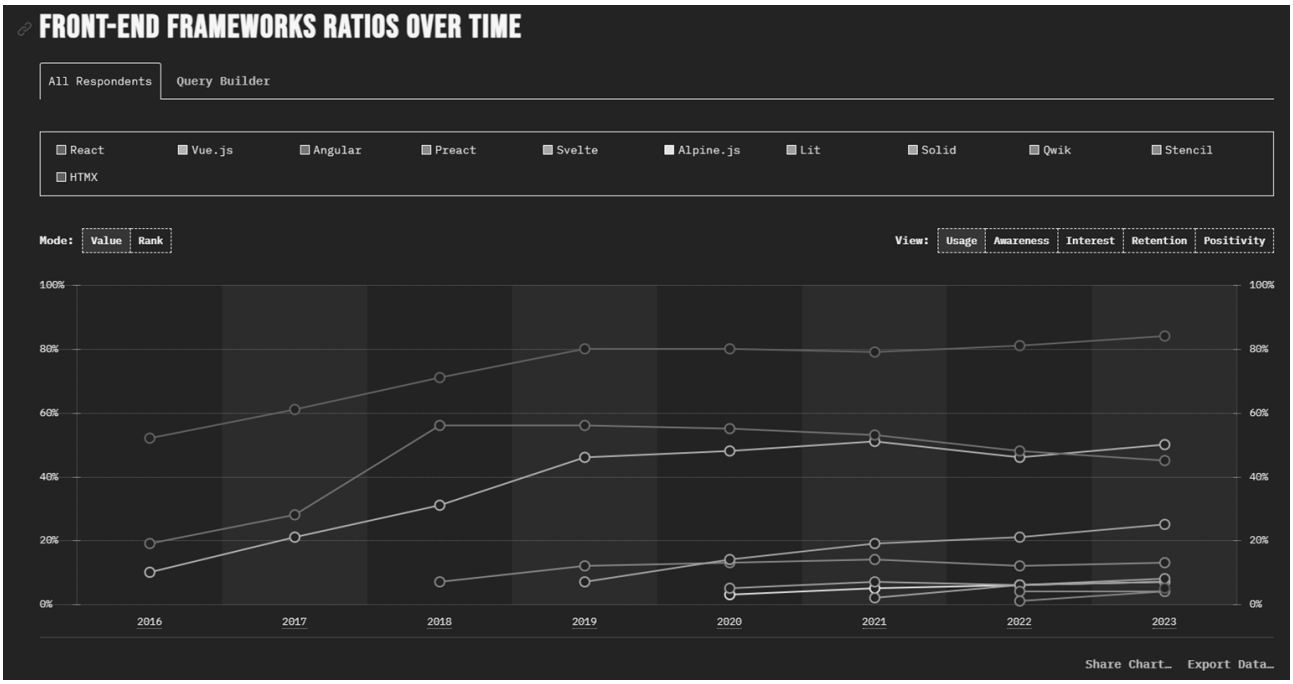

Рисунок 1.4 — Графік популярності фронтенд фреймворків

Також, інтеграція з бібліотекою Ant Design надає змогу швидко впроваджувати інтуїтивно зрозумілі та адаптивні елементи UI, що покращує користувацький досвід [7].

Для серверної частини обрана технологія ASP.NET Core Web API — сучасне рішення від Microsoft для створення продуктивних веб-сервісів [8]. Цей фреймворк підтримує асинхронну обробку запитів, що забезпечує високу швидкодію застосунку, а також має вбудовані механізми для валідації, авторизації, генерації документації (Swagger) та взаємодії з базами даних. У порівнянні з іншими фреймворками, такими як Express або Flask, ASP.NET Core є більш структурованим, типобезпечним і масштабованим рішенням.

База даних PostgreSQL використовується як надійне реляційне сховище, що підтримує складні запити, транзакції та розширення [9]. Завдяки високій продуктивності та стабільності ця СУБД є оптимальним вибором для зберігання структурованих облікових даних, особливо у виробничих системах. У порівнянні з MongoDB, яка краще підходить для нереляційних моделей, PostgreSQL забезпечує кращу узгодженість і зв’язність даних.

Порівняльна таблиця баз даних наведена у таблиці 1.2.

Таблиця 1.2 - Порівняльна таблиця PostgreSQL з іншими базами даних:

Характеристика	Postgresql	Mysql	Mongodb
Тип СУБД	Реляційна + об’єктно-реляційна	Реляційна	Документно-орієнтована (nosql)
Складні запити	Підтримує (CTE, віконні функції)	Обмежена підтримка	Обмежена підтримка
JSON підтримка	Рівень SQL + індексація	Обмежена	Основна модель
Масштабованість	Висока	Висока	Висока
Придатність для аналітики	Висока (аналітичні функції)	Середня	Обмежено
Підтримка розширень	(postgis, timescaledb, pg_partman тощо)	Обмежено	Немає
Швидкість на простих запитах	Трохи повільніший за mysql	Висока	Висока на простих читаннях
Підтримка зв’язків між таблицями	Повна (foreign keys, join'и)	Часткова	Немає
Найкраще підходить для	Складних систем, бізнесу, аналітики	Простих веб-додатків, CMS	Гнучких, масштабованих nosql рішень

Для зберігання файлів, зображень та інших медіаданих система використовує хмарне сховище Amazon S3 [10]. Такий підхід забезпечує надійне та масштабоване зберігання ресурсів із можливістю гнучкого керування доступом. Інтеграція з Amazon S3 дозволяє ефективно працювати з великим обсягом даних, зменшуючи навантаження на сервер і підвищуючи загальну продуктивність застосунку. Порівняно з локальним зберіганням, використання AWS S3 забезпечує вищий рівень доступності, безпеки та стійкості до збоїв.

Контейнеризація застосунку здійснюється за допомогою Docker, що дозволяє швидко розгортати систему на будь-якому сервері, підтримуючи однакове середовище для розробки, тестування та експлуатації [11]. Такий підхід знижує ймовірність помилок, пов’язаних із залежностями або несумісністю середовищ. Для невеликих і середніх проєктів Docker є зручним і зрозумілим інструментом.

Для організації платіжної системи обрано сервіс LiqPay — надійне та популярне рішення для прийому онлайн-платежів, розроблене компанією ПриватБанк. LiqPay підтримує широкий спектр методів оплати, включаючи банківські картки, мобільні гаманці та інші популярні способи, що забезпечує зручність для користувачів. Інтеграція LiqPay здійснюється через REST API, що дозволяє безпечно та ефективно обробляти платежі без необхідності зберігання платіжних даних на стороні сервера. Завдяки використанню підписів і токенів забезпечується захист від підробки даних і шахрайства. Використання LiqPay у проєкті сприяє швидкій та надійній реалізації платіжної функціональності з мінімальними витратами на розробку та підтримку.

Система реалізує ключові функції: створення та керування обліковими записами користувачів, робота з товарними одиницями, оформлення та облік замовлень, збереження клієнтської інформації. Така архітектура забезпечує масштабованість і гнучкість, дозволяючи розширювати функціонал системи відповідно до потреб бізнесу та ринку.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ВИРОБІВ З ПЛАСТИКУ

2.1 Аналіз та вибір архітектурних рішень

Для створення надійної, масштабованої та ефективної інформаційної системи обліку виробів з пластику було обрано клієнт-серверну архітектуру, яка є однією з найпоширеніших моделей побудови сучасних вебзастосунків. Її суть полягає у чіткому розподілі ролей між клієнтом (інтерфейсом користувача) та сервером (обробником логіки й даних). Клієнт надсилає запити до сервера, а сервер обробляє їх, звертаючись до бази даних і повертаючи відповідь. Такий підхід забезпечує розмежування обов'язків, що значно полегшує супровід, розширення та захист системи.

Основні переваги клієнт-серверної моделі:

— масштабованість — система легко адаптується до зростання навантаження, дозволяючи додавати нові клієнтські пристрої або сервери;

— безпека — дані надійно зберігаються на сервері, до якого клієнт має доступ лише через визначений api;

— гнучкість у розвитку — клієнтська і серверна частини можуть оновлюватися незалежно;

— централізоване зберігання та контроль — що спрощує адміністрування, резервне копіювання і контроль доступу;

— доступність — користувачі можуть працювати з системою з будь-якої точки світу через інтернет.

Клієнтська частина розроблена з використанням React.js — сучасної JavaScript-бібліотеки для побудови інтерфейсів користувача. Інтерфейс побудований за допомогою компонентної структури, що забезпечує повторне використання коду, гнучкість у розробці та високу продуктивність. Для побудови візуальних елементів використано Ant Design, що дозволяє створити сучасний і зручний інтерфейс. Клієнт надсилає HTTP-запити до сервера, обробляє отримані відповіді та відображає інформацію користувачеві.

Передбачено також адміністративну панель для управління обліковими даними, товарами та замовленнями.

Серверна частина реалізована на базі ASP.NET Core Web API, що забезпечує швидку, безпечну та надійну обробку HTTP-запитів. В основі серверної логіки використовується архітектурний шаблон MVC (Model–View–Controller), який структурно поділяє застосунок на три компоненти [12]:

Модель представляє дані та логіку програми. Вона інкапсулює операції над даними, включаючи їх зберігання, обробку та пошук. Ізолюючи дії, пов’язані з даними, Модель гарантує узгодженість і точність інформації в межах застосунку.

Представлення відповідає за відображення даних користувачеві. Воно подає інформацію у зручному та зрозумілому форматі, залишаючись незалежним від бізнес-логіки та структури зберігання даних. Таке розділення дозволяє створювати гнучкі та динамічні інтерфейси користувача.

Контролер виступає посередником між Моделлю та Представленням. Він приймає вхідні дані від користувача, передані через Представлення, обробляє їх, викликаючи відповідні методи Моделі, та формує відповідну відповідь. Контролер управляє потоком даних і визначає реакцію програми на дії користувача. Схема взаємодії компонентів архітектури MVC наведена на рисунку 2.1.

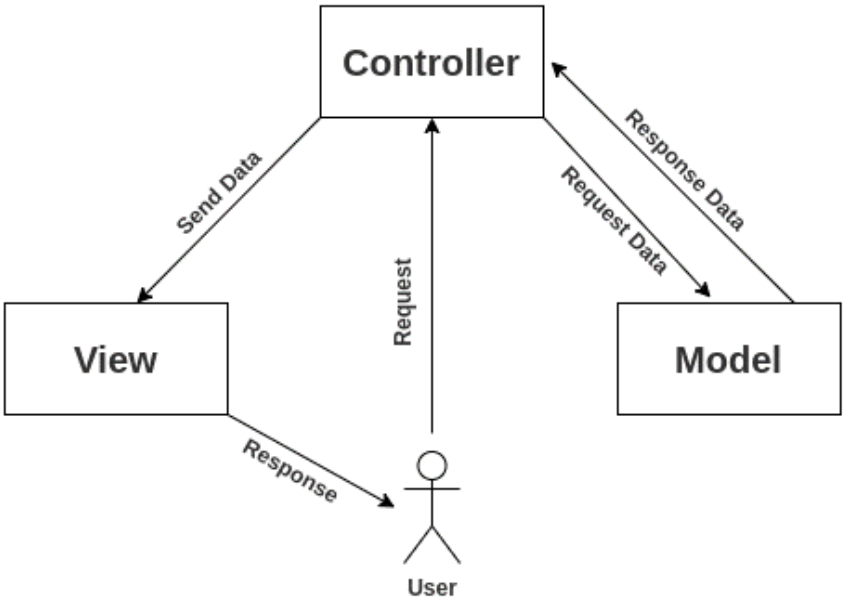

Рисунок 2.1 — Схема взаємодії компонентів архітектури MVC

Цей підхід дозволяє ізолювати бізнес-логіку від представлення, що значно полегшує підтримку, масштабування та модульне тестування системи.

Для зберігання інформації використовується PostgreSQL — надійна об’єктно-реляційна СУБД, що забезпечує високий рівень цілісності та узгодженості даних. У базі даних зберігається інформація про товари, користувачів, замовлення, сесії, права доступу тощо.

Уся система контейнеризована за допомогою Docker, що дає змогу запускати клієнтську і серверну частини, а також базу даних в окремих ізольованих середовищах. Такий підхід забезпечує простоту розгортання, однаковість середовищ на різних етапах розробки (dev/stage/prod), швидке масштабування і централізовану підтримку.

Типова взаємодія в системі має такий вигляд:

1) Користувач у веб-інтерфейсі формує запит;
2) Клієнтська частина відправляє HTTP-запит до API;
3) Контролер на сервері приймає запит, обробляє логіку;
4) Сервер звертається до бази даних через модель;
5) Дані повертаються до контролера, який формує відповідь;
6) Клієнт отримує дані та відображає їх користувачеві.

Отже, використання клієнт-серверної архітектури, патерну MVC та сучасного стеку технологій дозволяє створити надійну, підтримувану та ефективну інформаційну систему, яка відповідає потребам малого бізнесу у сфері обліку та керування виробами з пластику.

2.2 Структура бази даних інформаційної системи обліку виробів з пластику

На поточному етапі розробки інформаційної системи обліку виробів з пластику було створено базу даних, яка включає 12 таблиць, кожна з яких виконує конкретну роль у забезпеченні функціональності системи. ER-діаграма бази даних зображена у додатку В на рисунку В.2.

— Users — це ключова таблиця, що містить інформацію про зареєстрованих користувачів. У ній зберігаються дані про логін, email, хешований пароль, дату реєстрації, статус активності тощо. Вона має первинний

ключ Id, який використовується як зовнішній у багатьох інших таблицях, зокрема: Orders, Carts, RefreshTokens, UserRoleEntity і CustomProducts. Таким чином, вона є основною точкою ідентифікації користувача в усій системі.

— Roles — таблиця, яка описує усі ролі, що доступні в системі, наприклад: "користувач", "адміністратор", "модератор" тощо. Ролі дозволяють реалізувати механізм контролю доступу. Первинний ключ Id цієї таблиці використовується в UserRoleEntity (зв’язок з користувачами) і в RolePermissionEntity (зв’язок із дозволами).

— PermissionEntity — містить усі можливі права доступу, які існують у системі. Кожен дозвіл має унікальний ідентифікатор Id, який використовується як зовнішній ключ у таблиці RolePermissionEntity.

— UserRoleEntity — це проміжна таблиця, що реалізує зв’язок типу many-to-many між таблицями Users і Roles. Вона містить поля UserId і RoleId, що дозволяє одному користувачу мати декілька ролей, і, навпаки, кожну роль призначати багатьом користувачам.

— RolePermissionEntity — ще одна проміжна таблиця, яка реалізує зв’язок many-to-many, цього разу між Roles і PermissionEntity. Таким чином, кожна роль може мати набір дозволів, а кожен дозвіл може бути призначений кільком ролям.

— Products — таблиця, яка зберігає інформацію про усі товари в каталозі. Кожен товар має тип, тему, матеріал, опис, розмір, ціну, зображення тощо. Первинний ключ Id використовується як зовнішній ключ у таблицях CartItems і OrderItemsEntity, що дозволяє прив’язувати товар до конкретного кошика або замовлення.

— CustomProducts — Таблиця для зберігання індивідуально створених товарів користувачами. Вона містить інформацію про виріб, завантажене зображення, опис, тип, розмір, матеріал тощо. Кожен запис має зв’язок із користувачем, який створив продукт (UserId → Users.Id). Додатково може міститися зв’язок із реальним товаром (CreatedProductId → Products.Id), якщо кастомний виріб було схвалено та перетворено у стандартний товар.

— Carts — таблиця, яка відповідає за зберігання кошиків користувачів. Кожен користувач має власний кошик, який створюється під час реєстрації або при першому додаванні товару. Ключове поле UserId зв’язує кошик із конкретним користувачем.

— CartItems — ця Таблиця реалізує вміст кожного кошика. Вона містить зв’язки на CartId (кошик) та ProductId (товар), а також кількість одиниць кожного товару. Таким чином, вона дозволяє точно визначити, що саме користувач додав до кошика і в якій кількості.

— Orders — таблиця, що зберігає інформацію про всі оформлені замовлення. Кожне замовлення прив’язане до користувача (UserId → Users.Id), має власний статус, дату оформлення, суму, адресу доставки, спосіб оплати.

— OrderItemsEntity — ця Таблиця пов’язана з Orders і містить деталі про кожну товарну позицію в замовленні. Вона містить зв’язок із OrderId, а також зберігає ідентифікатор продукту, кількість та інші дані, що стосуються конкретного елемента замовлення.

— RefreshTokens — таблиця, яка зберігає токени оновлення (refresh tokens), що використовуються в механізмі автентифікації. Кожен токен пов’язаний з користувачем (UserId → Users.Id) і містить інформацію про дату створення, закінчення терміну дії, а також про те, чи був токен відкликаний.

2.3 Розробка UML-діаграм інформаційної системи обліку виробів з пластику

UML-діаграми (Unified Modeling Language) — це стандартизовані графічні нотації, що використовуються для візуального моделювання структури та поведінки програмних систем. UML допомагає розробникам, аналітикам і замовникам краще зрозуміти, як працює система, які компоненти вона має, як вони взаємодіють між собою та які функціональні можливості надає [13].

UML-діаграма прецедентів (use case diagram) — це діаграма, яка відображає функціональні можливості системи з точки зору користувача. Вона показує, які дії (прецеденти) може виконувати кожен тип користувача (актор) у системі. Діаграма прецедентів використовується для збору вимог, опису

сценаріїв взаємодії користувача із системою та визначення основної функціональності. Діаграма прецедентів інформаційної системи обліку виробів з пластику зображена на рисунку 2.3.

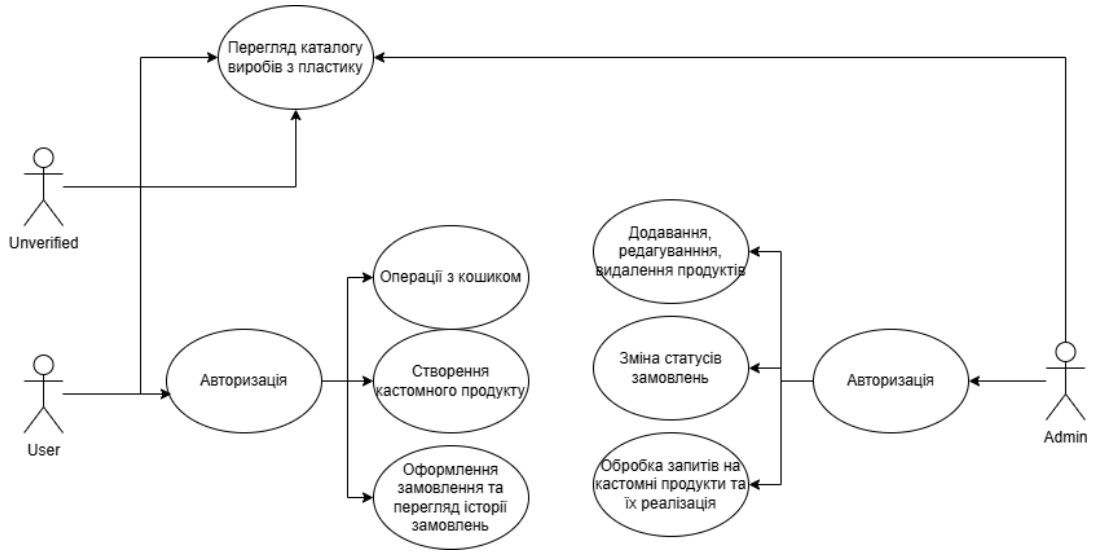

Рисунок 2.3 — Діаграма прецедентів інформаційної системи обліку виробів з пластику

На діаграмі зображено основні сценарії взаємодії користувачів з інформаційною системою обліку виробів з пластику. Учасниками є три типи акторів: гостьовий користувач, зареєстрований користувач та адміністратор. Кожен із них має свій набір дій , які він може виконувати в межах системи.

Гість має обмежений функціонал. Він може лише переглядати каталог виробів з пластику, тобто бачити доступні товари без можливості замовлення або взаємодії з кошиком. Також він має доступ до функції авторизації, яка дозволяє увійти в систему.

Зареєстрований користувач після авторизації отримує розширений функціонал. Він може здійснювати операції з кошиком, такі як додавання, редагування чи видалення товарів. Користувач має можливість створювати індивідуальні замовлення — це особливість системи, яка дозволяє надсилати запити на виготовлення виробів за власним дизайном. Крім того, користувач може оформлювати замовлення та переглядати їхню історію у своєму особистому кабінеті.

Адміністратор є користувачем з найвищими правами доступу. Він може додавати, редагувати та видаляти товари з каталогу — це дозволяє керувати асортиментом системи. Також адміністратор має право змінювати статуси замовлень та обробляти запити на кастомні продукти, приймаючи рішення про їх виготовлення або відхилення. Як і всі інші ролі, адміністратор проходить авторизацію перед отриманням доступу до системи.

UML-діаграма послідовностей (sequence diagram) — це поведінкова діаграма, яка демонструє, як об’єкти системи взаємодіють між собою в межах конкретного сценарію, в якому вони обмінюються повідомленнями. Вона зображає об’єкти, повідомлення між ними, порядок виконання дій та життєвий цикл кожного об'єкта під час взаємодії. Наприклад, для сценарію “Оформлення замовлення” діаграма покаже, як користувач надсилає запит до сервера, сервер перевіряє наявність товару, зберігає замовлення в базі даних і повертає підтвердження користувачу. Діаграма послідовності процесу реєстрації користувача в системі зображена на рисунку 2.4.

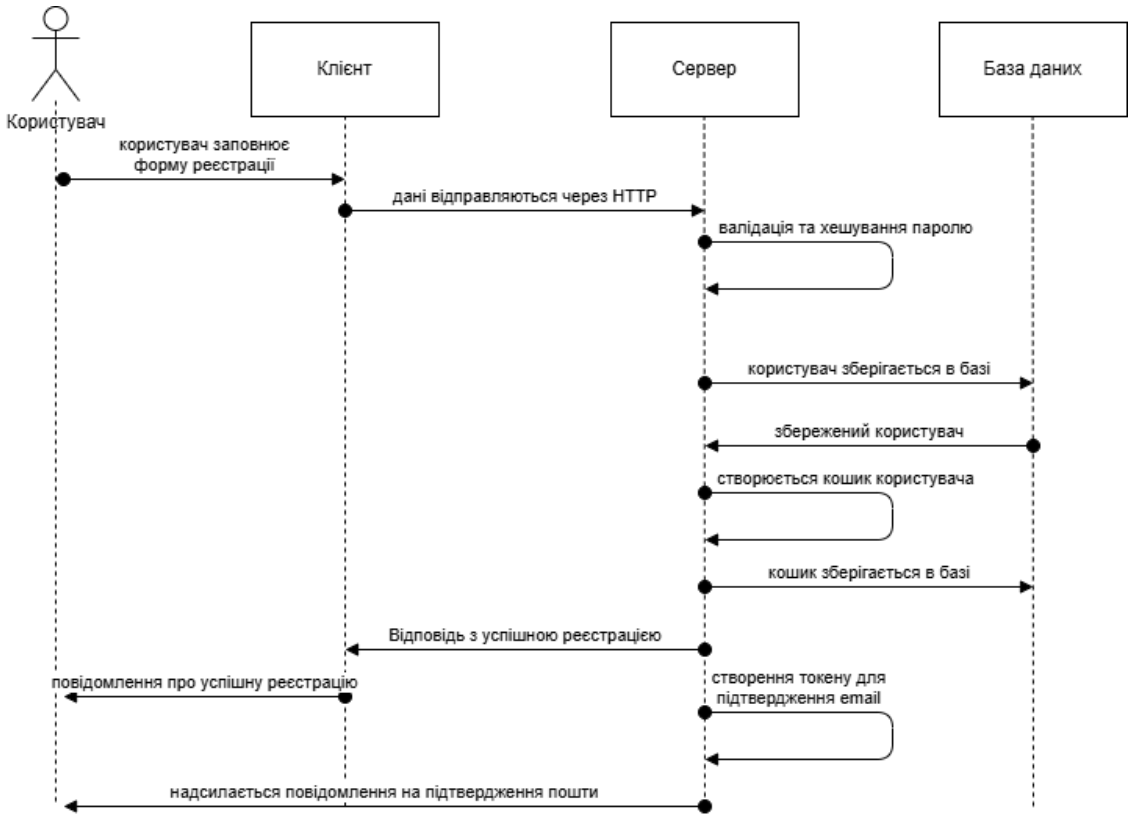

Рисунок 2.4 — Діаграма послідовності процесу реєстрації користувача в системі

На представленій UML-діаграмі послідовності зображено детальний процес реєстрації нового користувача в інформаційній системі обліку виробів з пластику. У процесі задіяні чотири основні учасники: користувач, клієнт, сервер та база даних. Кожен з учасників виконує свою роль у забезпеченні успішної реєстрації, створенні облікового запису та ініціалізації супутніх даних.

Процес починається з того, що користувач заповнює форму реєстрації на стороні клієнта. У цій формі зазвичай вказуються такі дані, як ім’я користувача, електронна адреса та пароль. Після заповнення форми, ці дані передаються від клієнта до сервера через HTTP-запит.

На стороні сервера першочергово виконується валідація введених даних. Якщо дані є коректними, наступним кроком є хешування пароля. Хешування необхідне для того, щоб пароль ніколи не зберігався у відкритому вигляді в базі даних.

Після підготовки об'єкта користувача, його дані зберігаються в базі даних. Сервер надсилає SQL-запит, щоб додати новий запис до таблиці Users.

Одразу після створення користувача система також ініціалізує супутні дані, зокрема створює порожній кошик, який буде використовуватися цим користувачем у майбутньому для формування замовлень. Об’єкт кошика зв’язується з ID користувача та також зберігається у базі даних. Таким чином, користувач уже має персоналізовану структуру для взаємодії з товарним каталогом.

Паралельно з цим сервер генерує спеціальний токен для підтвердження електронної пошти. Цей токен має обмежений термін дії та містить зашифровану інформацію, яка дозволяє ідентифікувати користувача при переході за посиланням. Створений токен вбудовується у посилання та надсилається на email користувача.

На завершення процесу користувач отримує повідомлення на свою пошту, де може перейти за посиланням та завершити активацію облікового запису. Цей крок є важливою складовою автентифікації, оскільки гарантує, що введена адреса дійсно належить користувачу, а обліковий запис не був створений зловмисником.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Реалізація авторизації за допомогою JWT

У сучасних веб-застосунках важливо забезпечити не лише функціональність, а й безпеку, гнучкість і зручність взаємодії між клієнтом та сервером. Для цього використовується REST API (Representational State Transfer Application Programming Interface) — це архітектурний стиль для створення веб-сервісів, який дозволяє системам обмінюватися даними через інтернет за допомогою стандартних HTTP-запитів. Простими словами, REST API — це набір правил, які визначають, як клієнт може звертатися до сервера для отримання, створення, оновлення чи видалення даних.

REST API базується на низці принципів, які забезпечують просту й стандартизовану взаємодію між клієнтом і сервером. Основними є використання HTTP-методів: GET — для отримання даних, POST — для створення нових об’єктів, PUT або PATCH — для їх оновлення, та DELETE — для видалення. Кожен ресурс у системі представляється у вигляді URL-адреси. Взаємодія між клієнтом і сервером відбувається переважно у форматі JSON, а кожен запит є безстанковим, тобто не залежить від попередніх — сервер не зберігає інформацію про стан клієнта між запитами [14].

Кожен запит у REST API надсилається за певною URL-адресою, що представляє ресурс. Запити формуються у форматі HTTP, у якому можна вказати заголовки, тіло запиту та метадані. У заголовках передається важлива службова інформація — наприклад, формат переданих даних (Content-Type: application/json) або токен авторизації (Authorization: Bearer <token>). Заголовки — це ключовий механізм для забезпечення автентифікації та авторизації користувача.

Перед тим як отримати доступ до особистого кабінету чи інших захищених розділів вебсайту, користувач повинен пройти процес автентифікації. Якщо користувач ще не має облікового запису, він спочатку реєструється за допомогою спеціальної форми, де вказує своє ім’я, електронну пошту та пароль. Після підтвердження електронної пошти обліковий запис активується. У разі,

якщо обліковий запис уже існує, користувач здійснює вхід через форму авторизації, вводячи email і пароль. Уся ця інформація передається на сервер у захищеному вигляді.

Автентифікація — це процес перевірки особи користувача, тобто підтвердження того, що саме та особа, за яку себе видає користувач, намагається увійти в систему. Як правило, це перевірка логіну та паролю. Авторизація — це процес надання або обмеження доступу до певних функцій або даних системи на основі ролі користувача.

Після того як користувач успішно входить у систему, сервер створює JWT (JSON Web Token) — це зашифрований цифровий токен, що містить у собі інформацію про користувача, його роль, ідентифікатор та термін дії токена. Цей JWT-токен клієнт зберігає в кукі (cookies) — спеціальному механізмі зберігання даних у браузері.

