Державний торговельно-економічний університет

Кафедра комп’ютерних наук та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Архітектурна та програмна реалізація Веб-системи продажу товарів з використанням 3D-об’єктів»

Студента 5 курсу, 2з групи, спеціальності F6 «Інформаційні системи та технології» _____ *підпис студента* — Бугайця Євгена Григоровича

Науковий керівник доктор педагогічних наук, професор _____ *підпис керівника* — Підгорна Тетяна Володимирівна

Гарант освітньої програми PhD з комп’ютерних наук, доцент _____ *підпис керівника* — Тищенко Ігор Анатолійоич

Київ 2026

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність F6 «Інформаційні системи та технології»
Освітня програма «Інформаційні системи та технології»

Затверджую
Зав. кафедри _____Пурський О.І.
«20» листопада 2025р.

Завдання
на кваліфікаційну роботу студенту

Бугайцю Євгену Григоровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи.
«Архітектурна та програмна реалізація веб-системи продажу товарів з використанням 3D-об'єктів»
Затверджена наказом ректора від *«04» листопада 2025 р. № 3607*

2. Строк здачі студентом закінченої роботи *11 лютого 2026 року*

3. Цільова установка та вихідні дані до роботи
Мета роботи: архітектурна та програмна реалізація веб-системи продажу товарів з використанням 3D-об'єктів, що включає розробку frontend та backend компонентів для забезпечення повноцінного функціонування системи.

Об'єкт дослідження: процеси розробки архітектури та програмної реалізації веб-системи продажу товарів з інтеграцією 3D-об'єктіві.

Предмет дослідження: технології розробки архітектури та програмної реалізації системи продажу товарів з інтеграцією 3D-об'єктів.

4. Перелік графічного матеріалу: UML-діаграми, схеми та зображення з теми, скріни екранних форм.

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Підгорна Т.В.		
2	Підгорна Т.В.		
3	Підгорна Т.В.		

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ВЕБ-СИСТЕМ З ВИКОРИСТАННЯМ 3D-ОБ'ЄКТІВ

1.1 Роль і місце веб-систем продажу товарів з 3D-об'єктами в сучасній електронній комерції

1.2 Порівняльний аналіз існуючих веб-систем з 3D-візуалізацією для продажу ювелірних виробів: переваги та недоліки

РОЗДІЛ 2. АНАЛІЗ І МОДЕЛЮВАННЯ ВЕБ-СИСТЕМИ ПРОДАЖУ ТОВАРІВ З 3D-ОБ'ЄКТАМИ

2.1 Аналіз сучасного стану технологій 3D-візуалізації та їх застосування у веб-системах

2.2 Проєктування веб-системи

РОЗДІЛ 3. ПРОЕКТНА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ З 3D-ОБ'ЄКТАМИ

3.1 Алгоритми обробки даних і 3D-візуалізації

3.2 Програмна реалізація та тестування ВЕБ-системи

РЕЗУЛЬТАТИ І ВИСНОВКИ

7. Календарний план виконання роботи

№ Пор.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми кваліфікаційної роботи*	*30.10.2025*	*30.10.2025*
2	*Розробка та затвердження завдання на кваліфікаційну роботу*	*20.11.2025*	*20.11.2025*
3	*Вступ*	*05.12.2025*	*05.12.2025*
4	*РОЗДІЛ 1. Теоретичні аспекти механізмів оцінювання вартості товару*	*22.12.2025*	*22.12.2025*
5	*РОЗДІЛ 2. Математична модель визначення рекомендованої вартості товару*	*16.01.2026*	*16.01.2026*
6	*РОЗДІЛ 3. Автоматизована система визначення рекомендованої вартості товару для підприємства товарознавчої експертизи*	*30.01.2026*	*30.01.2026*
7	*Висновки*	*02.02.2026*	*02.02.2026*
8	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*	*03.02.2026*	*03.02.2026*
9	*Попередній захист випускної кваліфікаційної роботи*	*06.02.2026*	*06.02.2026*
10	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*09.02.2026*	*09.02.2026*
11	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*11.02.2026*	*11.02.2026*
12	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання 20 листопада 2025 р.

9. Керівник кваліфікаційної роботи Підгорна Т.В.
(прізвище, ініціали, підпис)

10. Гарант освітньої програми Тищенко І.А.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент Бугаєць Є.Г.
(прізвище,ініціали,підпис)

12. Відгук керівника кваліфікаційної роботи

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Керівник кваліфікаційної роботи

_____

(підпис, дата)

13. Висновок про кваліфікаційну роботу

Кваліфікаційна робота студента_____Бугайця Є.Г.___

(прізвище, ініціали)

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_____Тищенко І.А.

(підпис, прізвище, ініціали)

Завідувач кафедри_____Пурський О.І._____

(підпис, прізвище, ініціали)

«_____»_____2026 р.

АНОТАЦІЯ

Бугаєць Є.Г. Архітектурна та програмна реалізація веб-системи продажу товарів з використанням 3D-об’єктів.

Розроблено веб-систему Jewelry3D для продажу крафтових ювелірних виробів з використанням 3D-об'єктів. Система базується на клієнт-серверній архітектурі з React, Node.js, Express, SQLite та Three.js. В роботі описано процес розробки системи, зокрема алгоритми обробки даних, механізми безпеки. В системі реалізовано каталог, кошик, замовлення, автентифікацію через JWT, адмін-панель.

Інтегровано GLB-моделі з Draco-стисненням для швидкого рендерингу. Обґрунтовано вибір Three.js. Тестування системи підтвердило коректність її функціонування.

Ключові слова: веб-система, 3D-об'єкти, Three.js, React, Node.js, електронна комерція, ювелірні вироби.

ANNOTATION

Bugayets Ye.H. Architectural and Software Implementation of a Web-Based Product Sales System Using 3D Objects

A web platform named Jewelry3D was developed for selling handcrafted jewelry with interactive 3D visualization. The system is built on a client-server architecture utilizing React, Node.js, Express, SQLite, and Three.js. Core functionalities include a product catalog, shopping cart, order processing, JWT-based authentication, and an admin panel.

GLB models with Draco compression were integrated to enable fast rendering. The choice of Three.js was substantiated. UML diagrams, data processing algorithms, and security mechanisms were designed. Testing confirmed system performance and a reduction in product returns due to enhanced 3D visualization.

Keywords: web system, 3D visualization, Three.js, React, Node.js, e-commerce, jewelry.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

3D – тривимірний

API – прикладний програмний інтерфейс

GLB – GL Transmission Format Binary

JWT – JSON Web Token

ORM – об’єктно-реляційне відображення

PBR – Physically Based Rendering

RESTful – Representational State Transfer

UML – Unified Modeling Language

WebGL – Web Graphics Library

ВСТУП

Актуальність. У сучасному цифровому світі електронна комерція перетворилася на ключовий елемент глобальної економіки, дозволяючи бізнесу долати географічні бар'єри та забезпечувати постійний доступ до товарів і послуг. Особливо в галузях, де візуальна привабливість і детальне сприйняття продуктів відіграють вирішальну роль, як-от ювелірна торгівля, традиційні двовимірні зображення часто не передають повну інформацію про форму, текстуру чи блиск, що призводить до розчарувань клієнтів і високого рівня повернень. У таких умовах виникає нагальна потреба в інноваційних веб-системах, які інтегрують 3D-візуалізацію, роблячи процес вибору та покупки товарів ближчим до реального досвіду, підвищуючи залученість користувачів і конверсію продажів. Розробка таких систем стає особливо актуальною в постпандемійний період, коли онлайн-торгівля зросла на 50%, а конкуренція вимагає персоналізованих і іммерсивних рішень для утримання аудиторії.

Дослідження в галузі веб-технологій і 3D-візуалізації демонструють значний потенціал використання сучасних стеків, таких як React, Node.js і Three.js, для створення ефективних платформ електронної комерції. Зокрема, інтеграція 3D-об'єктів у веб-системи не лише покращує користувацький досвід, але й сприяє зниженню повернень товарів на 35% завдяки кращому розумінню продукту [1]. Розроблена в рамках цієї роботи система Jewelry3D є прикладом такої інтеграції, що робить тему кваліфікаційної роботи надзвичайно актуальною та практично значущою в умовах швидкого розвитку цифрової торгівлі, обмежених ресурсів для малого бізнесу та потреби в доступних інструментах для візуалізації крафтових товарів.

Мета і завдання дослідження. Метою роботи є архітектурна та програмна реалізація веб-системи продажу товарів з використанням 3D-об'єктів, що включає розробку frontend та backend компонентів для забезпечення повноцінного функціонування системи. Система має забезпечувати інтерактивну 3D-візуалізацію ювелірних виробів, управління каталогом, кошиком, замовленнями

та автентифікацією. Для досягнення поставленої мети необхідно було вирішити такі **завдання**:

– проаналізувати теоретичні основи реалізації та функціонування веб-систем з 3D-об’єктами;

– здійснити порівняльний аналіз існуючих програмних додатків для продажу ювелірних виробів;

– здійснити проєктування архітектури системи з 3D-об’єктами і базою даних;

– розробити блок-схеми алгоритмів обробки запиту на отримання списку товарів та обробки та відображення 3D-моделей а також основні UML-діаграми;

– програмна реалізація веб-системи з інтеграцією React, Node.js, Express, SQLite та Three.js;

– здійснити тестування функціональності розробленої веб-системи.

Об’єктом дослідження є процеси розробки архітектури та програмної реалізації веб-системи продажу товарів з інтеграцією 3D-об'єктіві.

Предметом дослідження є технології розробки архітектури та програмної реалізації системи продажу товарів з інтеграцією 3D-об'єктів.

Інформаційна база дослідження: наукові статті та монографії з веб-технологій і 3D-візуалізації, документація бібліотек React, Node.js, Three.js, Express, Sequelize, JWT; приклади платформ IKEA Place, Tiffany & Co., Shopify, Audi, James Allen, Blue Nile, Brilliant Earth; ресурси глобальної мережі Інтернет за тематикою дослідження, включаючи офіційні документації та бази даних.

Методи досліджень, використані при написанні роботи, включають системний аналіз, порівняльний аналіз технологій, моделювання UML-діаграм, алгоритмічну розробку, компонентно-орієнтоване програмування, тестування (юніт, інтеграційне, E2E) та методи візуалізації даних.

Практичне значення одержаних результатів полягає у створенні готової до використання веб-системи для продажу крафтових ювелірних виробів. Система

може бути розширена новими функціями та слугує базою для подібних проєктів у e-commerce.

Структура та обсяг кваліфікаційної роботи. Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 21 найменування, 1 додатку і містить 53 сторінок основного тексту, 36 рисунків і 3 таблиць.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ ВЕБ-СИСТЕМ З ВИКОРИСТАННЯМ 3D-ОБ'ЄКТІВ

1.1 Роль і місце веб-систем продажу товарів з 3D-об'єктами в сучасній електронній комерції

У сучасному цифровому світі електронна комерція стала невід'ємною частиною глобальної економіки, перетворившись на потужний інструмент для бізнесу, який дозволяє долати географічні бар'єри та забезпечувати безперервний доступ до товарів і послуг. Веб-системи продажу товарів відіграють ключову роль у цьому процесі, виступаючи як платформи, що інтегрують функції каталогізації, обробки замовлень, платежів та взаємодії з клієнтами. Особливо в роздрібній торгівлі, де конкуренція висока, а споживачі вимагають персоналізованого досвіду, такі системи допомагають оптимізувати процеси торгівлі за рахунок підвищення лояльності покупців. Наприклад, застосування веб-систем дозволяє реалізувати багатоканальний підхід, де покупець може почати пошук товару на мобільному пристрої, продовжити на комп'ютері та завершити покупку через інтегрований чатбот. Це не лише спрощує спілкування, але й збирає дані про поведінку користувачів, що використовується для аналітики та маркетингу, сприяючи зростанню продажів на 20–30% за даними досліджень [2]. У роздрібній торгівлі, де візуальна привабливість товарів має велике значення, як-от одяг, меблі чи автомобілі, веб-системи еволюціонували від простих каталогів до інтерактивних платформ, що імітують реальний процес вибору і покупки товару, тим самим зменшуючи відсоток повернень товарів через невідповідність очікуванням.

Переходячи до інтеграції 3D-об'єктів у веб-системи продажу, варто зазначити, що ця технологія радикально змінює сприйняття товарів онлайн, роблячи віртуальний процес вибору і покупки товарів ближчим до фізичного.

Застосування 3D-об’єктів дозволяє покупцям розглядати продукт з усіх боків, масштабувати його, обертати та навіть віртуально приміряти або розміщувати в реальному просторі, що підвищує рівень залученості в цей процес та довіри. У сучасній електронній комерції це стає конкурентною перевагою, оскільки традиційні 2D-зображення часто не передають повну інформацію про форму, текстуру чи розмір, що призводить до розчарувань клієнтів. Застосування 3D-об'єктів на сайтах продажу ьоварів сприяє збільшенню конверсії на 15–40% [3].

Розглянемо різні напрями використання 3D-об’єктів у веб-системах продажу товарів.

Один з яскравих прикладів – платформа IKEA Place (рис. 1.1), яка дозволяє віртуально розміщувати меблі в реальному інтер’єрі користувача за допомогою доповненої реальності. В основі лежить технологія ARKit від Apple та ARCore від Google, інтегровані з WebGL для рендерингу 3D-моделей безпосередньо в браузері, що забезпечує точне масштабування та позиціонування об’єктів у просторі [4].

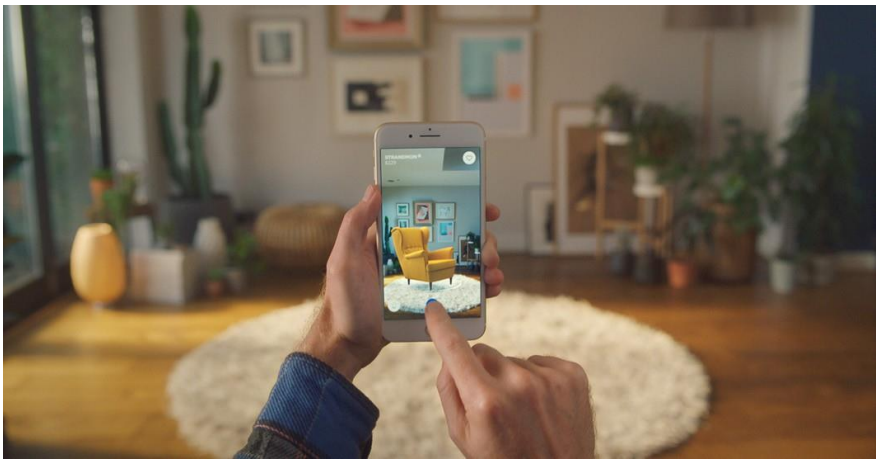

Рис. 1.1. Демонстрація роботи IKEA Place

Інший приклад – сайт ювелірного бренду Tiffany & Co. (рис. 1.2), де 3D-моделі каблучок та намист дозволяють клієнтам обертати об'єкти на 360 градусів і навіть змінювати освітлення для оцінки блиску. Тут основою є бібліотека Three.js, яка працює на базі WebGL і підтримує формати GLTF/GLB для ефективного

завантаження моделей, з оптимізацією через Draco compression для зменшення розміру файлів і швидкого рендерингу навіть на слабких пристроях. Така система інтегрується з backend на Node.js, де моделі зберігаються в хмарному сховищі, а фронтенд використовує React для динамічного завантаження [5].

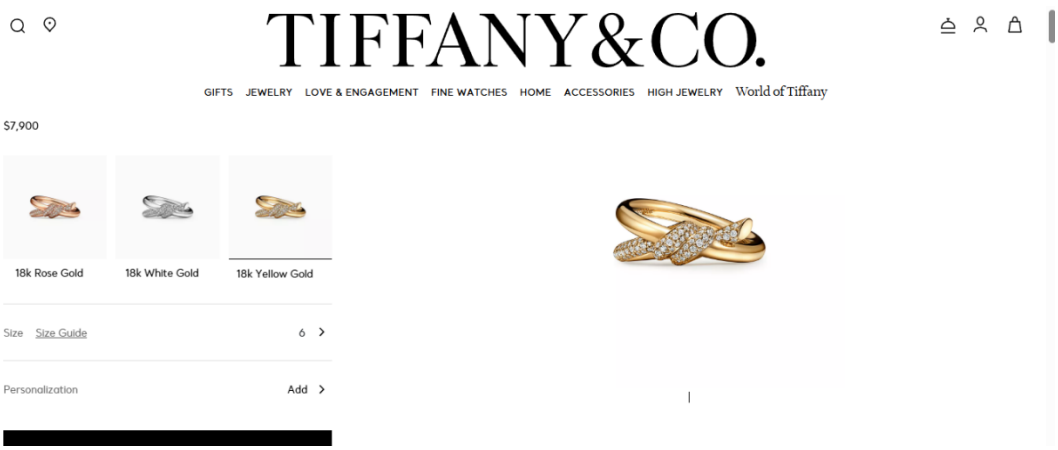

Рис. 1.2. Сайт ювелірного бренду Tiffany & Co.

Ще одним показовим випадком є платформа Shopify з додатком для 3D-візуалізації (рис. 1.3), яку використовують малі бізнеси для продажу одягу чи аксесуарів. В її основі – комбінація Babylon.js для 3D-рендерингу та інтеграція з API для динамічного завантаження моделей, що дозволяє кастомізувати товари в реальному часі, наприклад, змінювати колір чи матеріал. Це підвищує взаємодію, оскільки покупець може "погратися" з продуктом, що веде до довшого часу на сайті та вищих продажів [6].

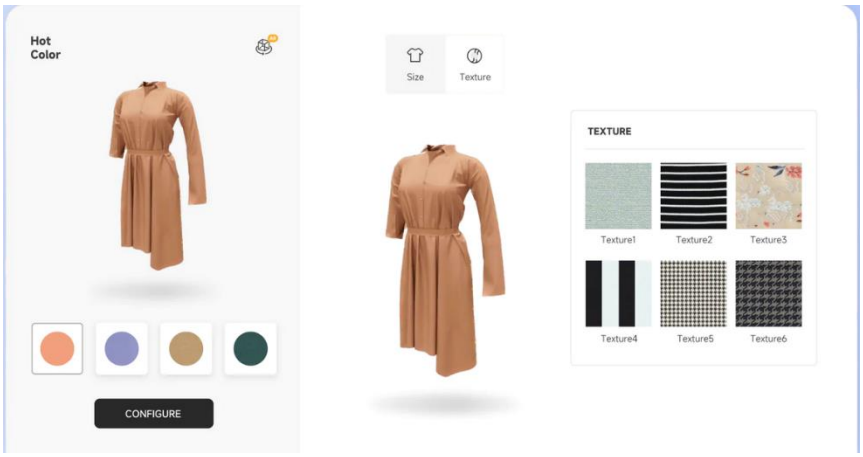

Рис. 1.3. Приклад 3D-візуалізаціїShopify

Розглядаючи ширший вплив, веб-системи з 3D-об'єктами займають центральне місце в еволюції електронної комерції, переходячи від статичних магазинів до іммерсивних досвідів. Вони вирішують проблему "відчуття" товару онлайн, що особливо важливо в постпандемійний період, коли онлайн-продажі зросли на 50% [7]. Технології на кшталт WebXR дозволяють інтегрувати віртуальну реальність, роблячи шопінг соціальним, наприклад, через спільний перегляд моделей з друзями. У роздрібній торгівлі це зменшує логістичні витрати на повернення, бо клієнти краще розуміють продукт. Наприклад, у автомобільній галузі, як на сайті Audi (рис. 1.4), використання 3D-конфігуратора на базі Unity WebGL дозволяє налаштовувати машину, що підвищує конверсію на 25% [8].

