МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ПРОДАЖУ ЮВЕЛІРНИХ ВИРОБІВ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувачка _____ **Маргарита ФОМЕНКО**

«__» _____ 2026 р.

Керівник роботи

Канд. техн. наук, доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«»_____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Фоменко Маргарити

1. Тема кваліфікаційної роботи Вебзастосунок продажу ювелірних виробів затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Очікуваним результатом є вебзастосунок продажу ювелірних виробів

4. Перелік питань, що підлягають розробці:

- оцінка стану ринку та огляд існуючих конкурентних платформ;
- вибір та аргументація технологічного стека для реалізації проєкту;
- розробка логічної та фізичної схеми бази даних;
- створення UI/UX дизайну клієнтської частини;

– проведення верифікації та тестування готового продукту.

5. Перелік графічних матеріалів:

Презентація

Дата видачі завдання «9» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебзастосунок продажу ювелірних виробів**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	22.11.2025	02.12.2025	Виконано
2.	Огляд літератури за темою роботи	21.01.2026	31.01.2026	Виконано
3.	Складання календарного плану КБР	08.02.2026	09.02.2026	Виконано
4.	Аналіз предметної області	11.02.2026	12.02.2026	Виконано
5.	Розробка проєктних рішень	13.02.2026	15.02.2026	Виконано
6.	Моделювання та конструювання ПЗ	27.02.2026	05.03.2026	Виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	15.03.2026	09.04.2026	Виконано
8.	Оформлення КБР та презентації	10.04.2026	26.05.2026	Виконано
9.	Відгук керівника КБР			
10.	Попередній захист	27.05.2026	27.05.2026	Виканан
11.	Рецензування			
12.	Завершення оформлення КБР та презентації			
13.	Захист кваліфікаційної роботи			

Здобувачка _____ **Маргарита ФОМЕНКО**

«9» лютого 2026 р.

Керівник роботи

Канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«9» лютого 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Вебзастосунок продажу ювелірних виробів»

Здобувачка 409 гр.: Маргарита Фоменко

Керівник: канд. техн. наук, доцент Євген Давиденко

Кваліфікаційна робота сфокусована на створенні онлайн-платформи для продажу ювелірних прикрас та виробів. Проєкт спрямований на формування ефективного та зручного цифрового середовища, де клієнти можуть оперативно підбирати прикраси, вивчати характеристики виробів та користуватися всіма перевагами актуальної електронної комерції.

Метою роботи є розробка вебзастосунку продажу ювелірних виробів. Система має гарантувати стабільну роботу з базою даних, надавати клієнтам доступ до вітрини прикрас, а менеджерам – інтуїтивно зрозумілі інструменти для оновлення товарів та контролю за контентом.

Об’єктом роботи є процес електронної комерції інтернет-магазину ювелірної продукції.

Предметом кваліфікаційної роботи є інструментарій розробки вебзастосунку ювелірних виробів.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі визначається актуальність теми, мета, предмет та об’єкт роботи.

У першому розділі виконано аналіз існуючих вебзастосунків-аналогів, визначено їхній функціонал, переваги, недоліки та описано технології, за допомогою яких створено вебзастосунок. Визначено структурні та функціональні особливості ПЗ, що розроблюється.

Другий розділ присвячено моделюванню об’єкту та предмету роботи. Тут формується специфікація вимог до програмного забезпечення, а також розробляються функціональні й інформаційні моделі вебзастосунку.

У третьому розділі описується загальна архітектура програмного продукту, а також проводиться моделювання та проєктування окремих компонентів:

структури бази даних, клієнтської та серверної частин вебзастосунку для продажу ювелірних виробів.

Четвертий розділ присвячений безпосередній програмній реалізації (кодуванню) застосунку. Описується процес тестування (верифікація, валідація) програмного забезпечення, виконання обчислень та аналіз їхніх результатів. Крім того, представлено детальне керівництво користувача для роботи з розробленою системою.

Кваліфікаційна робота викладена на 66 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 15 найменувань та 2 додатків. Праця містить 2 таблиці та 25 рисунків.