Кожен наступний запит клієнта до захищених ресурсів супроводжується додаванням JWT у заголовок Authorization або передається автоматично, якщо токен збережено в кукі з прапором HttpOnly. Це дозволяє серверу швидко ідентифікувати користувача, перевірити його права доступу й повернути відповідні дані, не запитуючи логін і пароль щоразу.

Для забезпечення безперервності сесії в більш безпечний спосіб сервер може використовувати пару токенів: access token (короткотривалий) і refresh token (довготривалий). Коли access token спливає, клієнт надсилає refresh token на окремий ендпоінт /refresh, і сервер видає нову пару токенів.

Спочатку створюємо у файлі конфігурації appsettings.json секцію JwtOptions, яка містить ключові параметри для роботи з JWT: SecretKey (секретний ключ для підпису), ExpiresMinutes (тривалість дії токена в хвилинах) та RefreshExpiresDays (строк дії refresh token'a в днях). Ці значення зчитуються через конфігураційний клас і використовуються при генерації та валідації токенів. Лістинг секції JwtOptions наведений у лістингу 3.1.

Лістинг 3.1 – Секція JwtOptions

```
"JwtOptions": {
  "SecretKey": "secretkeysecretkeysecretkeysecretkeysecretkeysecretkey",
```

Продовження лістингу 3.1

```
  "ExpiresMinutes": "1",
  "RefreshExpiresDays": "7"}
```

Далі ми створюємо конфігураційний клас JwtOptions, який використовується для зберігання та зчитування параметрів з файлу конфігурації, звертаючись до них як до властивостей об’єкту. Ми пов’язуємо наш клас з секцією в конфігураціях завдяки реєстрації конфігураційного класу в DI-контейнері, після чого ми можемо отримати доступ до налаштувань у будь-якому сервісі. IOptions<T> — це інтерфейс у .NET, який використовується для зручного отримання параметрів конфігурації з файлу appsettings.json. Коли ти викликаєш .Value, ти отримуєш заповнений екземпляр JwtOptions, зібраний з конфігураційного файлу. Код параметрів конфігурації наведений у лістингу 3.2.

Лістинг 3.2 — Отримання параметрів конфігурації за допомогою IOptions

```
private readonly JwtOptions _options;

public JwtProvider(IOptions<JwtOptions> options)
{
   _options = options.Value;
}
        public class JwtOptions
        {
                public string SecretKey {  get; set; } = string.Empty;
                public int ExpiresMinutes { get; set; }
                public int RefreshExpiresDays { get; set; }
        }
```

Оголошуємо метод Generate для генерації токену, де спочатку відбувається спроба отримати користувача з бази даних за унікальним ідентифікатором userId. Для цього використовується метод GetByUserId(...) із відповідного репозиторію. Якщо користувача з таким ID не знайдено, викидається виняток з повідомленням "Користувач не знайдений", що запобігає подальшій генерації токена для неавторизованої особи.

Далі з об'єкта користувача вилучаються його ролі, які перетворюються у список рядків. Ця інформація потрібна для подальшої авторизації. Після цього створюється список claims, у якому зберігатиметься інформація, що буде вкладена в токен. У claims обов’язково додається унікальний ідентифікатор

користувача (UserId), який дозволить серверу зчитувати, хто саме виконує запити. Також до списку claims додаються всі ролі користувача. Кожна роль зберігається у вигляді окремого об’єкта Claim. Це дозволяє системі пізніше зчитувати з токена, які саме права має користувач — наприклад, чи є він адміністратором, чи звичайним користувачем, і відповідно до цього обмежувати або дозволяти доступ до певних функцій API. Лістинг методу Generate наведений у лістингу 3.3.

Лістинг 3.3 — Метод Generate

```
public async Task<string> Generate(Guid userId)
{
	var user = await _usersRepository.GetByUserId(userId);

	if (user == null)
	{
		throw new Exception("Користувач не знайдений");
	}
	var roles = user.Roles.Select(r => r.ToString()).ToList();

	var claims = new List<Claim>
	{
		new Claim(CustomClaims.UserId, user.Id.ToString()),
	};

	foreach (var role in roles)
	{
		claims.Add(new Claim(CustomClaims.Roles, role));
	}
```

Далі створюємо об’єкт signingcredentials, який відповідає за підписування JWT-токена. Симетричний ключ береться з конфігурації (_options.secretkey) і перетворюється на байти, після чого використовується разом із алгоритмом HMAC SHA-256. Завдяки цьому токен неможливо змінити без знання секретного ключа, що гарантує його захищеність. Створюємо об'єкт JWT-токена. У параметрі claims передається список тверджень (claims) про користувача. Ці дані будуть закодовані в токен і дозволять серверу швидко розпізнавати, хто зробив запит, без додаткових перевірок у базі. Expires задає час, коли токен стане недійсним. Це захищає систему від використання старих токенів. Signingcredentials — це об’єкт, який містить криптографічну інформацію для підпису токена. Він гарантує, що токен дійсно виданий твоїм сервером, і не був

змінений кимось іншим. Продовження Лістинг методу Generate наведений у лістингу 3.4.

Лістинг 3.4 — Метод Generate

```
                var signingCredentials = new SigningCredentials(
                    new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_options.SecretKey)),
                    SecurityAlgorithms.HmacSha256);

                var token = new JwtSecurityToken(
                    claims: claims,
                    expires:
DateTime.UtcNow.AddMinutes(_options.ExpiresMinutes),
                    signingCredentials: signingCredentials);

                return new JwtSecurityTokenHandler().WriteToken(token);
            }
```

Також ми налаштовуємо JWT-автентифікацію у веб-застосунку на ASP.NET Core. Спочатку викликається метод AddAuthentication, який вмикає загальний механізм аутентифікації й повідомляє систему, що вона має використовувати схему JwtBearer, тобто перевіряти токени формату JWT при кожному запиті. Далі через метод AddJwtBearer відбувається конфігурація деталей, пов’язаних із перевіркою токена.

Усередині блоку конфігурації встановлюється кілька параметрів. Властивість RequireHttpsMetadata = true означає, що токени прийматимуться лише через захищене з’єднання HTTPS, що підвищує рівень безпеки. Параметр SaveToken = true дозволяє зберігати токен у контексті поточного запиту, щоб мати до нього доступ у внутрішньому коді сервера.

Найважливіший блок — TokenValidationParameters, де задаються правила перевірки токена. Параметр RoleClaimType = ClaimTypes.Role вказує системі, що інформація про роль користувача зберігається у клеймі role. Це дозволяє механізму авторизації правильно розпізнавати ролі користувачів і застосовувати до них політики доступу. Параметр ValidateLifetime = true забезпечує перевірку терміну дії токена. Тобто сервер перевірить, чи не минув час, зазначений у полі exp у JWT. Якщо токен прострочений — запит буде відхилено. ValidateIssuerSigningKey = true активує перевірку цифрового підпису токена, а

IssuerSigningKey = new SymmetricSecurityKey(...) передає секретний ключ, який використовується для цієї перевірки. Якщо підпис не збігається — токен вважається підробленим. Секретний ключ зчитується з конфігурації через jwtoptions.SecretKey, що дозволяє змінювати його без зміни коду.

Останній блок — це обробка події OnMessageReceived. Тут серверу вказується, звідки брати токен для перевірки. За замовчуванням JWT очікується в заголовку Authorization, але в цьому випадку токен зберігається в кукі з назвою "kokiie". Отже, при кожному запиті до сервера ця подія витягує токен із кукі та передає його далі в механізм автентифікації. Лістинг налаштування JWT-автентифікації наведений у лістингу 3.5.

Лістинг 3.5 — Налаштування JWT-автентифікації з отриманням токена з cookie

```
        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                                .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme,
options =>
                                {
                                        options.RequireHttpsMetadata = true;
                                        options.SaveToken = true;

                                        options.TokenValidationParameters = new()
                                        {
                                                RoleClaimType = ClaimTypes.Role,
                                                ValidateIssuer = false,
                                                ValidateAudience = false,
                                                ValidateLifetime = true,
                                                ValidateIssuerSigningKey = true,
                                                IssuerSigningKey = new
SymmetricSecurityKey(

        Encoding.UTF8.GetBytes(jwtoptions.SecretKey))
                                        };
                                        options.Events = new JwtBearerEvents
                                        {
                                                OnMessageReceived = context =>
                                                {
                                                        context.Token =
context.Request.Cookies["kokiie"];
                                                        return Task.CompletedTask;
                                                }
                                        };
                                });
```

Метод GetTokenPrincipal виконує перевірку JWT-токена, який зберігається, наприклад, у cookie, та повертає об’єкт типу ClaimsPrincipal, що містить усі claims (ідентифікаційні дані), витягнуті з цього токена. Цей метод відіграє ключову роль у підтвердженні автентичності користувача, особливо в системах, де токен не передається в заголовках, а зберігається в HttpOnly cookie.

У перших рядках створюється обробник токенів JwtSecurityTokenHandler та зчитується симетричний ключ, який використовується для перевірки цифрового підпису токена. Потім формуються параметри валідації (TokenValidationParameters), де вказано, що потрібно перевіряти підпис, але не обов’язково перевіряти видавця та аудиторію. Також активована перевірка строку дії, що запобігає використанню прострочених токенів.

Основна логіка виконується в блоці try. Якщо токен дійсний і відповідає ключу, його розбирають методом ValidateToken(...), який повертає ClaimsPrincipal — це об’єкт, що містить усі дані з токена у вигляді claims. Одразу після цього метод перевіряє claim "TokenType" — спеціальне поле, яке може використовуватись для розрізнення типів токенів (access або confirmation). Якщо тип токена не відповідає очікуваному, метод повертає null, що означає недійсний або неприйнятний токен. Лістинг методу GetTokenPrincipal наведений у лістингу 3.6.

Лістинг 3.6 — Метод GetTokenPrincipal

```
public ClaimsPrincipal? GetTokenPrincipal(string token, string
expectedTokenType)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.UTF8.GetBytes(_options.SecretKey);

    var validationParams = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true
    };
    try
    {
        var principal = tokenHandler.ValidateToken(token,
validationParams, out var validatedToken);
```

Продовження лістингу 3.6.

```
                    var tokenType = principal.FindFirst("TokenType")?.Value;

                    if (tokenType != expectedTokenType)
                    {
                        return null;
                    }
                    return principal; }
                catch{
return null; }}
```

3.2. Розробка REST API за допомогою Swagger

Для зручності розробки REST API інтегрується з інструментом Swagger (OpenAPI). Swagger автоматично генерує документацію до всіх доступних ендпоінтів API, дозволяє переглядати, які методи реалізовані (наприклад, POST /api/products, GET /api/orders), які параметри приймаються, яку відповідь сервер повертає, та які статуси можуть виникати (200 OK, 401 Unauthorized, 403 Forbidden). Swagger також дозволяє розробнику вручну протестувати запити просто з браузера, вказавши тіло запиту, заголовки та токени.

Swagger — це інструмент для автоматичної генерації документації до REST API, який значно спрощує розробку, тестування та взаємодію з веб-сервісами. Він створює зрозумілий веб-інтерфейс, у якому можна переглядати всі доступні ендпоінти, параметри запитів, типи відповідей і коди статусу. Завдяки інтеграції з ASP.NET Core Web API, Swagger формує документацію без додаткових зусиль — просто на основі атрибутів у коді. Крім того, він дозволяє напряму тестувати запити в браузері, що зручно для розробників і тестувальників, а також підтримує генерацію клієнтського коду для різних мов програмування.

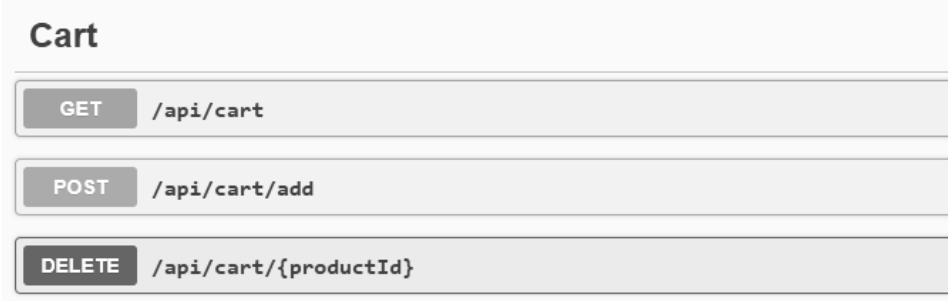

Рисунок 3.1 — Ендпоінти контролера кошика

На рисунку 3.1 зображено набір HTTP-роутів, що належать до модуля роботи з корзиною (Cart). У Swagger-інтерфейсі представлено три основні маршрути, які дозволяють взаємодіяти з корзиною користувача: перегляд вмісту, додавання товарів та їхнє видалення.

Запит GET /api/cart відповідає за отримання поточного вмісту корзини користувача. Це дозволяє динамічно відображати товари, які були додані до корзини раніше. Запит POST /api/cart/add використовується для додавання нового товару до корзини. У тілі запиту передається ідентифікатор товару та його кількість. Запит DELETE /api/cart/{productId} забезпечує видалення конкретного товару з корзини за його унікальним ідентифікатором, що передається як параметр у URL. Лістинг контролеру кошику наведений у додатку Д, лістинг Д.1.

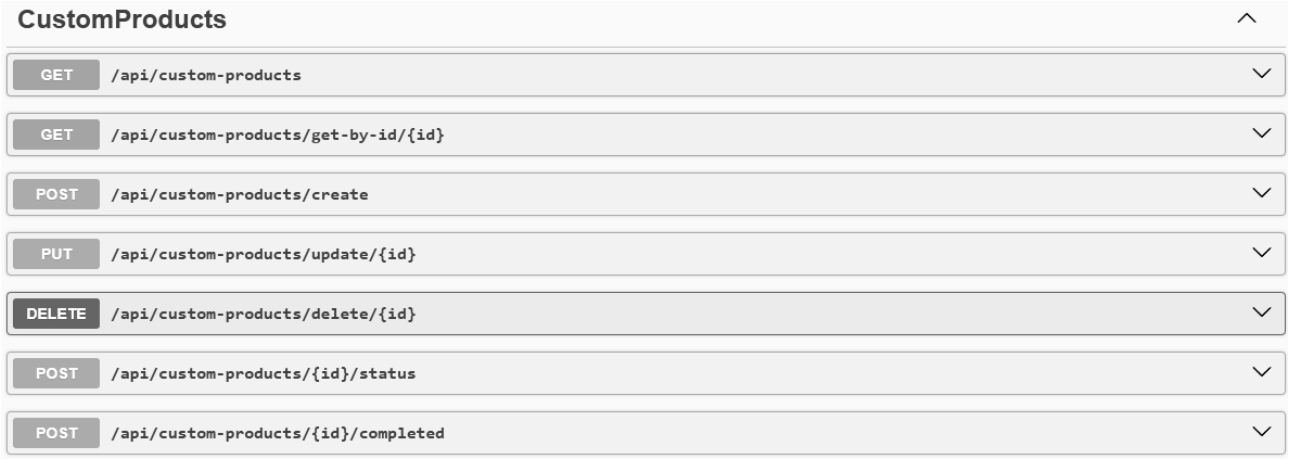

Рисунок 3.2 — Ендпоінти контролера кастомних продуктів

На рисунку 3.2 зображено набір API-роутів, які відповідають за роботу з модулем користувацьких продуктів. У Swagger-інтерфейсі представлені ендпоінти для повного управління життєвим циклом індивідуальних замовлень, починаючи зі створення, редагування і завершуючи зміною статусу.

Запит GET /api/custom-products дозволяє отримати перелік усіх користувацьких продуктів, а GET /api/custom-products/get-by-id/{id} — отримати детальну інформацію про конкретне замовлення за його ідентифікатором. Запит POST /api/custom-products/create відповідає за створення нового користувацького продукту на основі даних, введених користувачем. Запит PUT /api/custom-

products/update/{id} дозволяє редагувати існуючий продукт, поки його статус не змінено на фінальний. Запит DELETE /api/custom-products/delete/{id} видаляє користувацьке замовлення з системи. Запит POST /api/custom-products/{id}/status дозволяє змінювати статус користувацького продукту вручну. Запит POST /api/custom-products/{id}/completed дозволяє перетворити запит на кастомний продукт у існуючий продукт та додати його до загальних продуктів, також надіславши замовнику повідомлення та посилання на створений продукт. Лістинг контролеру кастомних продуктів наведений у додатку Д, лістинг Д.2.

Рисунок 3.3 — Ендпоінти для керування файлами

На рисунку 3.3 зображено API-роути, які належать до модуля роботи з файлами у AWS S3 bucket. Через ці ендпоінти реалізовано основні операції з управління файлами на сервері — завантаження, створення папок і видалення.

Запит POST /api/files/uploadFile використовується для завантаження файлу. Запит POST /api/files/create-folder/{prefix} дозволяє створити нову папку з указаним префіксом. Такий функціонал корисний для логічного групування файлів, наприклад, за категоріями або по користувачах. Запит POST /api/files/deleteFile призначений для видалення конкретного файлу з файлового сховища. Лістинг контролеру керування файлами у додатку Д, лістинг Д.3.

Рисунок 3.4 — Ендпоінти для керування замовленнями

На рисунку 3.4 зображено Swagger-інтерфейс із набором маршрутів, що відповідають за обробку замовлень у системі.

Запит POST /api/order/place-order відповідає за оформлення нового замовлення користувачем на основі вмісту його корзини. Він формує новий запис у системі замовлень і викликає додаткові дії, такі як надсилання листа-підтвердження. Запит GET /api/order/order-list дозволяє користувачу переглянути список лише своїх замовлень. Це зручно для історії покупок, повторних замовлень чи контролю статусу. Запит GET /api/order/all-orders-list відкриває доступ до повного списку замовлень у системі, зазвичай цей маршрут використовується адміністраторами для модерації. Запит POST /api/order/{orderId} призначений для оновлення окремого замовлення за ідентифікатором. Запит DELETE /api/order/{orderId} використовується для видалення замовлення з бази даних. Лістинг контролеру керування замовленнями наведений у додатку Д, лістинг Д.4.

Рисунок 3.5 — Ендпоінти керування товарами

На рисунку 3.5 зображено Swagger-інтерфейс із набором маршрутів, що належать до модуля керування товарами. Ці ендпоінти дозволяють здійснювати повний CRUD-функціонал (створення, читання, оновлення, видалення) над товарними одиницями, які зберігаються в системі.

Запит GET /api/products використовується для отримання списку всіх наявних продуктів. Він застосовується для відображення каталогу товарів на стороні клієнта або адміністратора. Запит POST /api/products/create дозволяє створити новий товар, надаючи необхідні параметри. Запит PUT /api/products/{id} забезпечує редагування вже існуючого товару за його

унікальним ідентифікатором. Запит DELETE /api/products/{id} дозволяє видалити продукт із системи. Запит GET /api/products/{productId} дає можливість отримати повну інформацію про конкретний товар. Використовується для перегляду детальної сторінки продукту. Лістинг контролеру керування товарами наведений у додатку Д, лістинг Д.5.

Рисунок 3.6 — Ендпоінти аутентифікації та управління користувачами

На рисунку 3.6 зображено Swagger-інтерфейс модуля Users, який реалізує повний набір функцій для автентифікації, авторизації та управління обліковими записами користувачів.

Запит POST /api/users/register використовується для реєстрації нового користувача. Запит POST /api/users/confirm-email забезпечує підтвердження електронної адреси за допомогою спеціального токена, який надсилається користувачу на email після реєстрації. Запит POST /api/users/login виконує вхід користувача в систему. У відповідь повертається токен доступу та оновлення, що використовуються для захисту подальших запитів. Запит POST /api/users/refresh відповідає за оновлення токенів доступу, коли access token спливає, але refresh token ще чинний. Запит POST /api/users/send-forget-password-email ініціює процедуру відновлення пароля — користувачу надсилається лист із посиланням для скидання пароля. Запит POST /api/users/change-password дозволяє встановити новий пароль на основі наданого токена, який був отриманий у попередньому кроці. Запит POST /api/users/logout використовується для виходу

користувача з системи та очищення токенів зберігання. Лістинг контролеру аутентифікації та управління користувачами наведений у додатку Д, лістинг Д.6.

3.3. Розробка сервісу електронних сповіщень

Наявність електронних сповіщень є важливим елементом сучасної інформаційної системи, оскільки значно покращує взаємодію між користувачем та застосунком. Завдяки автоматичному надсиланню листів можна налаштувати підтвердження реєстрації, зміну пароля, оповіщення про зміну статусу замовлення або повідомлення про видалення товару з кошика.

Спочатку нам потрібно створити наш обліковий запис з якого ми будемо надсилати наші повідомлення та працювати. Вигляд нашого облікового запису зображений на рисунку 3.7.

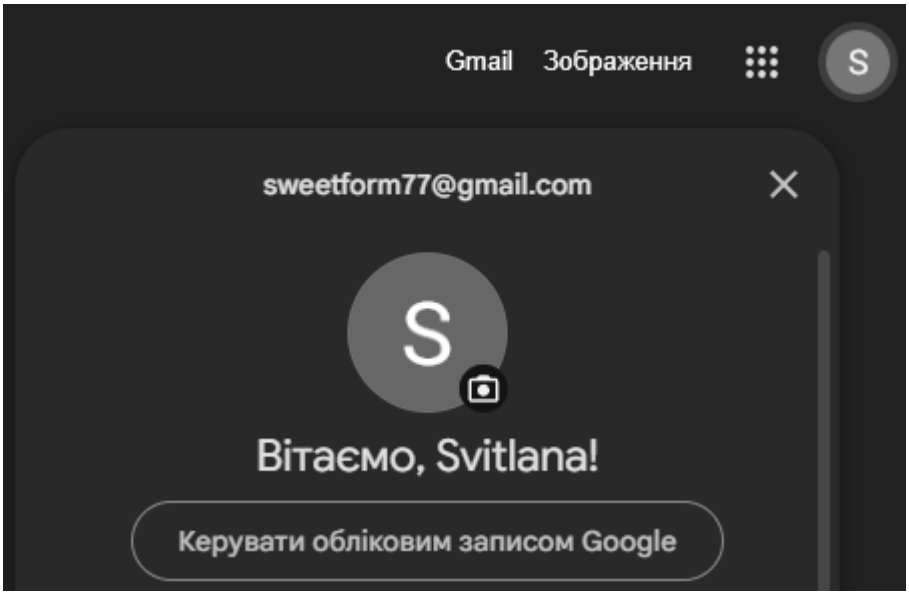

Рисунок 3.7 — Обліковий запис Google

Оголошуємо клас MailsService, який реалізує інтерфейс IMailsService і відповідає за логіку надсилання електронних листів у застосунку.

У класі оголошено два приватні поля: _templatePath — шлях до папки з HTML-шаблонами листів, та _emailSettings — об’єкт, що містить конфігураційні параметри створеної електронної пошти. Обидва поля ініціалізуються в конструкторі, у який передаються об’єкти IWebHostEnvironment та IOptions<EmailSettings>. Перший потрібен для визначення кореневої директорії

проєкту, а другий — для зчитування налаштувань пошти з конфігурації через DI. Шлях вказує на директорію MailTemplates. Лістинг класу MailsService наведений у лістингу 3.7.

Лістинг 3.7 — Клас MailsService

```
public class MailsService : IMailsService
{
	private readonly string _templatePath;
	private readonly EmailSettings _emailSettings;
	public MailsService(IWebHostEnvironment env, IOptions<EmailSettings>
emailSettings)
	{
		_emailSettings = emailSettings.Value;
		_templatePath = Path.Combine(env.ContentRootPath,"..",
			"SweetForm.Infrastructure", "MailService", "MailTemplates");
	}
```

Метод SendEmailAsync — це приватний асинхронний метод, який безпосередньо відповідає за відправлення електронного листа через SMTP-сервер. Він використовується як допоміжний усередині інших методів і приймає параметр message типу MimeMessage, який містить усі необхідні дані листа: відправника, одержувача, тему та тіло повідомлення. SMTP (Simple Mail Transfer Protocol) — це інтернет-протокол, який використовується для надсилання електронних листів із поштового клієнта або веб-додатку до поштового сервера, а також між серверами. SMTP-сервер приймає лист, перевіряє відправника, виконує доставку або пересилає його далі до адресата, відіграючи ключову роль у процесі електронного листування. Принцип роботи SMTP-сервера наведений на рисунку 3.8.

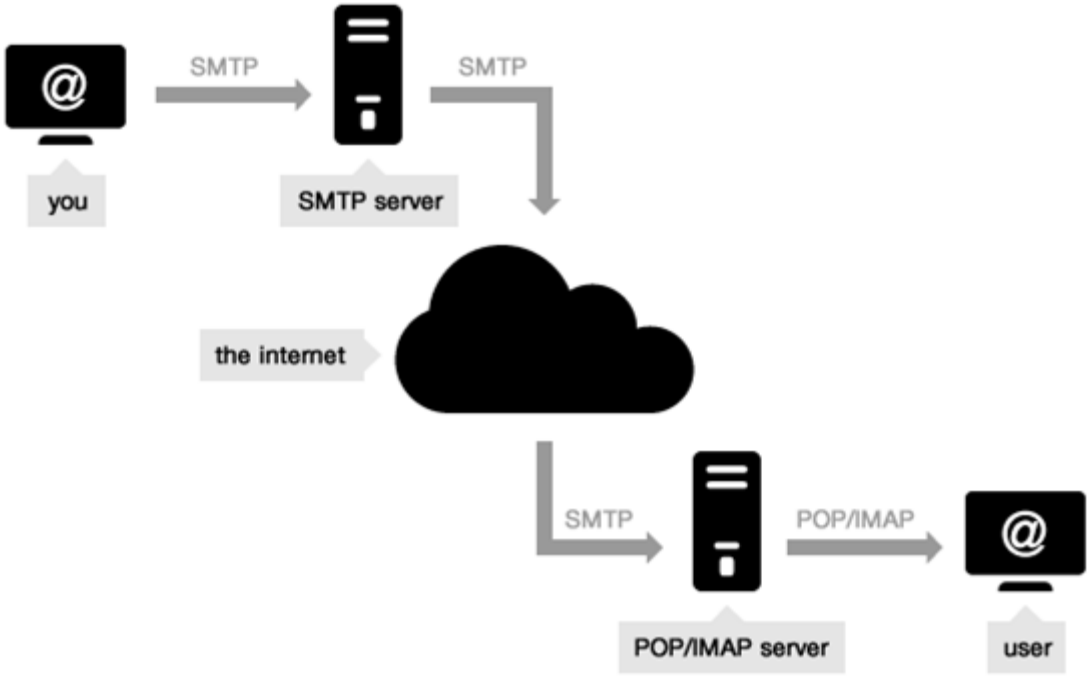

Рисунок 3.8 — Принцип роботи SMTP-сервера

У тілі методу створюється екземпляр SMTP-клієнта з бібліотеки MailKit. Далі виконується підключення до SMTP-сервера за адресою та портом, вказаними у налаштуваннях _emailSettings, із використанням захищеного з'єднання через StartTls. Після встановлення з'єднання метод автентифікується на сервері, використовуючи облікові дані відправника.

Після успішної автентифікації виконується відправлення повідомлення методом SendAsync. Після надсилання листа клієнт відключається від сервера з використанням DisconnectAsync. Лістинг методу SendEmailAsync наведений у лістингу 3.8.

Лістинг 3.8 — Метод SendEmailAsync

```
private async Task<bool> SendEmailAsync(MimeMessage message)
{
        try
        {
                using (var client = new SmtpClient())
                {
                        await client.ConnectAsync(_emailSettings.SmtpServer,
_emailSettings.Port, MailKit.Security.SecureSocketOptions.StartTls);
                        await
client.AuthenticateAsync(_emailSettings.SenderEmail, _emailSettings.Password);
                        await client.SendAsync(message);
                        await client.DisconnectAsync(true);
```

Продовження лістингу 3.8.

```
                    return true;
                }
            }
            catch (Exception ex)
            {
                throw new ApplicationException("Помилка під час
надсилання листа.", ex);
            }
        }
```

Метод SendConfirmationEmail — це публічний асинхронний метод, який відповідає за надсилання користувачу електронного листа для підтвердження реєстрації. Його мета — згенерувати HTML-лист із персоналізованим підтверджувальним посиланням і відправити його за вказаною email-адресою. Метод приймає два параметри: електронну адресу користувача та токен підтвердження, який використовується для формування унікального посилання.

На початку методу створюється MIME-повідомлення з відповідними полями: відправником, адресатом та темою "Підтвердження реєстрації". Далі генерується посилання для підтвердження, яке має вигляд http://localhost:3000/confirm-email?token={token}, і буде включене до тіла листа.

Після цього визначається шлях до HTML-шаблону листа EmailConfirmation.html. Якщо файл шаблону не знайдено, метод викидає виняток ApplicationException. Якщо файл існує, його вміст зчитується асинхронно, після чого в тексті шаблону відбувається заміна плейсхолдерів: {{Name}} на email користувача та {{ConfirmUrl}} на згенероване посилання для підтвердження.