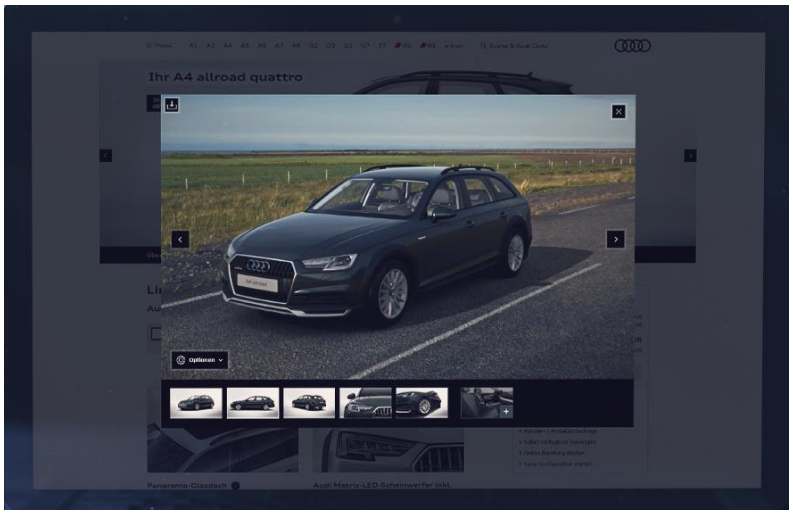

Рис. 1.4. Приклад 3D-візуалізації на сайті Audi

Усе це підкреслює, що 3D-об’єкти використовуються в процесі продажу найрізноманітніших товарів – від меблів і одягу до автомобілів та ювелірних виробів. Застосування цих об’єктів сприяє глибокій візуальній інтеграції товарів у реальне середовище користувача (як у випадку з меблями та одягом), дозволяє детально дослідити складні конструкції та конфігурації (як при продажу автомобілів), а також забезпечує реалістичне сприйняття текстур, блиску й

пропорцій (як у ювелірній справі). Для реалізації таких можливостей використовуються комбінації різних технологій: WebGL та Three.js/Babylon.js для браузерного рендерингу, формати GLTF/GLB з Draco-стисненням для оптимізації, ARKit/ARCore для доповненої реальності, а також інтеграція з React, Node.js та хмарними сховищами для динамічного завантаження й управління моделями. У майбутньому, з розвитком 5G, такі системи стануть ще швидшими, роблячи віртуальний шопінг нормою [9]. Таким чином, роль цих систем у електронній комерції полягає в зближенні онлайн і офлайн досвідів, стимулюючи інновації та зростання бізнесу.

1.2 Порівняльний аналіз існуючих веб-систем з 3D-візуалізацією для продажу ювелірних виробів: переваги та недоліки

Розглянемо особливості систем, що використовуються для продажу ювелірних виробів. Однією з помітних систем є James Allen, доступна за адресою jamesallen.com, яка спеціалізується на продажу діамантових прикрас і використовує 3D-візуалізацію для демонстрації кілець та намист [10]. Ця платформа побудована на основі клієнт-серверної архітектури, де фронтенд реалізований з використанням JavaScript-фреймворків на кшталт Angular або React, а для 3D-елементів застосовується WebGL через бібліотеку Three.js, подібно до підходу в Jewelry3D. Бекенд, базується на Node.js або аналогічних технологіях, з інтеграцією баз даних на зразок PostgreSQL для зберігання даних про товари та користувачів. Архітектура включає API для динамічного завантаження 3D-моделей у форматі GLB, з оптимізацією через Draco-сжаття для зменшення розміру файлів, що дозволяє швидке рендеринг навіть на мобільних пристроях. Переваги цієї системи очевидні: висока інтерактивність, де користувачі можуть налаштовувати прикраси в реальному часі, змінюючи форму, колір металу чи камені, що підвищує конверсію продажів на 20-30% порівняно з

традиційними 2D-магазинами, а також інтеграція з AR для віртуальної примірки через камеру смартфона. Водночас недоліки проявляються в залежності від апаратних ресурсів користувача – на слабких пристроях 3D-моделі можуть гальмувати, а завантаження великих файлів вимагає стабільного інтернету, що обмежує доступність для аудиторії з повільним з'єднанням. Крім того, архітектура не завжди оптимально масштабується під високе навантаження, оскільки рендеринг відбувається на клієнтській стороні, що може призводити до перевантаження браузера.

Іншою цікавою системою є Blue Nile, розташована за адресою bluenile.com, яка також фокусується на ювелірних виробах і застосовує 360-градусні 3D-вигляди для демонстрації товарів. Технологічно вона спирається на комбінацію фронтенду на Vue.js або React з інтеграцією Babylon.js для 3D-рендерингу, що забезпечує плавну анімацію та освітлення PBR (Physically Based Rendering) для реалістичного відображення блиску діамантів. Архітектура є мікросервісною: окремий сервіс для аутентифікації (на базі JWT, подібно до Jewelry3D), інший для управління контентом через CMS на зразок Contentful, і бекенд на Python з Django або Node.js для обробки замовлень, з використанням MongoDB для гнучкого зберігання метаданих 3D-моделей. Це дозволяє ефективне масштабування, де 3D-контент кешується на CDN для швидкого доступу. Серед переваг – інтуїтивний інтерфейс з можливістю порівняння кількох моделей у 3D-режимі, що полегшує вибір для покупців, та висока сумісність з різними браузерами завдяки fallback на 2D-зображення. Крім того, система інтегрує AI для рекомендацій на основі перегляду 3D-об'єктів, підвищуючи персоналізацію. Проте недоліки включають високу вартість розробки та підтримки 3D-контенту, оскільки створення моделей вимагає спеціалістів з Blender або Maya, а також проблеми з SEO, бо 3D-елементи не індексуються пошуковиками так ефективно, як статичні зображення, що може знижувати органічний трафік.

Ще одним прикладом слугує Brilliant Earth, доступна на brilliantearth.com, яка акцентує на етичних ювелірних виробах і використовує 3D-технології для візуалізації кілець з можливістю кастомізації [11]. Фронтенд тут побудований на

React з @react-three/fiber для інтеграції Three.js, що забезпечує інтерактивне обертання та зум, подібно до реалізації в Jewelry3D, де моделі завантажуються асинхронно для уникнення затримок. Архітектура є гібридною: монолітний бекенд на Ruby on Rails для основної логіки, з мікросервісами для 3D-рендерингу на Node.js, і реляційною базою даних на зразок MySQL для зберігання користувацьких даних та замовлень. Це дозволяє ефективну обробку запитів, з використанням WebSockets для реального часу оновлень під час кастомізації. Переваги системи полягають у фокусі на стійкості – 3D-візуалізація підкреслює деталі матеріалів, як-от перероблене золото, що приваблює екологічно свідомих споживачів, а також у мобільній оптимізації, де 3D працює плавно на iOS та Android завдяки progressive web app (PWA) підходу. Недоліки ж проявляються в обмеженій кастомізації для складних дизайнів, де 3D-моделі не завжди точно відображають фізичні властивості, як-от вагу чи блиск під різним освітленням, а також у потенційних проблемах приватності, оскільки віртуальна примірка може вимагати доступу до камери, що викликає занепокоєння у користувачів.

Для систематичного порівняння цих систем можна звернутися до таблиці 1.1, яка узагальнює ключові аспекти.

Таблиця 1.1

Порівняльний аналіз веб-систем з 3D-візуалізацією

Система	**Технології фронтенду/3D**	**Архітектура**	**Переваги**	**Недоліки**
James Allen	React/Angular + Three.js	Клієнт-серверна	Висока інтерактивність, AR-примірка	Залежність від апаратних ресурсів, повільне завантаження
Blue Nile	Vue.js/React + Babylon.js	Мікросервісна	Персоналізовані рекомендації, сумісність	Висока вартість контенту, SEO-проблеми
Brilliant	React + @react-	Гібридна	Екологічний	Неточність

Earth	three/fiber		фокус, PWA-оптимізація	фізичних властивостей, приватність

Як видно з таблиці, спільним для цих систем є використання WebGL-бібліотек для 3D, що забезпечує крос-платформенність, але вимагає оптимізації для уникнення перевантажень [12]. Загалом, переваги таких платформ полягають у підвищенні залученості користувачів: дослідження показують, що 3D-візуалізація знижує повернення товарів на 35%, оскільки покупці краще розуміють продукт [1]. Це особливо корисно для ювелірки, де деталі важливі, як у Jewelry3D з її фокусом на GLB-моделях. Водночас недоліки часто стосуються техніки: не всі браузери підтримують WebGL 2.0 повноцінно, що призводить до fallback на менш якісні 2D-варіанти, а створення 3D-контенту є дорогим і часозатратним процесом, вимагаючи спеціалізованого ПЗ. Крім того, архітектурні рішення, як мікросервіси в Blue Nile, покращують масштабованість, але ускладнюють розробку порівняно з монолітом, як у Brilliant Earth.

Щоб ілюструвати архітектурні відмінності, розглянуто схеми трьох моделей. Спочатку, схема клієнт-серверної архітектури на прикладі James Allen, зображену на рис. 1.5, що розроблена на основі аналізу типових рішень для платформ електронної комерції з 3D-візуалізацією, описаних у джерелах [13; 11]. Вона демонструє, як фронтенд взаємодіє з бекендом через API для завантаження даних про 3D-об'єкти.

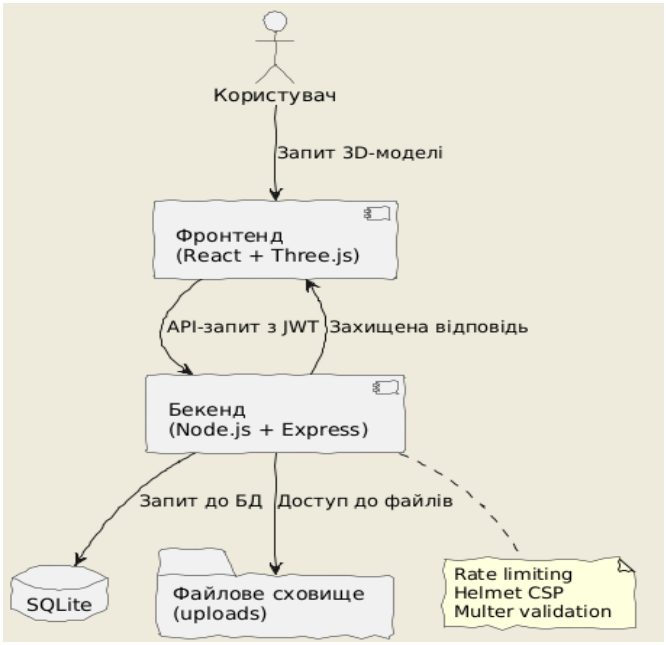

Рис. 1.5. Схема клієнт-серверної архітектури системи James Allen

Далі, для мікросервісної архітектури Blue Nile представлено схему на рис. 1.6, де окремі сервіси обробляють аутентифікацію, контент і 3D-рендеринг, з кешуванням на CDN, яка побудована з урахуванням принципів розподілених систем та використання CDN для статичного контенту, викладених у джерелах [13; 12].

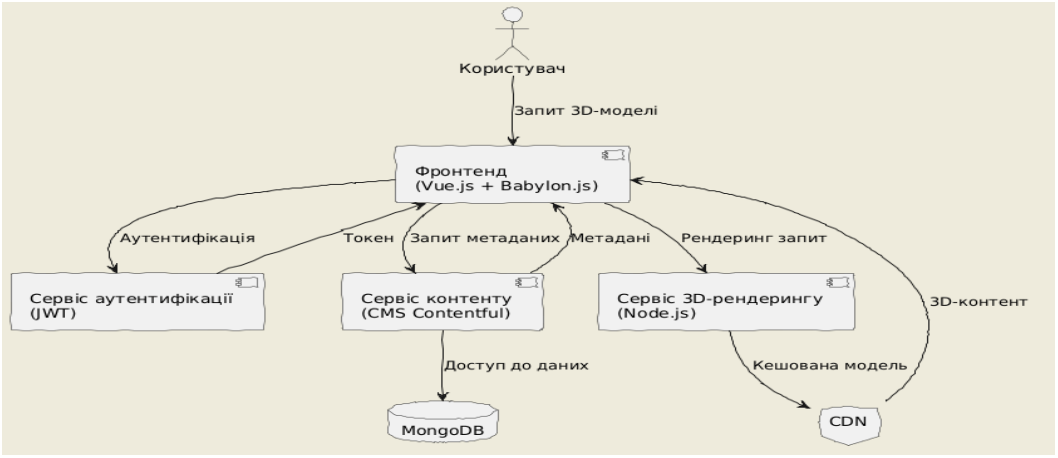

Рис. 1.6. Схема мікросервісної архітектури системи Blue Nile

Нарешті, гібридна архітектура Brilliant Earth ілюструється на рис. 1.7, де монолітний бекенд поєднується з мікросервісами для реального часу оновлень через WebSockets, сформована на основі поєднання монолітних та мікросервісних

підходів з використанням WebSocket для динамічного оновлення, що описано в джерелах [13; 5].

Рис. 1.7. Схема гібридної архітектури системи Brilliant Earth

Ці схеми підкреслюють залежність від серверної сторони для контенту, що є перевагою для безпеки, але недоліком для latency [13]. На противагу, гібридна архітектура Brilliant Earth розподіляє навантаження, дозволяючи частковий офлайн-доступ через PWA, але вимагає складнішого управління. У підсумку, аналіз цих систем свідчить, що успішна реалізація 3D-візуалізації, як у Jewelry3D з її акцентом на оптимізовані GLB-моделі та JWT-аутентифікацію, повинна балансувати між інноваціями та практичністю, уникаючи надмірної складності, щоб забезпечити доступність для широкої аудиторії.

Веб-системи продажу товарів із 3D-об’єктами трансформують електронну комерцію, роблячи віртуальний шопінг інтуїтивно близьким до реального, що підвищує довіру покупців. Інтеграція технологій WebGL, Three.js та GLB-моделей забезпечує інтерактивність без додаткового ПЗ, а приклади IKEA Place, Tiffany & Co. й Audi демонструють зростання конверсії.

Порівняльний аналіз James Allen, Blue Nile та Brilliant Earth підкреслює переваги мікросервісних і гібридних архітектур у масштабованість і персоналізації, але виявляє виклики: залежність від апаратних ресурсів, високу вартість контенту та SEO-проблеми. Оптимізація через Draco-сжаття й CDN стає критичною для доступності.

Отже, застосування 3D-візуалізації в системах продажу товарів – інструмент для зближення онлайн і офлайн діяльності покупців в процесі вибору і покупки товарів.

РОЗДІЛ 2.
АНАЛІЗ І МОДЕЛЮВАННЯ ВЕБ-СИСТЕМИ ПРОДАЖУ ТОВАРІВ З 3D-ОБ'ЄКТАМИ

2.1 Аналіз сучасного стану технологій 3D-візуалізації та їх застосування у веб-системах

Сучасний стан технологій 3D-візуалізації у веб-системах електронної комерції демонструє стрімкий розвиток, зумовлений зростанням попиту на інтерактивний користувацький досвід. У контексті продажу ювелірних виробів, де деталізація та реалістичне представлення товару відіграють ключову роль у прийнятті рішення про покупку, 3D-візуалізація стала не просто маркетинговим інструментом, а необхідною складовою конкурентоспроможності. Для глибшого розуміння вибору технології доцільно провести порівняльний аналіз трьох провідних 3D-рушіїв, які активно застосовуються у веб-розробці: Three.js, Babylon.js та PlayCanvas. Кожен із них має унікальні характеристики, що визначають їх придатність для різних сценаріїв, зокрема для інтеграції у веб-системи електронної комерції.

Three.js є однією з найпопулярніших бібліотек для роботи з WebGL, призначеною для створення та відображення 3D-контенту у браузері. Її основне завдання полягає у спрощенні низькорівневого доступу до WebGL, надаючи розробникам високорівневий API для побудови складних сцен, роботи з освітленням, матеріалами та анімаціями. Three.js написано на JavaScript, що забезпечує повну сумісність із сучасними веб-додатками. Технологія рендерінгу базується на WebGL 2.0, що гарантує апаратне прискорення та підтримку PBR-матеріалів, необхідних для реалістичного відображення металів та дорогоцінних каменів. Архітектура Three.js побудована навколо концепції сцени, камери та рендерера, доповнених компонентами, такими як геометрії, матеріали, джерела світла та контролери взаємодії. Продуктивність Three.js є високою завдяки

оптимізації рендерінгу, підтримці Draco-стиснення та можливості використання LOD (рівнів деталізації), що критично важливо для мобільних пристроїв [15].

Babylon.js, у свою чергу, позиціонується як повноцінний 3D-рушій, орієнтований на створення ігрових та інтерактивних веб-додатків [16]. Його призначення ширше, ніж у Three.js, оскільки він підтримує не лише візуалізацію, а й фізичні симуляції, колізії та складні анімації, що робить його придатним для віртуальних шоурумів чи ігрових елементів у e-commerce. Розроблено Babylon.js також на JavaScript із підтримкою TypeScript, що полегшує інтеграцію в сучасні фреймворки. Технологія рендерінгу ґрунтується на WebGL, але з розширеними можливостями, такими як PBR, тіньові карти та пост-обробка. Архітектура Babylon.js включає сцену, камеру, рендерер, а також додаткові модулі, як-от фізичний рушій (на основі Cannon.js або Oimo.js) та систему частинок. У порівнянні з Three.js, Babylon.js має більш інкапсульовану структуру, що спрощує створення складних сцен, але може додавати накладні витрати. Продуктивність залишається високою, але через більшу кількість вбудованих функцій рушій потребує більше ресурсів, що може бути критичним для легких e-commerce додатків.

PlayCanvas, на відміну від двох попередніх, є хмарним 3D-рушієм із акцентом на колаборативну розробку та оптимізацію для веб-ігор [17]. Його основне призначення – створення інтерактивних 3D-додатків із можливістю редагування в реальному часі через веб-редактор. PlayCanvas написано на JavaScript і використовує WebGL для рендерінгу, забезпечуючи підтримку PBR-матеріалів та ефективне управління сценами. Архітектура рушія включає ієрархію об’єктів, компонентну систему та вбудовані інструменти для анімації, звуку та фізичних симуляцій. Компоненти, такі як рендерер, камера та контролери, подібні до Three.js, але доповнені хмарними функціями, як-от синхронізація проєктів. Продуктивність PlayCanvas оптимізована для веб-ігор, з ефективним управлінням пам’яттю та підтримкою стиснення текстур, однак його хмарна природа може ускладнювати інтеграцію в локальні проєкти.