Ключові слова: *вебзастосунок, Python, Django, Django REST Framework, React, NextJS.*

ABSTRACT

to the qualifying bachelor's thesis

«Web application for selling jewelry»

Student of 409 group: Marharyta Fomenko

Supervisor: Candidate of Technical Sciences,

Associate Professor Davydenko Yevhen

The qualification work focuses on the creation of an online platform for selling jewelry and related products. The project aims to establish an efficient and user-friendly digital environment where customers can quickly select jewelry, review product specifications, and enjoy all the benefits of modern e-commerce.

The goal of this work is to develop a web application for selling jewelry. The system must ensure stable database operations, provide customers with access to the jewelry showcase, and offer managers intuitive tools for updating products and managing content.

The object of this work is a process of e-commerce for an online jewelry store.

The subject of this thesis is the toolkit for developing a jewelry web application.

The thesis consists of an introduction, four chapters, conclusions, and a list of references.

The introduction defines the relevance of the topic, the purpose, subject, and object of the work.

The first chapter analyzes existing similar web applications, identifies their functionality, advantages, and disadvantages, and describes the technologies used to create the web application. The structural and functional features of the software under development.

The second chapter is devoted to modeling the object and subject of the work. Here, the software requirements specification is formulated, and the functional and information models of the web application are developed.

The third section describes the overall architecture of the software product and includes the modeling and design of individual components: the database structure, and the client and server components of the web application for selling jewelry.

The fourth chapter is devoted to the actual software implementation (coding) of the application. It describes the process of testing (verification, validation) the software, performing calculations, and analyzing their results. In addition, a detailed user guide for working with the developed system is provided.

The conclusions provide an analysis of the performed work, an assessment of the degree of completion of the assigned tasks, and a description of the obtained results.

The qualification work is presented on 66 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 15 titles and 2 appendices. The work contains 2 tables and 25 figures.

Keywords: web application, Python, Django, Django REST Framework, React, NextJS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

DOM – Document Object Model

DRF – Django REST Framework

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

ORM – Object-Relational Mapping

REST – Representational State Transfer

SQL – Structured Query Language

SSR – Server-Side Rendering

UI – User Interface

URL – Uniform Resource Locator

UX – User Experience

БД – база даних

ПЗ – програмне забезпечення

СКБД – система керування базами даних

ВСТУП

Сучасний ринок електронної комерції переживає дуже бурхливий розвиток, який, у поєднанні зі зміною звичок споживачів, призвів до значного зростання популярності онлайн-покупок товарів преміум-сегменту, зокрема ювелірних виробів. Покупці в цифрову епоху вимагають від онлайн-комерції не тільки широкого вибору продукції, але й привабливого представлення товарів, інтуїтивно зрозумілої навігації та бездоганного захисту їхньої особистої та платіжної інформації. Розробка спеціалізованого вебзастосунку продажу ювелірних виробів є надзвичайно актуальною з науково-практичної точки зору. Така платформа стає незамінним інструментом для ювелірних брендів, індивідуальних майстрів та роздрібних продавців, які прагнуть розширити ринок збуту, автоматизувати торгівельні операції та підвищити загальний рівень задоволення уподобань споживачів.

Метою роботи є розробка вебзастосунку продажу ювелірних виробів. Система має гарантувати стабільну роботу з базою даних, надавати клієнтам доступ до вітрини прикрас, а менеджерам – інтуїтивно зрозумілі інструменти для оновлення товарів та контролю за контентом.

Для досягнення поставленої мети необхідно виконати наступний перелік завдань:

1) оцінка стану ринку та огляд існуючих конкурентних платформ;
2) вибір та аргументація інструментів для реалізації проєкту;
3) розробка інформаційної моделі та бази даних;
4) створення UI/UX дизайну клієнтської частини;
5) проведення верифікації та тестування готового продукту.

Об’єктом роботи є процес електронної комерції інтернет-магазину ювелірної продукції.