Готовий HTML-контент додається до листа як тіло з MIME-типом "html". Наприкінці метод викликає SendEmailAsync, який безпосередньо надсилає повідомлення через SMTP. Лістинг методу наведений у додатку Д, лістингу Д.22. Результат відправлення повідомлення підтвердження реєстрації на електронну пошту зображений на рисунку 3.9.

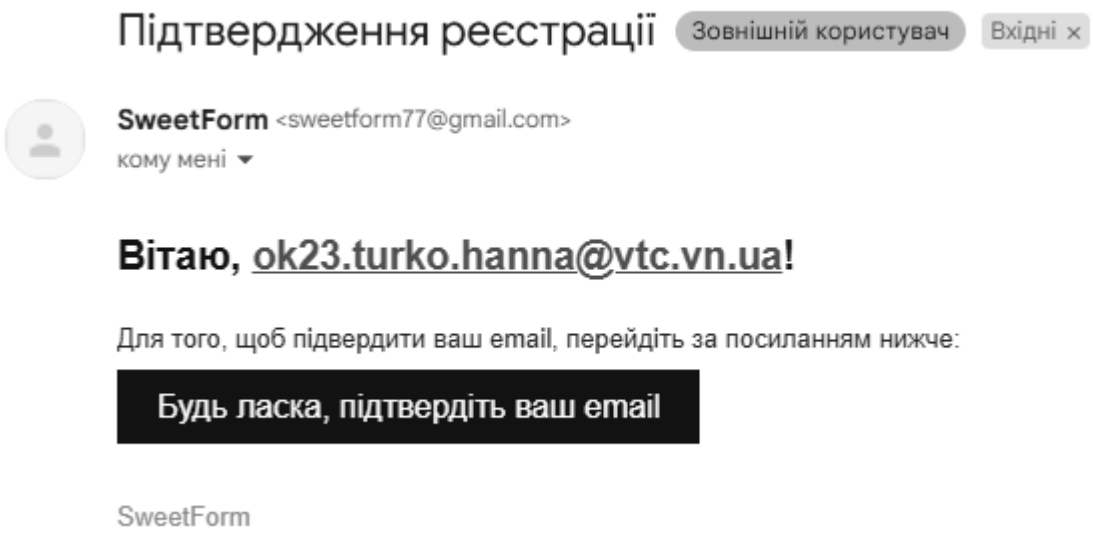

Рисунок 3.9 — Повідомлення підтвердження реєстрації

Метод SendResetPasswordEmail — це публічний асинхронний метод, який відповідає за надсилання електронного листа користувачу з посиланням для скидання пароля. Його основна мета — підготувати персоналізований HTML-лист на основі шаблону та надіслати його за вказаною адресою електронної пошти. Метод приймає два параметри: адресу одержувача і токен, який використовується для ідентифікації запиту на скидання пароля.

На початку методу створюється новий об'єкт MimeMessage, в якому вказується відправник, одержувач та тема листа — "Скидання пароля". Далі генерується спеціальне посилання для скидання пароля, яке включає переданий токен, і виглядає як http://localhost:3000/reset-password?token={token}.

Після цього визначається шлях до HTML-шаблону листа — файл ResetPasswordTemplate.html. Якщо цей файл не існує, метод викидає виняток ApplicationException з поясненням, що шаблон не знайдено. Якщо файл існує, метод зчитує його вміст і виконує підстановку динамічних значень — адреси користувача ({{Name}}) та посилання для скидання пароля ({{ResetUrl}}).

Сформований HTML-контент додається до MIME-повідомлення як тіло листа з типом html. Нарешті, метод викликає SendEmailAsync, який відповідає за фактичне відправлення листа через SMTP. Лістинг методу наведений у додатку

Д, лістингу Д.23. Результат відправлення повідомлення про скидання паролю на електронну пошту зображений на рисунку 3.10.

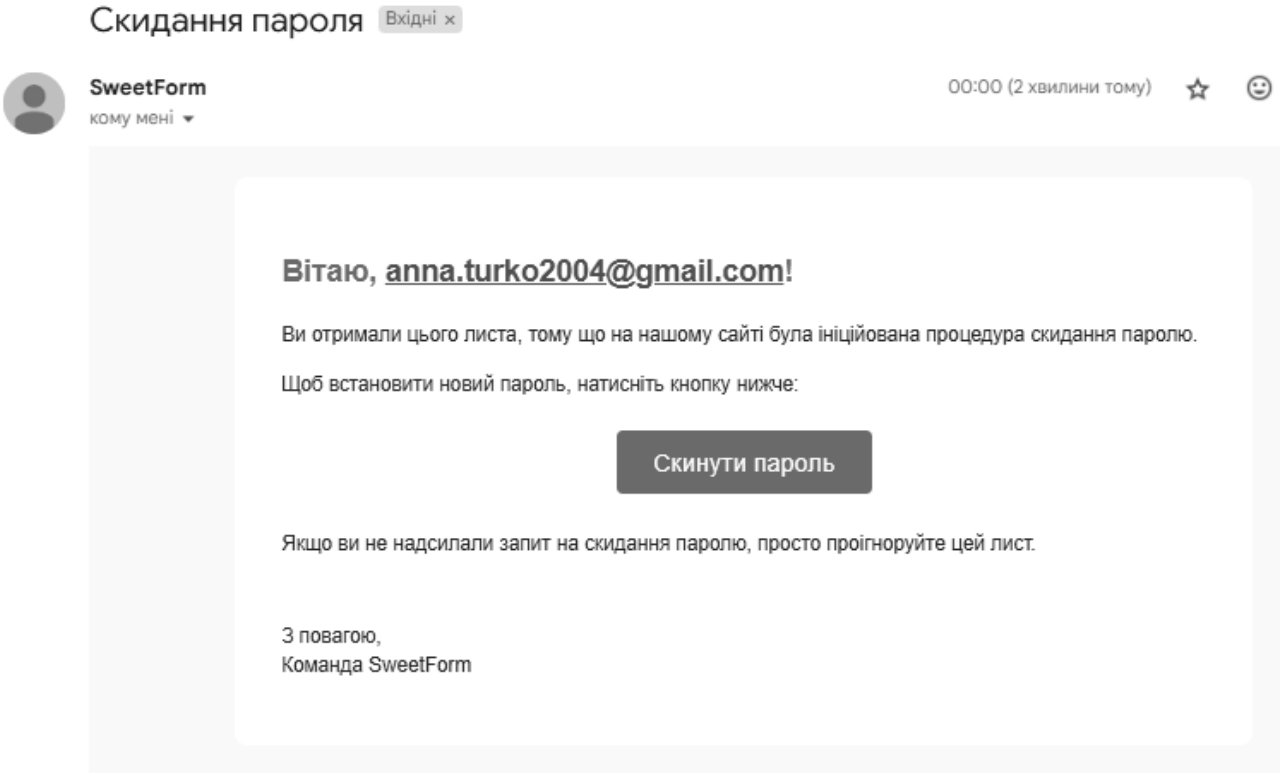

Рисунок 3.10 — Повідомлення про скидання паролю

Метод GenerateOrderItemsHtml є приватною допоміжною функцією, яка конвертує список товарів із замовлення у HTML-формат у вигляді ненумерованого списку. Для кожного товару створюється елемент <li>, що містить назву продукту, кількість, ціну за одиницю та загальну вартість позиції. Всі елементи поступово додаються до об’єкта StringBuilder, після чого сформований HTML-код повертається як рядок. Сформований список товарів пізніше вбудовується в тіло листа, який надсилається за допомогою методу SendEmailAsync. Лістинг методу GenerateOrderItemsHtml наведений у лістингу 3.9.

Лістинг 3.9 — Метод GenerateOrderItemsHtml

```
private string GenerateOrderItemsHtml(List<OrderItems> items)
{
        StringBuilder sb = new StringBuilder();
        sb.Append("<ul>");
        foreach (var item in items)
```

Продовження лістингу 3.9.

```
                {
                    sb.AppendFormat("<li>{0} - {1} {2:C2} {3:C2}</li>",
item.ProductName, item.Quantity, item.ProductPrice, item.Quantity * item.ProductPrice);
                }
                sb.Append("</ul>");
                return sb.ToString();
            }
```

Метод SendOrderConfirmationMail — це публічна асинхронна функція, призначена для надсилання електронного листа з підтвердженням замовлення користувачу. Вона приймає два параметри: адресу електронної пошти одержувача та об’єкт замовлення. Основним завданням методу є формування HTML-листа на основі шаблону, динамічна заміна даних про замовлення та передача листа через SMTP-сервер.

На початку створюється MIME-повідомлення з відповідними відправником, одержувачем та темою. Далі вказується шлях до HTML-шаблону, який зчитується, якщо існує. Якщо файл шаблону відсутній, генерується ApplicationException. У випадку успішного зчитування в шаблоні здійснюється заміна плейсхолдерів: {{OrderId}} — на унікальний номер замовлення, {{Name}} — на електронну адресу користувача, {{Amount}} — на загальну суму замовлення, а {{OrderItems}} — на згенеровану HTML-розмітку зі списком замовлених товарів, яку створює метод GenerateOrderItemsHtml. Повна реалізація коду наведена у додатку Д, лістингу Д.24. Результат відправлення повідомлення про створене замовлення на електронну пошту зображений на рисунку 3.11.

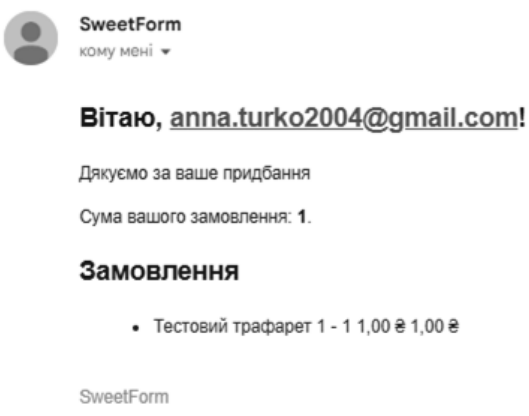

Рисунок 3.11 — Повідомлення про створене замовлення

Метод SendCustomProductNotificationToAdmin відповідає за надсилання адміністратору електронного листа, що містить детальну інформацію про новий кастомний продукт, створений користувачем на платформі. Це необхідно для того, щоб адміністрація сайту оперативно дізнавалася про нові замовлення, які вимагають модерації, виготовлення або подальшого опрацювання.

У першу чергу в методі створюється об’єкт MimeMessage, який використовується для формування електронного листа. В полі From додається відправник — ім’я та email, які зчитуються із конфігурації. В полі To додається одержувач — адміністратор. У даній реалізації, як отримувач листа, виступає все той же _emailSettings.SenderEmail з метою самосповіщення.

Далі метод формує тіло листа. Для цього він шукає HTML-шаблон за шляхом CustomProductNotification.html в директорії шаблонів, яка задається змінною _templatePath. Якщо такий файл не знайдено, кидається ApplicationException із відповідним поясненням — це захищає систему від надсилання порожніх або некоректних листів. Якщо шаблон знайдено, його вміст зчитується.

Після зчитування шаблону метод проводить заміну всіх цих змінних на реальні дані з об'єкта product і параметра userEmail. Таким чином, у фінальному HTML тілі листа буде видно повну інформацію про створений продукт: його назву, опис, тип, матеріал, розмір, статус, дату створення, зображення і електронну адресу автора замовлення. Це дозволяє адміністратору ознайомитися з усіма деталями прямо у листі без потреби переходити на сайт вручну.

Після цього сформований HTML текст обгортається у TextPart з MIME-типом "html" — це дозволяє відправляти форматований лист, який зручно переглядати у більшості поштових клієнтів. Далі метод викликає асинхронний метод SendEmailAsync(message), який фактично надсилає сформований лист через SMTP. Лістинг наведений у додатку Д, лістингу Д.25. Результат відправлення повідомлення про нове індивідуальне замовлення на електронну пошту адміністрації зображений на рисунку 3.12.

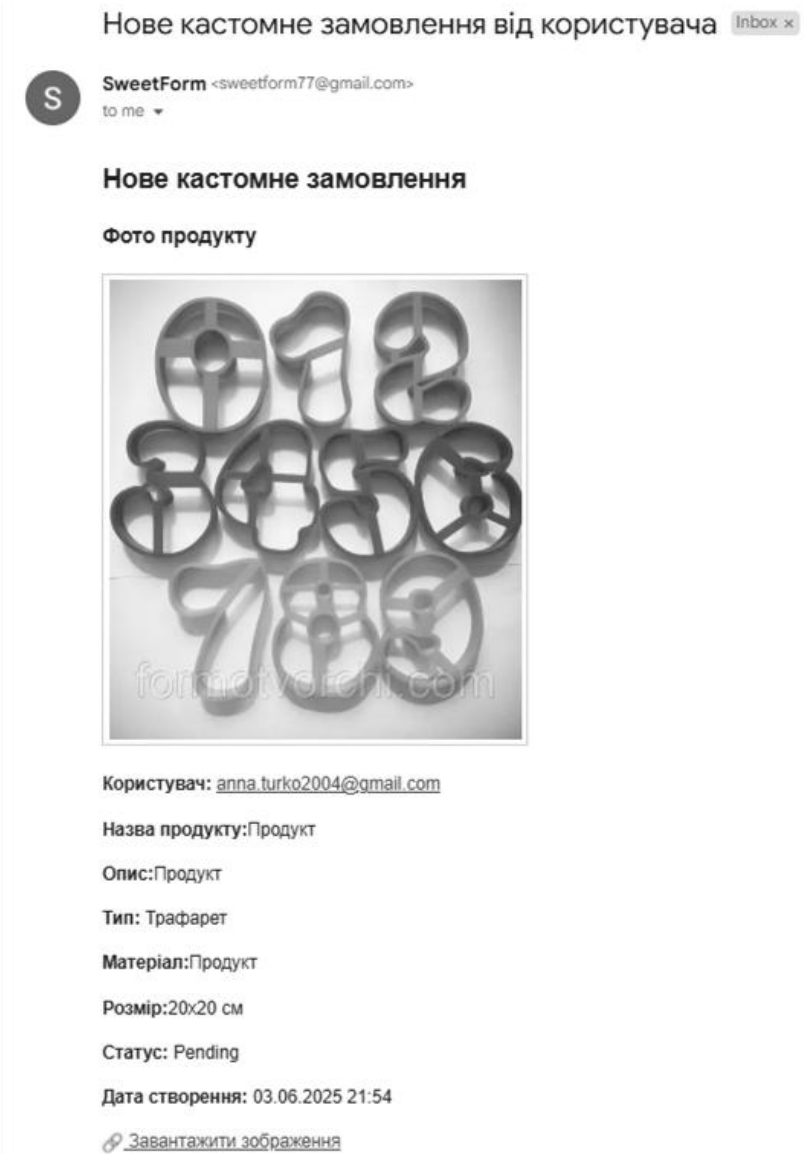

Рисунок 3.12 — Повідомлення про нове індивідуальне замовлення

3.4. Розробка користувацького інтерфейсу

Було створено головну сторінку сайту з формочками та трафаретами для випікання, виконаними на 3D-принтері. Вона орієнтована на неавторизованих користувачів, дозволяючи їм ознайомитися з продукцією, перейти до авторизації або зареєструватися.

У верхній частині сторінки розташоване головне меню навігації, що включає посилання на такі розділи, як "Головна", "Продукти". Це дозволяє користувачу швидко переходити між ключовими сторінками сайту. Вигляд головної сторінки сайту зображено на рисунках 3.13 – 3.14. Лістинг головної сторінки наведений у додатку Д, лістингу Д.7.

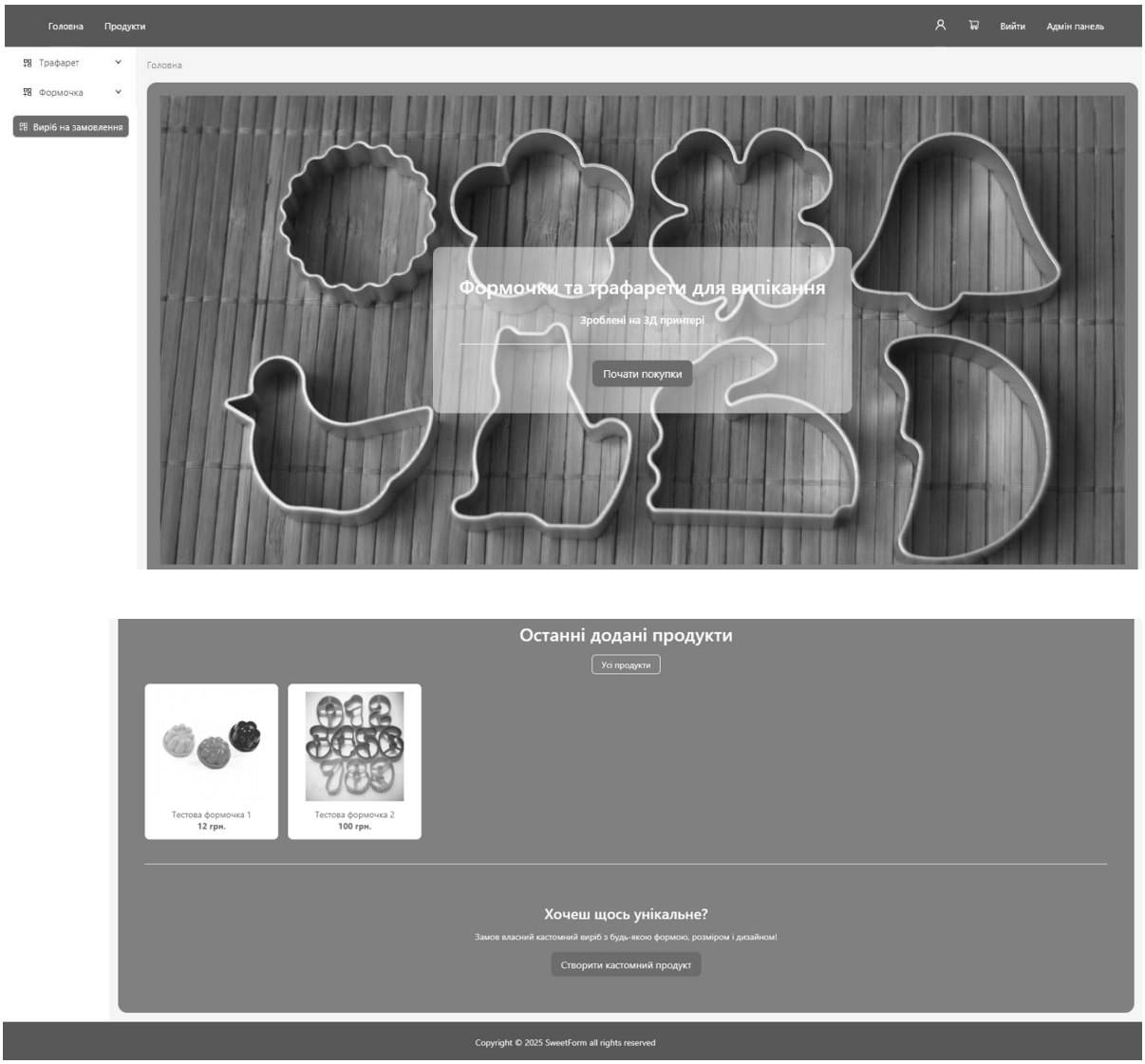

Рисунок 3.13 - 3.14 — Головна сторінка

Модальне вікно призначене для входу користувача до системи. У ньому передбачено два обов’язкових поля: для введення електронної пошти та пароля. Користувач також має змогу скористатися функцією відновлення паролю через посилання "Забули пароль?", яке розміщене під полями введення. Основна дія здійснюється через синю кнопку "Увійти", що запускає процес авторизації. Якщо в користувача ще немає облікового запису, він може натиснути на посилання "Створити акаунт", щоб перейти до сторінки реєстрації. Вигляд модального вікна входу до систему зображено на рисунку 3.15. Лістинг модального вікна входу до системи наведений у додатку Д, лістингу Д.8.

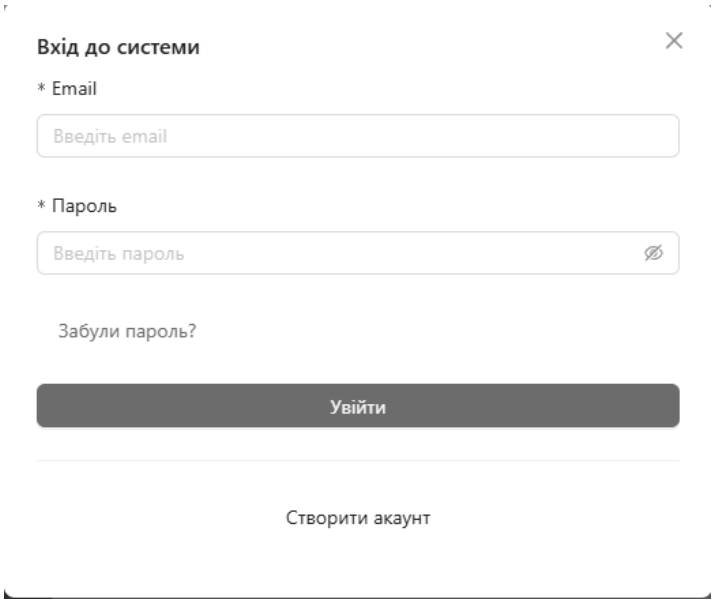

Рисунок 3.15 — Модальне вікно входу до системи

Модальне вікно реєстрації нового користувача. Воно містить три обов’язкові поля для заповнення: ім’я користувача, електронну пошту та пароль. Усі поля мають підписи й плейсхолдери, що підказують, яку інформацію потрібно ввести. Для пароля передбачено іконку, яка дозволяє показувати або приховувати введене значення. Після заповнення форми користувач може натиснути синю кнопку "Зареєструватися", щоб створити акаунт. Нижче розташоване посилання "Увійти в існуючий акаунт", яке дає змогу перейти до форми входу. Вигляд модального вікна реєстрації нового користувача зображено на рисунку 3.16. Лістинг модального вікна реєстрації нового користувача наведений у додатку Д, лістингу Д.9.

Сторінка профілю користувача на сайті. Вона поділена на два основні блоки: "Мої замовлення" та "Мої кастомні продукти".

У блоці "Мої замовлення" відображається список оформлених замовлень із зазначенням унікального ідентифікатора, дати покупки, статусу, переліку товарів, кількості та загальної суми замовлення. Вигляд блоку "Мої замовлення" зображено на рисунку 3.17.

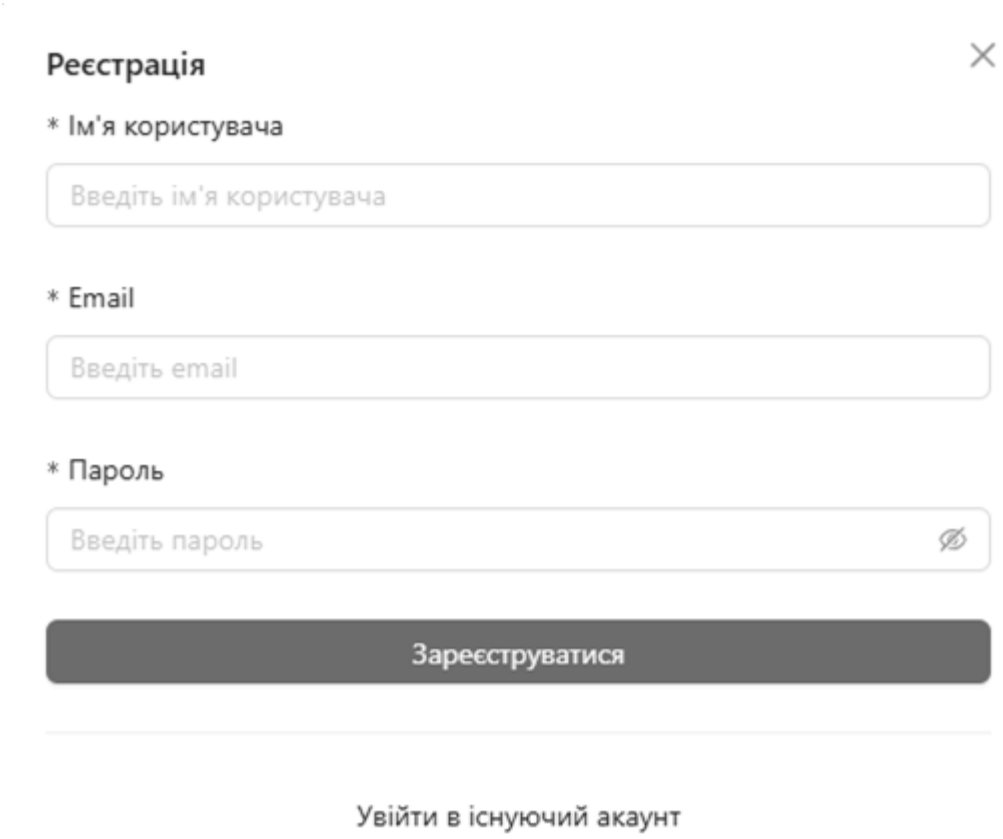

Рисунок 3.16 — Модальне вікно реєстрації нового користувача

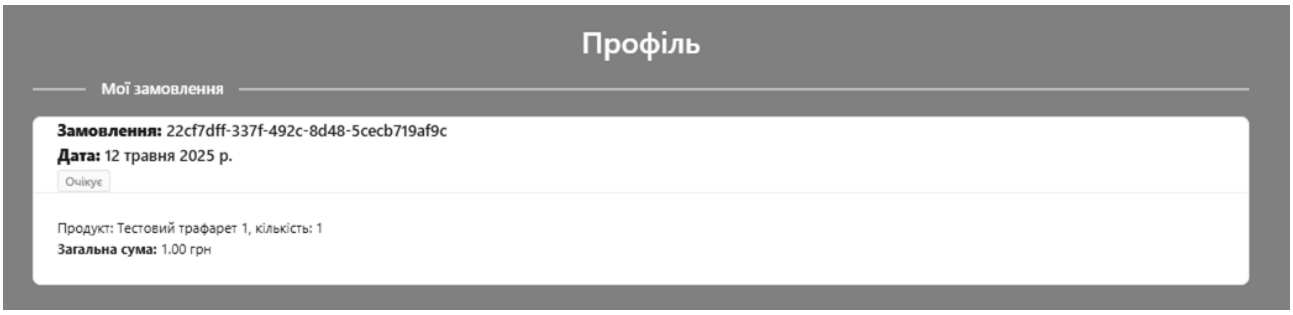

Рисунок 3.17 — Сторінка профілю користувача, блок «Мої замовлення»

У секції "Мої кастомні продукти" користувач бачить власноруч створений виріб. Вказано зображення продукту, його назву, опис, тип, матеріал та розмір. Також доступні дії: Редагувати або Видалити продукт, якщо він ще знаходиться на стадії очікування розгляду. Вигляд секції "Мої кастомні продукти" зображено на рисунку 3.18. Лістинг сторінки профілю користувача наведений у додатку Д, лістингу Д.10.