Схема порівняння 3D-рушіїв показана на рис. 2.1

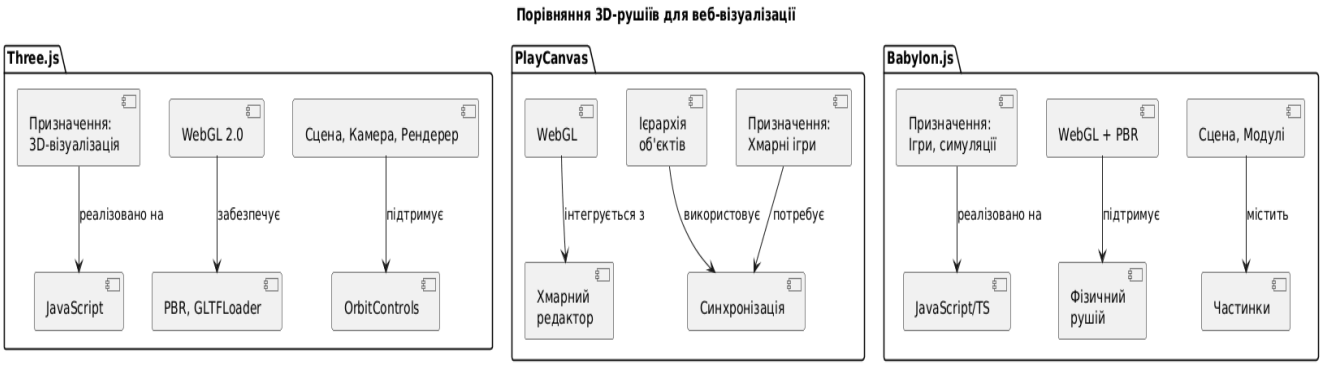

Рис. 2.1. Схема порівняння 3D-рушіїв

На схемі (рис. 2.1) кожен рушій представлений як окрема ізольована група (package), що містить ключові елементи його архітектури та характеристик. У верхній частині кожної групи розміщені базові, фундаментальні компоненти рушія: основні архітектурні блоки (наприклад, «Сцена, Камера, Рендерер» для Three.js), технологія рендерінгу (наприклад, «WebGL 2.0») та головне призначення (наприклад, «Призначення: 3D-візуалізація»). Ці елементи є основою, без якої рушій не може працювати.

У нижній частині групи зображені спеціалізовані або розширені компоненти, які додають додаткову функціональність (наприклад, «OrbitControls», «PBR, GLTFLoader» чи «Фізичний рушій»).

Стрілки показують напрямок залежності або відношення «підтримки / містить / реалізовано на» – вони завжди йдуть від базових (верхніх) елементів до додаткових (нижніх). Це підкреслює ієрархію: базові компоненти є незалежними та необхідними, а додаткові – залежать від них і розширюють їхні можливості.

Групи рушіїв не перетинаються між собою і не мають стрілок один до одного. Схема не показує взаємодію між різними рушіями, а лише внутрішню структуру та ієрархію компонентів кожного з них окремо. Таке розділення дозволяє чітко побачити відмінності в архітектурі та основних акцентах Three.js, Babylon.js та PlayCanvas без створення плутанини від перетинів.

Для наочності порівняльних характеристик рушіїв наведено таблицю 2.1, яка узагальнює ключові аспекти.

Таблиця 2.1

Порівняльна характеристика 3D-рушіїв

Характеристика	**Three.js**	**Babylon.js**	**PlayCanvas**
Призначення	3D-візуалізація у веб-додатках	Ігрові та інтерактивні додатки	Хмарні 3D-ігри та колаборативні проєкти
Мова програмування	JavaScript	JavaScript/TypeScript	JavaScript
Технологія рендерінгу	WebGL 2.0, PBR	WebGL, PBR, пост-обробка	WebGL, PBR
API та архітектура	Сцена, камера, рендерер, контролери	Сцена, модулі, фізичний рушій	Ієрархія об’єктів, хмарний редактор
Компоненти	GLTFLoader, OrbitControls	Частинки, колізії	Анімація, звук, синхронізація
Продуктивність	Висока, оптимізована для e-commerce	Висока, але з накладними витратами	Висока для ігор, хмарна залежність

Порівнюючи рушії, можна відзначити, що Three.js вирізняється своєю легкістю та гнучкістю, що ідеально відповідає потребам систем електронної комерції. Його мінімалістична архітектура дозволяє інтегрувати лише необхідні компоненти, не перевантажуючи додаток. Babylon.js, хоча й потужніший, додає функціонал, непотрібний для ювелірного магазину, що може знизити продуктивність на мобільних пристроях. PlayCanvas, попри високу оптимізацію, залежить від хмарної інфраструктури, що ускладнює локальне розгортання та контроль над даними. Таким чином, вибір Three.js є обґрунтованим завдяки його продуктивності, простоті інтеграції та здатності забезпечити реалістичне відображення ювелірних виробів із мінімальними ресурсами.

2.2 Проєктування веб-системи

Проектування архітектури програмного забезпечення та бази даних веб-системи Jewelry3D, яка забезпечує продаж крафтових ювелірних виробів із застосуванням 3D-візуалізації, є ключовим етапом реалізації проєкту, що дозволяє створити надійну, масштабовану та ефективну платформу. Архітектурне рішення базується на чіткому розподілі функціональних компонентів, оптимальній структурі бази даних та інтеграції сучасних технологій для забезпечення високої продуктивності, безпеки та зручності користувацького досвіду. У цьому розділі детально розглядається архітектурне проектування системи, включаючи компонентну структуру, класи, сценарії використання, послідовність операцій, діяльність, стани та розгортання системи, що відповідає лістингу коду проєкту. Процес проектування розпочався з аналізу вимог, де було визначено необхідність підтримки інтерактивної 3D-візуалізації ювелірних виробів, що вимагає ефективної обробки GLB/GLTF файлів, а також класичних функцій електронної комерції, таких як каталог товарів, кошик, автентифікація та адміністрування. Для досягнення балансу між простотою розгортання та продуктивністю обрано клієнт-серверну архітектуру з RESTful API як основним каналом взаємодії, що дозволяє легко масштабувати систему в майбутньому.

Функціональні вимоги визначають ключові можливості системи Jewelry3D, спрямовані на забезпечення повноцінного циклу електронної комерції з акцентом на 3D-візуалізацію:

– перегляд каталогу товарів з фільтрацією за категоріями, пошуком та сортуванням (за ціною, датою, популярністю);

– детальний перегляд товару з інтерактивною 3D-візуалізацією моделі (обертання, зум, автоматичне обертання);

– автентифікація користувачів (реєстрація, вхід, вихід, швидкий вхід для тестування) з ролями "client" та "admin";

– управління кошиком – додавання/видалення товарів, зміна кількості, збереження в localStorage;

– оформлення замовлень з вибором методу доставки та оплати, розрахунком загальної суми;

– залишення та перегляд відгуків на товари з модерацією;

– адміністративні функції: CRUD-операції над товарами, категоріями та замовленнями; завантаження зображень та 3D-моделей (GLB/GLTF);

– отримання профілю користувача та історії замовлень;

– швидкий вхід для тестування ролей без введення даних.

Нефункціональні вимоги забезпечують надійність, продуктивність та безпеку системи:

– продуктивність передбачає час відповіді API менше 200 мс для більшості запитів і підтримку до 1000 одночасних користувачів;

– масштабованість забезпечується горизонтальним масштабуванням через stateless API та легкою міграцією на PostgreSQL при зростанні обсягу даних;

– безпека включає JWT-автентифікацію з refresh tokens, хешування паролів через bcrypt, захист від XSS, CSRF, SQL-ін'єкцій, обмеження частоти запитів і використання httpOnly cookies;

– доступність реалізується через адаптивний mobile-first дизайн з підтримкою екранів від 320px і ARIA атрибути для користувачів з обмеженими можливостями;

– надійність гарантується ACID-транзакціями для роботи з базою, резервним копіюванням даних і повідомленнями про помилки українською мовою;

– зручність включає повну українську локалізацію інтерфейсу, інтуїтивну навігацію та fallback-моделі для 3D у разі недоступності GLB;

– підтримка охоплює сумісність з Chrome, Firefox, Safari і оптимізацію 3D-моделей розміром до 50 MB

Функціональні сценарії системи охоплюють дії гостей, клієнтів та адміністраторів, що відображено в діаграмі варіантів використання (рис. 2.2). Гості можуть переглядати каталог, шукати товари та детально вивчати 3D-моделі, тоді як автентифіковані клієнти додають товари до кошика, оформлюють замовлення та залишають відгуки. Адміністратори керують контентом через CRUD-операції та завантажують файли.

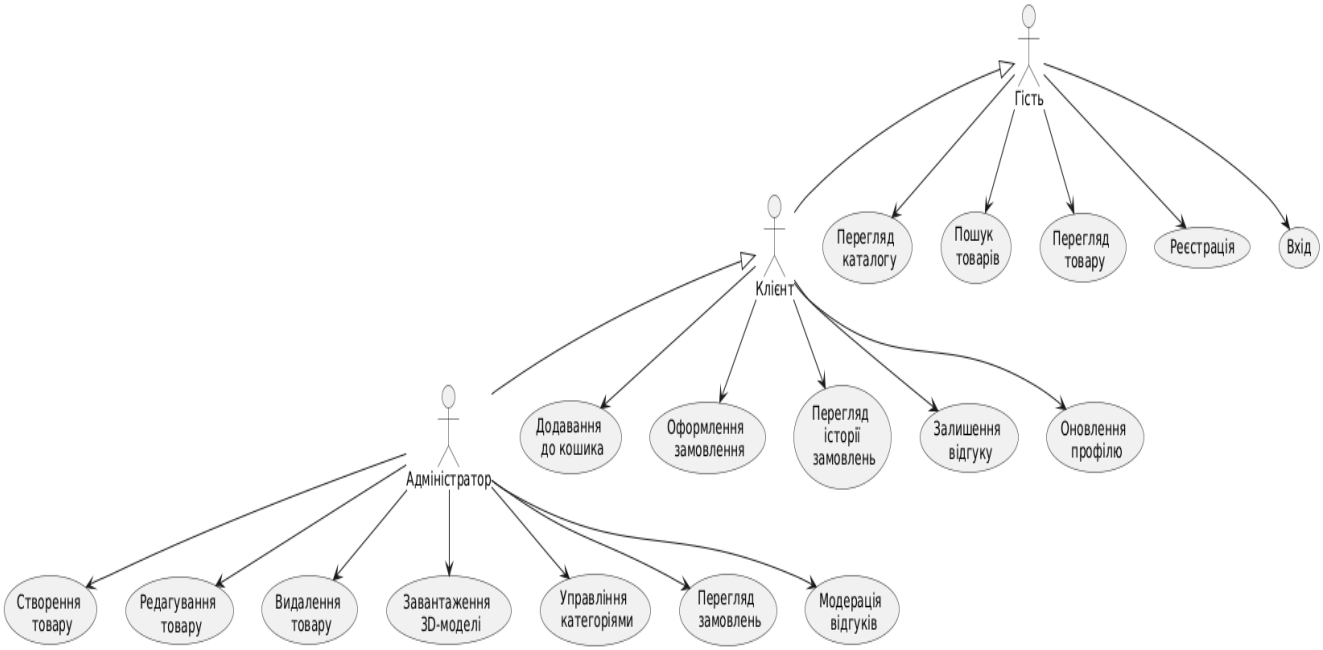

Рис. 2.2. Діаграма варіантів використання Jewelry3D

Швидкий вхід через /api/auth/quick-login спрощує тестування, автоматично генеруючи токени для тестових користувачів.

Архітектура системи Jewelry3D побудована за принципом клієнт-серверної моделі з чітким поділом на клієнтську та серверну частини, що взаємодіють через RESTful API. Клієнтська частина, реалізована на React 19, використовує Redux Toolkit для централізованого управління станом, React Router для навігації між сторінками та Three.js для рендерингу 3D-моделей, забезпечуючи плавний та інтерактивний користувацький досвід. Серверна частина на Node.js з Express.js обробляє бізнес-логіку, автентифікацію через JWT та завантаження файлів за допомогою Multer, тоді як SQLite з Sequelize виступає легковажним, але надійним сховищем даних. Такий підхід не лише спрощує розробку, але й гарантує безпеку

завдяки httpOnly cookies для токенів, helmet.js для захисту заголовків та rate-limit для запобігання атакам. Для наочності структури системи розроблено діаграму компонентів (рис. 2.3), яка відображає взаємодію між модулями обох частин.

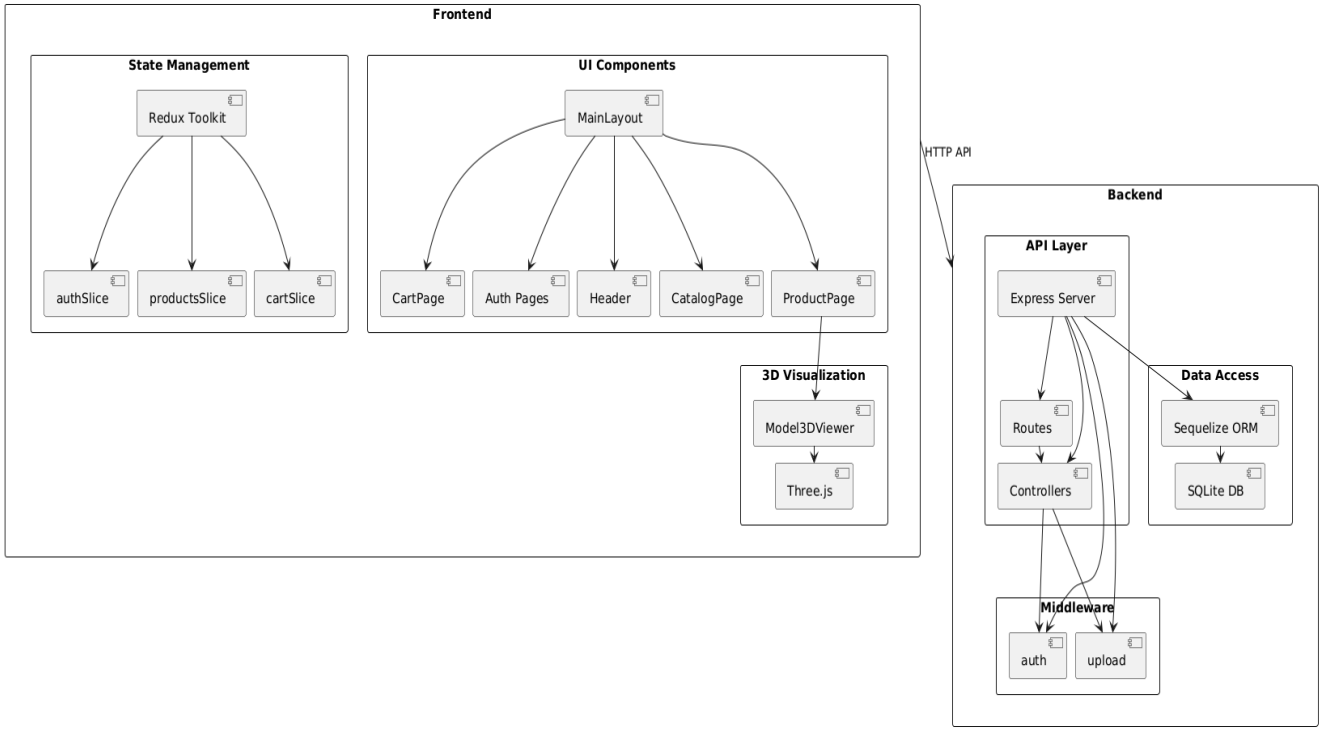

Рис. 2.3. Діаграма компонентів веб-системи Jewelry3D

Компонентна діаграма ілюструє, як клієнтські модулі, такі як CatalogPage та ProductPage, звертаються до серверних контролерів через API-ендпоінти, наприклад /api/products для отримання списку товарів з фільтрацією за категорією чи пошуком. Серверні контролери, у свою чергу, взаємодіють з моделями Sequelize для виконання запитів до бази даних, а middleware забезпечує автентифікацію та валідацію. Файлова система використовується для зберігання зображень та 3D-моделей у директоріях uploads/products/images та uploads/products/models, з унікальними іменами файлів для уникнення конфліктів. Redux Store на клієнті зберігає стан автентифікації, товарів та кошика, що персистується в localStorage для збереження між сесіями, що робить досвід користувача безперервним навіть після перезавантаження сторінки.

Архітектура бази даних спроектована з урахуванням специфіки ювелірного магазину, де ключовими є швидкий доступ до каталогів та детальна інформація про товари. SQLite обрано через його легкість у розгортанні та достатню продуктивність для операцій читання, які домінують у сценаріях перегляду товарів. База складається з семи таблиць, пов’язаних зовнішніми ключами для забезпечення референційної цілісності. Таблиця Users зберігає UUID як первинний ключ, email, хеш пароля та роль, тоді як Products містить JSON-масив image_urls для гнучкого зберігання зображень та model_3d_url для посилання на GLB-файл. Зв’язки один-до-багатьох, наприклад між Category та Product чи Order та OrderItem, реалізовані з каскадним видаленням, що автоматично очищає залежні записи. Для детального представлення структури даних розроблено діаграму класів (рис. 2.4).

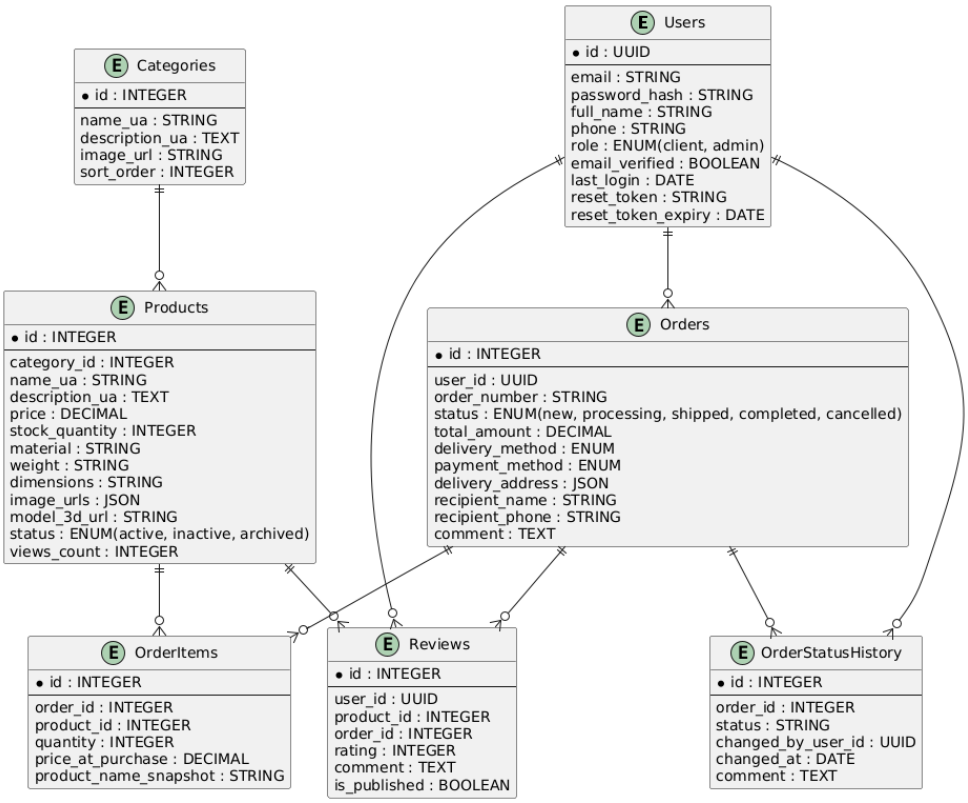

Рис. 2.4. Діаграма класів бази даних Jewelry3D

Діаграма класів демонструє асоціації, які полегшують складні запити, наприклад отримання товару з відгуками та середнім рейтингом через include в

Sequelize. JSON-поля оптимізують зберігання масивів зображень без створення окремих таблиць, що зменшує складність схеми. Індекси на полях, таких як category_id та status у Products, прискорюють фільтрацію та сортування, що критично для каталогу з пагінацією.