Предметом кваліфікаційної роботи є інструментарій розробки вебзастосунку ювелірних виробів.

1 СИСТЕМНИЙ АНАЛІЗ ОБРАНОЇ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Специфіка електронної комерції у ювелірній галузі

Сучасний ринок електронної комерції вражає своїми темпами зростання, проте продаж ювелірних виробів онлайн – це зовсім інша історія. На відміну від товарів повсякденного вжитку, ювелірні прикраси належать до преміального сегмента, де покупка – це не лише раціональний вибір, а й глибоко емоційний процес, що базується на довірі до бренду та особистому зв'язку з виробом.

Метою цього розділу є комплексний аналіз предметної області електронної комерції у сфері реалізації ювелірної продукції. Особлива увага приділяється дослідженню необхідних функціональних можливостей інтернет-магазинів, вибору оптимальних архітектурних рішень, а також аналізу критичних вимог до захисту транзакцій, збереження конфіденційності клієнтських даних та загальної продуктивності системи. Таке дослідження дозволить сформувати ґрунтовну теоретичну базу та визначити ключові технічні вимоги для подальшого проєктування власного вебзастосунку ювелірного бренду.

Аналітичний звіт «The State of Fashion: Watches and Jewellery» від McKinsey & Company свідчить про стійку цифровізацію світового ринку предметів розкоші. Прогнозується, що в найближчі роки частка онлайн-продажів ювелірних виробів значно збільшиться, перевищивши 20% від загального обсягу ринку [1].

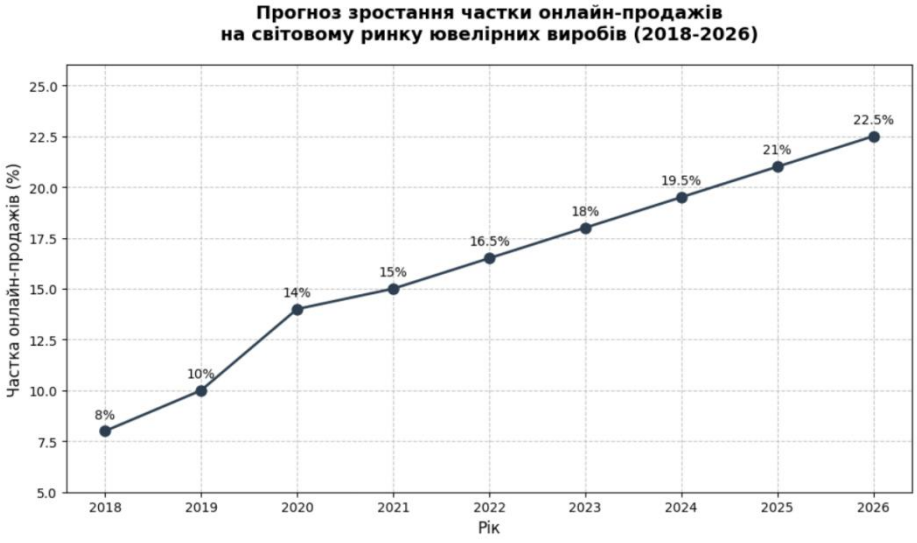

Рисунок 1.1 – Графік прогнозування розвитку ринку ювелірного бізнесу

Аналіз специфіки ювелірної галузі виявив ключові вимоги до програмних рішень:

1) висока якість візуалізації: Для демонстрації ювелірних виробів необхідна максимальна деталізація. Інтерактивні 3D-моделі та панорамні огляди підвищують залученість клієнтів та конверсію, одночасно знижуючи відсоток повернень дорогих товарів [2];

2) складність опису товарів: Ювелірні вироби мають багатий набір характеристик (метал, проба, вага, тип і якість каменів, огранювання, розмір). Це потребує гнучкої бази даних та ефективної системи багатофакторної фільтрації;

3) безпека платежів та даних: Висока вартість товарів вимагає абсолютних гарантій безпеки онлайн-платежів. Необхідно забезпечити надійне зберігання персональних даних та цифрових сертифікатів автентичності;

4) швидкість роботи та зручність користування: Клієнти преміум-сегменту очікують миттєвого завантаження сторінок та бездоганного інтерфейсу на всіх пристроях. Розподілена архітектура допомагає мінімізувати затримки та прискорити доставку контенту [3].