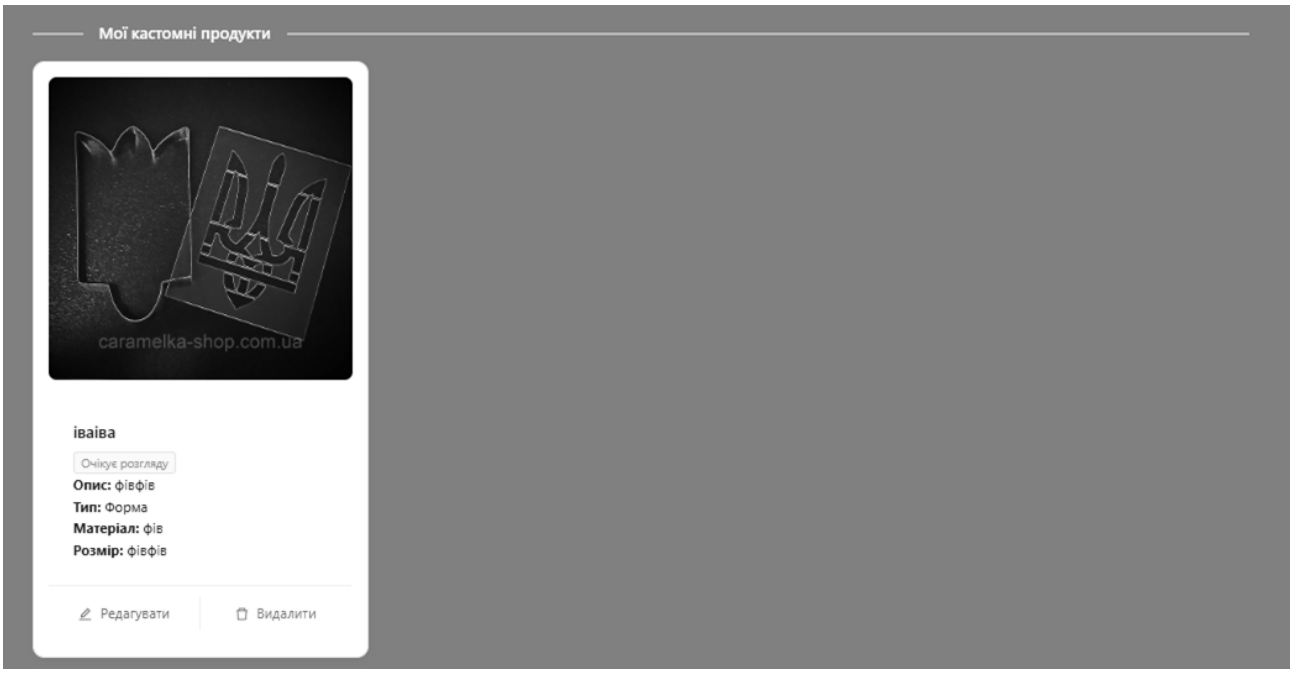

Рисунок 3.18 — Сторінка профілю користувача, секція «Мої кастомні продукти»

На рисунку 3.19 зображена сторінка каталогу інтернет-магазину, на якій користувачі можуть переглядати доступні трафарети та формочки для випікання. У центральній частині відображаються товари у вигляді окремих карток. Кожна картка містить зображення продукту, його назву, ціну, короткий опис, вказівку на матеріал, тему та розмір. Також передбачено можливість змінювати кількість одиниць товару за допомогою кнопок "+" і "–", після чого можна додати вибране до кошика натисканням на кнопку "Add to Cart". Внизу сторінки розміщена пагінація для перемикання між сторінками товарів. Лістинг головної сторінки продуктів наведений у додатку Д, лістингу Д.11.

Зліва розташований сайдбар з фільтрами. Він поділений на категорії "Трафарет" і "Формочка", кожна з яких містить підкатегорії з тематиками та зазначенням кількості товарів у дужках. Також присутня кнопка "Виріб на замовлення", яка веде на форму для створення індивідуального продукту. Нижче знаходиться фільтр за ціною у вигляді слайдера та кнопка "Застосувати" для підтвердження обраних параметрів.

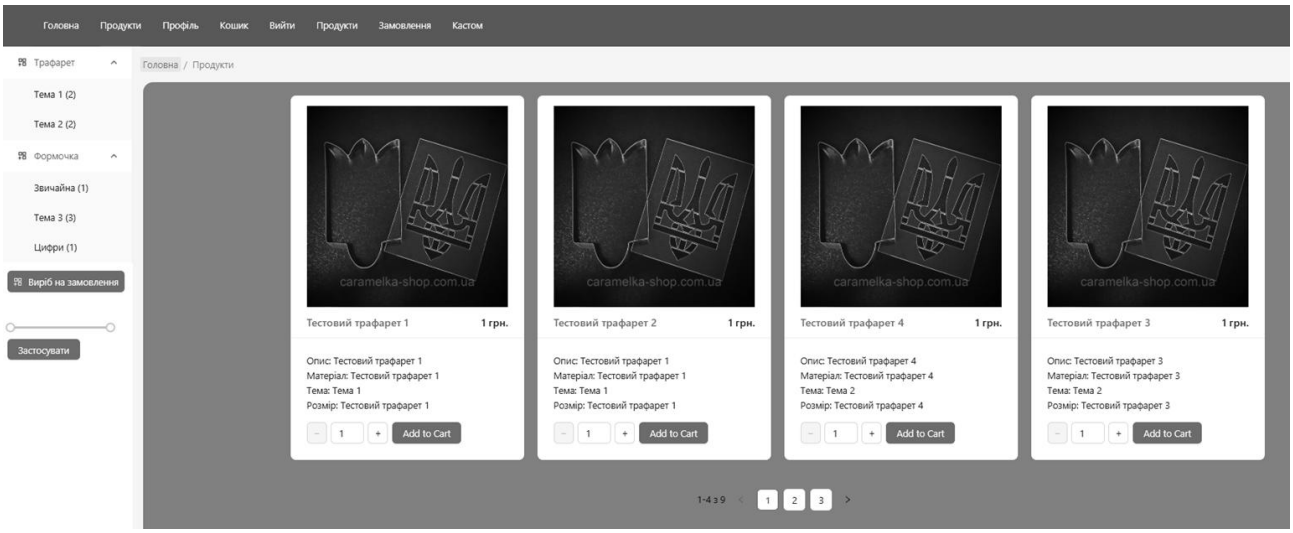

Рисунок 3.19 — Головна сторінка продуктів

На рисунку 3.20 зображена сторінка з переліком продуктів за їх типом. Продукти представлені у вигляді окремих карток з однаковим макетом. Кожна картка містить фотографію трафарету, назву, ціну, а також коротку інформацію: опис, матеріал, тему і розмір. У нижній частині кожної картки є кнопки для регулювання кількості товару та синя кнопка "Add to Cart" для додавання обраного трафарету до кошика. Лістинг сторінки продуктів за їх типом наведений у додатку Д, лістингу Д.12.

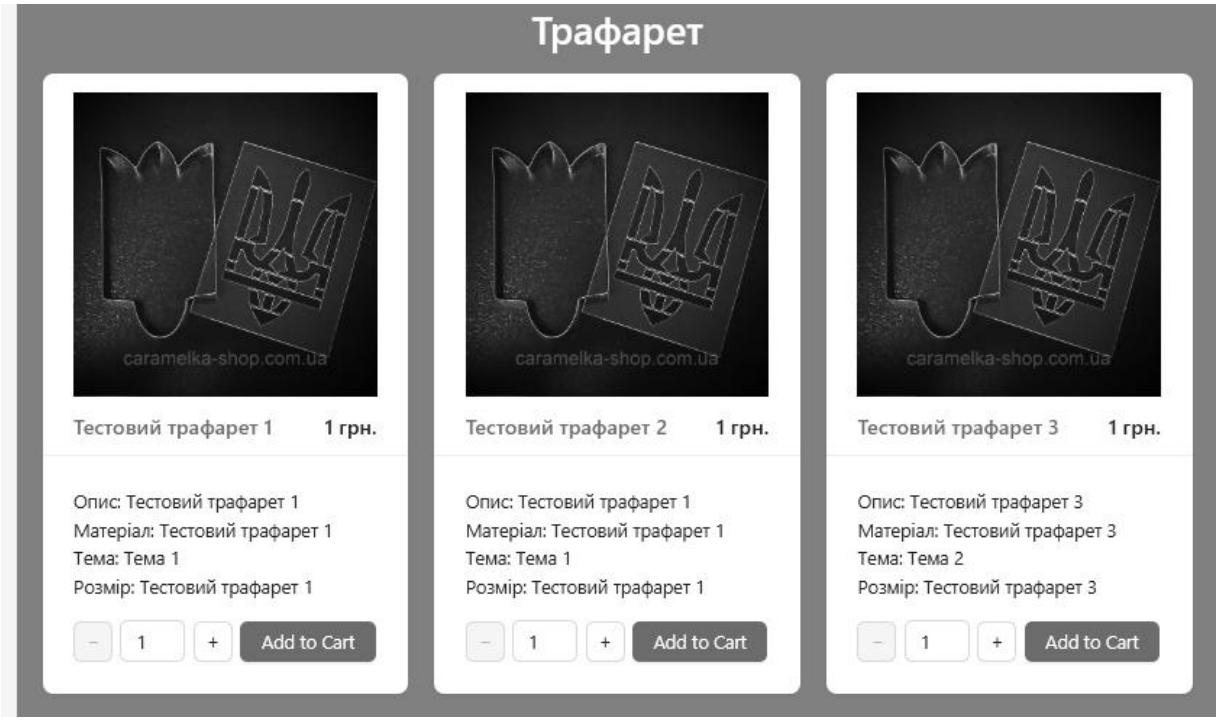

Рисунок 3.20 — Сторінка продуктів за їх типом

На рисунку 3.21 зображена сторінка відображення формочок, відфільтрованих за темою. Товарні картки на сторінці містять зображення продукту, його назву, ціну, а також додаткову інформацію: опис, матеріал, тему та розмір. Як і на інших сторінках, присутні елементи керування кількістю товару та синя кнопка "Add to Cart", яка дозволяє додати обраний виріб до кошика. Лістинг сторінки продуктів за темою наведений у додатку Д, лістингу Д.13.

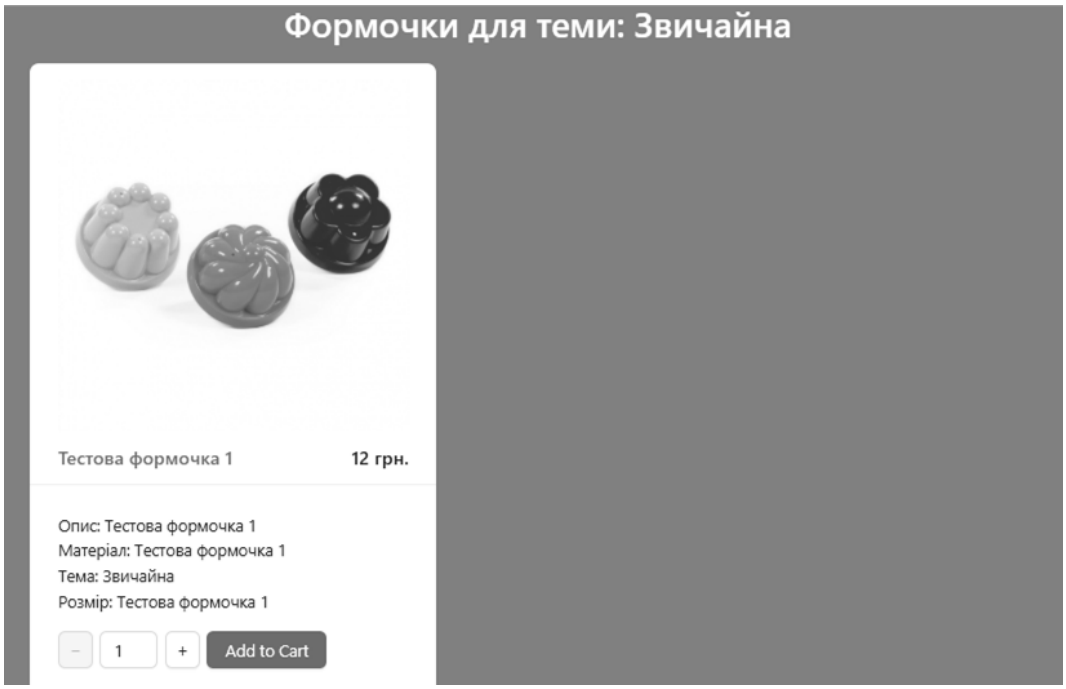

Рисунок 3.21 — Сторінка продукту за темою

На рисунку 3.22 зображена сторінка детального перегляду окремого продукту. У верхній частині присутній навігаційний ланцюжок, який вказує шлях користувача.

Сторінка поділена на дві частини: зліва — велике зображення продукту, справа — детальна інформація про товар. Тут вказано назву продукту, тему, ціну, матеріал, розмір і короткий опис — усі поля мають демонстраційні значення.

Нижче розміщені кнопки зміни кількості товару (– / +) і синя кнопка "Додати в кошик", яка дозволяє користувачеві оформити покупку прямо зі сторінки. Лістинг головної сторінки продукту наведений у додатку Д, лістингу Д.14.

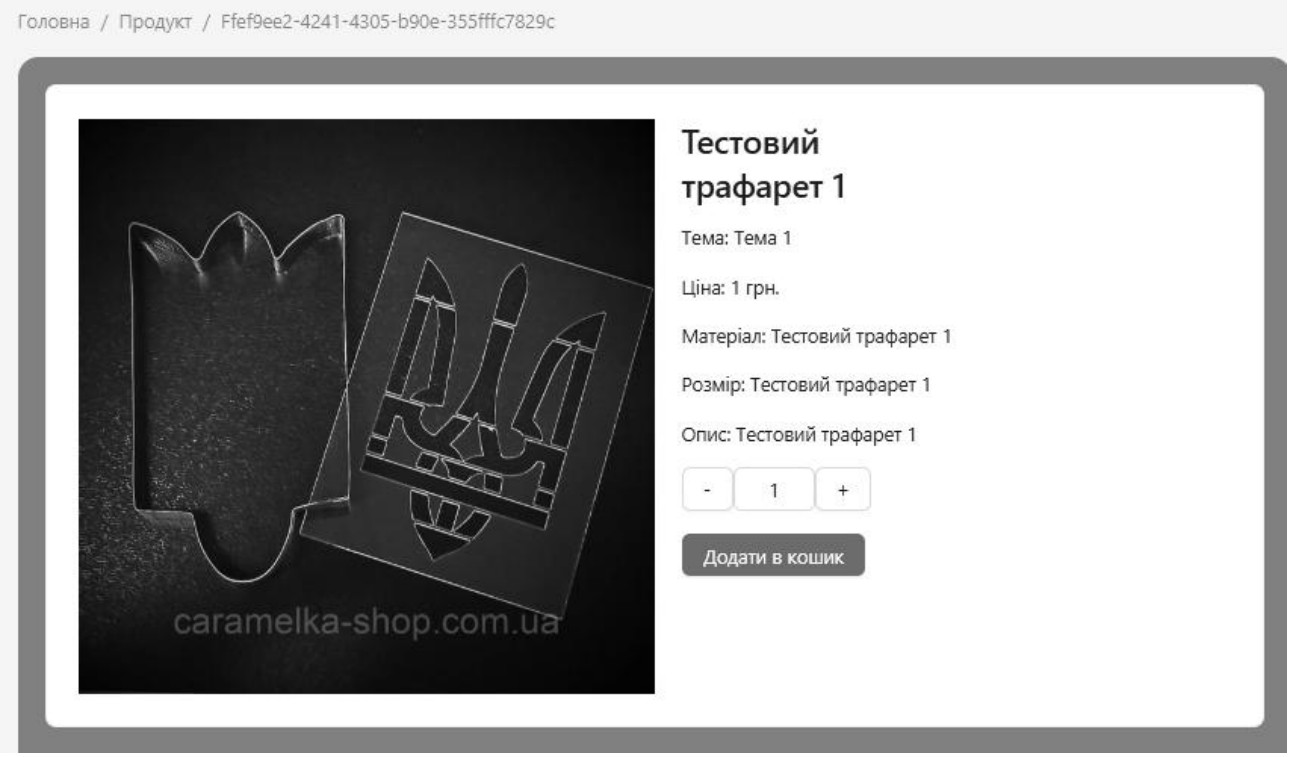

Рисунок 3.22 — Головна сторінка продукту

На рисунку 3.23 зображена сторінка створення кастомного виробу, яка дозволяє користувачеві надіслати запит на виготовлення індивідуального продукту. У верхній частині розміщено заголовок "Створити кастомний виріб" і коротка інструкція: “Заповни поля нижче, щоб надіслати запит на створення власного виробу.”

Форма складається з кількох обов’язкових полів:

— назва виробу — текстове поле з підказкою;

— тип продукту — випадаючий список для вибору або додавання типу;

— зображення — кнопка для завантаження файлу із візуалізацією виробу;

— опис — велике текстове поле для введення короткого опису виробу;

— розмір (мм) — поле з прикладом формату розмірів (наприклад, 100x80x10) ;

— матеріал — поле для введення бажаного матеріалу (pla, petg, силікон тощо).

У нижній частині розташована синя кнопка “Надіслати запит”, яка запускає обробку заявки. Лістинг сторінки створення запиту на індивідуальний виріб наведений у додатку Д, лістингу Д.15.

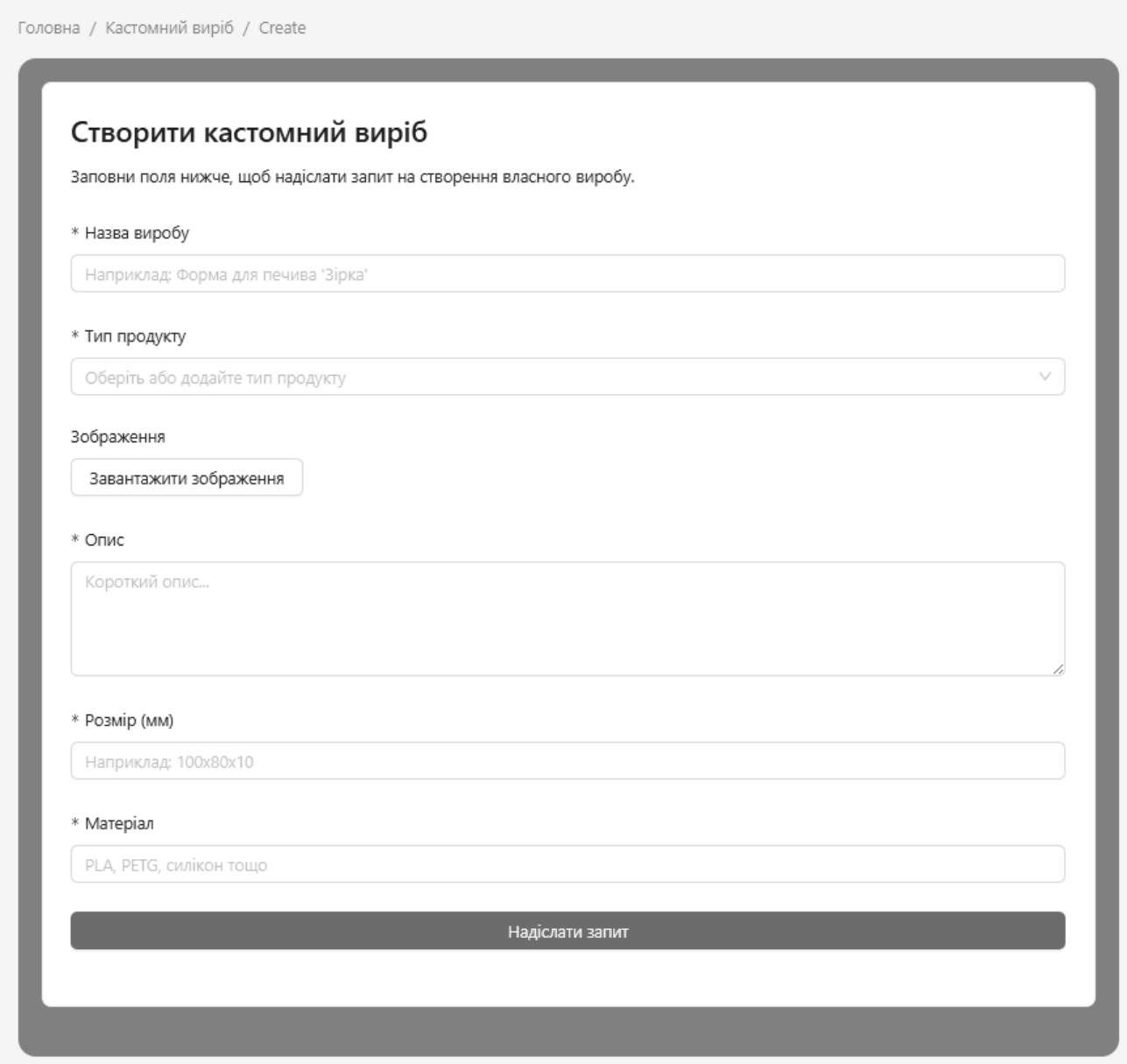

Рисунок 3.23 — Сторінка створення запиту на кастомний виріб

У додатку Г, на рисунку Г.1, зображений кошик у вигляді бічної панелі, яка відкривається поверх сторінки товарів і дозволяє користувачеві швидко переглянути вміст свого замовлення. У верхній частині відображається назва товару, його ціна, обрана кількість та підсумкова сума за позицію. Користувач може змінити кількість за допомогою кнопок “+” та “–” або видалити товар із кошика натисканням на червону кнопку “Видалити”. У нижній частині панелі підраховується загальна сума покупки, а також розміщена синя кнопка “Оформити замовлення” для переходу до фінального етапу покупки.

На рисунку 3.24 зображене модальне вікно оформлення замовлення, яке відкривається після додавання товару до кошика. У верхній частині виводиться

картка обраного продукту — зображення, назва, ціна за одиницю і підсумкова сума. Нижче розташована форма, яку користувач повинен заповнити для здійснення доставки. Форма включає обов’язкові поля: адреса, місто, поштовий індекс та країна. Також є поле вибору методу оплати, у цьому випадку обрано LiqPay. У нижній частині модального вікна знаходиться синя кнопка "Place order", яка підтверджує замовлення. Лістинг модального вікна оформлення замовлення наведений у додатку Д, лістингу Д.16.

Рисунок 3.24 — Модальне вікно оформлення замовлення

На рисунку 3.25 показано інтерфейс оплати через сервіс LiqPay. Зліва розміщений QR-код, який можна відсканувати через додаток Privat24 для швидкої оплати. Праворуч зверху наведені дані про замовлення: унікальний номер та сума до сплати — 1.00 гривня. Нижче розташовані кнопки для вибору способу оплати, зокрема LiqPay, Google Pay і банківська картка Visa. Користувач може обрати метод оплати — картка, Privat24 або рахунок — і ввести відповідні платіжні дані, такі як номер картки, термін дії і CVV2. Є також опція надіслати квитанцію на електронну пошту. Під формою знаходиться повідомлення про погодження з умовами публічного договору. Кнопка «Оплатити» активується після введення валідних даних і дозволяє завершити платіж. Такий інтерфейс генерується LiqPay автоматично після створення платіжної сесії і забезпечує зручний і безпечний спосіб оплати різними методами. Після успішної оплати статус замовлення змінюється на оплачене.

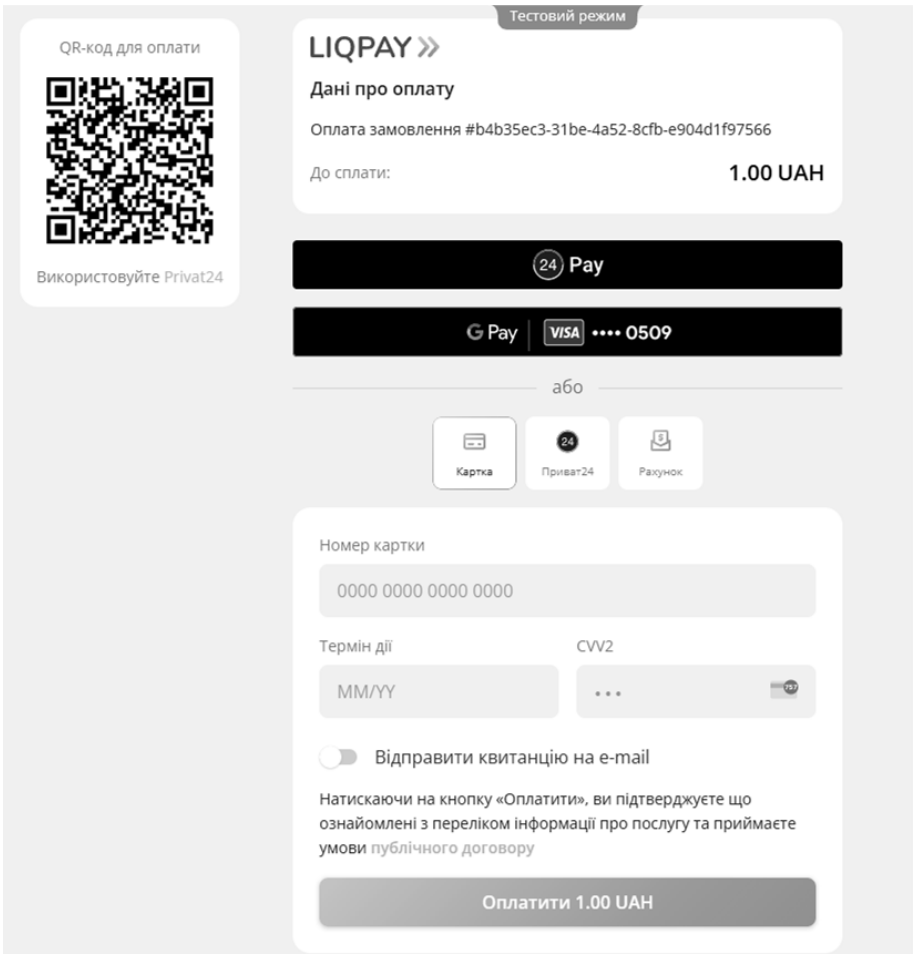

Рисунок 3.25 — Модальне вікно оформлення замовлення

На рисунку 3.26 відображено сторінку підтвердження успішної оплати через сервіс LiqPay у тестовому режимі. Вгорі знаходиться іконка галочки, яка символізує, що платіж пройшов успішно. Під нею розміщено текст подяки — «Дякуємо! Платіж успішно відправлено». Нижче наведена інформація про платіж: унікальний ідентифікатор (ID) транзакції, дата і час проведення операції, а також сума — 1.00 гривня. Кнопка для завантаження квитанції наразі недоступна (неактивна). Внизу знаходиться посилання або кнопка для повернення на сайт, що дозволяє користувачу завершити процес оплати і повернутися до інтерфейсу магазину.

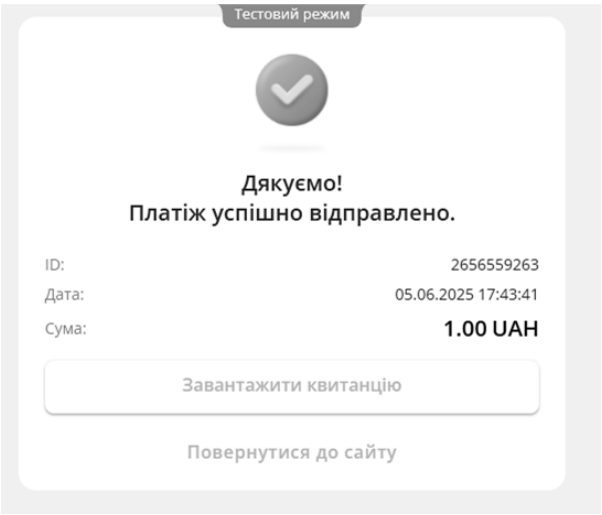

Рисунок 3.26 — Модальне вікно оформлення замовлення

Були реалізовані окремі інструменти для адмінів, які відображаються у навігаційному меню зверху якщо користувач має роль адміну.

На рисунку 3.27 зображений інструмент адміністраторів з переліком усіх наявних продуктів у системі. У верхній частині сторінки розташована кнопка “Add”, яка дозволяє адміністратору додати новий товар. Нижче представлений список наявних продуктів. Кожна картка містить зображення товару, його назву, інформацію про тип, тему, матеріал та розмір. Праворуч зазначено ціну і розміщені посилання edit та delete. Лістинг інструменту адміністратора для керування продуктами наведений у додатку Д, лістингу Д.17.

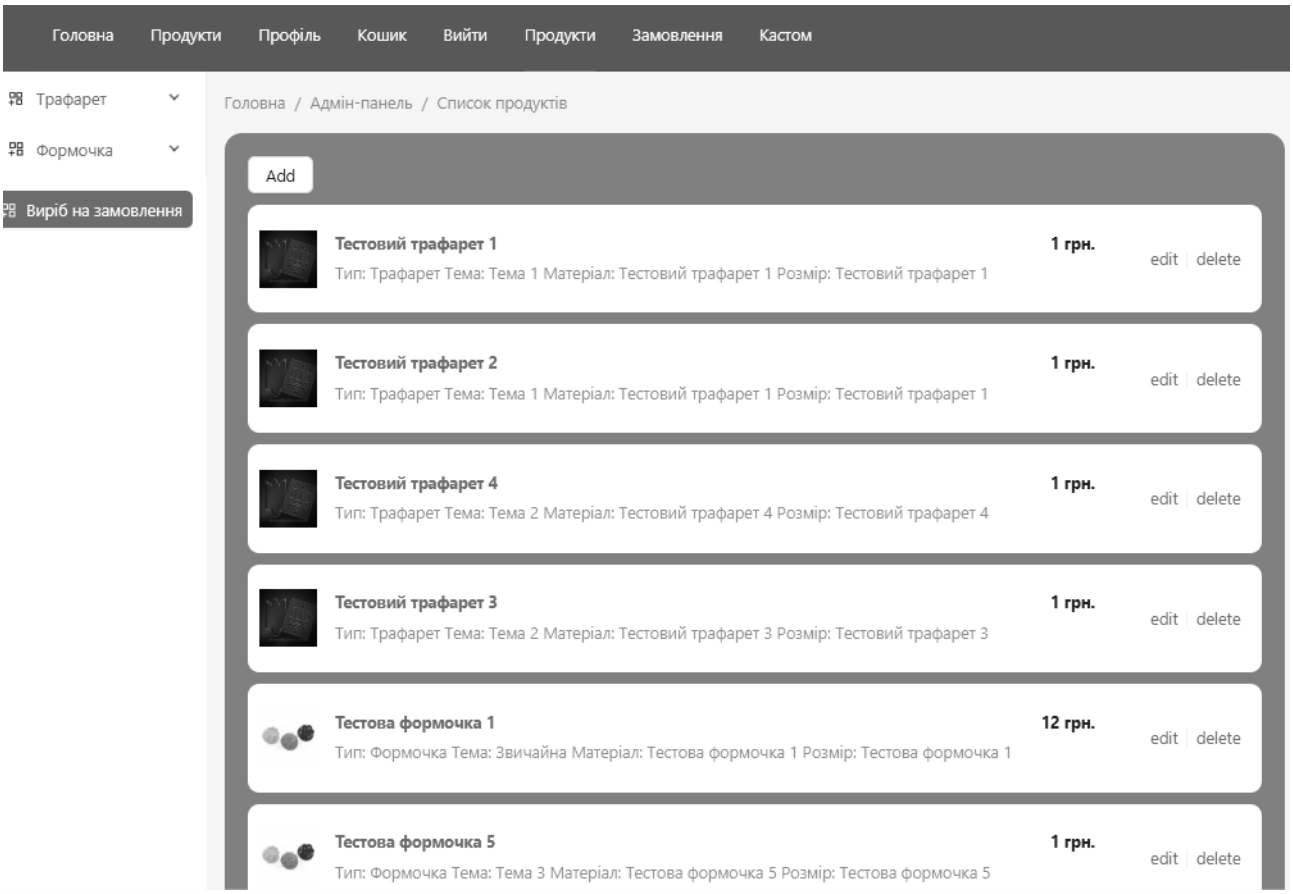

Рисунок 3.27 — Інструмент адміністратора для керування продуктами

На рисунку 3.28 зображена сторінка адмін-панелі з переглядом замовлення, яка дає адміністратору змогу бачити деталі конкретного замовлення користувача. У верхній частині відображено ID користувача, який оформив замовлення, а також унікальний номер самого замовлення.