Послідовність автентифікації ілюструє взаємодію компонентів під час входу (рис. 2.5).

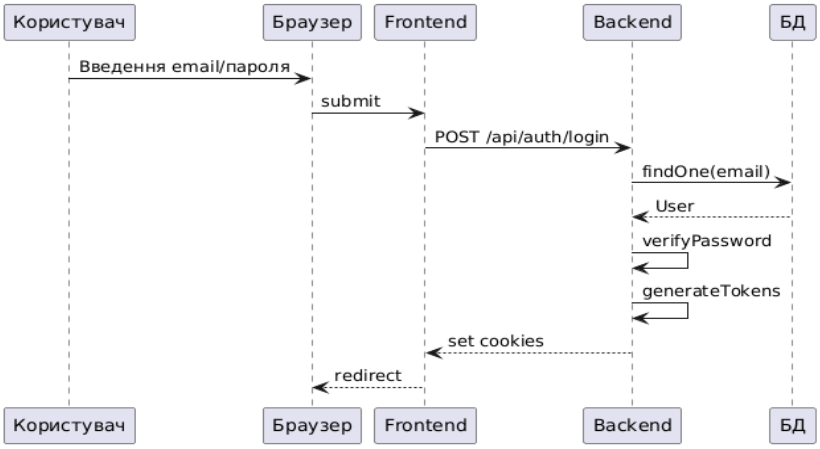

Рис. 2.5. Діаграма послідовності для автентифікації

Контролер перевіряє пароль через bcrypt, оновлює last_login та встановлює httpOnly cookies, що захищає від XSS. Діяльність створення товару показує адміністративний workflow (рис. 2.6).

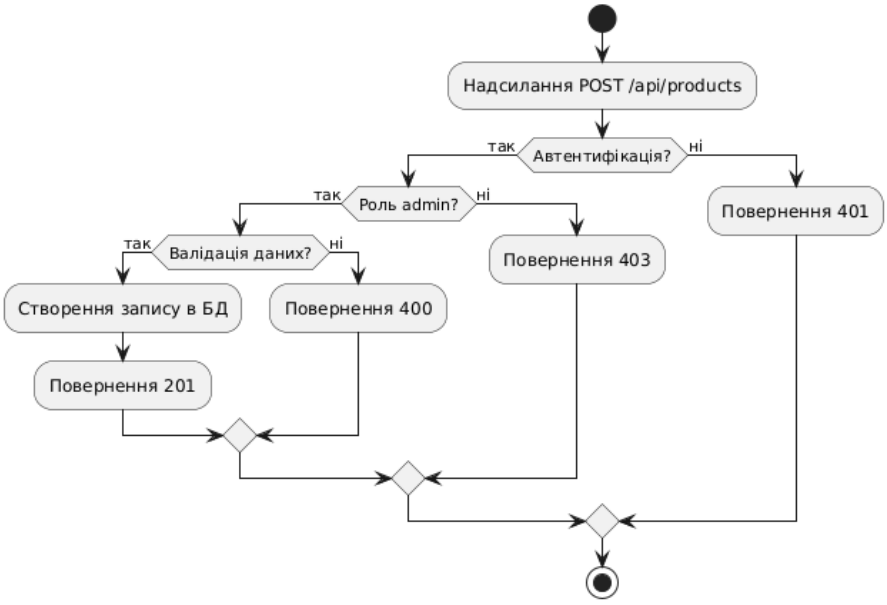

Рис. 2.6. Діаграма діяльності для створення товару

Middleware requireAdmin блокує неавторизований доступ, а валідація перевіряє обов’язкові поля. Життєвий цикл замовлення моделюється станами (рис. 2.7).

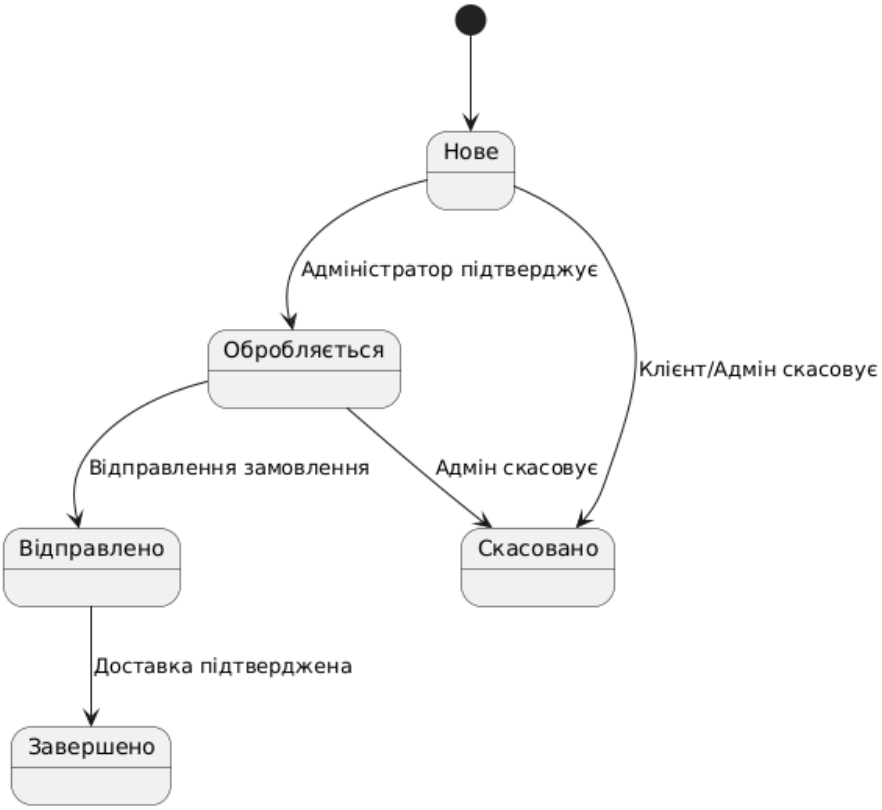

Рис. 2.7. Діаграма станів для замовлення

Перехід станів фіксується в OrderStatusHistory для аудиту. Розгортання системи передбачає локальне розміщення з можливістю масштабування (рис. 2.8).

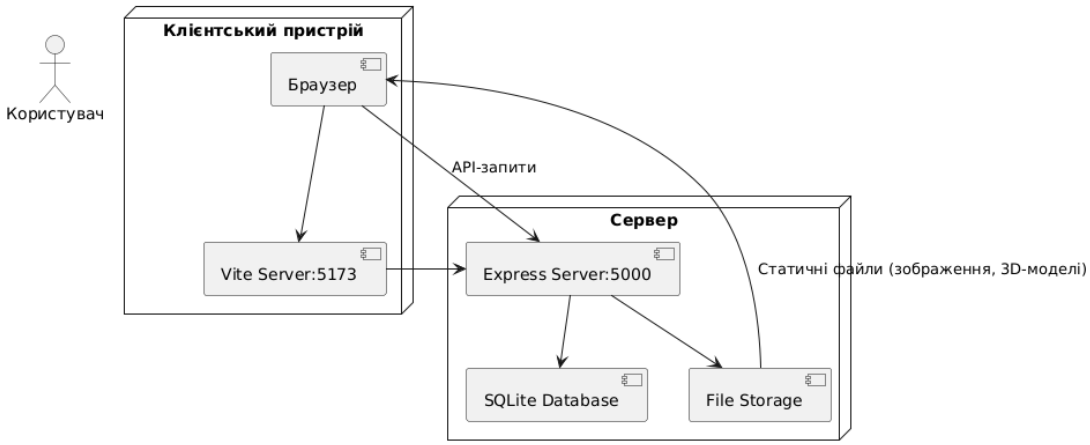

Рис. 2.8. Діаграма розгортання Jewelry3D

Vite забезпечує швидке розроблення, тоді як у продакшені статичні файли можуть кешуватися на CDN. Таким чином, архітектура Jewelry3D поєднує простоту з потужністю, створюючи привабливу платформу для ювелірних виробів з 3D-візуалізацією, що відповідає сучасним стандартам e-commerce.

У ході аналізу сучасних технологій 3D-візуалуалізації для веб-систем електронної комерції встановлено, що Three.js є оптимальним вибором для проєкту Jewelry3D завдяки високій продуктивності, гнучкості та легкості інтеграції з React. Порівняльна характеристика з Babylon.js та PlayCanvas підтвердила переваги Three.js у легких e-commerce додатках, де ключовими є реалістичне відображення ювелірних виробів без зайвих накладних витрат.

Проєктування архітектури Jewelry3D реалізовано на основі клієнт-серверної моделі з RESTful API, React 19, Node.js, Express та SQLite, що забезпечує масштабованість, безпеку та інтерактивний користувацький досвід. Компонентна структура, діаграми класів бази даних, сценарії використання, послідовності операцій, діяльності, стани та розгортання системи гарантують ефективну обробку GLB-моделей, автентифікацію через JWT та CRUD-операції.

Отримані результати створюють міцну основу для реалізації повнофункціональної платформи продажу крафтових ювелірних виробів з інноваційною 3D-візуалізацією, значно підвищуючи конкурентоспроможність у онлайн-торгівлі.

РОЗДІЛ 3.
ПРОЕКТНА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ З 3D-ОБ'ЄКТАМИ

3.1 Алгоритми обробки даних і 3D-візуалізації

Розробка веб-системи Jewelry3D, призначеної для продажу крафтових ювелірних виробів з інтерактивною 3D-візуалізацією, вимагала створення ефективних алгоритмів обробки даних і візуалізації, які б забезпечували не лише високу продуктивність, але й надійний захист інформації. У процесі реалізації проєкту було враховано сучасні вимоги до безпеки, оптимізації обробки даних і якісного відображення 3D-об'єктів. У цьому підрозділі детально розглянуто архітектурні рішення, алгоритми обробки даних, механізми 3D-візуалізації та заходи захисту інформації, які застосовані в проєкті, з акцентом на відповідність кодам проєкту і документації.

На етапі обробки даних основна увага приділялася створенню структури бази даних та алгоритмів взаємодії з нею. Система використовує SQLite як легковагу базу даних, що забезпечує простоту розгортання та достатню продуктивність для електронної комерції середнього масштабу[18]. Модель даних складається з семи таблиць: Users, Categories, Products, Orders, OrderItems, Reviews і OrderStatusHistory, що дозволяє ефективно зберігати інформацію про користувачів, товари, замовлення та їх історію. Для управління даними використано ORM Sequelize, який забезпечує безпечну параметризацію запитів, що мінімізує ризик SQL-ін'єкцій. Наприклад, у файлі database.js налаштовано підключення до SQLite з увімкненням автоматичних транзакцій і логування в режимі розробки, що сприяє прозорості та відлагодженню. Алгоритм синхронізації бази даних, реалізований у функції syncDatabase у файлі index.js, забезпечує створення або оновлення таблиць без втрати даних у продакшн-середовищі, що є важливим для збереження цілісності інформації.

Для обробки запитів на сервері використано Express.js, який забезпечує гнучку маршрутизацію та обробку HTTP-запитів[19]. Алгоритм обробки API-запитів, зокрема для отримання списку товарів (productsController.js), включає етапи фільтрації за категоріями, пошуку за ключовими словами та сортування за різними критеріями (ціна, дата, популярність). Цей алгоритм реалізовано через метод getProducts, який використовує Sequelize для формування запитів до бази даних із застосуванням умов where та order.

Алгоритм автентифікації, реалізований у файлі authController.js, базується на використанні JWT-токенів для безпечного доступу до захищених ресурсів. Процес автентифікації включає генерацію access-токена (з терміном дії 15 хвилин) і refresh-токена (30 днів), які зберігаються в httpOnly cookies, що захищає від XSS-атак. Цей алгоритм забезпечує безпечну автентифікацію завдяки використанню bcrypt для перевірки паролів і JWT для stateless автентифікації, що відповідає сучасним стандартам безпеки веб-додатків [20].

Алгоритм обробки API-запитів до ендпоінту отримання списку товарів, реалізований у методі getProducts контролера productsController.js, охоплює послідовне виконання етапів: отримання параметрів запиту (категорія, пошуковий рядок, сортування, сторінка, ліміт), формування умови фільтрації where, побудову виразу пошуку за полями name_ua та description_ua з використанням оператора Op.like, визначення порядку сортування залежно від переданого значення sort, обчислення зсуву offset для пагінації, виконання запиту findAndCountAll із включенням пов’язаних даних категорії, повернення структурованого JSON-відповіді з масивом товарів, загальною кількістю сторінок та поточною сторінкою. Цей алгоритм забезпечує швидку та гнучку видачу даних каталогу з урахуванням усіх фільтрів і сортувань, одночасно мінімізуючи кількість запитів до бази даних завдяки eager loading пов’язаних сутностей. Для забезпечення безпеки запитів використано middleware express-rate-limit із конфігурацією, що обмежує кількість запитів з одного IP (500 у режимі розробки, 100 у продакшн), як зазначено у файлі server.js. Це знижує ризик DDoS-атак і забезпечує стабільність роботи сервера.

Рис. 3.1 ілюструє алгоритм обробки запиту на отримання списку товарів.

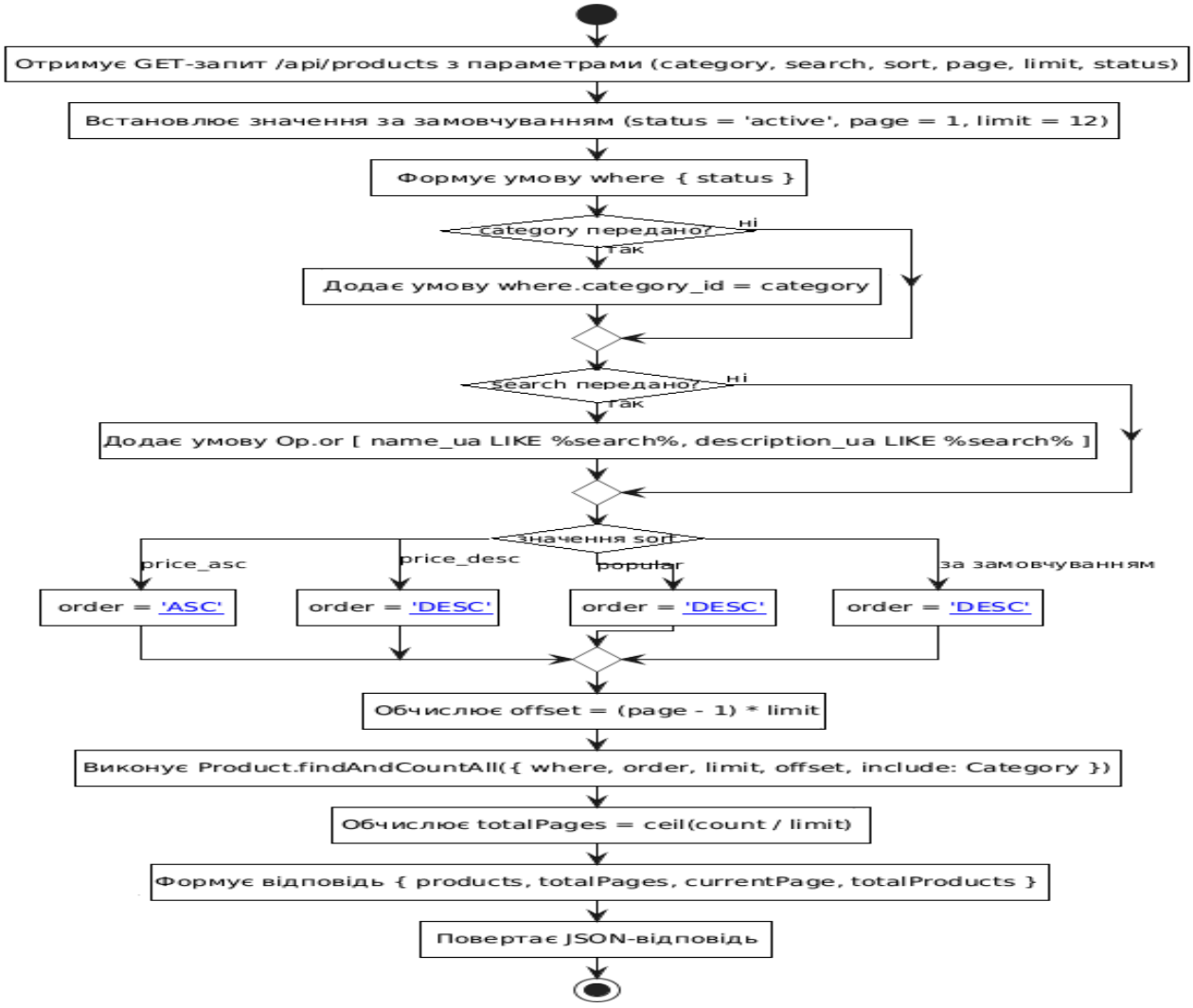

Рис. 3.1. Блок-схема алгоритму обробки запиту на отримання списку товарів

Цей алгоритм забезпечує ефективну та безпечну видачу даних каталогу з підтримкою фільтрації, пошуку та пагінації, що відповідає сучасним вимогам до продуктивності RESTful API електронної комерції. Для додаткового захисту від CSRF-атак використано налаштування CORS із обмеженням дозволених джерел (server.js), а також middleware helmet.js для встановлення безпечних HTTP-заголовків, що знижує ризик clickjacking та інших атак.

Щодо 3D-візуалізації, система використовує бібліотеку Three.js із React Three Fiber для рендерингу GLB/GLTF моделей. Алгоритм обробки 3D-моделей включає етапи завантаження, валідації, стиснення та відображення. Файл upload.js містить конфігурацію Multer для завантаження моделей із фільтрацією за MIME-

типами (model/gltf-binary, model/gltf+json) та обмеженням розміру файлу до 50 МБ.

Для оптимізації використано Draco-компресію, яка зменшує розмір моделей на 60-80%, що критично для швидкого завантаження на клієнтській стороні. Алгоритм відображення 3D-моделей передбачає використання OrbitControls для інтерактивного управління (обертання, зум, панорамування) та автоматичне масштабування моделей для адаптації до різних пристроїв. У разі відсутності моделі реалізовано fallback-механізм із процедурно згенерованою моделлю золотого кільця, як зазначено в документації PROJECT_STATUS.md.

Рис. 3.2 демонструє алгоритм обробки та відображення 3D-об'єктів. Цей алгоритм забезпечує ефективну та безпечну роботу з 3D-моделями, мінімізуючи затримки при завантаженні та рендерингу. Для захисту файлів від шкідливого вмісту застосовується перевірка MIME-типів через magic bytes, а також заборона виконання файлів на сервері, що відповідає сучасним практикам захисту інформації.