Таким чином, успішний вебзастосунок для ювелірного бутика має бути не просто інтернет-магазином, а високопродуктивною, візуально привабливою та захищеною, що забезпечує преміальний цифровий досвід.

1.2 Складові частини системи керування продажами ювелірних виробів

У сучасних умовах розвитку цифрової економіки результативність ювелірного бізнесу в значній мірі залежить від грамотного управління електронною комерцією. Ювелірна інтернет-платформа є складною системою, що включає взаємопов'язані бізнес-процеси.

Аналіз структури системи управління ювелірною онлайн-платформою виявив її поділ на п'ять основних функціональних блоків:

1) адміністрування: Цей блок відповідає за керування асортиментом товарів, їх наявністю на складі, розробку цінової стратегії, а також за маркетингові заходи та оптимізацію для пошукових систем;

2) взаємодія з клієнтами: Тут реалізовано процеси реєстрації та входу користувачів (з використанням безпечних JWT-токенів), управління їхніми особистими кабінетами, функціонал списків бажань та історії покупок, а також програму лояльності для особливих клієнтів;

3) управління контентом та пошуком: Цей блок забезпечує якісне представлення візуальної інформації про вироби, дозволяє здійснювати розширений пошук за характеристиками каменів та генерувати цифрові сертифікати;

4) фінанси та операції: Відповідає за обробку платежів, автоматичне коригування цін відповідно до ринкових коливань вартості металів, а також за ведення бухгалтерського обліку та формування фінансової звітності;

5) технічна інфраструктура (Headless-архітектура): Цей блок, побудований за принципом відокремлення фронтенду від бекенду, забезпечує роботу серверного API (на базі Django REST) та підтримує клієнтський інтерфейс (розроблений на React/Next.js). Як зазначає Salesforce, такий підхід значно прискорює випуск нових функцій та полегшує масштабування системи [4].

Аналіз функцій управління електронною комерцією показує складну систему взаємопов’язаних операцій. Блок, що працює з клієнтами, відповідає не лише за продажі, а й за створення довіри до бренду. Фінансово-операційна

складова забезпечує економічну стабільність платформи. Через високий середній чек ювелірного магазину цей блок потребує максимальної інтеграції з надійними платіжними системами та ефективними засобами боротьби з шахрайством. Інформаційно-технічний блок відповідає за підтримку всіх технічних процесів.

Огляд сучасних методів онлайн-продажів у ювелірній галузі виявляє певні недоліки традиційних систем:

1) застарілу монолітну структуру, яка уповільнює завантаження сторінок при складних фільтрах;

2) недостатню деталізацію атрибутів товарів, що обмежує можливості покупців у пошуку специфічних виробів (наприклад, фільтрація алмазів за рівнями чистоти VVS1/VVS2);

3) відсутність інтерактивного клієнтського досвіду, зокрема неможливість докладного огляду виробів.

Виходячи з аналізу існуючих проблем, для створення ефективного вебзастосунку для продажу ювелірних виробів, визначено наступні ключові вимоги:

1) оптимізована продуктивність: Застосувати Headless-архітектуру для забезпечення високої швидкості роботи, з чітким розмежуванням серверної частини (Python, Django REST) та інтерфейсу користувача;

2) гнучке управління товарами: Розробити надійну реляційну базу даних, яка зможе ефективно зберігати та обробляти складні дані про характеристики ювелірних виробів;

3) безпека на вищому рівні: Гарантувати найвищий ступінь захисту інформації та надійну систему автентифікації.

Цей проєкт є значним інженерним завданням, але його успішна реалізація принесе суттєві переваги: покращення клієнтського досвіду, зростання довіри покупців та позитивний вплив на прибутковість ювелірного бізнесу.