Під ним вказано основну інформацію: дата оформлення, сума і адреса доставки. Нижче відображено перелік товарів у замовленні.

Праворуч знаходиться випадаючий список для зміни статусу замовлення і червоне посилання "Видалити", яке дозволяє адміну повністю видалити це замовлення. Лістинг інструменту адміністратора для керування замовленнями наведений у додатку Д, лістингу Д.18.

Рисунок 3.28 — Інструмент адміністратора для керування замовленнями

На рисунку 3.29 зображена сторінка адмін-панелі зі списком кастомних виробів, яка дозволяє адміністратору переглядати та керувати індивідуальними запитами користувачів. У відображеній картці показано назву кастомного виробу, зображення, тип продукту, матеріал та розмір. Справа розташований випадаючий список для зміни статусу виробу, кнопка "Видалити" для видалення запиту та кнопка "Опублікувати", яка робить кастомний виріб доступним для продажу. Лістинг інструменту адміністратора для керування кастоними виробами наведений у додатку Д, лістингу Д.19.

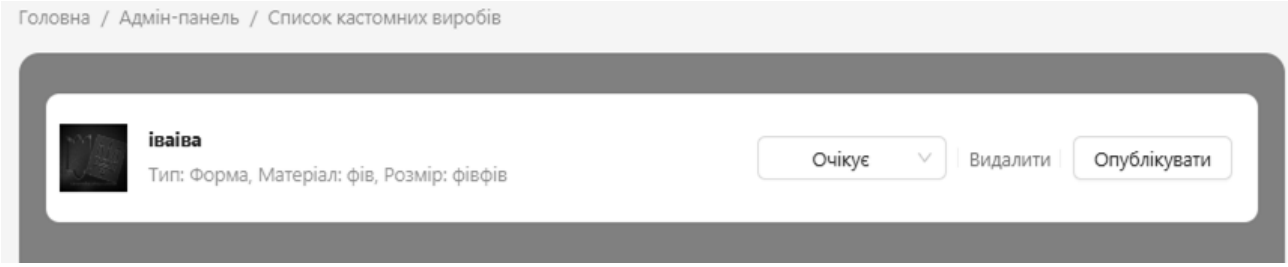

Рисунок 3.29 — Інструмент адміністратора для керування кастоними виробами

У додатку Г, на рисунку Г.2 зображено модальне вікно редагування продукту в адміністративній панелі. У верхній частині вікна відображено зображення товару, який редагується.

Форма редагування містить такі обов’язкові поля:

— назва продукту;

— тип продукту (випадаючий список) ;

— зображення — кнопка для завантаження нового файлу;

— матеріал;

— тема;

— розмір;

— опис;

— ціна.

У нижній частині розташовані дві кнопки: "Cancel" для скасування змін і "OK" для збереження оновлень. Лістинг модального вікна адміністратора для редагування продукту наведений у додатку Д, лістингу Д.20.

3.5. Модульне тестування компонентів серверної частини

Модульне тестування є важливою частиною процесу розробки програмного забезпечення, оскільки дозволяє перевірити коректність роботи окремих компонентів системи в ізольованому середовищі. Основна мета модульного тестування — виявлення помилок на ранніх етапах розробки, підвищення надійності коду та спрощення його підтримки в майбутньому.

Основними етапами процесу модульного тестування є Arrange, Act, Assert.

Arrange, Act, Assert (або AAA) — це поширений термін, що описує структуру юніт-тесту: підготовку середовища, виконання тесту та перевірку результату. Це вважається найкращою практикою у модульному тестуванні. Кожен юніт-тест, як правило, має складатися з трьох частин:

— arrange (підготовка): налаштування необхідних об’єктів і стану;

— act (дія): виконання методу, який тестується;

— assert (перевірка): перевірка результатів і очікувань.

У межах даної бакалаврської роботи було проведено модульне тестування основних компонентів серверної частини: мапінгів (AutoMapper), репозиторіїв, сервісів та контролерів, моделей бізнес-логіки.

Модульні тести для моделей бізнес є особливо важливими, адже саме ці моделі містять ключові правила, які визначають допустиму поведінку об'єктів у системі. Тестування таких моделей дозволяє переконатись, що створення об'єктів відбувається лише з валідними даними, всі обмеження дотримуються, а зміна стану (наприклад, статусу) працює коректно. Це допомагає вчасно виявити

помилки у бізнес-правилах, забезпечує передбачуваність логіки та захищає систему від некоректного використання об’єктів ззовні. Результати тестування моделей бізнес-логіки наведені на рисунку 3.28.

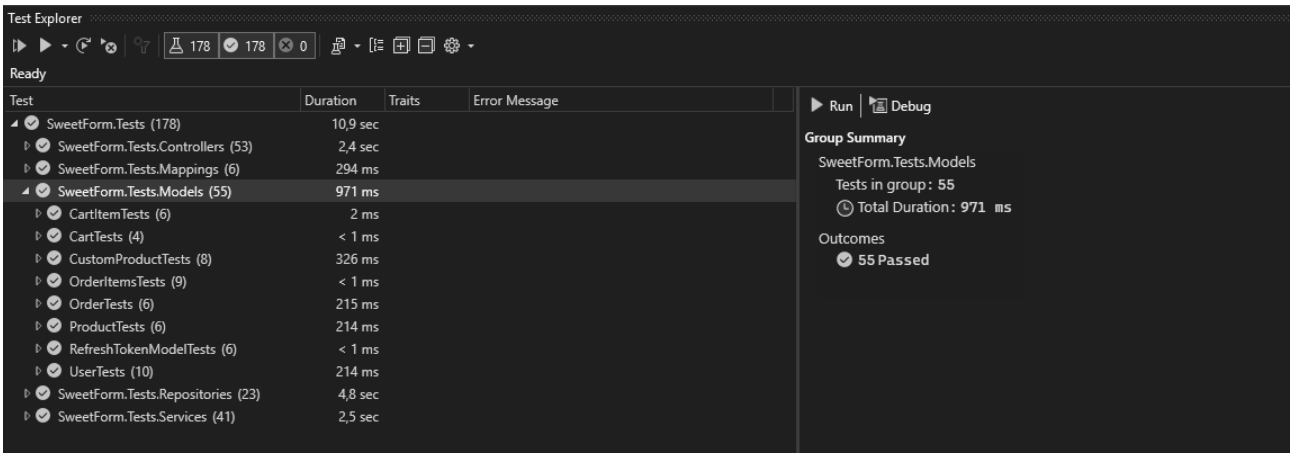

Рисунок 3.28 — Результат тестування моделей бізнес-логіки

Для забезпечення коректного перетворення даних між різними моделями було налаштовано та протестовано мапінги з використанням бібліотеки AutoMapper. Модульні тести перевіряють правильність конфігурації мапера, а також відповідність значень після перетворення. Це дозволяє уникнути логічних помилок при передачі даних між шарами застосунку. Результати тестування мапінгів наведені на рисунку 3.29.

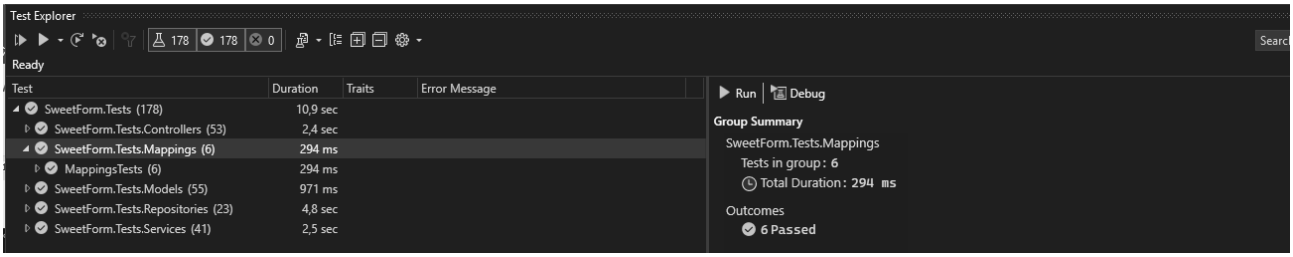

Рисунок 3.29 — Результат тестування мапінгів

Репозиторії є посередниками між бізнес-логікою і базою даних, тому їхня стабільна робота є критично важливою. Для модульного тестування використовувалися мок-об’єкти бази даних, що дає змогу тестувати методи збереження, оновлення, видалення та отримання даних без реального

підключення до бази. Тести охоплюють як позитивні сценарії, так і обробку помилок.

Мокінг — це процес, який дозволяє створити мок-об’єкт (підроблений об’єкт), що може імітувати поведінку справжнього об’єкта. Мок-об’єкт можна використовувати для перевірки того, чи був справжній об’єкт викликаний з очікуваними параметрами, а також для перевірки, що об’єкт не був викликаний з неочікуваними параметрами. Крім того, можна перевірити, скільки разів об’єкт був викликаний. Таким чином, ви можете впевнитися, що виклики відбуваються саме так, як потрібно [15]. Результати тестування репозиторіїв наведені на рисунку 3.30.

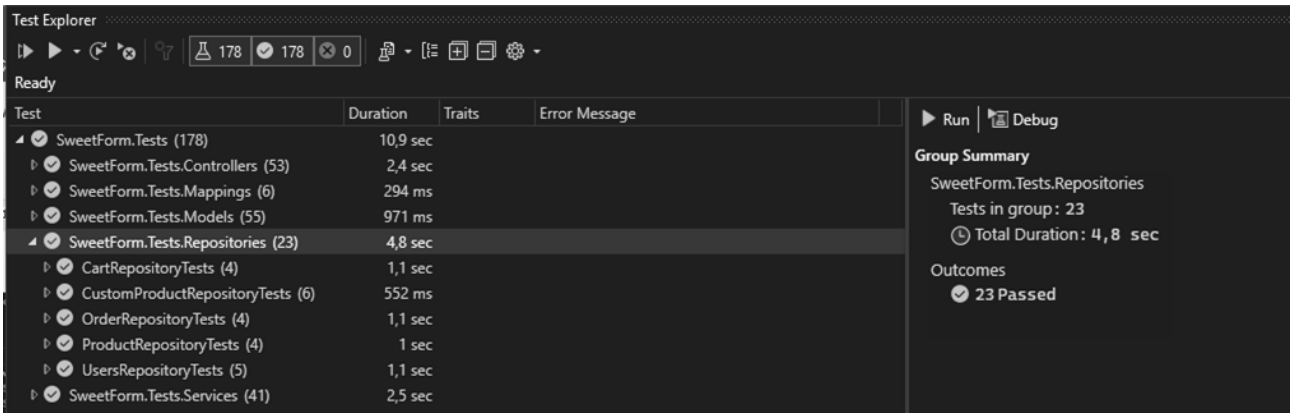

Рисунок 3.30 — Результат тестування репозиторіїв

Сервіси реалізують бізнес-логіку системи. У тестах перевіряється, як саме сервіси обробляють вхідні дані, взаємодіють із репозиторіями, та повертають результат. Для ізоляції сервісів від зовнішніх залежностей активно застосовувалися моки. Це дозволяє зосередитись виключно на логіці всередині сервісу, не враховуючи вплив зовнішніх компонентів. Результати тестування сервісів наведені на рисунку 3.31.

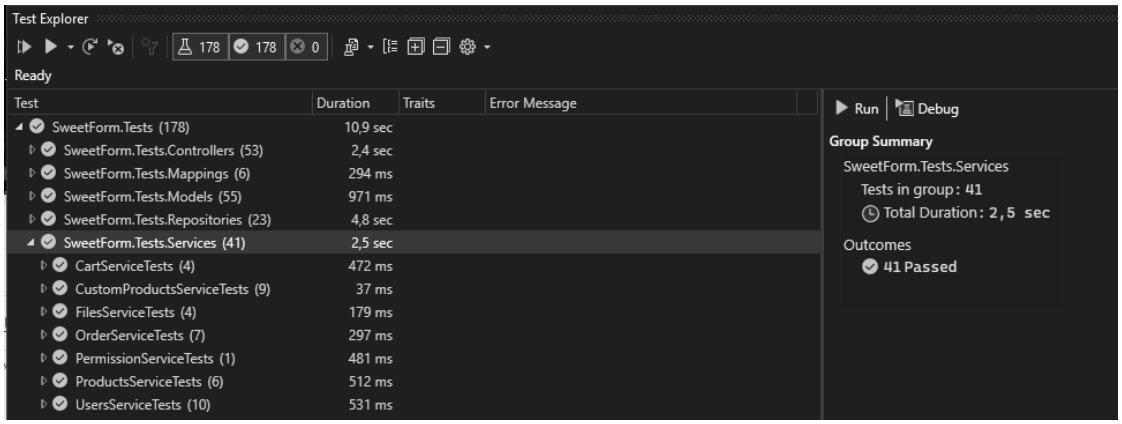

Рисунок 3.31 — Результат тестування сервісів

Контролери є точкою входу в API, тому важливо впевнитися, що вони коректно обробляють HTTP-запити, викликають відповідні методи сервісів та формують коректні HTTP-відповіді. У модульних тестах перевіряється логіка маршрутизації, обробка параметрів, повернення відповідних кодів статусу та вмісту відповіді. Результати тестування контролерів наведені на рисунку 3.32.

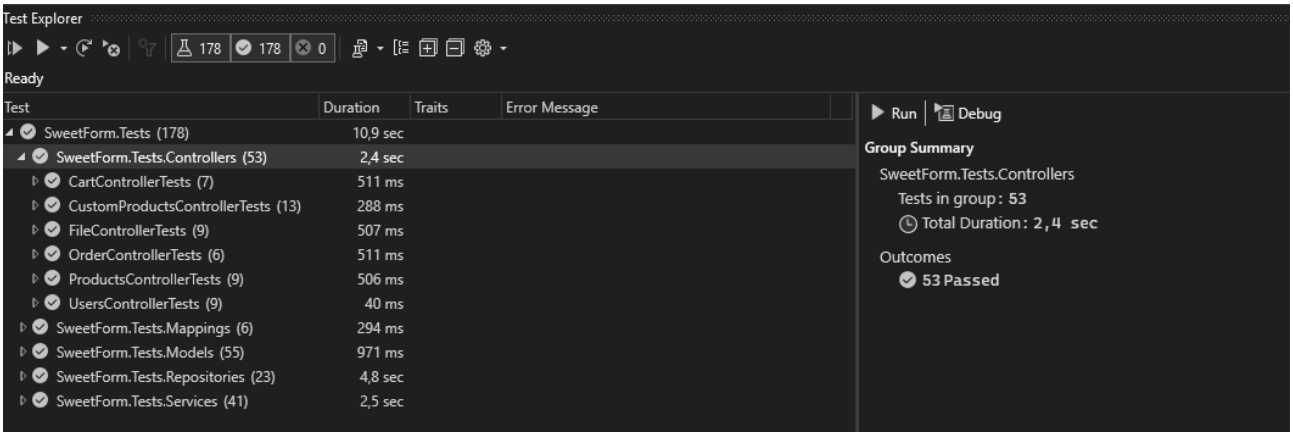

Рисунок 3.32 — Результат тестування контролерів

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було повністю реалізовано інформаційну систему обліку виробів з пластику, яка охоплює усі ключові етапи розробки сучасного веб-застосунку — від аналізу предметної області до програмної реалізації та тестування.

Розглянута інформація в межах першого розділу свідчить про необхідність розробки інформаційної системи для ведення обліку пластикової продукції з урахуванням сучасних IT-рішень. Обрана сукупність технологій — React, ASP.NET Core Web API, PostgreSQL та Docker — дозволяє забезпечити стабільну роботу системи, її адаптивність до змін та високу продуктивність. Такий технологічний підхід сприяє автоматизації облікових процесів, зменшенню ймовірності помилок у роботі з даними та створює комфортні умови для взаємодії як адміністраторів, так і звичайних користувачів. Для зберігання зображень, файлів та мультимедійних даних передбачено використання хмарного сховища Amazon S3 Bucket, що гарантує швидкий доступ, масштабованість і високу надійність. Для забезпечення безпечної та зручної організації онлайн-платежів у системі інтегровано сервіс LiqPay, який підтримує різноманітні методи оплати, включаючи банківські картки та мобільні гаманці, і забезпечує надійний захист платіжних даних. Упровадження подібної системи має потенціал значно покращити ефективність управлінських процесів на підприємстві, зміцнити комунікацію з клієнтами та сприяти цифровому розвитку малого бізнесу.

Другий розділ був присвячений архітектурному та структурному проєктуванню. У результаті аналізу та проєктування системи обліку виробів з пластику було обґрунтовано доцільність використання клієнт-серверної архітектури з технологіями React.js, ASP.NET Core Web API, PostgreSQL і Docker. Такий підхід забезпечує розмежування функцій, зручність супроводу, масштабованість і надійність. Застосування шаблону MVC дозволило логічно структурувати серверну частину, що полегшує підтримку та розширення функціоналу. Структура бази даних охоплює ключові сутності системи, включаючи користувачів, ролі, товари, замовлення та інші компоненти, з чітко

визначеними зв’язками, що забезпечує цілісність даних. UML-діаграми прецедентів і послідовностей візуалізували функціональність і бізнес-процеси, створивши базу для коректної реалізації вимог. Усі архітектурні та технічні рішення спрямовані на створення гнучкого, стабільного та ефективного веб-застосунку.

У третьому розділі представлено програмну реалізацію інформаційної системи. Було створено REST API із документацією в Swagger, що дозволило забезпечити зручність тестування, розширюваність і підтримку системи. Реалізовано окремі модулі для роботи з товарами, кастомними виробами, кошиком, замовленнями, авторизацією, профілем користувача та управлінням файлами. Для надсилання автоматичних електронних листів (наприклад, підтвердження реєстрації, відновлення паролю, повідомлення про зміну статусу замовлення тощо) був реалізований окремий сервіс повідомлень із підтримкою HTML-шаблонів і повною інтеграцією в бізнес-процеси системи. Користувацький інтерфейс розроблено з використанням React та бібліотеки Ant Design, із чітким поділом функціоналу для клієнтів і адміністраторів. Інтерфейс забезпечує зручну навігацію, можливість перегляду товарів, створення замовлень, кастомізації виробів і керування профілем.

Було проведено модульне тестування основних частин системи: репозиторіїв, сервісів, доменних моделей і поштового сервісу. Тестування підтвердило правильність реалованих бізнес-процесів, стабільність системи та її відповідність функціональним вимогам. У результаті створено повноцінну веб-систему, готову до реального використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційна система обліку виробів з пластику. [Електронний ресурс] / Кисюк Д. В., Турко Г. А. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)», Вінниця, 16 червня 2025р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25272.

2. Проблеми та потреби українських підприємств малого та середнього бізнесу в умовах війни. URL: https://dif.org.ua/article/problemi-ta-potrebi-ukrainskikh-pidpriemstv-malogo-ta-serednogo-biznesu-v-umovakh-viyni (дата звернення: 17.03.2024).

3. Інтернет-магазин cakeshop.com.ua. URL: https://cakeshop.com.ua/ua/ (дата звернення: 22.03.2024).

4. Інтернет-магазин sweet4ucutters. URL: https://sweet4ucutters.com/?srsltid=AfmBOooPjdWv564bPb4ftGRu3OgMrPo8NDVL7DCGA7ErOvJ3RO7CFZed (дата звернення: 22.03.2024).

5. Інтернет-магазин formochki.com.ua. URL: https://ua.formochki.com.ua/k/formyi-dlya-vyipechki_c2051/plastik_c3838/nerzhaveyuschaya-stal_c3810 (дата звернення: 26.03.2024).

6. React. A JavaScript library for building user interfaces. URL: https://legacy.reactjs.org/ (дата звернення: 04.04.2024).

7. Ant Design 5.0. URL: https://ant.design/ (дата звернення: 04.04.2024).

8. APIs with ASP.NET Core. Build secure REST APIs on any platform with C#. URL https://dotnet.microsoft.com/en-us/apps/aspnet/apis (дата звернення: 04.04.2024).

9. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL https://www.postgresql.org/ (дата звернення: 10.04.2024).

10. What is Amazon S3?. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (дата звернення: 15.04.2024).

11. Learn to build and deploy your distributed applications easily to the cloud with Docker. URL: https://docker-curriculum.com/ (дата звернення: 15.04.2024).

12. MVC Design Pattern. URL: https://medium.com/@sandesh__30_/mvc-design-pattern-ff40d66990e3 (дата звернення: 15.04.2024).

13. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 26.04.2024).

14. Вступ до REST API — RESTful вебсервіси. URL: https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi (дата звернення: 26.04.2024).

15. How To Simplify Your C# Unit Testing With a Mocking Framework. URL: https://www.telerik.com/blogs/how-to-simplify-your-csharp-unit-testing-mocking-framework (дата звернення: 05.05.2024).

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., Азаров О.Д.
21 березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи
«Інформаційна система обліку виробів з пластику»

08-54.БКР.010.00.000 ТЗ

Науковий керівник: ст. викл. каф. ОТ
_____ Кисюк Д. В.
(прізвище та ініціали)
Виконала: студентка 2 (4) курсу, групи КІ-23мсз
_____ Турко Г. А.
(прізвище та ініціали)

м. Вінниця — 2025

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність обраної теми зумовлена стрімким розвитком цифрових технологій, що вимагає від підприємств адаптації до нових умов ведення бізнесу. У наш час наявність власного веб-ресурсу та інструментів для взаємодії з клієнтами є необхідною умовою ефективної комерційної діяльності. Зокрема, підприємства, що спеціалізуються на виготовленні виробів з пластику, потребують спеціалізованих рішень для обробки замовлень і взаємодії з клієнтами.

1.2 Наказ про затвердження теми БКР.

2 Мета БКР і призначення розробки

2.1 Метою бакалаврської кваліфікаційної роботи є створення інформаційної системи обліку виробів з пластику, яка забезпечить автоматизацію процесів замовлення, підвищить ефективність взаємодії з клієнтами та сприятиме цифровізації бізнесу.

2.2 Призначення розробки — забезпечення малого та середнього бізнесу зручним та ефективним веб-застосунком для ведення обліку пластикової продукції, що включає функціонал реєстрації користувачів, управління замовленнями та товарними позиціями, збереження історії взаємодії з клієнтами, а також можливість створення індивідуальних продуктів.

3 Вихідні дані для виконання БКР

3.1 Проведення аналізу сучасного стану програмних рішень для обліку продукції та потреб малого бізнесу в автоматизації процесів.

3.2 Розробка архітектури інформаційної системи з урахуванням клієнт-серверної моделі, використанням React, ASP.NET Core, PostgreSQL, Docker та Amazon S3.

3.3 Проєктування та реалізація основних функціональних модулів системи: облік товарів, управління замовленнями, авторизація користувачів, адміністрування, створення індивідуальних виробів.

3.4 Аналіз економічної доцільності впровадження системи для підприємств малого бізнесу у сфері виготовлення пластикової продукції.