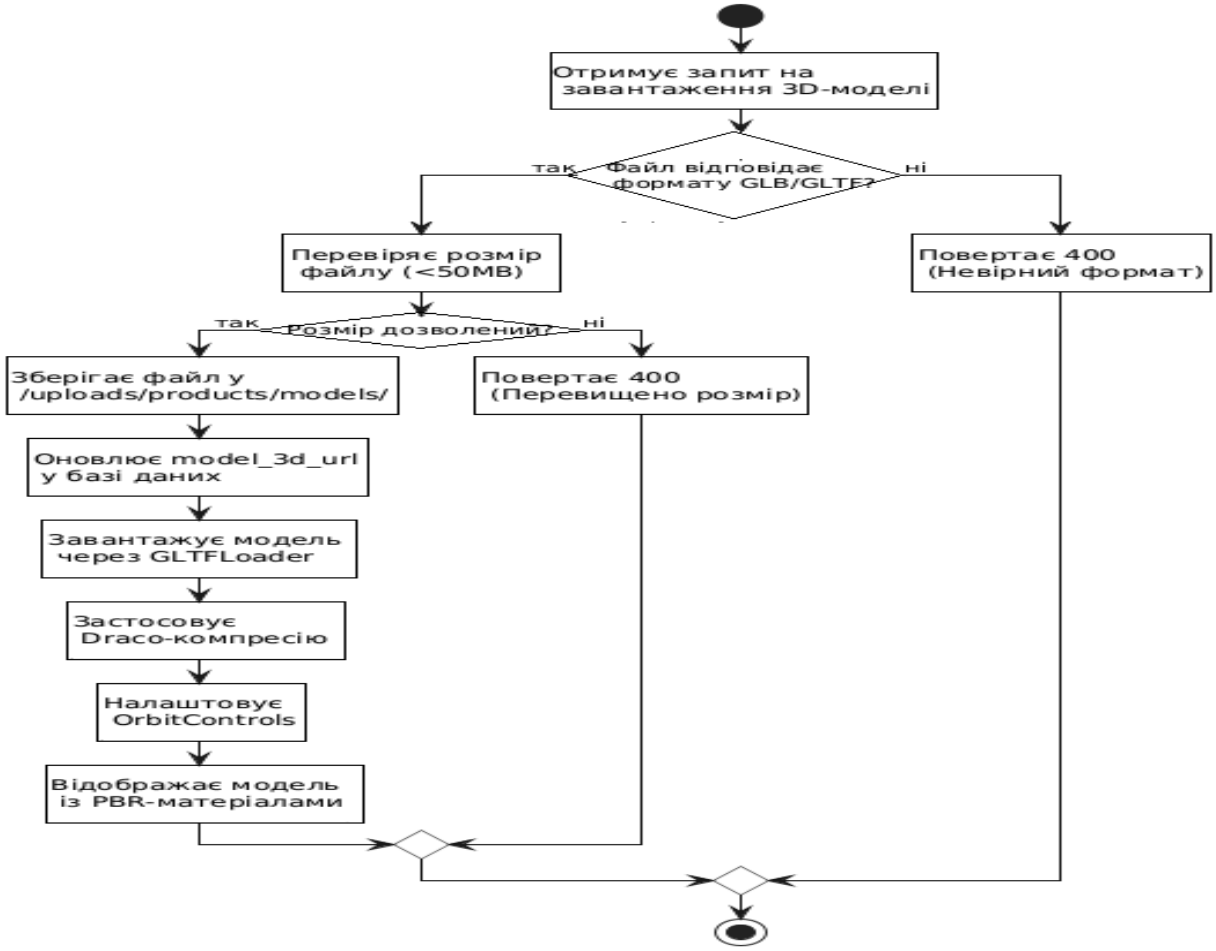

Рис. 3.2. Блок-схема алгоритму обробки та відображення 3D-об'єктів

Обробка даних у кошику покупок реалізована через Redux Toolkit із персистентністю в localStorage (cartSlice), що дозволяє зберігати стан кошика між сесіями. Алгоритм додавання товару до кошика включає перевірку наявності товару в базі даних, оновлення кількості та перерахунок загальної суми, що реалізовано в productsController.js та orders.js. Для захисту від несанкціонованого доступу до даних кошика використано middleware requireClient, який обмежує доступ до оформлення замовлень лише для користувачів із роллю "client". Таблиця 3.1 підсумовує ключові заходи безпеки, застосовані в системі.

Таблиця 3.1

Заходи захисту інформації в системі Jewelry3D

Захід безпеки	**Опис**	**Реалізація**
Хешування паролів	Використання bcrypt із cost factor 12 для захисту паролів	User.js, метод beforeCreate

JWT-автентифікація	Використання access та refresh токенів у httpOnly cookies	authController.js, auth.js
Обмеження запитів	Rate limiting для захисту від DDoS-атак	server.js, express-rate-limit
Захист від XSS/CSRF	Використання helmet.js та CORS із обмеженням джерел	server.js
Валідація файлів	Перевірка MIME-типів і розміру для 3D-моделей та зображень	upload.js, Multer

Ці заходи забезпечують комплексний захист інформації, що відповідає сучасним стандартам кібербезпеки [21]. Для оптимізації продуктивності використано індекси бази даних (наприклад, на полях category_id, status у таблиці Products), що прискорює виконання запитів. Крім того, реалізована пагінація в каталозі товарів (getProducts) зменшує навантаження на сервер і клієнтську частину.

Таким чином, розробка веб-системи Jewelry3D поєднує ефективні алгоритми обробки даних, інтерактивну 3D-візуалізацію та надійні механізми захисту інформації. Алгоритми автентифікації та обробки 3D-моделейдемонструють структурований підхід до реалізації функціоналу з урахуванням безпеки. Використання сучасних технологій, таких як Sequelize, Three.js і JWT, забезпечує високу продуктивність і захист даних, що робить систему придатною для реального використання в електронній комерції.

3.2 Програмна реалізація та тестування веб-системи

Розробка веб-системи Jewelry3D для продажу крафтових ювелірних виробів з інтерактивною 3D-візуалізацією є прикладом сучасного підходу до створення платформ електронної комерції, що поєднують зручність користувацького

інтерфейсу, безпеку даних та інноваційні технології. Програмна реалізація системи базується на чітко структурованій архітектурі, яка включає фронтенд на React 19 з Redux Toolkit для управління станом, бекенд на Node.js із Express.js, базу даних SQLite з ORM Sequelize та інтеграцію 3D-візуалізації через Three.js. У цьому розділі детально розглядається реалізація всіх екранних форм, оцінюється безпека системи та описуються методи тестування результатів.

У системі Jewelry3D 3D-моделі ювелірних виробів не створювалися спеціально для проєкту, а були взяті з відкритих безкоштовних ресурсів, таких як Sketchfab, CGTrader, Free3D та TurboSquid, де доступні моделі в форматі GLB/GLTF з можливістю завантаження. Процес інтеграції цих моделей включав:

– пошук релевантних моделей за ключовими словами на кшталт "ring jewelry", "earrings 3D model" або "necklace GLB" з фільтром на downloadable та безкоштовні;

– завантаження файлів у форматі GLB (рекомендований для Three.js через компактність і підтримку PBR-матеріалів);

– оптимізацію моделей у Blender (зменшення кількості полігонів за допомогою модифікатора Decimate, стиснення текстур до 1024x1024 px та застосування Draco-компресії для зменшення розміру файлів на 60-80%);

– перейменування файлів відповідно до товарів (наприклад, ring1.glb для золотого кільця);

– розміщення в директорії backend/uploads/products/models/ для автоматичного завантаження через API.

Для випадків відсутності моделі використовується процедурна fallback-модель, згенерована в Three.js (простий об’єкт кільця з золотим матеріалом та геометрією, створеною через CylinderGeometry та MeshStandardMaterial). Це забезпечує гнучкість і не вимагає створення моделей з нуля, дозволяючи фокусуватися на інтеграції.

Програмна реалізація системи починається з фронтенду, який забезпечує інтуїтивно зрозумілий інтерфейс для користувачів. Головна сторінка системи, як

показано на рис. 3.3, містить верхній блок із назвою "Jewelry3D" та коротким описом платформи, що підкреслює її унікальність у сфері продажу ювелірних виробів із 3D-візуалізацією. У цьому блоці розташовані дві кнопки: "Переглянути каталог", яка перенаправляє користувача на сторінку каталогу, та "Почати", що веде до сторінки авторизації для швидкого доступу до функціоналу. Дизайн виконано в адаптивному стилі з використанням CSS Grid і Flexbox, що забезпечує коректне відображення на різних пристроях.

Рис. 3.3. Головна сторінка веб-системи Jewelry3D

Нижче на головній сторінці розташовано блок "Вироби у 3D" (рис. 3.4), який демонструє можливості 3D-візуалізації. Цей блок містить інтерактивний елемент із Three.js, що дозволяє користувачам переглядати приклади ювелірних виробів у тривимірному форматі. Реалізація базується на бібліотеці @react-three/fiber, яка забезпечує інтеграцію WebGL через GLTFLoader для завантаження моделей формату GLB. Користувачі можуть обертати моделі за допомогою OrbitControls, масштабувати їх колесом миші та переглядати з різних кутів, що підвищує залученість і демонструє якість товарів.

Рис. 3.4. Блок "Вироби у 3D" на головній сторінці

Наступний блок "Чому саме Jewelry3D?" (рис. 3.5) підкреслює переваги платформи, такі як унікальність українських ювелірних виробів, підтримка 3D-візуалізації та зручність користувацького досвіду. Цей блок виконано у форматі інформаційних карток із градієнтними фонами та hover-ефектами, що додають інтерактивності. Текстова інформація локалізована українською мовою, що відповідає цільовій аудиторії платформи.

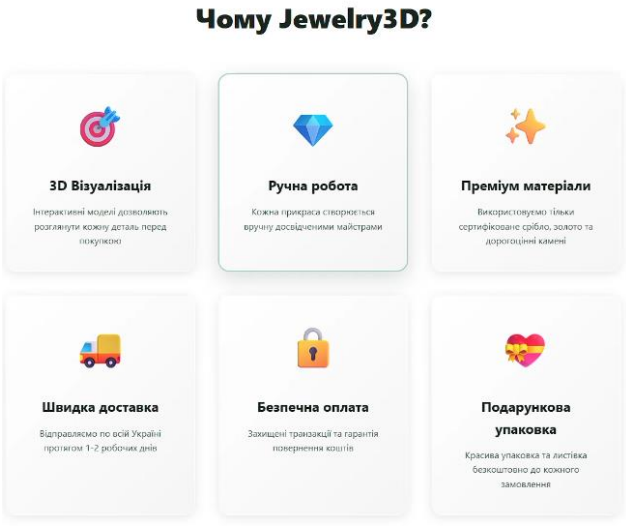

Рис. 3.5. Блок "Чому саме Jewelry3D?"

Нижній блок головної сторінки, названий "Готові обрати свою прикрасу?" (рис. 3.6), містить заклик до дії з кнопкою для переходу до каталогу. У найнижчій частині цього блоку розміщено футер із інформацією про рік випуску (2024), згадку про захист авторських прав та назву платформи "Jewelry3D". Цей блок виконано з використанням CSS Modules для ізоляції стилів, що запобігає конфліктам імен класів.

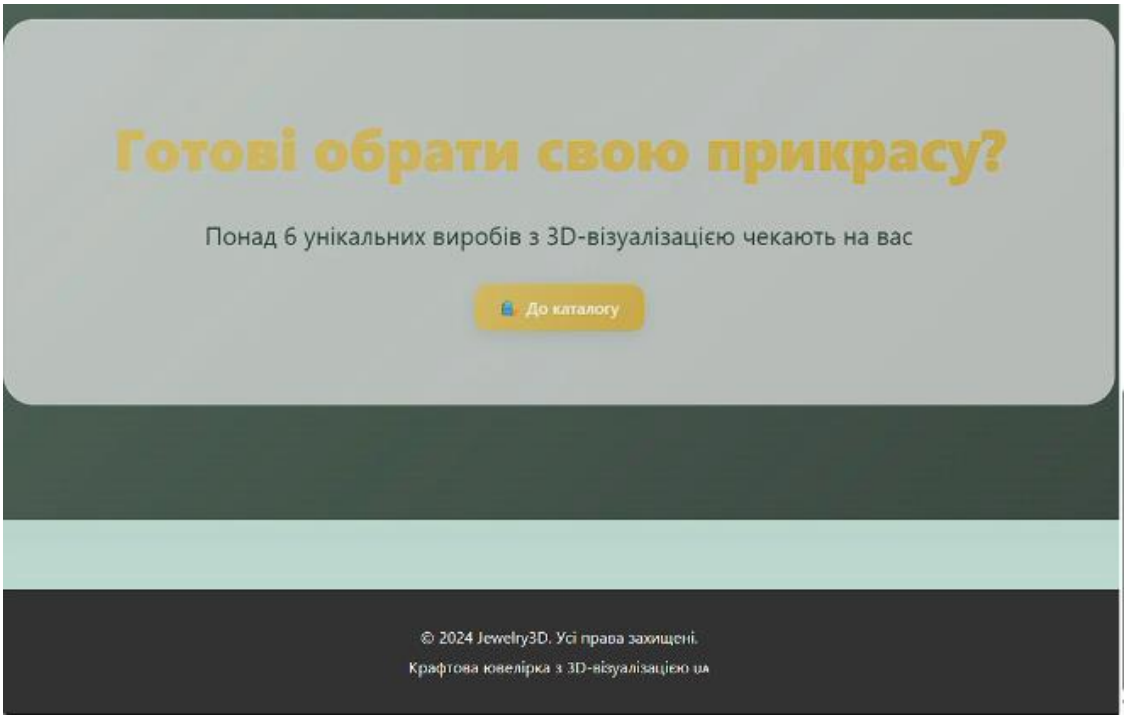

Рис. 3.6. Нижній блок "Готові обрати свою прикрасу?"

Сторінка каталогу товарів (рис. 3.7) реалізує функціонал перегляду ювелірних виробів із можливістю фільтрації за категоріями, пошуку за ключовими словами та сортування за ціною, популярністю чи датою додавання. Логіка фільтрації та сортування реалізована через API-запити до ендпоінту /api/products, який повертає дані з бази SQLite. Картки товарів містять зображення, назву, ціну та короткий опис, а також кнопку для швидкого додавання до кошика, що зберігається в localStorage через Redux Toolkit.

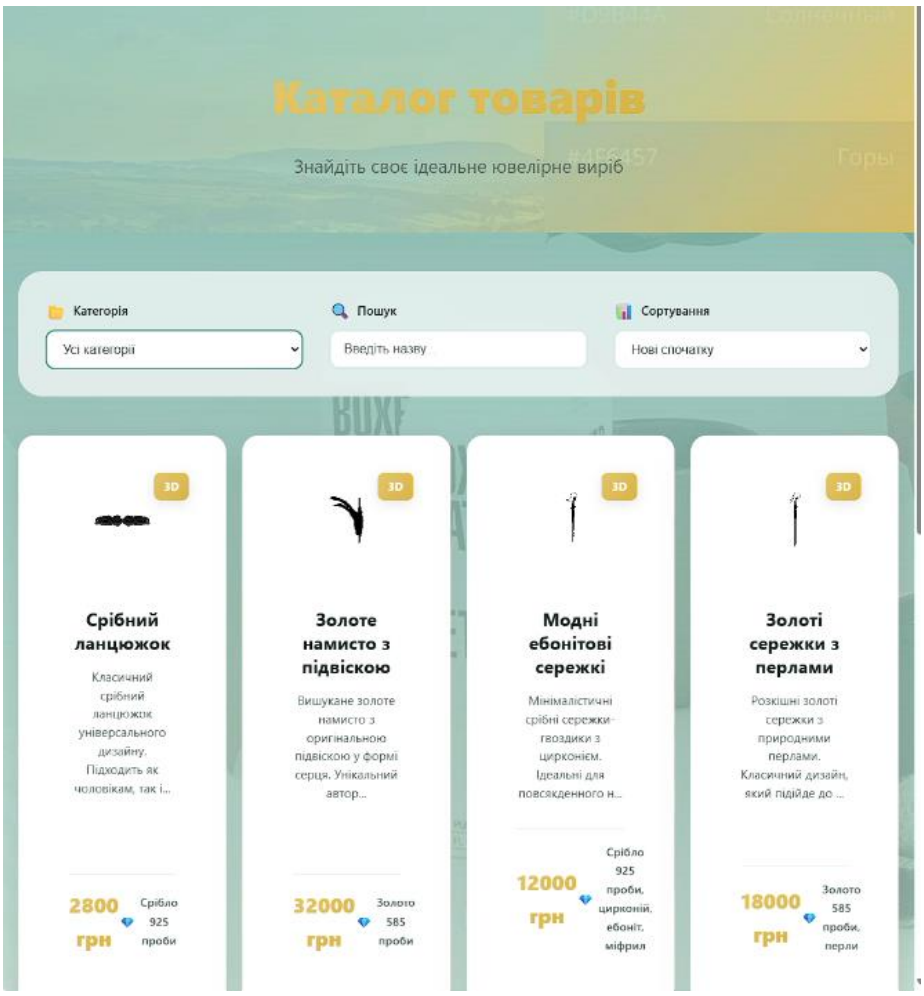

Рис. 3.7. Сторінка каталогу товарів

Сторінка перегляду окремого товару (рис. 3.8) надає детальну інформацію про виріб, включаючи 3D-візуалізацію через компонент Model3DViewer. Користувач може переглядати модель у форматі GLB, взаємодіяти з нею за допомогою жестів або миші, а також ознайомитися з характеристиками товару (матеріал, вага, розміри) та відгуками. Лічильник переглядів оновлюється через PATCH-запит до сервера, що відображає популярність товару.

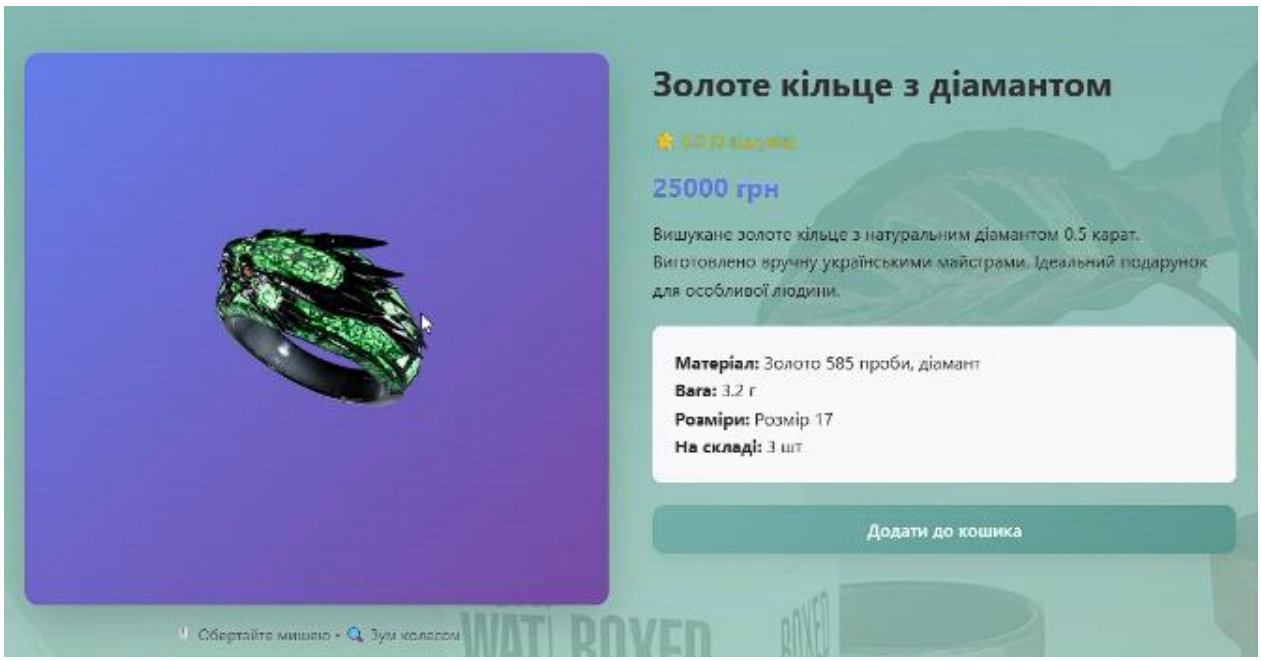

Рис. 3.8. Сторінка перегляду товару

Вікно авторизації (рис. 3.9) містить форму для введення email та пароля, а також дві кнопки швидкого входу: "Увійти як покупець" та "Увійти як адміністратор". Ці кнопки використовують ендпоінт /api/auth/quick-login для автоматичного входу з тестовими акаунтами (client@test.ua та admin@test.ua). Форма підтримує валідацію на клієнтській стороні через Yup, а токени зберігаються в httpOnly cookies для захисту від XSS-атак.