1.3 Аналіз застосунків-аналогів

Для обґрунтування технічних та функціональних вимог до розроблюваного програмного забезпечення було проведено аналіз трьох провідних міжнародних платформ у сфері онлайн-продажу ювелірних виробів. Кожна з обраних систем є лідером у своїй ніші та демонструє різні підходи до організації електронної комерції преміум-сегмента.

Інтернет-магазин «Pandora»

Pandora – це всесвітньо відомий бренд та глобальна електронно-комерційна платформа, яка дозволяє користувачам переглядати каталог, персоналізувати прикраси (наприклад, збирати браслети із шармами) та здійснювати покупки онлайн.

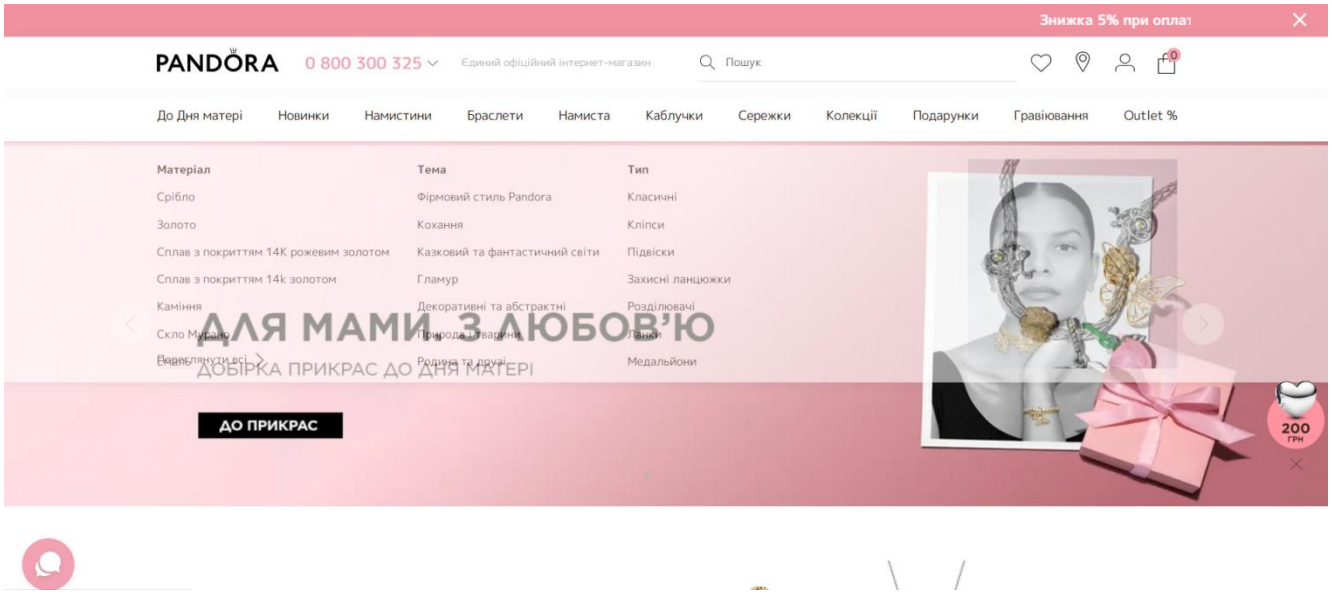

Рисунок 1.2 – Інтерфейс вебзастосунку «Pandora»

Застосунок вирізняється високим рівнем інтерактивності, естетичною візуалізацією товарів та деталізованою системою фільтрації. Платформа також пропонує функції створення списків бажань (wishlists), інтеграцію з програмами лояльності (Pandora Club) та безпечне оформлення замовлень із використанням сучасних протоколів шифрування платіжних даних [5].

Інтернет-магазин ювелірних виробів «Blue Nile»

Blue Nile – один із провідних світових онлайн-продавців діамантів та ювелірних прикрас, що забезпечує комплексний, безпечний та високо персоналізований процес купівлі преміальних товарів.