4 Вимоги до виконання БКР

Головна вимога — забезпечити високу продуктивність, масштабованість та гнучкість системи, реалізувати модулі з урахуванням принципів безпеки та відповідно до сучасних стандартів веб-розробки. Особливу увагу приділено взаємодії між серверною й клієнтською частинами, коректному обробленню запитів та наданню зручного інтерфейсу для кінцевого користувача.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи БКР

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми БКР	03.09.2024	06.03.2025	
2.	Аналіз існуючих програмних рішень	16.03.2025	18.03.2025	
3.	Проєктування інформаціної системи обліку виробів з пластику	19.03.2025	27.03.2025	
4.	Програмна реалізація інформаціної системи обліку виробів з пластику	28.03.2025	26.04.2025	
5.	Тестування інформаціної системи обліку виробів з пластику	27.04.2025	05.05.2025	
6.	Оформлення додатків	06.05.2025	09.05.2025	
7.	Попередній захист БКР, доопрацювання, рецензування БКР	13.05.2025		

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія» (освітня програма «Комп’ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: _____ Інформаційна система обліку виробів з пластику_____

Тип роботи: _____ бакалаврська кваліфікаційна робота _____
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism __1,5___%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

____Крупельницький Л.В., доц. каф ОТ_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ ___Захарченко С.М.____
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____Кисюк Д. В. ст. викл. каф. ОТ_____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____Турко Г. А._____
(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

ІНФОРМАЦІЙНА СИСТЕМА ОБЛІКУ ВИРОБІВ З ПЛАСТИКУ

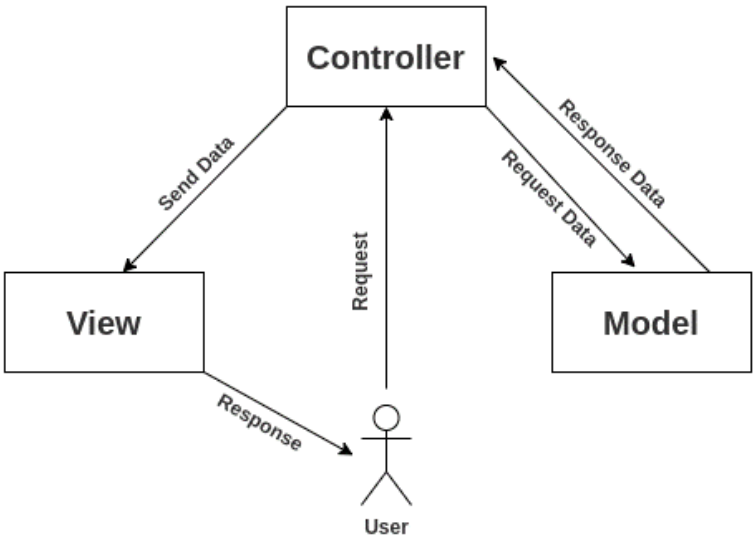

Рисунок В.1 — Схема взаємодії компонентів архітектури MVC

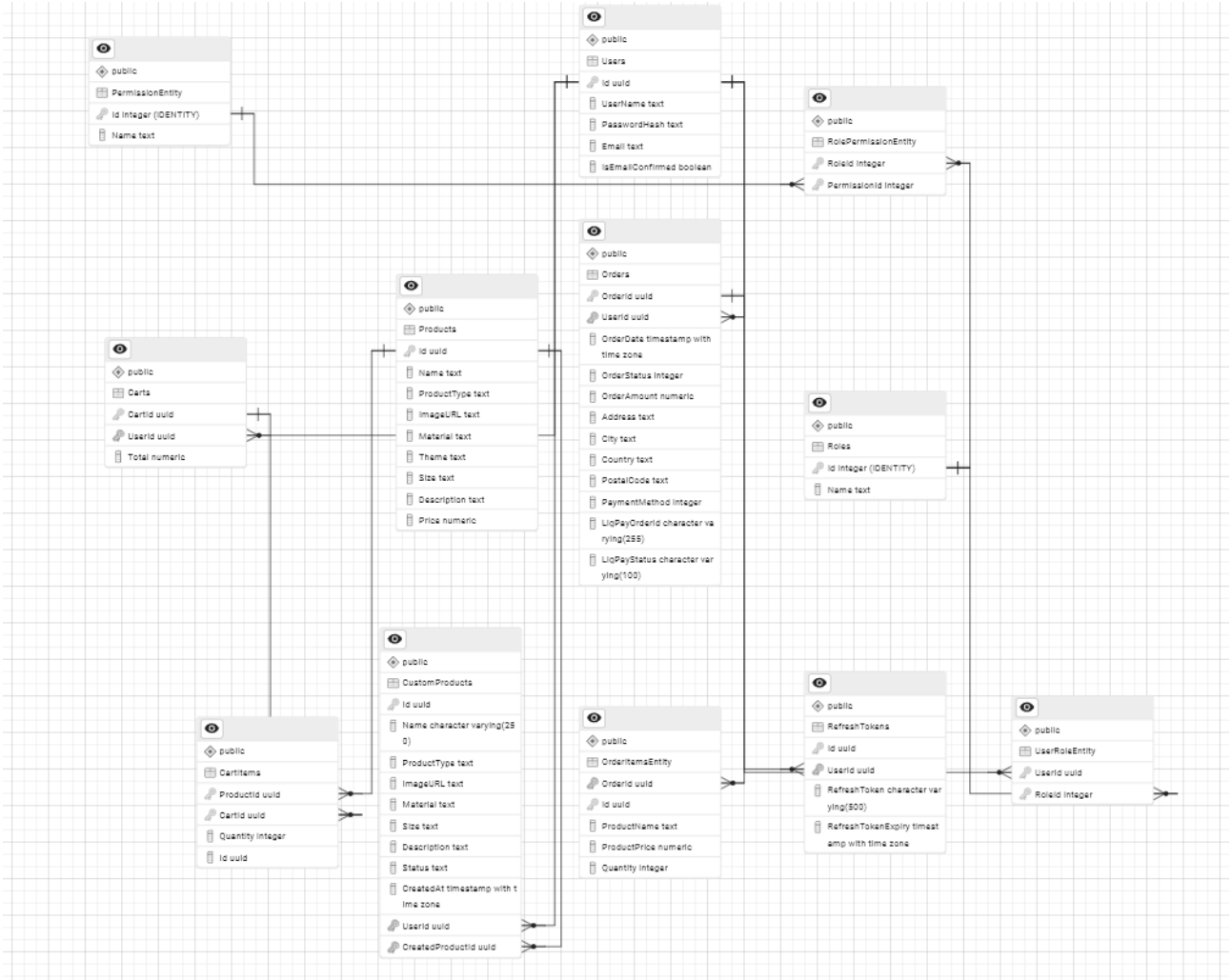

Рисунок В.2 — ER-діаграма бази даних системи обліку

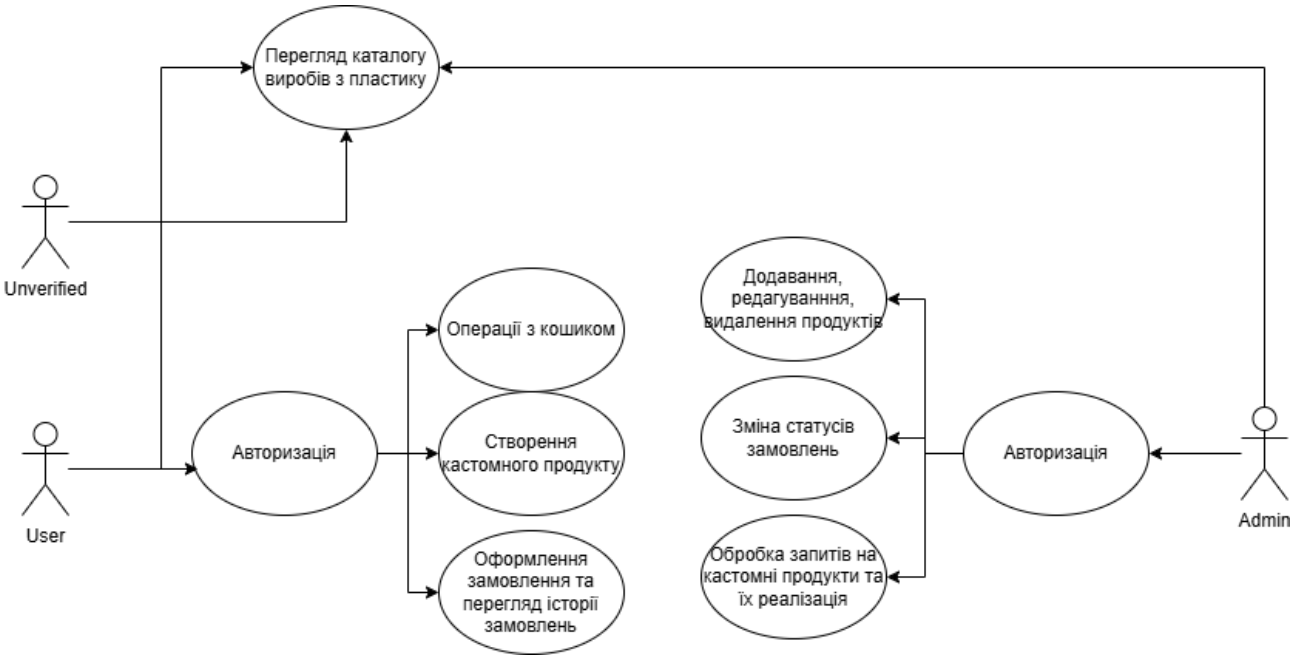

Рисунок В.3 — Діаграма прецедентів інформаційної системи обліку виробів з пластику

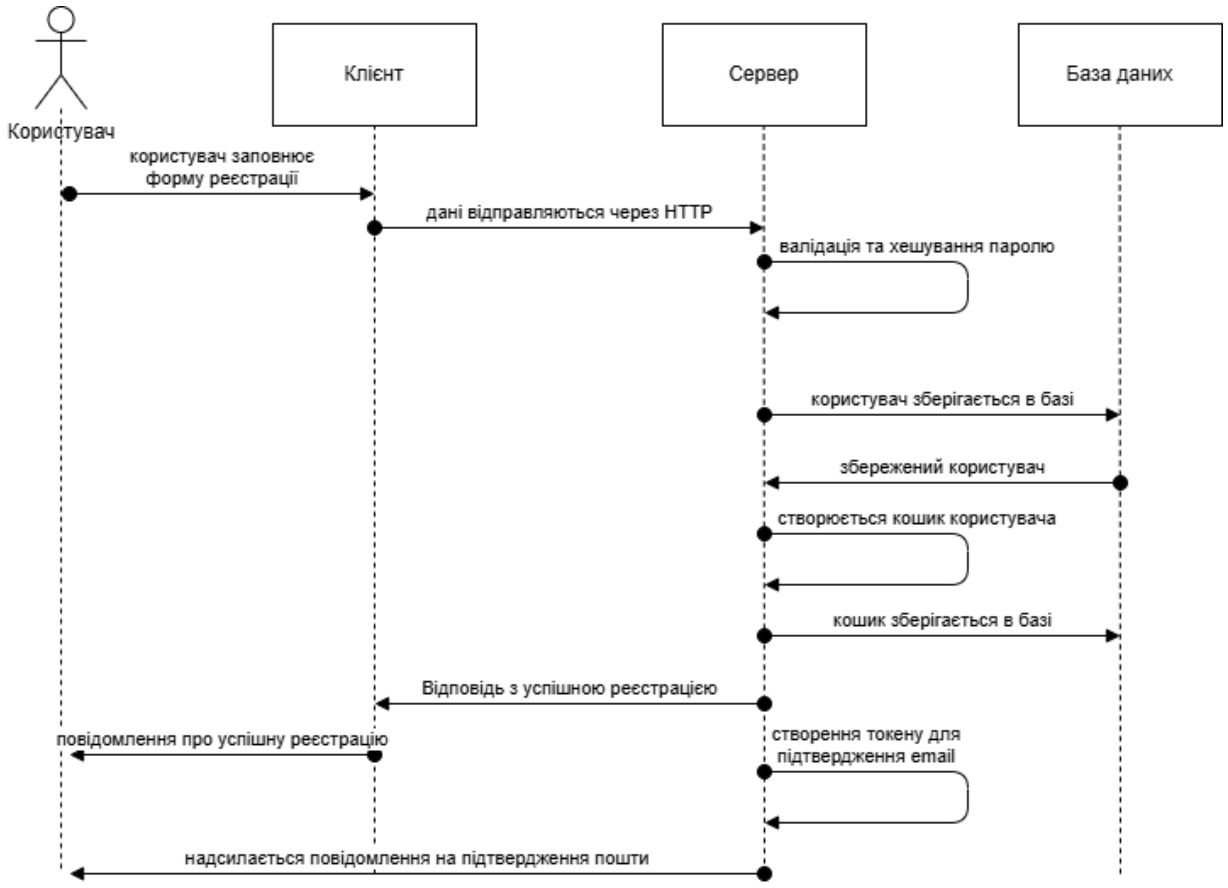

Рисунок В.4 — Діаграма послідовності процесу реєстрації користувача в системі

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА СИСТЕМА ОБЛІКУ ВИРОБІВ З ПЛАСТИКУ

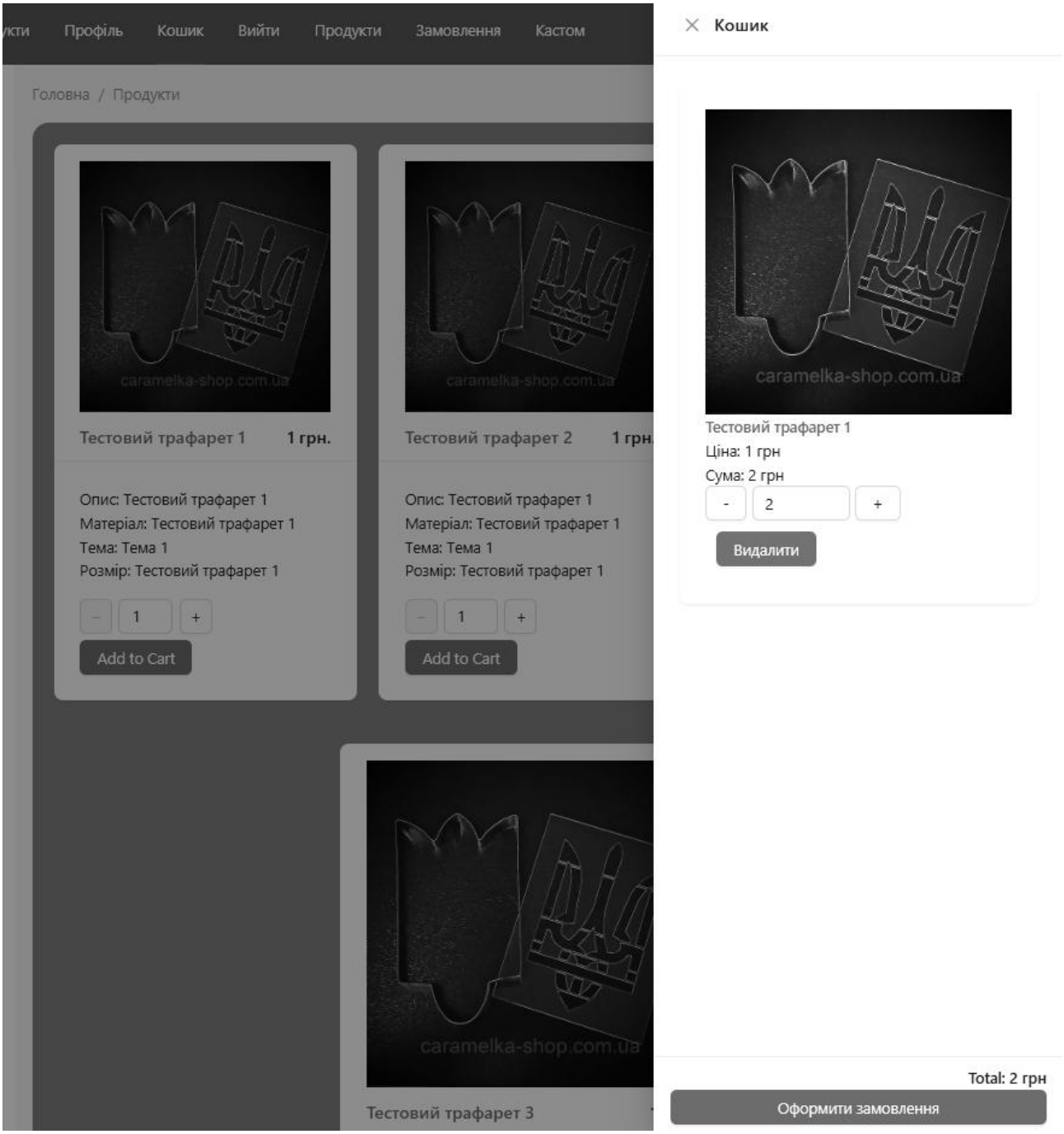

Рисунок Г.1 — Бічна панель кошику користувача

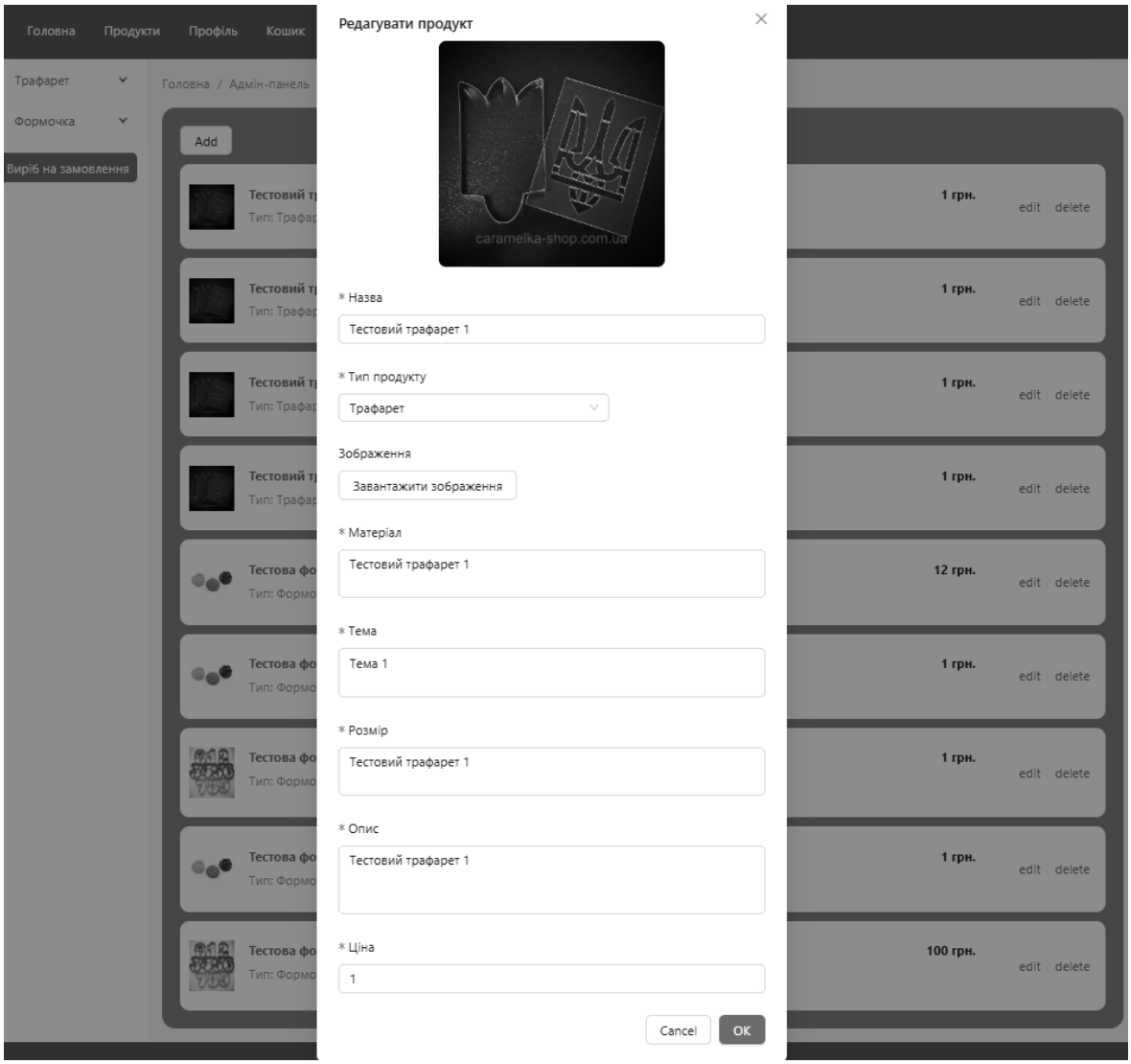

Рисунок Г.2 — Інструмент адміністратора для редагування продукту

АКТ

про впровадження результатів

комплексної кваліфікаційної роботи

на здобуття ступеня «бакалавр»

Турко Ганни Андріївни

На тему «Інформаційна система обліку виробів з пластику»

У бакалаврській кваліфікаційній роботі здобувачки освіти Турко Ганни Андріївни, виконаній на базі ТОВ «В.Д.Т.» під керівництвом директора Турко Андрія Олександровича, розроблено веб-платформу, що спрямована на оптимізацію ключових бізнес-процесів підприємства, зокрема формування та обробки замовлень, кастомізації продукції та управління товарним асортиментом.

У практичній частині роботи передбачається використання результатів дослідження для впровадження повноцінної інформаційної системи, призначеної для оптимізації бізнес-процесів підприємства, пов'язаних із прийомом і обробкою замовлень, кастомізацією виробів, а також управлінням товарним асортиментом. Отримані результати можуть бути застосовані для покращення взаємодії з клієнтами та масштабування цифрових рішень підприємства.

Наразі система проходить внутрішнє тестування в компанії та демонструє стабільну роботу. Заплановане впровадження веб-платформи в реальні бізнес-процеси ТОВ «В.Д.Т.» орієнтовно відбудеться у серпні–вересні 2025 року, після завершення всіх етапів тестування та доопрацювання функціоналу відповідно до потреб підприємства.

Директор
ТОВ «В.Д.Т.»

Товариство з обмеженою відповідальністю «ВАНДЕР ДЕВ ТЕЧ» 45139794 Україна, м.Вінниця

А. О. Турко

ДОДАТОК Д

Лістинг програмного забезпечення

Лістинг Д.1 — Контролер CartController

```
	namespace SweetForm.API.Controller
{
	[ApiController]
	[Route("api/cart")]
	[Authorize]
	public class CartController : ControllerBase
	{
		private readonly ICartService _cartService;
		private readonly IUsersService _usersService;

		public CartController(ICartService cartService, IUsersService usersService)
		{
			_cartService = cartService;
			_usersService = usersService;
		}

		[Authorize(Policy = "UserPolicy")]
		[HttpGet]
		public async Task<ActionResult<Cart>> GetUserCart()
		{
			try
			{
				var userId = _usersService.GetCurrentUserId(HttpContext);

				var cart = await _cartService.GetCart(userId.Value);

				if (cart == null)
				{
					return NotFound(new { message = "Корзина не
знайдена." });
				}

				return Ok(cart);
			}
			catch (Exception ex)
			{
				Console.WriteLine($"[GetUserCart] Ошибка:
{ex.Message}");
				return BadRequest(new { message = "Сталася помилка при
обробці запиту.", error = ex.Message });
			}
		}

		[Authorize(Policy = "UserPolicy")]
		[HttpPost("add")]
		public async Task<ActionResult<Cart>> AddProductToCart([FromBody]
CartItemRequest newItem)
		{
```

```
                try
                {
                        var userId = _usersService.GetCurrentUserId(HttpContext);

                        await _cartService.AddProduct(userId.Value,
newItem.ProductId, newItem.Quantity);
                        return Ok(new { message = "ok" });
                }
                catch (Exception ex)
                {
                        Console.WriteLine($"[AddProductToCart] Помилка:
{ex.Message}");
                        if (ex.InnerException != null)
                        {
                                Console.WriteLine($"[Inner Exception]
{ex.InnerException.Message}");
                        }
                        return BadRequest(new
                        {
                                message = "Сталася помилка при додаванні товару
до кошика.",
                                error = ex.InnerException?.Message ?? ex.Message
                        });
                }
        }

        [Authorize(Policy = "UserPolicy")]
        [HttpDelete("{productId:guid}")]
        public async Task<ActionResult<Guid>> RemoveFromCart( Guid productId)
        {
                try
                {
                        var userId = _usersService.GetCurrentUserId(HttpContext);

                        await _cartService.RemoveFromCart(userId.Value, productId);
                        return Ok(new { message = "ok" });
                }
                catch (Exception ex)
                {
                        return BadRequest(new { message = ex.Message });
                }
        }

    }
}
```

Лістинг Д.2 — Контролер CustomProductsController

```
    namespace SweetForm.API.Controllers
{
    [ApiController]
    [Route("api/custom-products")]
    [Authorize]
```

```
	public class CustomProductsController : ControllerBase
	{
		private readonly ICustomProductsService _customProductService;
		private readonly IUsersService _usersService;


		public CustomProductsController(ICustomProductsService service,
IUsersService usersService)
		{
			_customProductService = service;
			_usersService = usersService;
		}

		[Authorize(Policy = "UserPolicy")]
		[HttpGet]
		public async Task<ActionResult<List<CustomProduct>>>
GetAll([FromQuery] ProductStatus? status)
		{
			var userInfo = await _usersService.GetUserInfo(HttpContext);
			Guid userId = userInfo.Id;

			var products = await
_customProductService.GetAllCustomProducts(userId, status);
			return Ok(products);
		}

		[Authorize(Policy = "UserPolicy")]
		[HttpGet("get-by-id/{id:guid}")]
		public async Task<ActionResult<CustomProduct>> GetById(Guid id)
		{
			var product = await
_customProductService.GetCustomProductById(id);
			if (product == null)
				return NotFound(new { message = "Кастомний продукт не
знайдено." });

			return Ok(product);
		}
		[Authorize(Policy = "UserPolicy")]
		[HttpPost("create")]
		public async Task<ActionResult<CustomProduct>> Create([FromBody]
CustomProductRequest request)
		{
			try
			{
				var userInfo = await _usersService.GetUserInfo(HttpContext);
				Guid userId = userInfo.Id;
				string userEmail = userInfo.Email;

				if (userId == null)
				{
					return Unauthorized(new { message = "Не вдалося
знайти ID користувача." });
				}
```

```
                                var product = CustomProduct.Create(
                                        Guid.NewGuid(),
                                        userId,
                                        request.Name,
                                        request.ProductType,
                                        request.ImageURL,
                                        request.Material,
                                        request.Size,
                                        request.Description);

                                var created = await
_customProductService.CreateCustomProduct(product);
                                return Ok(created);
                        }
                        catch (ArgumentException ex)
                        {
                                return BadRequest(new { message = ex.Message });
                        }
                }

                [Authorize(Policy = "UserPolicy")]
                [HttpPut("update/{id:guid}")]
                [Authorize]
                public async Task<ActionResult<Guid>> Update(Guid id, [FromBody]
CustomProductRequest request)
                {
                        try
                        {
                                var userInfo = await _usersService.GetUserInfo(HttpContext);
                                Guid userId = userInfo.Id;
                                string userEmail = userInfo.Email;

                                if (userId == null)
                                {
                                        return Unauthorized(new { message = "Не вдалося
знайти ID користувача." });
                                }

                                var existing = await
_customProductService.GetCustomProductById(id);
                                if (existing == null)
                                        return NotFound(new { message = "Кастомний
продукт не знайдено." });

                                var updated = await
_customProductService.UpdateCustomProduct(
                                        id,
                                        userId,
                                        request.Name,
                                        request.ProductType,
                                        request.ImageURL,
                                        request.Material,
                                        request.Size,
```

```
                                        request.Description);

                                return Ok(updated);
                        }
                        catch (Exception ex)
                        {
                                return BadRequest(new { message = ex.Message });
                        }
                }

                [Authorize(Policy = "UserPolicy")]
                [HttpDelete("delete/{id:guid}")]
                [Authorize]
                public async Task<ActionResult> Delete(Guid id)
                {
                        var existing = await
_customProductService.GetCustomProductById(id);
                        if (existing == null)
                                return NotFound(new { message = "Кастомний продукт не
знайдено." });

                        await _customProductService.DeleteCustomProduct(id);
                        return Ok();
                }

                [Authorize(Policy = "AdminPolicy")]
                [HttpPost("{id:guid}/status")]
                public async Task<ActionResult> ChangeStatus(Guid id, [FromBody]
ProductStatus newStatus)
                {
                        try
                        {
                                var userInfo = await _usersService.GetUserInfo(HttpContext);
                                string userEmail = userInfo.Email;

                                await _customProductService.ChangeCustomProductStatus(id,
newStatus, userEmail);
                                return Ok();
                        }
                        catch (Exception ex)
                        {
                                return BadRequest(new { message = ex.Message });
                        }
                }

                [Authorize(Policy = "AdminPolicy")]
                [HttpPost("{id:guid}/completed")]
                public async Task<ActionResult<Guid>>
CreateProductByCustomProduct(Guid id, [FromBody] ProductRequest request)
                {
                        try
                        {
                                var createdProductId = await
_customProductService.CreateProductByCustomProduct(id, request.Theme, request.Price);
```

```
                return Ok(createdProductId);
            }
            catch (Exception ex)
            {
                return BadRequest(new { message = ex.Message });
            }
        }
    }
}
```

Лістинг Д.3 — Контролер FileController

```
    public class FileController : ControllerBase
    {
        private readonly IFilesService _filesService;
        private readonly IProductsService _productsService;

        public FileController(IFilesService filesService, IProductsService
productsService)
        {
            _filesService = filesService;
            _productsService = productsService;
        }

        [HttpPost("uploadFile")]
        [Consumes("multipart/form-data")]
        public async Task<IActionResult> UploadFile(IFormFile file, [FromForm]
string prefix)
        {
            if (file == null || file.Length == 0)
                return BadRequest("Файл не был выбран.");

            if (file.Length > 5 * 1024 * 1024)
                return BadRequest("Размер файла превышает 5MB.");

            if (string.IsNullOrEmpty(prefix))
                return BadRequest("Префикс не передан");
            try
            {
                var fileName = await _filesService.UploadFileAsync(file,
prefix);

                return Ok(new { fileName});
            }
            catch (Exception ex)
            {
                return StatusCode(500, new { message = "Ошибка при
загрузке файла", error = ex.Message });
            }
        }

        [HttpPost("create-folder/{prefix}")]
        public async Task<IActionResult> CreateFolder(string prefix)
        {
```

```
                var products = await _productsService.GetProducts();
                bool productTypeExists = products.Any(p => p.ProductType ==
prefix);

                if (!productTypeExists)
                {
                        await _filesService.CreateFolder(prefix);
                }

                return Ok(new { prefix });

        }

        [HttpPost("deleteFile")]
        public async Task<IActionResult> DeleteFile(string fileName)
        {
                if (string.IsNullOrEmpty(fileName))
                        return BadRequest("Имя файла не указано.");

                try
                {
                        var response = await _filesService.DeleteFileAsync(fileName);
                        return Ok(new { message = "Файл успешно удалён",
response });
                }
                catch (Exception ex)
                {
                        return StatusCode(500, new { message = "Ошибка при
удалении файла", error = ex.Message });
                }
        }

    }
}
}
```

Лістинг Д.4 — Контролер OrderController

```
namespace SweetForm.API.Controller
{
    [ApiController]
    [Route("api/order")]
    [Authorize]
    public class OrderController : ControllerBase
    {
        private readonly IOrderService _orderService;
        private readonly IUsersService _usersService;
        private readonly IMailsService _mailsService;
        public OrderController(IOrderService orderService, IUsersService
usersService, IMailsService mailsService)
        {
            _orderService = orderService;
            _usersService = usersService;
            _mailsService = mailsService;
```

```
        }
        [Authorize(Policy = "UserPolicy")]
        [HttpPost("place-order")]
        public async Task<IActionResult> PlaceOrder([FromBody] OrderRequest
newOrder)
        {
                var userInfo = await _usersService.GetUserInfo(HttpContext);
                Guid userId = userInfo.Id;
                string userEmail = userInfo.Email;

                if (userId == null)
                {
                        return Unauthorized(new { message = "Не вдалося знайти ID
користувача." });
                }
                if (newOrder.Address == null)
                {
                        throw new Exception("newOrder.Address is NULL при вході
в PlaceOrder");
                }

                try
                {
                        var order = await _orderService.PlaceOrder(userId, userEmail,
newOrder.Address, newOrder.City, newOrder.PostalCode, newOrder.Country);
                        return Ok(order);
                }
                catch (Exception ex)
                {
                        return BadRequest(new { message = "Помилка при створенні
замовлення.", error = ex.Message });
                }
        }

        [Authorize(Policy = "UserPolicy")]
        [HttpGet("order-list")]
        public async Task<IActionResult> GetOrdersHistory()
        {
                var userId = _usersService.GetCurrentUserId(HttpContext);

                if (userId == null)
                {
                        return Unauthorized(new { message = "Не вдалося знайти ID
користувача." });
                }

                var orders = await _orderService.GetOrderHistory(userId.Value);
                return Ok(orders);
        }

        [Authorize(Policy = "AdminPolicy")]
        [HttpGet("all-orders-list")]
        public async Task<IActionResult> GetAllOrders()
        {
```

```
                return Ok(await _orderService.AdminGetAllOrders());
            }

            [Authorize(Policy = "AdminPolicy")]
            [HttpPost("{orderId:guid}")]
            public async Task<IActionResult> UpdateOrderStatus( Guid orderId,
OrderStatus newStatus)
            {
                var userInfo = await _usersService.GetUserInfo(HttpContext);

                var id = await _orderService.UpdateOrderStatus(orderId, newStatus,
userInfo.Id);
                return Ok(id);
            }

            [Authorize(Policy = "AdminPolicy")]
            [HttpDelete("{orderId:guid}")]
            public async Task<IActionResult> DeleteOrder(Guid orderId)
            {
                var id = await _orderService.DeleteOrder(orderId);
                return Ok(id);
            }
        }
}
```