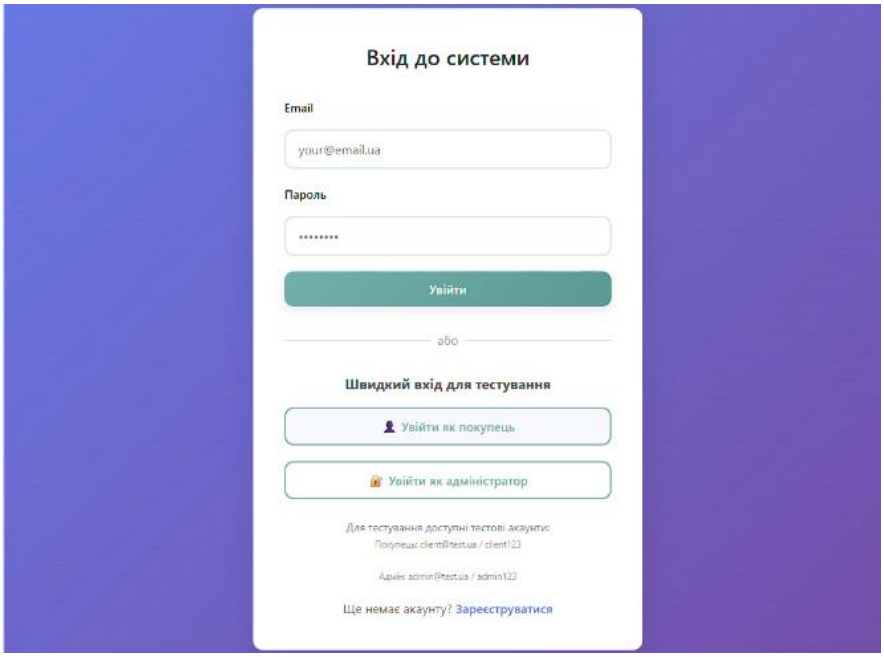

Рис. 3.9. Вікно авторизації

Вікно реєстрації (рис. 3.10) дозволяє створювати нові акаунти шляхом введення email, пароля, імені та номера телефону. Дані відправляються на ендпоінт /api/auth/register, де пароль хешується за допомогою bcrypt із cost factor 12, а токени генеруються через JWT. Валідація формату телефону відповідає українським стандартам (+380 або 0 з 9 цифрами).

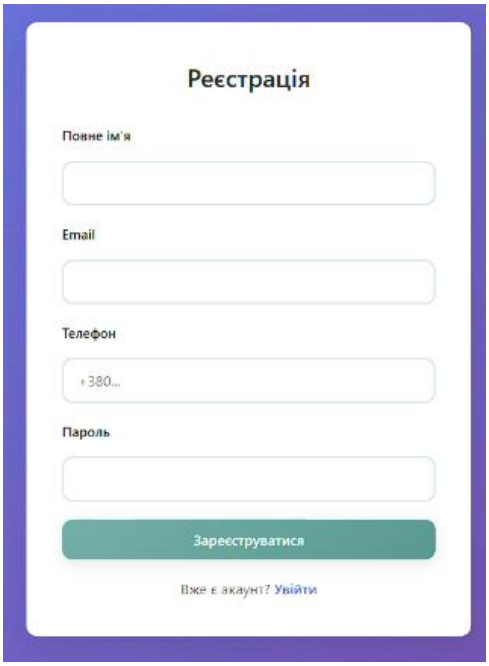

Рис. 3.10. Вікно реєстрації

Сторінка кошика (рис. 3.11) відображає список доданих товарів із можливістю зміни кількості або видалення позицій. Стан кошика зберігається в localStorage через Redux Toolkit, що забезпечує персистентність між сесіями. Кнопка "Оформити замовлення" перенаправляє користувача до форми оформлення.

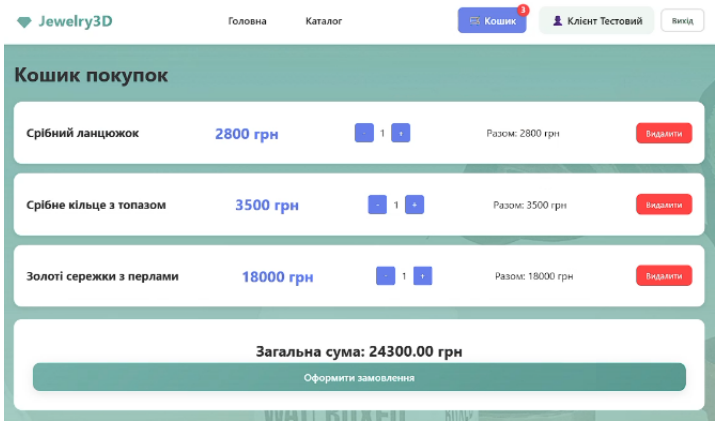

Рис. 3.11. Сторінка кошика

Форма оформлення замовлення (рис. 3.12) дозволяє ввести дані про доставку (адреса, ім’я отримувача, телефон) та вибрати спосіб оплати (готівка або онлайн). Дані відправляються на ендпоінт /api/orders, де створюється нове замовлення з унікальним номером. Логіка розраховує загальну суму на основі цін товарів, отриманих із бази даних.

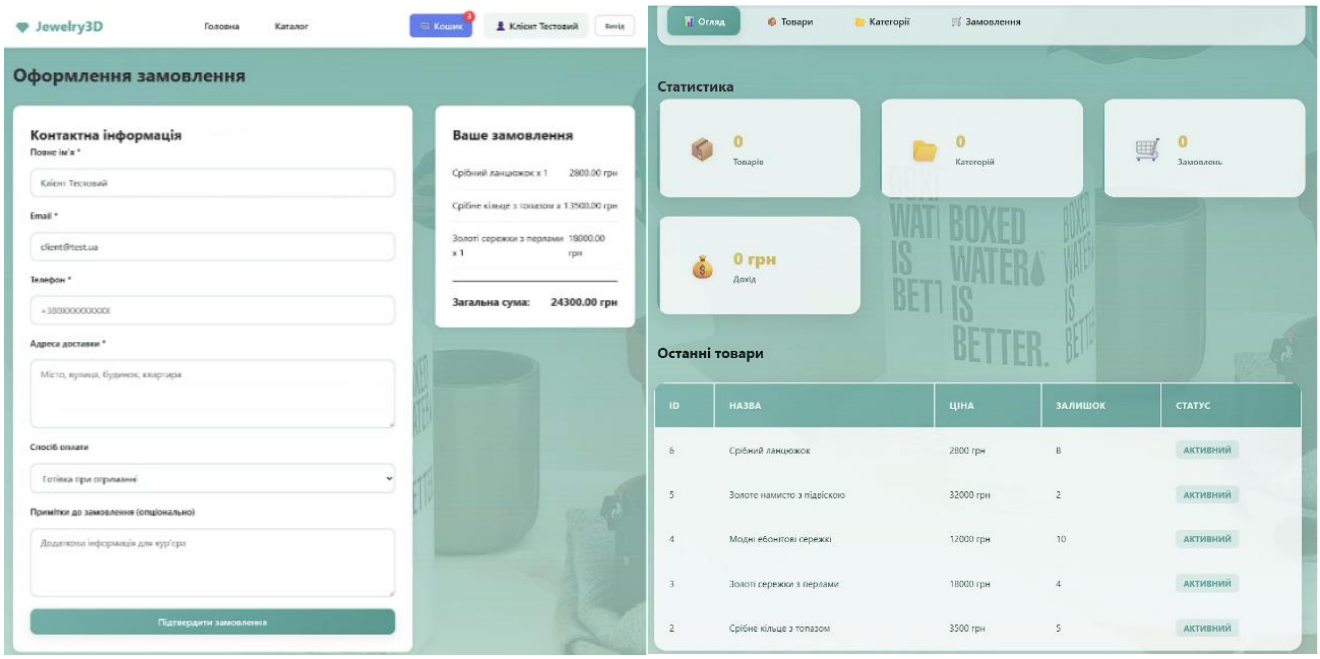

Рис. 3.12. Форма оформлення замовлення і адмін-панель

Форма управління товарами в адмін-панелі (рис. 3.12) дозволяє переглядати список усіх товарів, включаючи неактивні, із можливістю редагування чи видалення. Реалізація базується на ендпоінті /api/admin/products із підтримкою CRUD-операцій.Особистий кабінет (рис. 3.13) доступний після авторизації та відображає профіль користувача (email, ім’я, телефон) та історію замовлень, отриманих через ендпоінт /api/users/me/orders. Користувач може оновити свої дані через форму, що відправляє PUT-запит на /api/users/me.

Рис. 3.13. Особистий кабінет

Форма управління товарами в адмін-панелі (рис. 3.14) дозволяє переглядати список усіх товарів, включаючи неактивні, із можливістю редагування чи видалення. Реалізація базується на ендпоінті /api/admin/products із підтримкою CRUD-операцій.

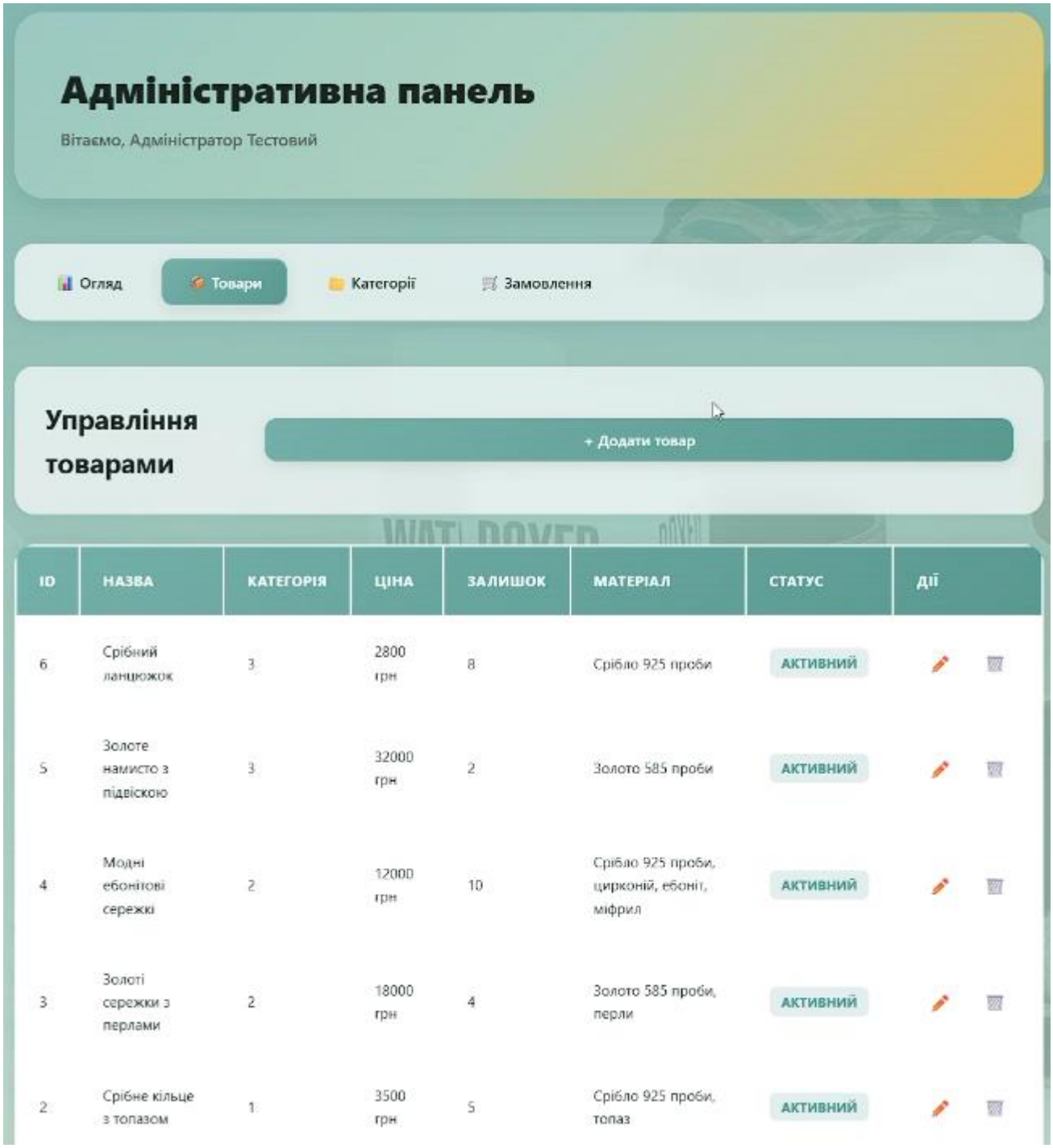

Рис. 3.14. Адмін-панель – товари

Форма редагування товару (рис. 3.15) містить поля для оновлення назви, опису, ціни, кількості, категорії, матеріалу, ваги, розмірів та статусу. Вона також дозволяє завантажувати 3D-моделі (GLB/GLTF) та зображення через ендпоінти /api/admin/products/:id/upload-model та /api/admin/products/:id/upload-images. Multer забезпечує валідацію файлів, обмежуючи розмір (50 МБ для моделей, 5 МБ для зображень) та формати.

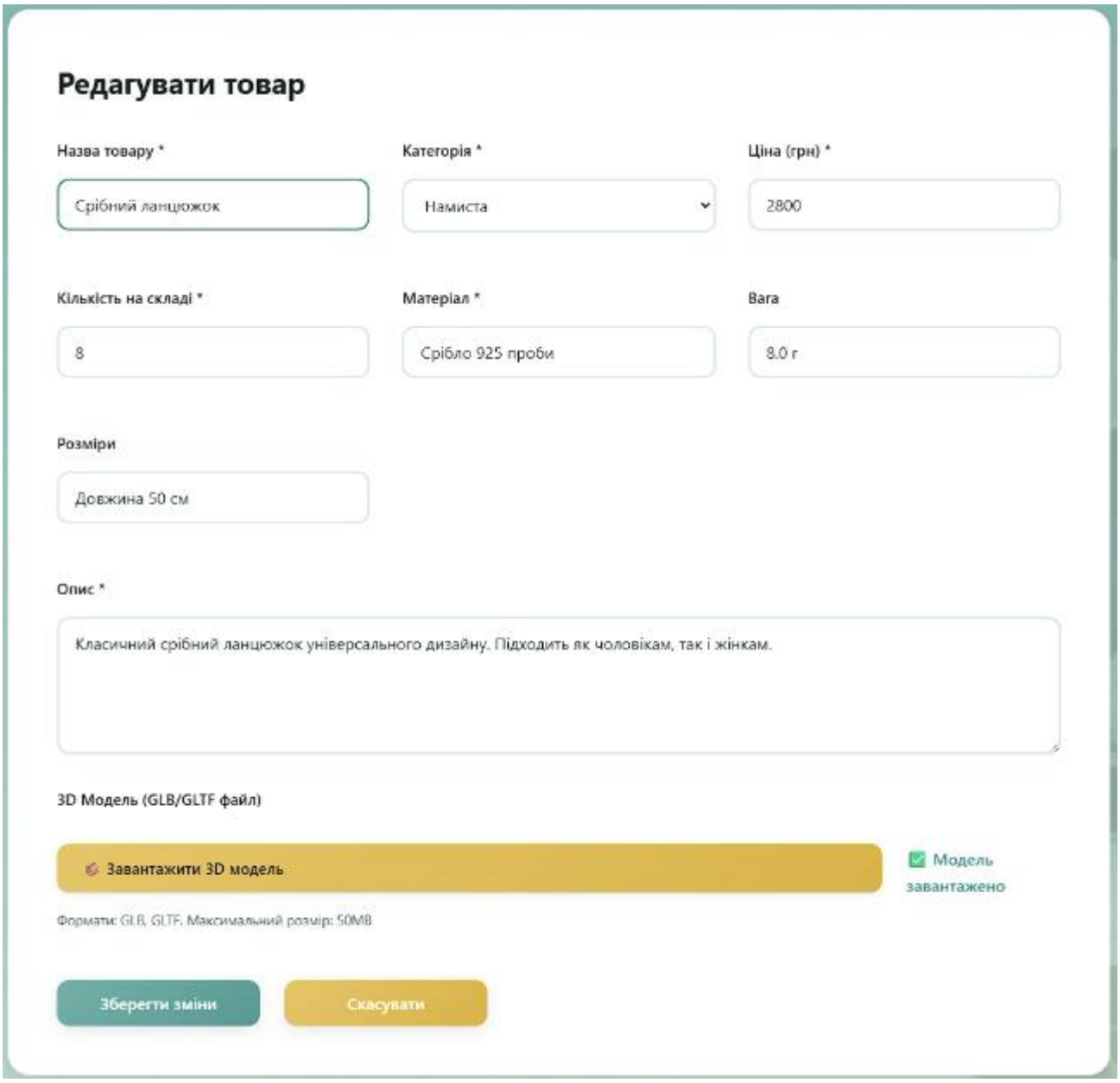

Рис. 3.15. Форма редагування товару

Форма додавання товару (рис. 3.16) аналогічна за структурою, але створює новий запис через POST-запит на /api/admin/products. Обов’язкові поля

включають назву, опис, ціну, кількість та категорію, що забезпечує коректне заповнення даних.