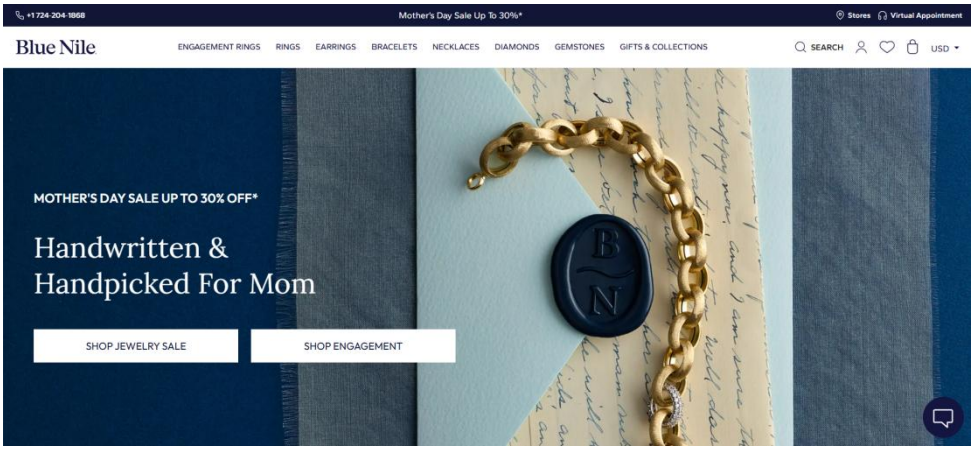

Рисунок 1.3 – Інтерфейс вебзастосунку Blue Nile

Важливою перевагою вебзастосунку Blue Nile є інноваційна функція «Створи свою каблучку» (Build Your Own Ring), що дозволяє покупцям самостійно комбінувати сертифіковані діаманти та оправи в режимі реального часу. Платформа також підтримує високоякісну 360-градусну візуалізацію каменів, інтеграцію з надійними міжнародними системами логістики та надає прямий доступ до сертифікатів якості від незалежних гемологічних лабораторій (наприклад, GIA) [6].

Інтернет-магазин ювелірних виробів «Brilliant Earth»

Brilliant Earth – це електронно-комерційна платформа, яка спеціалізується на продажі етичних ювелірних виробів.