Лістинг Д.5 — Контролер ProductsController

```
namespace SweetForm.API.Controller
{
    [ApiController]
    [Route("api/products")]
    [Authorize]
    public class ProductsController : ControllerBase
    {
        private readonly IProductsService _productsService;
        private readonly IFilesService _filesService;

        public ProductsController(IProductsService productsService, IFilesService
filesService)
        {
            _productsService = productsService;
            _filesService = filesService;
        }

        [AllowAnonymous]
        [HttpGet]
        public async Task<ActionResult<List<ProductResponse>>> GetProducts(
            [FromQuery] string? productType,
            [FromQuery] string? theme,
            [FromQuery] decimal? minPrice,
            [FromQuery] decimal? maxPrice)
        {
            var products = await _productsService.GetProducts(productType,
theme, minPrice, maxPrice);
```

```
            return Ok(products);
        }

        [HttpPost("create")]
        [Authorize(Policy = "AdminPolicy")]
        public async Task<ActionResult<Product>> CreateProduct([FromBody]
ProductRequest request)
        {
            try
            {
                var product = Product.Create(
                    Guid.NewGuid(),
                    request.Name,
                    request.ProductType,
                    request.ImageURL,
                    request.Material,
                    request.Theme,
                    request.Size,
                    request.Description,
                    request.Price);
                var productData = await
_productsService.CreateProduct(product);
                return Ok(productData);

            }
            catch (ArgumentException ex)
            {
                return BadRequest(new { message = ex.Message });
            }
        }


        [HttpPut("{id:guid}")]
        [Authorize(Policy = "AdminPolicy")]
        public async Task<ActionResult<Guid>> UpdateProduct(Guid id,
[FromBody] ProductRequest request)
        {
            var existingProduct = await _productsService.GetProductById(id);
            if (existingProduct == null)
            {
                return NotFound(new { message = "Продукт не знайдено."
});
            }

            var productId = await _productsService.UpdateProduct(id,
request.Name, request.ProductType, request.ImageURL, request.Material, request.Theme,
request.Size, request.Description, request.Price);
            return Ok(productId);

        }

        [HttpDelete("{id:guid}")]
        [Authorize(Policy = "AdminPolicy")]
```

```
		public async Task<ActionResult<Guid>> DeleteProduct(Guid id)
		{

			var product = await _productsService.GetProductById(id);

			if (product == null)
			{
				return NotFound(new { message = "Продукт не знайдено."
});
			}

			await _productsService.DeleteProduct(id);

			if (!string.IsNullOrEmpty(product.ImageURL))
			{
				await _filesService.DeleteFileAsync(product.ImageURL);
			}

			return Ok();
		}

		[HttpGet("{productId:guid}")]
		public async Task<ActionResult<Product>> GetProductById(Guid
productId)
		{
			try
			{
				var product = await
_productsService.GetProductById(productId);
				if (product == null)
				{
					return NotFound();
				}

				return Ok(product);
			}
			catch (Exception ex)
			{
				return StatusCode(500, new { message = ex.Message });
			}
		}
	}
}
```

Лістинг Д.6 — Контролер UsersController

```
	namespace SweetForm.API.Controller
{
	[ApiController]
	[Route("api/users")]
	public class UsersController : ControllerBase
	{
		private readonly IUsersService _usersService;
		private readonly IAuthorizationService _authorizationService;
```

```
        private readonly IMailsService _mailsService;
        public UsersController(IUsersService usersService,
            IAuthorizationService authorizationService)
        {
            _usersService = usersService;
            _authorizationService = authorizationService;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody]
RegisterUserRequest request)
        {
            if (string.IsNullOrEmpty(request.UserName) ||
string.IsNullOrEmpty(request.Password) || string.IsNullOrEmpty(request.Email))
            {
                return BadRequest(new { message = "Всі поля є
обов'язковими для заповнення." });
            }

            var token = await _usersService.Register(request.UserName,
request.Password, request.Email);
            Response.Cookies.Append("kokiie", token.AccessToken);
            Response.Cookies.Append("refresh", token.RefreshToken);
            return Ok(new { accessToken = token.AccessToken, refreshToken =
token.RefreshToken });
        }

        [HttpPost("confirm-email")]
        public async Task<IActionResult> ConfirmEmail([FromQuery] string token)
        {
            if (token == null || string.IsNullOrEmpty(token))
            {
                return BadRequest(new { message = "Токен відсутні" });
            }

            var isConfirm = await _usersService.ConfirmEmailAddress(token);

            if (!isConfirm)
            {
                return BadRequest(new { message = "Невірний або
прострочений токен." });
            }

            return Ok(new { message = "Ваш email підтверджено!" });
        }


        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] LoginUserRequest
request)
        {
            if (string.IsNullOrEmpty(request.Password) ||
string.IsNullOrEmpty(request.Email))
            {
```

```
                return BadRequest(new { message = "Всі поля є
обов'язковими для заповнення." });
            }

            try
            {
                var token = await _usersService.Login(request.Email,
request.Password);
                if (string.IsNullOrEmpty(token.AccessToken) ||
string.IsNullOrEmpty(token.RefreshToken))
                    return Unauthorized(new { message = "Неправильний
email або пароль." });

                Response.Cookies.Append("kokiie", token.AccessToken);
                Response.Cookies.Append("refresh", token.RefreshToken);
                return Ok(new { accessToken = token.AccessToken,
refreshToken = token.RefreshToken });
            }
            catch (Exception ex)
            {
                return Unauthorized(new { message = "Неправильний email
або пароль.", error = ex.Message });
            }
        }

        [HttpPost("refresh")]
        public async Task<IActionResult> RefreshToken()

        {
            var refreshToken = Request.Cookies["refresh"];
            var accessToken = Request.Cookies["kokiie"];

            if (string.IsNullOrEmpty(accessToken) ||
string.IsNullOrEmpty(refreshToken))
                return BadRequest(new { message = "Токени не надано" });

            try
            {
                var (newAccessToken, newRefreshToken) = await
_usersService.RefreshTokenAsync(accessToken, refreshToken);

                Response.Cookies.Append("refresh", newRefreshToken);
                Response.Cookies.Append("kokiie", newAccessToken);

                return Ok(new
                {
                    accessToken = newAccessToken,
                    refreshToken = newRefreshToken,

                });
            }
            catch (Exception ex)
            {
                return Unauthorized(new { message = ex.Message });
```

```
                }
        }

        [HttpPost("send-forget-password-email")]
        public async Task<IActionResult> SendForgetPasswordEmail([FromBody]
SendForgetPasswordEmailRequest request)
        {
                try
                {
                        if (string.IsNullOrEmpty(request.Email))
                                return BadRequest(new { message = "Email
обов'язковий." });

                        await
_usersService.ForgetPasswordEmailAsync(request.Email);
                        return Ok();
                }
                catch (Exception ex)
                {
                        return Unauthorized(new { message = ex.Message });
                }
        }

        [HttpPost("change-password")]
        public async Task<IActionResult> ChangePassword([FromBody]
ChangePasswordRequest request)
        {
                try
                {
                        await _usersService.ConfirmForgetPassword(request.Token,
request.newPassword);
                        return Ok();
                }
                catch (Exception ex)
                {
                        return Unauthorized(new { message = ex.Message });
                }
        }

        [HttpPost("logout")]
        public IActionResult Logout()
        {
                Response.Cookies.Delete("kokiie");
                Response.Cookies.Delete("refresh");

                return Ok(new { Message = "Logout successful" });
        }

    }
}
```

Лістинг Д.7 — Головна сторінка

```
export default function Home() {
```

```
  const [fetchProducts, { data: products }] = useLazyGetProductsQuery();

  useEffect(() => {
    fetchProducts({});
  }, []);

  return (
    <div>
<div className={styles.heroWrapper}>
  <div className={styles.heroImage}>
    <div className={styles.heroOverlay}>
      <Title level={2} className={styles.heroTitle}>Формочки та трафарети для
випікання</Title>
      <Title level={5} className={styles.heroSubtitle}>Зроблені на 3Д принтері</Title>
      <Divider style={{ backgroundColor: 'white' }} />
      <Link href="/products">
        <Button type="primary" size="large">Почати покупки</Button>
      </Link>
    </div>
  </div>
</div>

      {/* Recently Added */}
      <div className={styles.recent}>
        <Title level={2} className={styles.styleTitle}>Останні додані продукти</Title>
        <Link href="/products">
          <Button type="default" className={styles.whiteButton}>Усі продукти</Button>
        </Link>

        <div className={styles.scrollContainer}>
          {products?.map((product) => (
            <div key={product.id} className={styles.productCard}>
              <Link href={`/mold-page/${product.id}`}>
                <img
                  src={`https://sweetform-s3-bucket.s3.amazonaws.com/${product.imageURL}`}
                  alt={product.name}
                />
                <p>{product.name}</p>
                <strong>{product.price} грн.</strong>
              </Link>
            </div>
          ))}
        </div>

        {/* Custom Product Info Block */}
          <Divider style={{ backgroundColor: 'white' }} />
        <div className={styles.customBlock}>
          <Title level={3} className={styles.styleTitle}>Хочеш щось унікальне?</Title>
          <Paragraph className={styles.styleTitle}>Замов власний кастомний виріб з будь-
якою формою, розміром і дизайном!</Paragraph>
          <Link href="/custom-product/create">
            <Button type="primary" size="large">Створити кастомний продукт</Button>
          </Link>
        </div>
```

```
    </div>
   </div>
  );
}
```

Лістинг Д.8 — Модальне вікно входу до системи

```
interface Props {
  isVisible: boolean;
  onClose: () => void;
  openRegisterModal: () => void;
}

export const LoginModal = ({ isVisible, onClose, openRegisterModal }: Props) => {
  const [isForgotOpen, setForgotOpen] = useState(false);
  const [form] = Form.useForm();
  const [loginMutation] = useLoginMutation();

  const handleSubmit = async (values: { email: string; password: string }) => {
    try {
      await loginMutation(values).unwrap();
      onClose();
    } catch (error) {
      message.error("Помилка входу");
    }
  };

  return (
    <Modal title="Вхід до системи" open={isVisible} onCancel={onClose} footer={null}>
      <Form form={form} name="login" onFinish={handleSubmit} layout="vertical">
        <Form.Item
          label="Email"
          name="email"
          rules={[{ required: true, message: "Будь ласка, введіть email!", type: "email" }]}
        >
          <Input placeholder="Введіть email" />
        </Form.Item>

        <Form.Item
          label="Пароль"
          name="password"
          rules={[{ required: true, message: "Будь ласка, введіть пароль!" }]}
        >
          <Input.Password placeholder="Введіть пароль" />
        </Form.Item>

        <Form.Item>
        <Button type="link" onClick={() => setForgotOpen(true) }>Забули пароль?</Button>
        <ForgotPasswordModal isVisible={isForgotOpen} onClose={() =>
setForgotOpen(false)} />

        </Form.Item>

        <Form.Item>
```

```
        <Button type="primary" htmlType="submit" block>
          Увійти
        </Button>
      </Form.Item>

      <Divider />

      <Form.Item>
        <Button type="text" block onClick={openRegisterModal}>
          Створити акаунт
        </Button>
      </Form.Item>
    </Form>
  </Modal>
 );
};
```

Лістинг Д.9 — Модальне вікно реєстрації нового користувача

```
interface Props {
 isVisible: boolean;
 onClose: () => void;
 openLoginModal: () => void;
}

export const RegisterModal = ({ isVisible, onClose, openLoginModal }: Props) => {
 const [form] = Form.useForm();

 const [registerUser] = useRegisterMutation();
 const handleSubmit = async (values: { userName: string; email: string; password: string })
=> {
   try {
    await registerUser(values).unwrap();
    form.resetFields();
    onClose();
   } catch (error) {
    message.error("Помилка реєстрації");
   }
 };

 return (
  <Modal title="Реєстрація" open={isVisible} onCancel={onClose} footer={null}>
    <Form form={form} name="register" onFinish={handleSubmit} layout="vertical">
      <Form.Item
        label="Ім'я користувача"
        name="userName"
        rules={[{ required: true, message: "Будь ласка, введіть ім'я користувача!" }]}
      >
        <Input placeholder="Введіть ім'я користувача" />
      </Form.Item>

      <Form.Item
        label="Email"
        name="email"
```

```
        rules={[{ required: true, message: "Будь ласка, введіть email!", type: "email" }]}
      >
        <Input placeholder="Введіть email" />
      </Form.Item>

      <Form.Item
        label="Пароль"
        name="password"
        rules={[
          { required: true, message: "Будь ласка, введіть пароль!" },
          { min: 6, message: "Пароль повинен містити мінімум 6 символів!" },
        ]}
      >
        <Input.Password placeholder="Введіть пароль" />
      </Form.Item>

      <Form.Item>
        <Button type="primary" htmlType="submit" block>
          Зареєструватися
        </Button>
      </Form.Item>

      <Divider />

      <Form.Item>
        <Button type="text" block onClick={openLoginModal}>
          Увійти в існуючий акаунт
        </Button>
      </Form.Item>
    </Form>
  </Modal>
 );
};
```

Лістинг Д.10 — Сторінка профілю користувача

```
const ProfilePage: React.FC = () => {
 const router = useRouter();

 const { data: orders, isLoading, isError } = useGetOrderHistoryQuery();
 const { data: customProducts, isLoading: isLoadingCustoms, isError: isErrorCustoms,
refetch } = useGetCustomProductsQuery({ byUser: true });
 const [deleteCustomProduct] = useDeleteCustomProductMutation();

 const handleEdit = (id: string) => {
  router.push(`/custom-product/${id}?edit=true`);
 };

 const handleDelete = async (productId: string) => {
  try {
   await deleteCustomProduct({ productId }).unwrap();
   message.success("Кастомний продукт видалено");
   refetch();
  } catch {
```

```
    message.error("Не вдалося видалити продукт");
  }
 };

 return (
  <div className={styles.wrapper}>
   <Title level={2} className={styles.title}>Профіль</Title>

   <Divider className={styles.divider} orientation="left">Мої замовлення</Divider>
   {isLoading ? (
    <p>Завантаження замовлень...</p>
   ) : isError ? (
    <p>Сталася помилка при завантаженні.</p>
   ) : !orders?.length ? (
    <p>Замовлень не знайдено.</p>
   ) : (
    <div className={styles.fullWidthGrid}>
     {orders.map((order) => {
      const { color, text } = getOrderStatusInfo(order.orderStatus);
      return (
       <Card
        key={order.orderId}
        style={{ marginBottom: 20 }}
        title={
         <>
          <p><b>Замовлення:</b> {order.orderId}</p>
          <p><b>Дата:</b> {new Date(order.orderDate).toLocaleDateString("uk-UA", {
           day: "2-digit", month: "long", year: "numeric"
          })}</p>
          <Tag color={color}>{text}</Tag>
         </>
        }
       >
        {order.orderItems.map((item: OrderItem) => (
         <p key={item.id}>Продукт: {item.productName}, кількість:
{item.quantity}</p>
        ))}
        <p><b>Загальна сума:</b> {order.orderAmount.toFixed(2)} грн</p>
       </Card>
      );
     })}
    </div>
   )}

   <Divider className={styles.divider} orientation="left">Мої кастомні
продукти</Divider>
   {isLoadingCustoms ? (
    <p>Завантаження кастомних продуктів...</p>
   ) : isErrorCustoms ? (
    <p>Помилка при завантаженні кастомних продуктів.</p>
   ) : !customProducts?.length ? (
    <p>Немає кастомних продуктів.</p>
   ) : (
    <div className={styles.cards}>
```

```
        {customProducts.map((product) => {
          const { color, text } = getCustomProductStatusInfo(product.status);
          return (
            <Card
              key={product.id}
              className={styles.cardStyle}
              cover={
                <Image
                  src={`https://sweetform-s3-bucket.s3.amazonaws.com/${product.imageURL}`}
                  alt={product.name}
                  fallback="/no-image.png"
                  className={styles.cardImage}
                  preview={false}
                />
              }
              actions={product.status === ProductStatus.Pending ? [
                <Button type="link" icon={<EditOutlined />} onClick={() =>
handleEdit(product.id)}>Редагувати</Button>,
                <Popconfirm
                  title="Видалити продукт?"
                  description="Цю дію не можна скасувати."
                  onConfirm={() => handleDelete(product.id)}
                  okText="Так"
                  cancelText="Ні"
                >
                  <Button danger type="link" icon={<DeleteOutlined />}>Видалити</Button>
                </Popconfirm>,
              ] : undefined}
            >
              <Title level={5}>
                {product.status === ProductStatus.Completed ? (
                  <Link href={`/mold-
page/${product.createdProductId}`}>{product.name}</Link>
                ) : (
                  product.name
                )}
              </Title>

              <Tag color={color}>{text}</Tag>

              <p><b>Опис:</b> {product.description}</p>
              <p><b>Тип:</b> {product.productType}</p>
              <p><b>Матеріал:</b> {product.material}</p>
              <p><b>Розмір:</b> {product.size}</p>
            </Card>
          );
        })}
      </div>
    )}
  </div>
 );
}

export default ProfilePage;
```

Лістинг Д.11 — Головна сторінка продуктів

```
const pageSize = 8;

const ProductsPage: React.FC = () => {
  const { minPrice, maxPrice } = useAppSelector(state => state.filters);
  const [trigger, { data: products = [], isLoading }] = useLazyGetProductsQuery();
  const [currentPage, setCurrentPage] = useState(1);

  useEffect(() => {
    trigger({ minPrice, maxPrice });
  }, [minPrice, maxPrice, trigger]);

  const paginatedProducts = products.slice((currentPage - 1) * pageSize, currentPage *
pageSize);

  return isLoading ? (
    <div className={styles.spinnerWrapper}>
      <Spin />
    </div>
  ) : (
    <div className={styles.moldsPage}>
      <Products products={paginatedProducts} />

      {products.length > pageSize && (
        <div className={styles.paginationWrapper}>
          <Pagination
            current={currentPage}
            pageSize={pageSize}
            total={products.length}
            onChange={setCurrentPage}
            showTotal={(total, range) => `${range[0]}-${range[1]} з ${total}`}
          />
        </div>
      )}
    </div>
  );
}

export default ProductsPage;
```

Лістинг Д.12 — Сторінка продуктів за їх типом

```
const pageSize = 4;

const ProductsByProductTypePage: React.FC = () => {
  const [trigger, { data: products = [], isLoading }] = useLazyGetProductsQuery();
  const searchParams = useSearchParams();
  const productType = searchParams.get('type') || '';

  const [currentPage, setCurrentPage] = useState(1);

  useEffect(() => {
    if (productType) {
```

```
    trigger({ productType });
    setCurrentPage(1);
   }
  }, [productType, trigger]);

  const paginatedProducts = products.slice((currentPage - 1) * pageSize, currentPage *
pageSize);

  return isLoading ? (
   <div className={styles.spinnerWrapper}>
    <Spin size="large" />
   </div>
  ) : (
   <div className={styles.pageWrapper}>
    <Title level={2} className={styles.title}>{productType}</Title>

    <Products products={paginatedProducts} />

    {products.length > pageSize && (
     <div className={styles.pagination}>
      <Pagination
       current={currentPage}
       pageSize={pageSize}
       total={products.length}
       onChange={setCurrentPage}
       showTotal={(total, range) => `${range[0]}-${range[1]} з ${total}`}
      />
     </div>
    )}
   </div>
  );
}
export default ProductsByProductTypePage;
```

Лістинг Д.13 — Сторінка продукту за темою

```
const ProductsByTheme: React.FC = () => {

  const [trigger, { data: products = [], isLoading }] = useLazyGetProductsQuery();
  const searchParams = useSearchParams();
  const productType = searchParams.get('type') || '';
  const theme = searchParams.get('theme') || '';

  const [currentPage, setCurrentPage] = useState(1);

  useEffect(() => {
   if (productType || theme) {
    trigger({ productType, theme });
    setCurrentPage(1);
   }
  }, [productType, theme, trigger]);

  const paginatedProducts = products.slice((currentPage - 1) * pageSize, currentPage *
pageSize);
```

```
  return isLoading ? (
    <div className={styles.spinnerWrapper}>
      <Spin size="large" />
    </div>
  ) : (
    <div className={styles.pageWrapper}>
    <Title level={2} className={styles.title}>
      {productType} за темою {theme}
    </Title>

      <Products products={paginatedProducts} />

      {products.length > pageSize && (
        <div className={styles.pagination}>
          <Pagination
            current={currentPage}
            pageSize={pageSize}
            total={products.length}
            onChange={setCurrentPage}
            showTotal={(total, range) => `${range[0]}-${range[1]} з ${total}`}
          />
        </div>
      )}
    </div>
  );
}

export default ProductsByTheme;
```

Лістинг Д.14 — Головна сторінка продукту

```
const ProductDetailPage: React.FC = () => {
  const [productId, setProductId] = useState<string>('');
  const [quantity, setQuantity] = useState<number>(1);

  useEffect(() => {
    const currentProductId =
decodeURIComponent(window.location.pathname.split('/').pop() || '');
    setProductId(currentProductId);
  }, []);

  const { data: product, isLoading } = useGetProductByIdQuery(productId);
  const [addToCart] = useAddToCartMutation();

  const handleAddToCart = async () => {
    try {
      await addToCart({ productId, quantity }).unwrap();
      message.success("Товар додано до кошика");
    } catch (error) {
      message.error("Помилка при додаванні до кошика");
    }
  };
```

```
  const handleChangeQuantity = (value: number) => {
    if (value < 1) setQuantity(1);
    else if (value > 99) setQuantity(99);
    else setQuantity(value);
  };

  return isLoading || !product ? (
    <div className={styles.spinnerWrapper}>
      <Spin size="large" />
    </div>
  ) : (
    <div className={styles.pageWrapper}>
      <Card>
        <div className={styles.infoWrapper}>
          <div className={styles.productImageWrapper}>
            <Image
              src={`https://sweetform-s3-bucket.s3.amazonaws.com/${product.imageURL}`}
              alt={product.name}
              preview={false}
              className={styles.productImage}
              onError={(e) => (e.currentTarget.src = "...")}
            />
          </div>

          <div className={styles.productInfoWrapper}>
            <Title level={3}>{product.name}</Title>
            <Paragraph>Тема: {product.theme}</Paragraph>
            <Paragraph>Ціна: {product.price} грн.</Paragraph>
            <Paragraph>Матеріал: {product.material}</Paragraph>
            <Paragraph>Розмір: {product.size}</Paragraph>
            <Paragraph>Опис: {product.description}</Paragraph>

            <div className={styles.quantityControl}>
              <Button onClick={() => setQuantity(Math.max(1, quantity - 1))}>-</Button>
              <Input
                value={quantity}
                onChange={(e) => handleChangeQuantity(Number(e.target.value))}
                className={styles.quantityInput}
              />
              <Button onClick={() => setQuantity(Math.min(99, quantity + 1))}>+</Button>
            </div>

            <Button type="primary" onClick={handleAddToCart}
className={styles.addToCartButton}>
              Додати в кошик
            </Button>
          </div>
        </div>
      </Card>
    </div>
  );
};

export default ProductDetailPage;
```

Лістинг Д.15 — Сторінка створення запиту на кастомний виріб

```
const CustomProductCreatePage: React.FC = () => {
  const [form] = Form.useForm();
  const [items, setItems] = useState<string[]>(['Форма', 'Трафарет', 'Іграшка']);
  const [fileList, setFileList] = useState<UploadFile[]>([]);
  const [previewUrl, setPreviewUrl] = useState<string | null>(null);

  const [createCustomProduct, { isLoading }] = useCreateCustomProductMutation();

  const params = useParams();
  const searchParams = useSearchParams();
  const productId = params?.id as string | undefined;
  const isEditMode = searchParams.get('edit') === 'true';

  const { data: existingProduct, isLoading: isLoadingProduct } =
useGetCustomProductByIdQuery(productId!, {
    skip: !isEditMode || !productId,
  });

  useEffect(() => {
    if (existingProduct && isEditMode) {
      form.setFieldsValue({
        name: existingProduct.name,
        productType: existingProduct.productType,
        description: existingProduct.description,
        size: existingProduct.size,
        material: existingProduct.material,
        imageURL: existingProduct.imageURL,
      });

      if (existingProduct.imageURL) {
        setPreviewUrl(existingProduct.imageURL);
      }
    }
  }, [existingProduct, isEditMode]);

  const handleFinish = async (values: any) => {
    try {
      await createCustomProduct(values).unwrap();
      message.success('Кастомний виріб успішно створено!');
      form.resetFields();
      setPreviewUrl(null);
      setFileList([]);
    } catch (error) {
      console.error('Error creating product:', error);
      message.error('Помилка під час створення виробу');
    }
  };

  const handleUpload = async (file: File, prefix: string) => {
    try {
      const result = await uploadFile(file, prefix);
      message.success('Файл завантажено успішно!');
```

```
      form.setFieldsValue({ imageURL: result });
      setPreviewUrl(result);
    } catch (error: unknown) {
      if (axios.isAxiosError(error)) {
        message.error(error.response?.status || error.message);
      } else {
        message.error('Невідома помилка завантаження');
      }
    }
  };

  const handleUploadImg = async ({ file, onSuccess, onError }: any) => {
    const productType = form.getFieldValue('productType');

    if (!productType) {
      message.warning('Спочатку виберіть тип продукту');
      return onError?.('Тип продукту не обрано');
    }

    const isNewType = !items.some((item) => item.toLowerCase() ===
productType.toLowerCase());

    if (isNewType) {
      await createFolder(productType);
      message.success('Папку створено');
    }

    await handleUpload(file, productType);
  };

  if (isEditMode && isLoadingProduct) {
    return <p>Завантаження виробу для редагування...</p>;
  }

  if (isEditMode && existingProduct && existingProduct.status !== 0) {
    return (
      <Card className={styles.pageWrapper}>
        <Title level={4} type="danger">Редагування заборонено</Title>
        <Paragraph>Цей виріб вже опрацьовано й не може бути змінений.</Paragraph>
      </Card>
    );
  }

  return (
    <div className={styles.pageWrapper}>
      <Card>
        <Title level={3}>{isEditMode ? 'Редагувати кастомний виріб' : 'Створити
кастомний виріб'}</Title>
        <Paragraph>
          {isEditMode
            ? 'Відредагуй дані виробу нижче.'
            : 'Заповни поля нижче, щоб надіслати запит на створення власного виробу.'}
        </Paragraph>
```

```
      {previewUrl && (
        <div style={{ textAlign: 'center', marginBottom: 24 }}>
          <img
            src={`https://sweetform-s3-bucket.s3.amazonaws.com/${previewUrl}`}
            alt="Попередній перегляд"
            style={{ maxWidth: '100%', maxHeight: 300, objectFit: 'contain', borderRadius: 8
}}
          />
        </div>
      )}

      <Form layout="vertical" form={form} onFinish={handleFinish}
classNameName={styles.form}>
        <Form.Item
          label="Назва виробу"
          name="name"
          rules={[{ required: true, message: 'Введіть назву виробу' }]}
        >
          <Input placeholder="Наприклад: Форма для печива 'Зірка'" />
        </Form.Item>

        <Form.Item
          label="Тип продукту"
          name="productType"
          rules={[{ required: true, message: 'Введіть або виберіть тип продукту' }]}
        >
          <Select
            style={{ width: '100%' }}
            placeholder="Оберіть або додайте тип продукту"
            dropdownRender={(menu) => (
              <>
                {menu}
                <Divider style={{ margin: '8px 0' }} />
                <Space style={{ padding: '0 8px 4px' }}>
                  <Input
                    placeholder="Новий тип"
                    value={form.getFieldValue('productType')}
                    onChange={(e) => form.setFieldValue('productType', e.target.value)}
                    onKeyDown={(e) => e.stopPropagation()}
                  />
                  <Button
                    type="text"
                    onClick={() => {
                      const newType = form.getFieldValue('productType');
                      if (newType && !items.includes(newType)) {
                        setItems([...items, newType]);
                        message.success('Тип додано до списку');
                      }
                    }}
                  >
                    Додати
                  </Button>
                </Space>
              </>
```

```
            )}
            options={items.map((item) => ({ label: item, value: item }))}
           />
          </Form.Item>

          <Form.Item label="Зображення" name="imageURL">
           <Upload
            customRequest={handleUploadImg}
            showUploadList={true}
            fileList={fileList}
            onChange={({ file }) => {
             setFileList([file]);
            }}
           >
            <Button>Завантажити зображення</Button>
           </Upload>
          </Form.Item>

          <Form.Item
           label="Опис"
           name="description"
           rules={[{ required: true, message: 'Введіть опис виробу' }]}
          >
           <Input.TextArea rows={4} placeholder="Короткий опис..." />
          </Form.Item>

          <Form.Item
           label="Розмір (мм)"
           name="size"
           rules={[{ required: true, message: 'Вкажіть розмір' }]}
          >
           <Input placeholder="Наприклад: 100x80x10" />
          </Form.Item>

          <Form.Item
           label="Матеріал"
           name="material"
           rules={[{ required: true, message: 'Вкажіть матеріал' }]}
          >
           <Input placeholder="PLA, PETG, силікон тощо" />
          </Form.Item>

          <Form.Item>
           <Button type="primary" htmlType="submit" loading={isLoading} block>
            {isEditMode ? 'Зберегти зміни' : 'Надіслати запит'}
           </Button>
          </Form.Item>
         </Form>
        </Card>
       </div>
    );
};

export default CustomProductCreatePage;
```