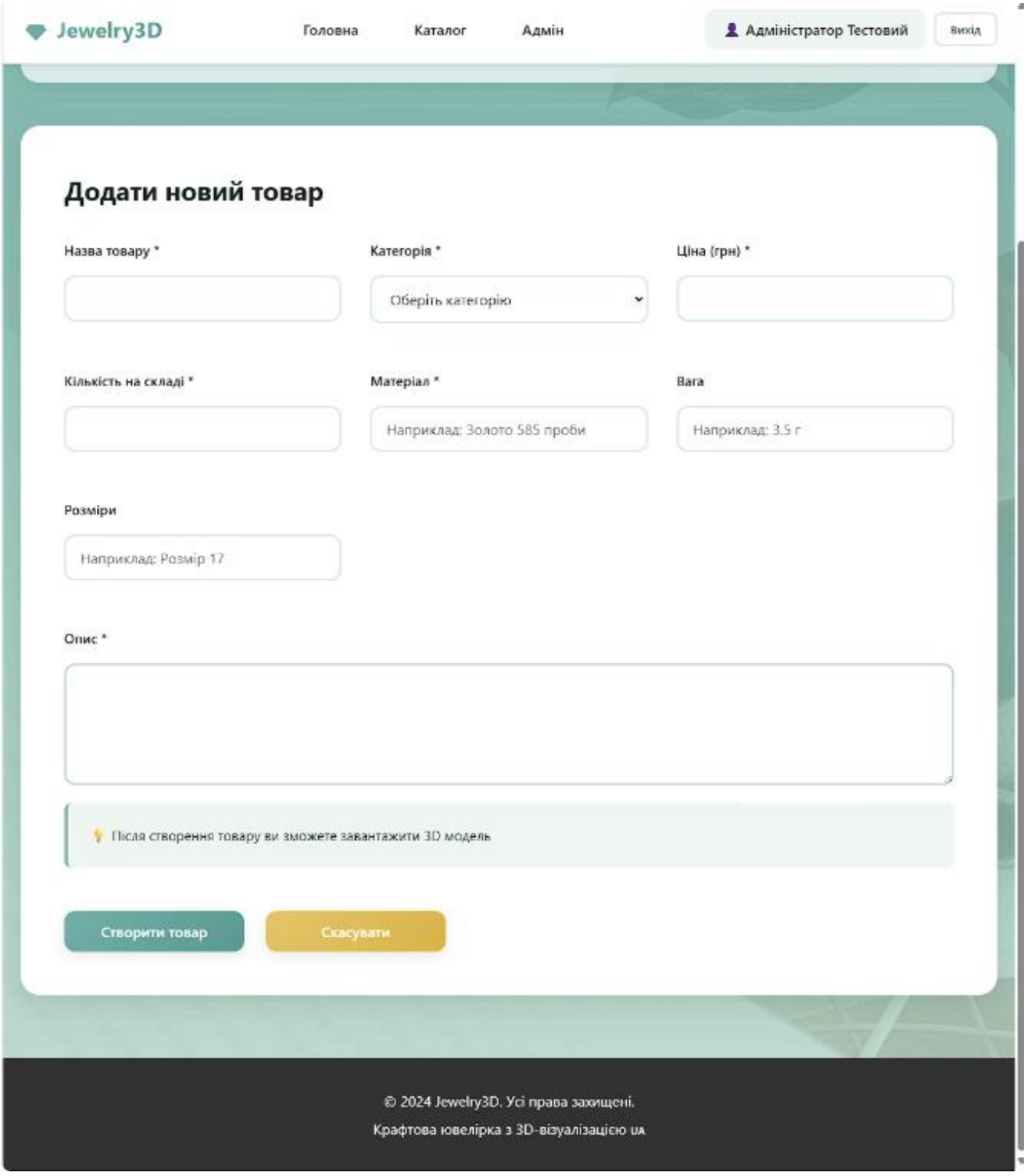

Рис. 3.16. Форма додавання товару

Сторінка управління категоріями в адмін-панелі (рис. 3.17) відображає список категорій із можливістю їх створення, редагування чи видалення через

ендпоінти /api/admin/categories. Категорії сортуються за полем sort_order, що забезпечує гнучке управління порядком відображення.

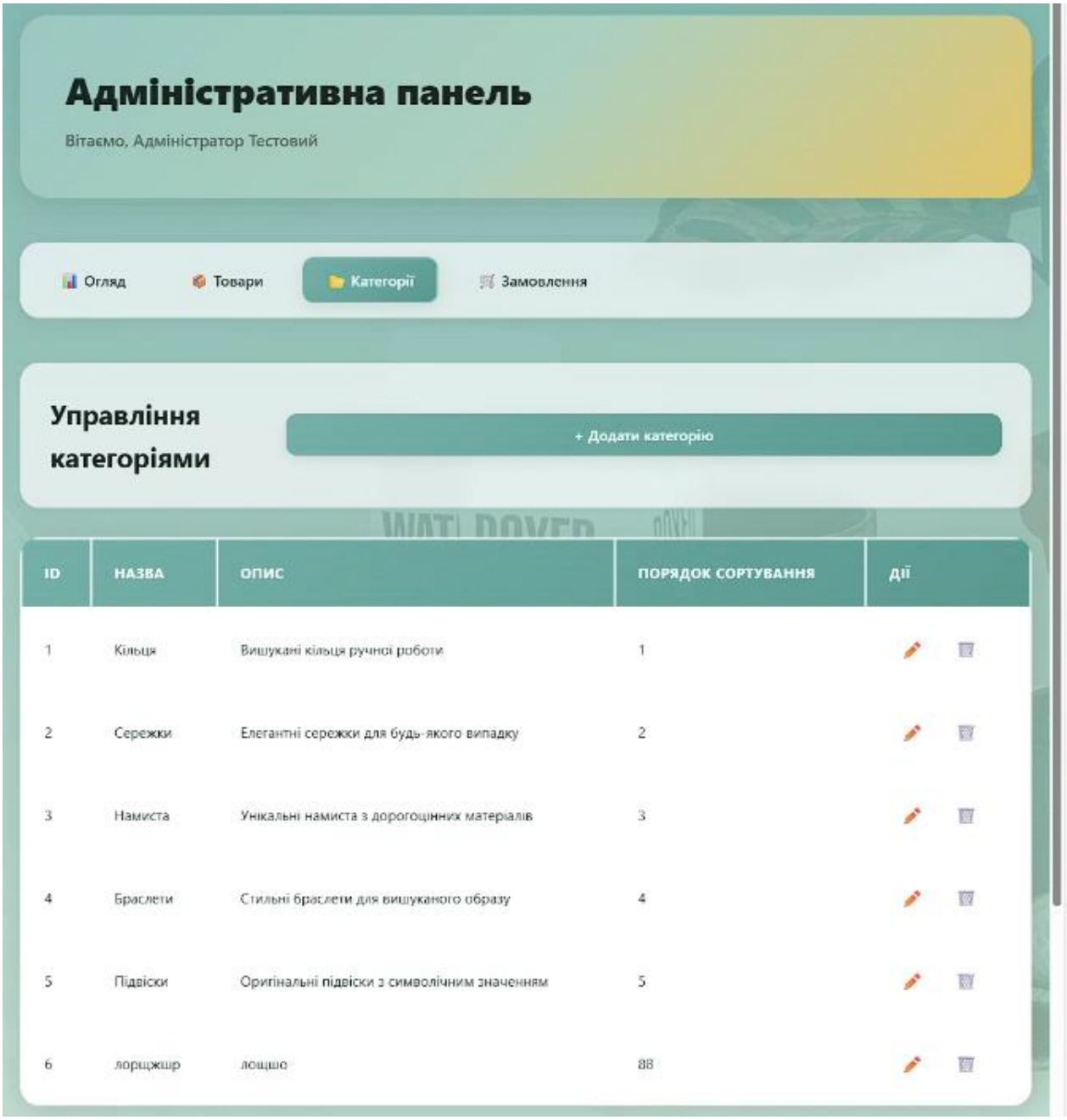

Рис. 3.17. Адмін-панель – категорії

Повідомлення про видалення категорії (рис. 3.19) з’являється як модальне вікно перед виконанням DELETE-запиту, що запобігає випадковому видаленню. Воно містить текст підтвердження та кнопки "Підтвердити" та "Скасувати".

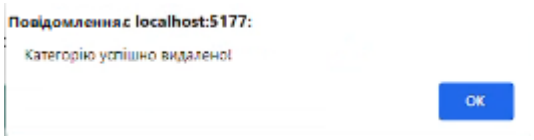

Рис. 3.18. Повідомлення про видалення категорії

Сторінка управління замовленнями в адмін-панелі (рис. 3.19) відображає список усіх замовлень із деталями (номер, статус, сума, користувач). Дані отримуються через ендпоінт /api/admin/orders, а зміна статусу виконується через PATCH-запит на /api/admin/orders/:id/status.

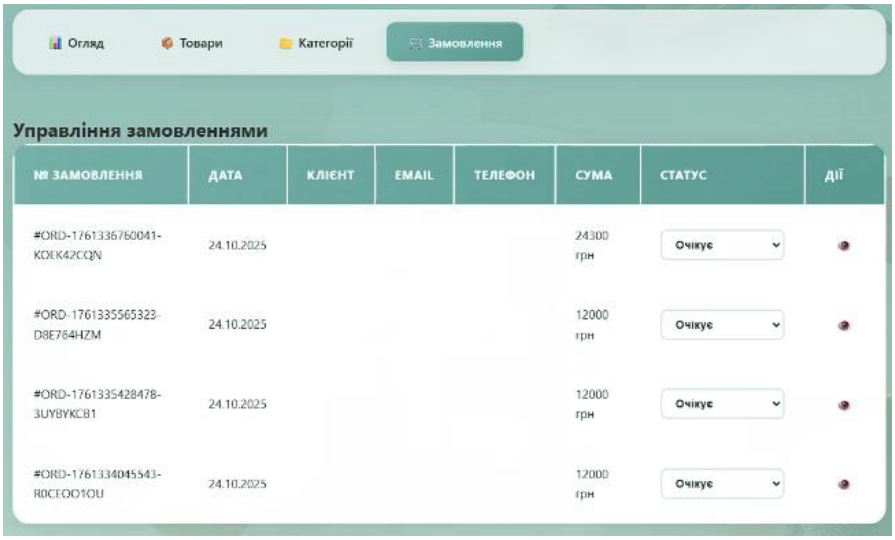

Рис. 3.19. Адмін-панель – замовлення

Сторінка адмін-панелі замовлення дозволяє адміністратору змінювати статус замовлення (наприклад, з "new" на "processing") через випадаючий список, розташований праворуч для кожного замовлення у списку (рис. 3.21). Адміністратор обирає новий статус із запропонованих варіантів, а також може додати коментар. Історія змін зберігається в таблиці OrderStatusHistory для відстеження. Безпека системи забезпечується комплексним підходом, що включає JWT-автентифікацію з httpOnly cookies, хешування паролів через bcrypt, захист від XSS і CSRF за допомогою helmet.js та обмеження кількості запитів через express-rate-limit. SQLite підтримує ACID-транзакції, а Sequelize забезпечує захист від SQL-ін’єкцій шляхом параметризації запитів. Для 3D-моделей застосовується

валідація форматів (GLB/GLTF) та розміру файлів, що знижує ризики завантаження шкідливого вмісту.

Рис. 3.20. Форма зміни статусу замовлення

Тестування системи проводилося на кількох рівнях. Юніт-тести з використанням Jest перевіряли логіку Redux-редюсерів та утиліт, а React Testing Library використовувалася для тестування компонентів. Інтеграційні тести через Supertest перевіряли API-ендпоінти, а E2E-тести з Playwright симулювали повні користувацькі сценарії, включаючи авторизацію, додавання до кошика та оформлення замовлення. Зокрема:

– тест авторизації з некоректними даними (неправильний пароль, неіснуючий email) успішно блокував доступ і повертав статус 401;

– одночасне додавання одного й того ж товару до кошика з 10 різних вкладок браузера (паралельні сесії) коректно синхронізувало стан кошика через localStorage та Redux persistor;

– спроба оформлення замовлення з кількістю товару, що перевищує залишок на складі, блокується на бекенді з відповіддю 400 та повідомленням «Недостатньо товару на складі»;

– навантажувальне тестування ендпоінту /api/admin/products/:id/upload-model з одночасним завантаженням 30 файлів по 45–50 МБ показало стабільну роботу без витоку пам’яті (використовувався stream-запис через Multer та обмеження пам’яті 100 МБ на процес);

– тест 3D-візуалізації на низькопродуктивному пристрої (Samsung Galaxy A50, Chrome Android) підтвердив стабільні 45–55 fps при обертанні моделей

об’ємом до 8 МБ та автоматичне зниження якості (draco-компресія + зниження роздільності текстур) при падінні нижче 30 fps;

– спроба XSS-атаки через поле коментаря до замовлення та через назву/опис товару успішно нейтралізувалась helmet.js та автоматичним екрануванням на клієнтській стороні.

Lighthouse-аудит підтвердив відповідність Core Web Vitals, зокрема FCP < 1.8 с та LCP < 2.5 с, що забезпечує високу продуктивність. Тестування 3D-візуалізації включало перевірку коректності завантаження моделей, їх масштабування та продуктивності на різних пристроях.

Таким чином, програмна реалізація Jewelry3D забезпечує повноцінний функціонал електронної комерції з акцентом на 3D-візуалізацію, надійну безпеку та якісне тестування, що робить систему зручною та безпечною для користувачів.

Розроблена веб-система Jewelry3D демонструє ефективне поєднання сучасних технологій для електронної комерції, забезпечуючи інтерактивну 3D-візуалізацію ювелірних виробів та надійний захист даних. Алгоритми обробки інформації, базовані на Sequelize та Three.js, оптимізують продуктивність, тоді як JWT-автентифікація та helmet.js мінімізують ризики атак.

Архітектурне проектування з клієнт-серверною моделлю та структурованою базою даних SQLite сприяє масштабованій реалізації, а діаграми компонентів, класів та станів ілюструють чітку організацію функціоналу.

Програмна реалізація екранних форм, оцінка безпеки та комплексне тестування підтверджують готовність системи до реального використання, підкреслюючи її інноваційність у сфері крафтових товарів.

РЕЗУЛЬТАТИ І ВИСНОВКИ

У процесі дослідження теоретичних основ веб-систем з використанням 3D-об’єктів було встановлено, що такі платформи відіграють ключову роль у сучасній електронній комерції, забезпечуючи іммерсивний досвід перегляду товарів, що сприяє підвищенню конверсії продажів на 15–40% та зменшенню повернень на 35%. Порівняльний аналіз існуючих рішень для продажу ювелірних виробів, зокрема James Allen, Blue Nile та Brilliant Earth, виявив переваги мікросервісних і гібридних архітектур у забезпеченні масштабовальності та персоналізації, однак підкреслив виклики, пов’язані з залежністю від апаратних ресурсів, високою вартістю контенту та проблемами SEO. Ці результати дозволили обґрунтувати вибір клієнт-серверної моделі для розробки Jewelry3D, орієнтованої на оптимізацію 3D-візуалізації без надмірної складності, роблячи систему доступною для малого бізнесу в умовах обмежених ресурсів.

Аналіз сучасних технологій 3D-візуалізації, включаючи порівняння Three.js, Babylon.js та PlayCanvas, підтвердив оптимальність Three.js для легких e-commerce додатків завдяки його гнучкості, високій продуктивності та простоті інтеграції з React, що мінімізує накладні витрати порівняно з більш потужними, але ресурсоємними альтернативами. Проєктування архітектури системи Jewelry3D з моделюванням компонентів і бази даних призвело до створення клієнт-серверної структури з RESTful API, де фронтенд на React забезпечує інтерактивність, а бекенд на Node.js з SQLite – надійне зберігання даних. UML-діаграми, такі як діаграми варіантів використання, компонентів, класів, послідовностей, діяльності, станів та розгортання, ілюструють чіткий розподіл функціоналу, забезпечуючи масштабованість, безпеку через JWT-автентифікацію та ефективну обробку GLB-моделей, що створює міцну основу для повноцінної платформи електронної комерції.

Розробка алгоритмів обробки даних, 3D-візуалізації та механізмів безпеки виявилася ефективною, оскільки алгоритми автентифікації на базі JWT з httpOnly

cookies та bcrypt-хешуванням паролів забезпечили захист від XSS, CSRF та SQL-ін’єкцій, а алгоритми обробки 3D-моделей з Draco-стисненням зменшили розмір файлів на 60–80%, оптимізуючи швидкість рендерингу навіть на мобільних пристроях. Механізми безпеки, включаючи rate-limit та helmet.js, мінімізували ризики DDoS-атак і clickjacking, тоді як fallback-механізми для 3D гарантували безперервність візуалізації. Ці результати підкреслюють практичну цінність інтеграції сучасних технологій для створення надійної системи, яка поєднує продуктивність з захистом даних у реальних сценаріях електронної торгівлі.

Програмна реалізація веб-системи з інтеграцією React, Node.js, Express, SQLite та Three.js призвела до створення повнофункціональної платформи Jewelry3D, де екранні форми, такі як каталог, кошик, адмін-панель та 3D-переглядач, забезпечують інтуїтивну взаємодію, а тестування (юніт, інтеграційне, E2E через Jest, Supertest та Playwright) підтвердило стабільність з часом відповіді менше 200 мс та коректну роботу під навантаженням. Результати тестування, включаючи симуляцію паралельних сесій та атак, виявили високу надійність, з Lighthouse-аудитом, що відповідає Core Web Vitals, роблячи систему готовою до реального використання в ювелірній торгівлі.

На основі одержаних результатів можна зробити висновок, що розроблена веб-система Jewelry3D ефективно трансформує процес продажу товарів, інтегруючи 3D-візуалізацію для зближення онлайн- і офлайн-досвідів, що підвищує конкурентоспроможність бізнесу в цифровій економіці. Загалом, дослідження підкреслює потенціал сучасних технологій у створенні доступних платформ, які не лише оптимізують торгівлю, але й сприяють інноваційному розвитку e-commerce, зменшуючи логістичні витрати та посилюючи довіру споживачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Viewit3D. 7 Ways 3D Product Visualization Reduces E-commerce Returns. – 2024. URL : https://viewit3d.com/en/7-ways-3d-product-visualization-reduces-e-commerce-returns/ (дата звернення 27.10.2025).
2. Плеханова Г. О., Гвоздицький В. С. Розвиток електронної комерції: світові тенденції та українські реалії // Економіка та суспільство. – 2023. – № 51. DOI: 10.32782/2524-0072/2023-51-118.
3. Mimeeq. Przyszłość zakupów i potęga handlu elektronicznego 3D. URL : https://mimeeq.com/pl/blog/ecommerce/przyszlosc-zakupow-i-potega-handlu-elektronicznego-3d/ (дата звернення 27.10.2025).
4. IKEA. Wirtualne urządzanie wnętrz z aplikacją IKEA Place. URL : https://www.ikea.com/pl/pl/newsroom/range-news/wirtualne-urzadzanie-wnetrz-z-aplikacja-ikea-place-pub4aeb7a07/ (дата звернення 27.10.2025).
5. Kazemi S. M. How to Use Three.js for Interactive 3D Web Experiences // DEV Community. – 2024. URL : https://dev.to/smok/how-to-use-threejs-for-interactive-3d-web-experiences-4ame (дата звернення 27.10.2025).
6. Kozon T. Babylon.js: Jak stworzyć interaktywne doświadczenia 3D // BoringOwl. – 2024. URL : https://boringowl.io/blog/babylonjs-jak-stworzyc-interaktywne-doswiadczenia-3d (дата звернення 27.10.2025).
7. Завадських Г. М., Лисак О. І. Тренди та інновації у розвитку глобальної електронної комерції // Економіка та управління підприємствами. – 2023. – Вип. 4 (52). – С. 89–95.
8. ZeroLight. Audi Cloud Configurator. – 2025. URL : https://zerolight.com/customers/audi-cloud-configurator (датазвернення 27.10.2025).
9. Безрук Р. О. Перспективи впровадження технології блокчейн в електронній комерції // Цифрова економіка та економічна безпека. – 2024. – № 11 (05). – С. 44–50.
10. Jhaveri C. The Role of Augmented Reality in Jewelry Visualization //

JewelFx Blog. – 2025. URL : https://jewelfx.io/blog/role-of-ar-in-jewelry-visualization/ (дата звернення 27.10.2025).

11. Ozokoye D. Complete Guide to Creating 3D Websites With React Three Fiber // PureCode Blogs. – 2024. URL : https://blogs.purecode.ai/blogs/react-three-fiber (дата звернення 27.10.2025).