Лістинг Д.16 — Модальне вікно оформлення замовлення

```
interface Props {
   isVisible: boolean;
   onClose: () => void;
   onSubmit: () => void;
}

export const OrderModal = ({ isVisible, onClose, onSubmit }: Props) => {

   const [form] = Form.useForm();

   const {data:cart} = useGetCartQuery();
   const [placeUserOrder] = usePlaceOrderMutation();

   const handlePlaceOrder = async (values: OrderRequest) => {
      try {
         await placeUserOrder(values).unwrap();
         form.resetFields();
         onSubmit();
      } catch (error) {
         console.error("Ошибка:", error);
      }
   };

   return (
      <Modal title="Order" open={isVisible} onCancel={onClose} footer={null}>
         <div style={{ display: "flex", flexDirection: "column", gap: "10px" }}>
            {cart?.cartItems.map((item) => (
               <Card key={item.productId} style={{ width: 300, marginBottom: "10px" }}
bordered={false}>
                  <Image src={`https://sweetform-s3-
bucket.s3.amazonaws.com/${item.product.imageURL}`} alt={item.product.name} style={{
width: "100%" }} />
                  <Typography.Text strong>
                     <Link
href={`/moldPage/${item.product.name}`}>{item.product.name}</Link>
                  </Typography.Text>
                  <div>Ціна: {item.product.price} грн</div>
                  <div>Сума: {item.product.price * item.quantity} грн</div>
               </Card>
            ))}
         </div>

         <Form form={form} onFinish={handlePlaceOrder}>
            <Form.Item
               label="Address"
               name="address"
               rules={[{ required: true, message: "Введите адрес!" }]}
            >
               <Input placeholder="Enter your address" />
            </Form.Item>

            <Form.Item
```

```
            label="City"
            name="city"
            rules={[{ required: true, message: "Введите город!" }]}
          >
            <Input placeholder="Enter your city" />
          </Form.Item>

          <Form.Item
            label="Postal Code"
            name="postalCode"
            rules={[
              { required: true, message: "Введите почтовый индекс!" },
              { pattern: /^[0-9]{4,6}$/, message: "Введите корректный индекс (4-6
цифр)" }
            ]}
          >
            <Input placeholder="Enter your postal code" />
          </Form.Item>

          <Form.Item
            label="Country"
            name="country"
            rules={[{ required: true, message: "Введите страну!" }]}
          >
            <Input placeholder="Enter your country" />
          </Form.Item>

          <Form.Item>
            <Button type="primary" htmlType="submit">Place order</Button>
          </Form.Item>
        </Form>
      </Modal>
    );
};
```

Лістинг Д.17 — Інструмент адміністратора для керування продуктами

```
export default function ProductListPage() {
  const defaultValues = {
    name: "",
    productType: "",
    imageURL: "",
    material: "",
    theme: "",
    size: "",
    description: "",
    price: 1,
  } as Product;

  const [values, setValues] = useState<Product>(defaultValues);
  const [isVisible, setIsVisible] = useState(false);
  const [mode, setMode] = useState(Mode.Create);

  const [trigger, { data: products = [], isLoading, isError }] = useLazyGetProductsQuery();
```

```
const [deleteProduct] = useDeleteProductMutation();

useEffect(() => {
  trigger({});
}, []);

const handleDeleteProduct = async (productId: string) => {
  try {
    await deleteProduct({ productId }).unwrap();
  } catch (error) {
    console.error(error);
  }
};

const openModal = () => {
  setIsVisible(true);
};

const closeModal = () => {
  setValues(defaultValues);
  setIsVisible(false);
};

const openEditModal = (mold: Product) => {
  setMode(Mode.Edit);
  setValues(mold);
  setIsVisible(true);
};

if (isLoading) return <p>Завантаження...</p>;
if (isError || !products) return <div>Помилка при завантаженні продуктів</div>;

return (
  <div>
    <Button onClick={openModal}>Додати</Button>

    {products.map((product) => (
      <List
        key={product.id}
        style={{
          backgroundColor: 'white',
          borderRadius: 10,
          padding: 10,
          marginTop: 10,
        }}
      >
        <List.Item
          actions={[
            <a key="edit" onClick={() => openEditModal(product)}>редагувати</a>,
            <a key="delete" onClick={() => handleDeleteProduct(product.id)}>видалити</a>,
          ]}
        >
          <List.Item.Meta
            avatar={
```

```
            <Image
              style={{ width: 50 }}
              src={`https://sweetform-s3-bucket.s3.amazonaws.com/${product.imageURL}`}
              alt={product.name}
            />
          }
          title={
            <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center'
}}>
              <Link href={`/moldPage/${product.id}`}>{product.name}</Link>
              <Typography.Text strong>{product.price} грн.</Typography.Text>
            </div>
          }
          description={`Тип: ${product.productType} Тема: ${product.theme} Матеріал:
${product.material} Розмір: ${product.size}`}
        />
      </List.Item>
    </List>
  ))}

  <CreateUpdateMold
    mode={mode}
    products={products}
    values={values}
    isVisible={isVisible}
    handleCancel={closeModal}
  />
 </div>
 );
}
```

Лістинг Д.18 — Інструмент адміністратора для керування замовленнями

```
'use client';

import { useGetAllOrdersQuery, useUpdateOrderStatusMutation, useDeleteOrderMutation
} from "@/app/store/order/orders.slice";
import { Order, OrderStatus } from "@/app/Models/Order";
import { Button, List, Select, message } from "antd";

export default function AdminOrderList() {
 const { data: orders, isLoading } = useGetAllOrdersQuery();
 const [updateOrderStatus] = useUpdateOrderStatusMutation();
 const [deleteOrder] = useDeleteOrderMutation();

 const handleChangeStatus = async (orderId: string, newStatus: OrderStatus) => {
  try {
   await updateOrderStatus({ orderId, newStatus }).unwrap();
   message.success("Статус оновлено");
  } catch {
   message.error("Помилка при оновленні статусу");
  }
 };
```

```
  const handleDeleteOrder = async (orderId: string) => {
    try {
      await deleteOrder({ orderId }).unwrap();
      message.success("Замовлення видалено");
    } catch {
      message.error("Не вдалося видалити замовлення");
    }
  };

  if (isLoading) return <p>Завантаження...</p>;
  if (!orders || orders.length === 0) return <p>Замовлень не знайдено.</p>;

  const groupedOrders: Record<string, Order[]> = {};
  for (const order of orders) {
    if (!groupedOrders[order.userId]) groupedOrders[order.userId] = [];
    groupedOrders[order.userId].push(order);
  }

  return (
    <div>
      {Object.entries(groupedOrders).map(([userId, userOrders]) => (
        <List
          key={userId}
          header={<b>Користувач: {userId}</b>}
          bordered
          style={{ marginBottom: 24, backgroundColor: 'white', borderRadius: 10, padding: 10
}}
          dataSource={userOrders}
          renderItem={(order) => (
            <List.Item
              actions={[
                <Select
                  key="status"
                  style={{ width: 160 }}
                  value={order.orderStatus}
                  onChange={(value) => handleChangeStatus(order.orderId, value)}
                  options={[
                    { label: 'Очікує', value: OrderStatus.Pending },
                    { label: 'Оплачено', value: OrderStatus.Paid },
                    { label: 'Відправлено', value: OrderStatus.Shipped },
                    { label: 'Доставлено', value: OrderStatus.Delivered },
                    { label: 'Скасовано', value: OrderStatus.Canceled },
                  ]}
                />,
                <Button key="delete" danger type="link" onClick={() =>
handleDeleteOrder(order.orderId)}>
                  Видалити
                </Button>,
              ]}
            >
              <List.Item.Meta
                title={`Замовлення: ${order.orderId}`}
                description={
                  <>
```

```
                <p>Дата: {new Date(order.orderDate).toLocaleDateString('uk-UA')}</p>
                <p>Сума: {order.orderAmount.toFixed(2)} грн</p>
                <p>Адреса: {order.address}, {order.city}, {order.country},
{order.postalCode}</p>
                <ul style={{ marginTop: 5 }}>
                  {order.orderItems.map((item) => (
                    <li key={item.id}>{item.productName} — {item.quantity} шт</li>
                  ))}
                </ul>
              </>
            }
          />
        </List.Item>
      )}
    />
  ))}
 </div>
);
}
```

Лістинг Д.19 — Інструмент адміністратора для керування кастоними виробами

```
export default function AdminCustomProductListPage() {
 const defaultValues: CustomProduct = {
  id: '',
  name: '',
  productType: '',
  imageURL: '',
  material: '',
  size: '',
  description: '',
  status: 0,
  createdProductId: ''
 };

 const [values, setValues] = useState<CustomProduct>(defaultValues);
 const [isVisible, setIsVisible] = useState(false);
 const [mode, setMode] = useState(Mode.Create);
 const [pendingPublishProduct, setPendingPublishProduct] = useState<CustomProduct |
null>(null);

 const { data: products, isLoading, isError, refetch } = useGetCustomProductsQuery({});
 const [deleteCustomProduct] = useDeleteCustomProductMutation();
 const [updateCustomProduct] = useUpdateCustomProductMutation();

 const handleDeleteProduct = async (productId: string) => {
  try {
   await deleteCustomProduct({ productId }).unwrap();
  } catch (error) {
   console.error(error);
  }
 };
```

```
  const handleChangeStatus = async (productId: string, newStatus: ProductStatus) => {
    try {
      await updateCustomProduct({ productId, data: { status: newStatus } as any }).unwrap();
      message.success('Статус оновлено');
    } catch (error) {
      message.error('Помилка при зміні статусу');
    }
  };

  const handlePublishClick = (customProduct: CustomProduct) => {
    setMode(Mode.Create);
    setValues({
      ...customProduct,
      price: 1,
      theme: ''
    } as any);
    setPendingPublishProduct(customProduct);
    setIsVisible(true);
  };

  const closeModal = () => {
    setValues(defaultValues);
    setIsVisible(false);
    setPendingPublishProduct(null);
  };

  if (isLoading) return <p>Завантаження...</p>;
  if (isError || !products) return <div>Помилка завантаження продуктів</div>;

  return (
    <div>
      {products.map((product) => (
        <List
          key={product.id}
          style={{ backgroundColor: 'white', borderRadius: 10, padding: 10, marginTop: 10 }}
        >
          <List.Item
            actions={[
              <Select
                key="status"
                style={{ width: 140 }}
                value={product.status}
                onChange={(value) => handleChangeStatus(product.id, value)}
                options={[
                  { label: 'Очікує', value: ProductStatus.Pending },
                  { label: 'Схвалено', value: ProductStatus.Approved },
                  { label: 'Відхилено', value: ProductStatus.Rejected },
                  { label: 'Завершено', value: ProductStatus.Completed },
                ]}
              />,
              <a key="delete" onClick={() => handleDeleteProduct(product.id)}>Видалити</a>,
              <Button key="publish" onClick={() =>
handlePublishClick(product)}>Опублікувати</Button>
```

```
          ]}
        >
          <List.Item.Meta
            avatar={
              <Image
                style={{ width: 50 }}
                src={`https://sweetform-s3-bucket.s3.amazonaws.com/${product.imageURL}`}
                alt={product.name}
              />
            }
            title={
              <Link href={`/custom-product/${product.id}`}>{product.name}</Link>
            }
            description={`Тип: ${product.productType}, Матеріал: ${product.material},
Розмір: ${product.size}`}
          />
        </List.Item>
      </List>
    ))}

    <CreateUpdateMold
      mode={mode}
      products={products as any}
      values={values as any}
      isVisible={isVisible}
      handleCancel={closeModal}
    />
  </div>
 );
}
```

Лістинг Д.20 — Інструмент адміністратора для редагування продукту

```
interface Props {
 mode: Mode;
 products: Product[];
 values: Product;
 isVisible: boolean;
 handleCancel: () => void;
 onCreated?: (created: Product) => void;
}

export enum Mode {
 Create,
 Edit,
}

export const CreateUpdateMold = ({ mode, products, values, isVisible, handleCancel,
onCreated }: Props) => {
 const [form] = Form.useForm();
 const [fileList, setFileList] = useState<any[]>([]);
 const [previewUrl, setPreviewUrl] = useState<string | null>(null);

 const [createProduct] = useCreateProductMutation();
```

```
  const [updateProduct] = useUpdateProductMutation();

  const items = Array.from(new Set(products.map((p) => p.productType)));

  useEffect(() => {
    if (isVisible) {
      form.setFieldsValue({ ...values });
      setPreviewUrl(values.imageURL || null);
    }
  }, [values, isVisible]);

  const handleUpload = async (file: File, prefix: string) => {
    try {
      const result = await uploadFile(file, prefix);
      message.success("Файл завантажено");
      form.setFieldsValue({ imageURL: result });
      setPreviewUrl(result);
    } catch (error) {
      const err = axios.isAxiosError(error) ? error.message : "Помилка завантаження";
      message.error(err);
    }
  };

  const handleUploadImg = async ({ file, onSuccess, onError }: any) => {
    const productType = form.getFieldValue("productType");

    if (!productType) {
      message.warning("Сначала выберите тип продукта");
      return onError?.("Тип продукта не выбран");
    }

    const isNewType = !items.some(item => item.toLowerCase() ===
productType.toLowerCase());

    if (isNewType) {
      await createFolder(productType);
      message.success("Папка создана");
    }

    await handleUpload(file, productType);
  };

const createDefaultProduct = async (formValues: Product) => {
  const created = await createProduct(formValues).unwrap();
  message.success("Продукт створено!");
  if (onCreated) {
    onCreated(created);
  }
};

  const handleOnOk = async () => {
    try {
      const formValues = await form.validateFields();
      if (mode === Mode.Create) {
```

```
        await createDefaultProduct(formValues);
      } else {
        if (!values.id) {
          return message.error("ID продукту не знайдено");
        }

        await updateProduct({ productId: values.id, data: formValues }).unwrap();
        message.success("Продукт оновлено!");
      }

      form.resetFields();
      setPreviewUrl(null);
      setFileList([]);
      handleCancel();
    } catch (error) {
      console.error(error);
      message.error("Помилка збереження");
    }
  };

  return (
    <Modal
      title={mode === Mode.Create ? "Створити новий продукт" : "Редагувати продукт"}
      open={isVisible}
      onOk={handleOnOk}
      onCancel={handleCancel}
    >
      {previewUrl && (
  <div style={{ textAlign: "center", marginBottom: 16 }}>
    <img
      src={`https://sweetform-s3-bucket.s3.amazonaws.com/${previewUrl}`}
      alt="Превью изображения"
      style={{ maxWidth: "100%", maxHeight: 250, objectFit: "contain", borderRadius: 8 }}
    />
  </div>
)}

      <Form form={form} layout="vertical">
        <Form.Item label="Назва" name="name" rules={[{ required: true, message: "Введите
название" }]}>
          <Input placeholder="Назва" />
        </Form.Item>

        <Form.Item
          label="Тип продукту"
          name="productType"
          rules={[{ required: true, message: "Введіть або виберіть тип продукту" }]}
        >
          <Select
            style={{ width: 300 }}
            placeholder="Введіть або виберіть тип продукту"
            dropdownRender={(menu) => (
              <>
                {menu}
```

```
            <Divider style={{ margin: '8px 0' }} />
            <Space style={{ padding: '0 8px 4px' }}>
              <Input
                placeholder="Введіть новий тип"
                value={form.getFieldValue("productType")}
                onChange={(e) => form.setFieldValue("productType", e.target.value)}
                onKeyDown={(e) => e.stopPropagation()}
              />
            </Space>
          </>
        )}
        options={items.map((item) => ({ label: item, value: item }))}
      />
    </Form.Item>

    <Form.Item label="Зображення" name="imageURL">
      <Upload
        customRequest={handleUploadImg}
        showUploadList={true}
        fileList={fileList}
        onChange={({ fileList }) => setFileList(fileList)}
      >
        <Button>Завантажити зображення</Button>
      </Upload>
    </Form.Item>

    <Form.Item label="Матеріал" name="material" rules={[{ required: true, message:
"Введите материал" }]}>
      <Input.TextArea placeholder="Материал" autoSize={{ minRows: 2, maxRows: 3 }}
/>
    </Form.Item>

    <Form.Item label="Тема" name="theme" rules={[{ required: true, message: "Введите
тему" }]}>
      <Input.TextArea placeholder="Тема" autoSize={{ minRows: 2, maxRows: 3 }} />
    </Form.Item>

    <Form.Item label="Розмір" name="size" rules={[{ required: true, message: "Введите
размер" }]}>
      <Input.TextArea placeholder="Размер" autoSize={{ minRows: 2, maxRows: 3 }} />
    </Form.Item>

    <Form.Item label="Опис" name="description" rules={[{ required: true, message:
"Введите описание" }]}>
      <Input.TextArea placeholder="Описание" autoSize={{ minRows: 3, maxRows: 5 }}
/>
    </Form.Item>

    <Form.Item label="Ціна" name="price" rules={[{ required: true, message: "Введите
цену" }]}>
      <InputNumber min={1} max={99999} style={{ width: "100%" }}
placeholder="Цена" />
    </Form.Item>
  </Form>
```

```
    </Modal>
  );
};
```

Лістинг Д.21 — Головний макет сторінки з Redux

```
export default function RootLayout({ children }: { children: React.ReactNode }) {

    return (
        <html lang="en">
            <body>
                <Provider store={store}>
                    <AppLayout>{children}</AppLayout>
                </Provider>
            </body>
        </html>
    );
}
```

Лістинг Д.22 – Навігаційне меню

```
export default function Navigation() {
  const [isLoginModalOpen, setIsLoginModalOpen] = useState(false);
  const [isRegistrateModalOpen, setIsRegistrateModalOpen] = useState(false);
  const [isCartOpen, setIsCartOpen] = useState(false);

  const token = useAppSelector((state) => state.auth.accessToken);
  const roles = useAppSelector((state) => state.auth.roles);

  const isLoggedIn = !!token;
  const isAdmin = roles?.includes("Admin");

  const [logout] = useLogoutMutation();

  const leftItems = [
    {
      key: "home",
      label: <Link style={{ color: "white" }} href="/">Головна</Link>,
    },
    {
      key: "molds",
      label: <Link style={{ color: "white" }} href="/products">Продукти</Link>,
    },
  ];

  const rightItems = [];

  if (!isLoggedIn) {
    rightItems.push({
      key: "login",
      label: (
        <span style={{ color: "white" }} onClick={() => setIsLoginModalOpen(true)}>
          Login
        </span>
```

```
    ),
  });
} else {
  rightItems.push(
    {
      key: "profile",
      label: (
        <Link href="/profile" style={{ color: "white" }}>
          <UserOutlined style={{ fontSize: 18 }} />
        </Link>
      ),
    },
    {
      key: "cart",
      label: (
        <span style={{ color: "white" }} onClick={() => setIsCartOpen(true)}>
          <ShoppingCartOutlined style={{ fontSize: 18 }} />
        </span>
      ),
    },
    {
      key: "logout",
      label: (
        <span style={{ color: "white" }} onClick={() => logout()}>
          Вийти
        </span>
      ),
    }
  );

  if (isAdmin) {
    rightItems.push({
      key: "admin",
      label: <span style={{ color: "white" }}>Адмін панель</span>,
      children: [
        {
          key: "admin-products",
          label: <Link href="/admin/product-list">Продукти</Link>,
        },
        {
          key: "admin-orders",
          label: <Link href="/admin/order-list">Замовлення</Link>,
        },
        {
          key: "admin-custom",
          label: <Link href="/admin/custom-products-list">Кастом</Link>,
        },
      ],
    });
  }
}

return (
  <>
```

```
    <div style={{ display: "flex", justifyContent: "space-between", backgroundColor: "#67564A" }}>
      <Menu mode="horizontal" items={leftItems} style={{ backgroundColor: "#67564A", flex: 1 }} />
      <Menu
        mode="horizontal"
        items={rightItems}
        style={{
          backgroundColor: "#67564A",
          flex: 1,
          display: "flex",
          justifyContent: "flex-end",
        }}
      />
    </div>

    <CartDrawer isVisible={isCartOpen} onClose={() => setIsCartOpen(false)} />

    <LoginModal
      isVisible={isLoginModalOpen}
      onClose={() => setIsLoginModalOpen(false)}
      openRegisterModal={() => {
        setIsLoginModalOpen(false);
        setIsRegistrateModalOpen(true);
      }}
    />

    <RegisterModal
      isVisible={isRegistrateModalOpen}
      onClose={() => setIsRegistrateModalOpen(false)}
      openLoginModal={() => {
        setIsRegistrateModalOpen(false);
        setIsLoginModalOpen(true);
      }}
    />
   </>
  );
}
```

Лістинг Д.23 — Метод SendConfirmationEmail

```
                public async Task<bool> SendConfirmationEmail(string toEmail, string token)
                {
                        try
                        {
                                var message = new MimeMessage();
                                message.From.Add(new MailboxAddress(_emailSettings.SenderName, _emailSettings.SenderEmail));
                                message.To.Add(new MailboxAddress("", toEmail));
                                message.Subject = "Підтвердження реєстрації";

                                string confirmUrl = $"http://localhost:3000/confirm-email?token={token}";
```

```
                        string templatePath = Path.Combine(_templatePath,
"EmailConfirmation.html");

                        if (!File.Exists(templatePath))
                        {
                                throw new ApplicationException($"Помилка: Файл
шаблону не знайдено:{templatePath}");
                        }

                        string htmlBody = await File.ReadAllTextAsync(templatePath,
Encoding.UTF8);
                        htmlBody = htmlBody.Replace("{{Name}}", toEmail)
                                                .Replace("{{ConfirmUrl}}",
confirmUrl);
                        message.Body = new TextPart("html") { Text = htmlBody };

                        return await SendEmailAsync(message);
                }
                catch (Exception ex)
                {
                        throw new ApplicationException("Помилка під час
надсилання листа.", ex);
                }
        }

public async Task<bool> SendResetPasswordEmail(string toEmail, string token)
        {
                try
                {
                        var message = new MimeMessage();
                        message.From.Add(new
MailboxAddress(_emailSettings.SenderName, _emailSettings.SenderEmail));
                        message.To.Add(new MailboxAddress("", toEmail));
                        message.Subject = "Скидання пароля";

                        string resetUrl = $"http://localhost:3000/reset-
password?token={token}";

                        string templatePath = Path.Combine(_templatePath,
"ResetPasswordTemplate.html");

                        if (!File.Exists(templatePath))
                        {
                                throw new ApplicationException($"Помилка: Файл
шаблону не знайдено:{templatePath}");
                        }

                        string htmlBody = await File.ReadAllTextAsync(templatePath,
Encoding.UTF8);

                        htmlBody = htmlBody.Replace("{{Name}}", toEmail)
                                                .Replace("{{ResetUrl}}",
resetUrl);
```

```
                message.Body = new TextPart("html") { Text = htmlBody };

                Console.WriteLine(templatePath);

                return await SendEmailAsync(message);
            }
            catch (Exception ex)
            {
                throw new ApplicationException("Помилка під час
надсилання листа.", ex);
            }
        }
```

Лістинг Д.24 — Метод SendResetPasswordEmail

```
        public async Task<bool> SendResetPasswordEmail(string toEmail, string
token)
        {
            try
            {
                var message = new MimeMessage();
                message.From.Add(new
MailboxAddress(_emailSettings.SenderName, _emailSettings.SenderEmail));
                message.To.Add(new MailboxAddress("", toEmail));
                message.Subject = "Скидання пароля";

                string resetUrl = $"http://localhost:3000/reset-
password?token={token}";

                string templatePath = Path.Combine(_templatePath,
"ResetPasswordTemplate.html");

                if (!File.Exists(templatePath))
                {
                    throw new ApplicationException($"Помилка: Файл
шаблону не знайдено:{templatePath}");
                }

                string htmlBody = await File.ReadAllTextAsync(templatePath,
Encoding.UTF8);

                htmlBody = htmlBody.Replace("{{Name}}", toEmail)
                                   .Replace("{{ResetUrl}}",
resetUrl);

                message.Body = new TextPart("html") { Text = htmlBody };

                Console.WriteLine(templatePath);

                return await SendEmailAsync(message);
            }
            catch (Exception ex)
            {
```

```
                throw new ApplicationException("Помилка під час
надсилання листа.", ex);
            }
        }
```

Лістинг Д.25 — Метод SendOrderConfirmationMail

```
        public async Task<bool> SendOrderConfirmationMail(string toEmail, Order
order)
        {
            var message = new MimeMessage();
            message.From.Add(new
MailboxAddress(_emailSettings.SenderName, _emailSettings.SenderEmail));

            message.To.Add(new MailboxAddress("", toEmail));
            message.Subject = "Order";

            string templatePath = Path.Combine(_templatePath,
"OrderTemplate.html");

            if (!File.Exists(templatePath))
            {
                throw new ApplicationException($"Помилка: Файл шаблону
не знайдено:{templatePath}");
            }

            string htmlBody = await File.ReadAllTextAsync(templatePath,
Encoding.UTF8);

            htmlBody = htmlBody.Replace("{{OrderId}}",
order.OrderId.ToString())
                               .Replace("{{Name}}", toEmail)
                               .Replace("{{Amount}}",
order.OrderAmount.ToString())
                               .Replace("{{OrderItems}}",
GenerateOrderItemsHtml(order.OrderItems));


            message.Body = new TextPart("html") {Text = htmlBody };

            return await SendEmailAsync(message);
        }
```

Лістинг Д.26 — Метод SendCustomProductNotificationToAdmin

```
        public async Task<bool>
SendCustomProductNotificationToAdmin(CustomProduct product, string userEmail)
        {
            try
            {

                var message = new MimeMessage();
                message.From.Add(new
MailboxAddress(_emailSettings.SenderName, _emailSettings.SenderEmail));
```

```
                message.To.Add(new MailboxAddress("Admin",
_emailSettings.SenderEmail));
                message.Subject = "Нове кастомне замовлення від
користувача";

                string templatePath = Path.Combine(_templatePath,
"CustomProductNotification.html");

                if (!File.Exists(templatePath))
                {
                        throw new ApplicationException($"Помилка: Файл
шаблону не знайдено: {templatePath}");
                }

                string htmlBody = await File.ReadAllTextAsync(templatePath,
Encoding.UTF8);

                string imageUrl = $"https://sweetform-s3-
bucket.s3.amazonaws.com/{product.ImageURL}";

                htmlBody = htmlBody.Replace("{{UserEmail}}", userEmail)
                                                .Replace("{{ProductName}}",
product.Name)
                                                .Replace("{{Description}}",
product.Description)
                                                .Replace("{{ProductType}}",
product.ProductType)
                                                .Replace("{{Material}}",
product.Material)
                                                .Replace("{{Size}}",
product.Size)
                                                .Replace("{{Status}}",
product.Status.ToString())
                                                .Replace("{{CreatedAt}}",
product.CreatedAt.ToString("dd.MM.yyyy HH:mm"))
                                                .Replace("{{ImageUrl}}",
imageUrl);

                message.Body = new TextPart("html") { Text = htmlBody };

                return await SendEmailAsync(message);
            }
            catch (Exception ex)
            {
                throw new ApplicationException("Помилка під час
надсилання листа адміну.", ex);
            }
        }
```