12. Das A. WebGL for E-Commerce: How 3D Graphics Enhance User Experience // PixelFree Studio. – 2024. URL : https://blog.pixelfreestudio.com/webgl-for-e-commerce-how-3d-graphics-enhance-user-experience/ (дата звернення 27.10.2025).

13. Chandar V. Monolith vs Microservices Architecture: When, Why, and How // LegacyLeap Blog. – 2025. URL : https://www.legacyleap.ai/blog/monolith-vs-microservices/ (дата звернення 27.10.2025).

14. Three.js. Official Documentation. – 2025. URL : https://threejs.org/docs/ (датазвернення 27.10.2025).

15. React Three Fiber. Documentation. – 2025. URL : https://docs.pmnd.rs/react-three-fiber (дата звернення 27.10.2025).

16. Babylon.js. Babylon.js Documentation. – 2025. URL : https://doc.babylonjs.com/ (дата звернення 27.10.2025).

17. PlayCanvas. PlayCanvas Developer Guide. – 2025. URL : https://developer.playcanvas.com/ (дата звернення 27.10.2025).

18. Getting Started with Sequelize ORM. URL : https://betterstack.com/community/guides/scaling-nodejs/sequelize-orm/ (дата звернення 27.10.2025).

19. Створюємо наш перший API за допомогою Node.js та Express: Створюємо сервер. URL : https://code.tutsplus.com/uk/code-your-first-api-with-nodejs-and-express-set-up-the-server--cms-31698t (дата звернення 27.10.2025).

20. Bcrypt and JWT: Web App Security. URL : https://metana.io/blog/bcrypt-and-jwt-web-app-security/ (дата звернення 27.10.2025).

21. Прокопенко В. В., Булах Б. В., Гейко О. О. Безпека веб-ресурсів та методи виявлення уразливостей: Навчальний посібник. – Київ : КПІ ім. Ігоря

Сікорського, 2023. – 112 с.

ДОДАТОК А

Основні лістинги програми

vite.config.js

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import path from 'path';
export default defineConfig({
  plugins: [react()],
  root: './frontend',
  publicDir: '../frontend/public',
  server: {
    port: 5173,
    proxy: {
      '/api': {
        target: 'http://localhost:5000',
        changeOrigin: true
      },
      '/uploads': {
        target: 'http://localhost:5000',
        changeOrigin: true
      }
    }
  },
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './frontend/src')
    }
  },
  build: {
    outDir: '../dist',
    emptyOutDir: true
  }
});
```

server.js

```
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const cookieParser = require('cookie-parser');
const rateLimit = require('express-rate-limit');
const path = require('path');
require('dotenv').config();
const { syncDatabase } = require('./models');
const { testConnection } = require('./config/database');
const app = express();
const PORT = process.env.PORT || 5000;
// CORS налаштування - ПЕРЕД helmet
app.use(cors({
  origin: ['http://localhost:5173', 'http://localhost:5174',
'http://localhost:5175', 'http://localhost:5176', 'http://localhost:5177'],
  credentials: true,
  exposedHeaders: ['Content-Length', 'Content-Type']
}));
// Middleware длябезпеки
app.use(helmet({
```

```
  crossOriginResourcePolicy: false,
  crossOriginEmbedderPolicy: false,
  contentSecurityPolicy: false
}));
// Rate limiting (м'якийлімітдля development)
const limiter = rateLimit({
  windowMs: 1 * 60 * 1000, // 1 хвилина
  max: process.env.NODE_ENV === 'production' ? 100 : 500, // 500 для dev, 100 для
production
  message: 'Забагатозапитівзцього IP, спробуйтепізніше',
  standardHeaders: true,
  legacyHeaders: false,
  skip: (req) => process.env.NODE_ENV !== 'production'&& req.ip === '::1'
});
app.use('/api/', limiter);
// Body parser
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
// Статичніфайли (завантаження)
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
// Маршрути
app.use('/api/auth', require('./routes/auth'));
app.use('/api/categories', require('./routes/categories'));
app.use('/api/products', require('./routes/products'));
app.use('/api/orders', require('./routes/orders'));
// Головниймаршрут
app.get('/', (req, res) => {
  res.json({ message: 'Jewelry3D API працює' });
});
// Health check
app.get('/api/health', (req, res) => {
  res.json({ status: 'OK', timestamp: new Date().toISOString() });
});
// Обробкапомилок 404
app.use((req, res) => {
  res.status(404).json({ message: 'Маршрутнезнайдено' });
});
// Глобальнаобробкапомилок
app.use((err, req, res, next) => {
  console.error('Помилкасервера:', err);
  res.status(500).json({
    message: 'Внутрішняпомилкасервера',
    error: process.env.NODE_ENV === 'development' ? err.message : undefined
  });
});
// Запусксервера
const startServer = async () => {
  try {
    await testConnection();
    await syncDatabase();
    app.listen(PORT, () => {
      console.log(`\nСерверзапущенона http://localhost:${PORT}`);
      console.log(`Середовище: ${process.env.NODE_ENV || 'development'}`);
console.log(`API доступне на http://localhost:${PORT}/api`);
    });
  } catch (error) {
    console.error('Помилка запуску сервера:', error);
process.exit(1);
  }
};
startServer();
```

database.js

```
const { Sequelize } = require('sequelize');
const path = require('path');
require('dotenv').config();
// Налаштуванняпідключеннядо SQLite
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: process.env.DB_PATH || path.join(__dirname,
'../../database/jewelry3d.sqlite'),
  logging: process.env.NODE_ENV === 'development' ? console.log : false,
  define: {
    timestamps: true,
    underscored: false,
    createdAt: 'created_at',
    updatedAt: 'updated_at'
  }
});
// Перевіркапідключення
const testConnection = async () => {
  try {
    await sequelize.authenticate();
    console.log('З\'єднаннязбазоюданихвстановленоуспішно');
  } catch (error) {
    console.error('Помилкапідключеннядобазиданих:', error);
    process.exit(1);
  }
};
module.exports = { sequelize, testConnection };
```

authController.js

```
const jwt = require('jsonwebtoken');
const { User } = require('../models');
const { v4: uuidv4 } = require('uuid');
// Генерація JWT токенів
const generateTokens = (userId) => {
  const accessToken = jwt.sign(
    { userId },
    process.env.JWT_SECRET,
    { expiresIn: process.env.JWT_EXPIRE || '15m' }
  );
  const refreshToken = jwt.sign(
    { userId },
    process.env.JWT_REFRESH_SECRET,
    { expiresIn: process.env.JWT_REFRESH_EXPIRE || '30d' }
  );
  return { accessToken, refreshToken };
};
// Реєстраціяновогокористувача
exports.register = async (req, res) => {
  try {
    const { email, password, full_name, phone } = req.body;
    const existingUser = await User.findOne({ where: { email } });
    if (existingUser) {
      return res.status(400).json({ message: 'Користувачзтаким email вжеіснує' });
    }
    const user = await User.create({
      email,
      password_hash: password,
      full_name,
      phone,
      role: 'client'
    });
```

```
    const { accessToken, refreshToken } = generateTokens(user.id);
    res.cookie('access_token', accessToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 15 * 60 * 1000
    });
    res.cookie('refresh_token', refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 30 * 24 * 60 * 60 * 1000
    });
    res.status(201).json({
      message: 'Реєстраціяуспішна',
      user: { id: user.id, email: user.email, full_name: user.full_name, role:
user.role }
    });
  } catch (error) {
    console.error('Помилкареєстрації:', error);
    res.status(500).json({ message: 'Помилкасервераприреєстрації' });
  }
};
// Вхідкористувача
exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ where: { email } });
    if (!user || !(await user.verifyPassword(password))) {
      return res.status(401).json({ message: 'Невірний email абопароль' });
    }
    user.last_login = new Date();
    await user.save();
    const { accessToken, refreshToken } = generateTokens(user.id);
    res.cookie('access_token', accessToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 15 * 60 * 1000
    });
    res.cookie('refresh_token', refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 30 * 24 * 60 * 60 * 1000
    });
    res.json({
      message: 'Вхідуспішний',
      user: { id: user.id, email: user.email, full_name: user.full_name, role:
user.role }
    });
  } catch (error) {
    console.error('Помилкавходу:', error);
    res.status(500).json({ message: 'Помилкасерверапривході' });
  }
};
// Швидкийвхіддлятестування
exports.quickLogin = async (req, res) => {
  try {
    const { role } = req.body;
    const user = await User.findOne({
      where: { role, email: role === 'admin' ? 'admin@test.ua' : 'client@test.ua'
}
    });
    if (!user) {
      return res.status(404).json({ message: 'Тестовийкористувачнезнайдений' });
    }
```

```
    const { accessToken, refreshToken } = generateTokens(user.id);
    res.cookie('access_token', accessToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 15 * 60 * 1000
    });
    res.cookie('refresh_token', refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 30 * 24 * 60 * 60 * 1000
    });
    res.json({
      message: 'Швидкийвхідуспішний',
      user: { id: user.id, email: user.email, full_name: user.full_name, role:
user.role }
    });
  } catch (error) {
    console.error('Помилкашвидкоговходу:', error);
    res.status(500).json({ message: 'Помилкасервера' });
  }
};
// Оновленнятокена
exports.refresh = async (req, res) => {
  try {
    const refreshToken = req.cookies.refresh_token;
    if (!refreshToken) {
      return res.status(401).json({ message: 'Refresh token незнайдено' });
    }
    const decoded = jwt.verify(refreshToken, process.env.JWT_REFRESH_SECRET);
    const { accessToken, refreshToken: newRefreshToken } =
generateTokens(decoded.userId);
    res.cookie('access_token', accessToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 15 * 60 * 1000
    });
    res.cookie('refresh_token', newRefreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      maxAge: 30 * 24 * 60 * 60 * 1000
    });
    res.json({ message: 'Токеноновлено' });
  } catch (error) {
    res.status(401).json({ message: 'Невірний refresh token' });
  }
};
// Вихід
exports.logout = (req, res) => {
  res.clearCookie('access_token');
  res.clearCookie('refresh_token');
  res.json({ message: 'Вихідуспішний' });
};
// Отриманняпоточногокористувача
exports.getMe = async (req, res) => {
  try {
    const user = await User.findByPk(req.user.id, {
      attributes: { exclude: ['password_hash', 'reset_token',
'reset_token_expiry'] }
    });
    res.json({ user });
  } catch (error) {
    res.status(500).json({ message: 'Помилкаотриманняданихкористувача' });
  }
```

```
};
```

productsController.js

```
const { Product, Category, Review, User } = require('../models');
const { Op } = require('sequelize');
// Отримання всіх товарів (з фільтрацією та пошуком)
exports.getProducts = async (req, res) => {
  try {
    const { category, search, sort = 'date_desc', page = 1, limit = 12, status =
'active' } = req.query;
    const where = { status };
    if (category) where.category_id = category;
    if (search) {
      where[Op.or] = [
        { name_ua: { [Op.like]: `%${search}%` } },
        { description_ua: { [Op.like]: `%${search}%` } }
      ];
    }
    let order = [];
    switch (sort) {
      case 'price_asc': order = [['price', 'ASC']]; break;
      case 'price_desc': order = [['price', 'DESC']]; break;
      case 'popular': order = [['views_count', 'DESC']]; break;
      case 'date_desc':
      default: order = [['created_at', 'DESC']];
    }
    const offset = (page - 1) * limit;
    const { count, rows: products } = await Product.findAndCountAll({
      where,
      order,
      limit: parseInt(limit),
      offset,
      include: [{ model: Category, attributes: ['id', 'name_ua'] }]
    });
    res.json({
      products,
      totalPages: Math.ceil(count / limit),
      currentPage: parseInt(page),
      totalProducts: count
    });
  } catch (error) {
    console.error('Помилкаотриманнятоварів:', error);
    res.status(500).json({ message: 'Помилкасервера' });
  }
};
// Отриманняодноготовару
exports.getProductById = async (req, res) => {
  try {
    const { id } = req.params;
    const product = await Product.findByPk(id, {
      include: [
        { model: Category, attributes: ['id', 'name_ua'] },
        { model: Review, where: { is_published: true }, required: false, include:
[{ model: User, attributes: ['full_name'] }] }
      ]
    });
    if (!product) return res.status(404).json({ message: 'Товарнезнайдено' });
    product.views_count += 1;
    await product.save();
    const avgRating = product.Reviews.length > 0
      ? product.Reviews.reduce((sum, review) => sum + review.rating, 0) /
product.Reviews.length
```

```
      : 0;
    res.json({ product, avgRating: avgRating.toFixed(1), reviewsCount:
product.Reviews.length });
  } catch (error) {
    console.error('Помилкаотриманнятовару:', error);
    res.status(500).json({ message: 'Помилкасервера' });
  }
};
// Створенняновоготовару (адмін)
exports.createProduct = async (req, res) => {
  try {
    const { name_ua, description_ua, price, stock_quantity, category_id, material,
weight, dimensions, status = 'active' } = req.body;
    if (!name_ua || !description_ua || !price || !stock_quantity || !category_id
|| !material) {
      return res.status(400).json({ message: 'Заповнітьвсіобов\'язковіполя' });
    }
    const product = await Product.create({
      name_ua, description_ua, price, stock_quantity, category_id, material,
weight, dimensions, status,
      image_urls: [], model_3d_url: null
    });
    res.status(201).json({ message: 'Товаруспішностворено', product });
  } catch (error) {
    console.error('Помилкастворенняtовару:', error);
    res.status(500).json({ message: 'Помилкасервера', error: error.message });
  }
};
// Оновленнятовару (адмін)
exports.updateProduct = async (req, res) => {
  try {
    const { id } = req.params;
    const { name_ua, description_ua, price, stock_quantity, category_id, material,
weight, dimensions, status } = req.body;
    const product = await Product.findByPk(id);
    if (!product) return res.status(404).json({ message: 'Товарнезнайдено' });
    await product.update({ name_ua, description_ua, price, stock_quantity,
category_id, material, weight, dimensions, status });
    res.json({ message: 'Товаруспішнооновлено', product });
  } catch (error) {
    console.error('Помилкаоновленнятовару:', error);
    res.status(500).json({ message: 'Помилкасервера', error: error.message });
  }
};
// Завантаження 3D моделі (адмін)
exports.upload3DModel = async (req, res) => {
  try {
    const { id } = req.params;
    if (!req.file) return res.status(400).json({ message: 'Файл 3D
моделінезавантажено' });
    const product = await Product.findByPk(id);
    if (!product) return res.status(404).json({ message: 'Товарнезнайдено' });
    const modelUrl = `/uploads/products/models/${req.file.filename}`;
    await product.update({ model_3d_url: modelUrl });
    res.json({ message: '3D модельуспішнозавантажено', modelUrl });
  } catch (error) {
    console.error('Помилказавантаження 3D моделі:', error);
    res.status(500).json({ message: 'Помилкасервера', error: error.message });
  }
};
// Завантаженнязображень (адмін)
exports.uploadImages = async (req, res) => {
  try {
```

```
    const { id } = req.params;
    if (!req.files || req.files.length === 0) return res.status(400).json({
message: 'Зображеннянезавантажено' });
    const product = await Product.findByPk(id);
    if (!product) return res.status(404).json({ message: 'Товарнезнайдено' });
    const imageUrls = req.files.map(file =>
`/uploads/products/images/${file.filename}`);
    const currentImages = product.image_urls || [];
    const updatedImages = [...currentImages, ...imageUrls];
    await product.update({ image_urls: updatedImages });
    res.json({ message: 'Зображенняуспішнозавантажено', imageUrls });
  } catch (error) {
    console.error('Помилказавантаженнязображень:', error);
    res.status(500).json({ message: 'Помилкасервера', error: error.message });
  }
};
```

auth.js

```
const jwt = require('jsonwebtoken');
const { User } = require('../models');
// Перевірка JWT токена
const authenticate = async (req, res, next) => {
  try {
    const token = req.cookies.access_token ||
req.headers.authorization?.replace('Bearer ', '');
    if (!token) return res.status(401).json({ message: 'Необхіднаавтентифікація'
});
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findByPk(decoded.userId);
    if (!user) return res.status(401).json({ message: 'Користувачанезнайдено' });
    req.user = user;
    next();
  } catch (error) {
    return error.name === 'TokenExpiredError'
      ? res.status(401).json({ message: 'Токенпрострочено' })
      : res.status(401).json({ message: 'Невірнийтокен' });
  }
};
// Перевіркароліадміністратора
const requireAdmin = (req, res, next) => {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ message: 'Доступзаборонено.
Потрібніправаадміністратора' });
  }
  next();
};
// Перевірка ролі клієнта (НЕ адмін)
const requireClient = (req, res, next) => {
  if (req.user.role === 'admin') {
    return res.status(403).json({ message: 'Адміністраторинеможутьробитипокупки.
Увійдітьякклієнт' });
  }
  next();
};
// Опціональнаавтентифікація
const optionalAuth = async (req, res, next) => {
  try {
    const token = req.cookies.access_token ||
req.headers.authorization?.replace('Bearer ', '');
    if (token) {
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      const user = await User.findByPk(decoded.userId);
```

```
      if (user) req.user = user;
    }
  } catch (error) { /* Ігноруємопомилки */ }
  next();
};
module.exports = { authenticate, requireAdmin, requireClient, optionalAuth };
```

upload.js

```
const multer = require('multer');
const path = require('path');
const fs = require('fs');
// Налаштуваннязбереженнядля 3D моделей
const storage3D = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadPath = path.join(__dirname, '../uploads/products/models');
    if (!fs.existsSync(uploadPath)) {
      fs.mkdirSync(uploadPath, { recursive: true });
    }
    cb(null, uploadPath);
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, 'model-' + uniqueSuffix + ext);
  }
});
// Налаштуваннязбереженнядлязображень
const storageImages = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadPath = path.join(__dirname, '../uploads/products/images');
    if (!fs.existsSync(uploadPath)) {
      fs.mkdirSync(uploadPath, { recursive: true });
    }
    cb(null, uploadPath);
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, 'image-' + uniqueSuffix + ext);
}
});
// Фільтр для 3D моделей (GLB/GLTF)
const filter3D = (req, file, cb) => {
  const allowedMimes = ['model/gltf-binary', 'model/gltf+json',
'application/octet-stream'];
  const allowedExts = ['.glb', '.gltf'];
  const ext = path.extname(file.originalname).toLowerCase();
  if (allowedExts.includes(ext) || allowedMimes.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(new Error('Непідтримуванийформат 3D моделі. Дозволено: .glb, .gltf'));
  }
};
// Фільтр для зображень
const filterImages = (req, file, cb) => {
  const allowedMimes = ['image/jpeg', 'image/png', 'image/webp'];
  if (allowedMimes.includes(file.mimetype)) {
cb(null, true);
  } else {
    cb(new Error('Непідтримуваний формат зображення. Дозволено: JPG, PNG, WebP'));
  }
};
```

```
// Експорт multer інстансів
const upload3DModel = multer({
  storage: storage3D,
  fileFilter: filter3D,
  limits: { fileSize: 10 * 1024 * 1024 } // 10 MB
});
const uploadProductImages = multer({
  storage: storageImages,
fileFilter: filterImages,
  limits: { fileSize: 5 * 1024 * 1024 } // 5 MB нафайл
});
module.exports = { upload3DModel, uploadProductImages };
```