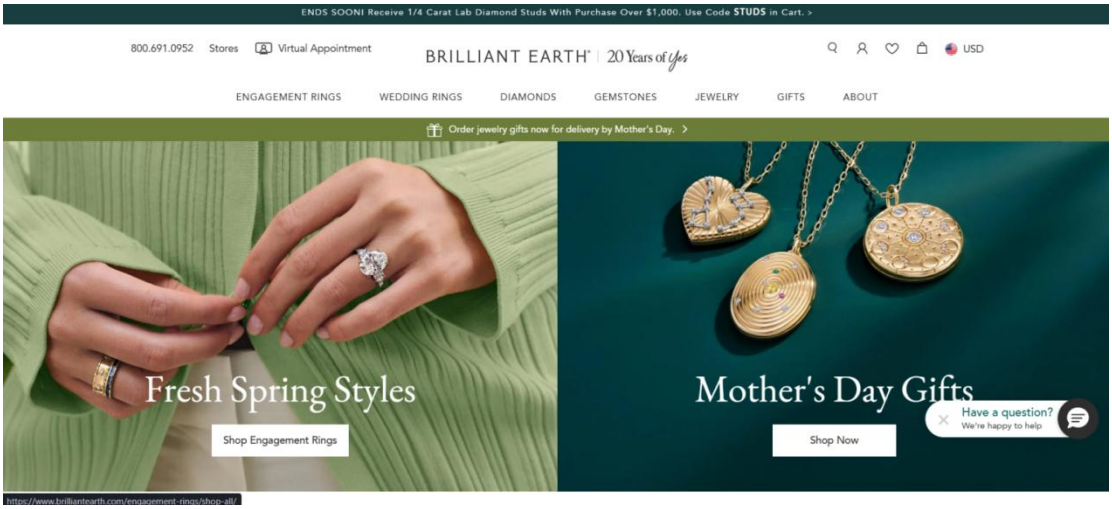

Рисунок 1.4 – Інтерфейс вебзастосунку Brilliant Earth

Вебзастосунок компанії вирізняється інноваційним підходом до прозорості ланцюга постачань, використовуючи технологію блокчейну для відстеження походження каменів. Завдяки інтеграції функцій доповненої реальності (AR) для віртуальної примірки та бездоганному клієнтському сервісу (наприклад, віртуальним відеоконсультаціям), платформа задає високі стандарти технологічності на ринку цифрового ювелірного ритейлу [7].

Таблиця 1.1 – Детальний аналіз застосунків-аналогів

Характеристика	**Pandora**	**Brilliant Earth**	**Blue Nile**
Розробник	Pandora A/S (Данія), глобальна корпорація	Brilliant Earth LLC (США)	Blue Nile Inc. (США), піонер галузі

Кінець таблиці 1.1

Архітектура	Ентерпрайз-моноліт на хмарній інфраструктурі з поступовим переходом до MACH (Microservices, API-first)	Хмарна мікросервісна архітектура з динамічною інтеграцією зовнішніх баз даних	Традиційна монолітна (Legacy System), гібридне розгортання
Стек технологій	Salesforce Commerce Cloud, JavaScript (React для нових модулів), інтеграція з SAP ERP	Ruby on Rails (backend), React.js (frontend), PostgreSQL, Amazon Web Services (AWS)	Oracle ATG Web Commerce, Java, масивні реляційні SQL-бази даних
Ключовий функціонал	Інтерактивний конструктор браслетів, AR-примірка (Virtual Try-On), розвинена омніканальна програма лояльності (My Pandora)	Деталізований пошук діамантів за критеріями 4C, конструктор «Create Your Own Ring», відстеження походження каменів (блокчейн)	Глобальна база діамантів із відео 360°, прямий доступ до цифрових сертифікатів GIA, вбудована освітня база
Переваги	Високий рівень візуальної персоналізації, глибокий емоційний сторітелінг, безшовна інтеграція онлайн-магазину з офлайн-бутиками	Неперевершена гнучкість кастомізації, величезна база каменів у реальному часі, високий рівень довіри завдяки етичному позиціонуванню	Максимальна деталізація інформації про кожен виріб, бездоганна надійність транзакцій, статус довіреного лідера індустрії
Недоліки	Обмежені можливості фільтрації за складними гемологічними властивостями; високе навантаження на браузер через важкий медіаконтент	Інтерфейс, перевантажений технічними даними, що може відлякувати новачків; періодичні затримки відгуку при складних запитах до API	Консервативний та застарілий UI-дизайн, недостатня оптимізація швидкодії на смартфонах, швидкодії на смартфонах, висока вартість впровадження нових функцій

Аналіз сучасного програмного забезпечення для реалізації ювелірних прикрас показує, що кожна система має свої переваги та недоліки.

Вибір відповідного програмного забезпечення залежить від конкретних вимог ювелірного магазину, його масштабів, фінансових можливостей та організаційних цілей.

Детальна оцінка функціональності систем, а також їхніх переваг і недоліків дозволяє вибрати найбільш підходяще рішення для ефективного управління ювелірним онлайн-бізнесом.

Висновки до розділу 1

Перший розділ присвячений глибокому дослідженню специфіки онлайн-продажу ювелірних виробів. Цей преміальний ринок потребує від програмних рішень не просто базових функцій для здійснення покупок, а й виняткової візуалізації товарів, ефективного управління детальними характеристиками виробів та найвищого рівня безпеки даних.

Аналіз провідних світових платформ, таких як Pandora, Brilliant Earth та Blue Nile, виявив, що, попри їхню функціональність, більшість побудовані на застарілих монолітних архітектурах. Це обмежує їхню швидкість, масштабованість та погіршує користувацький досвід, особливо при складних пошуках або завантаженні великих зображень.

Зважаючи на ці недоліки, визначено необхідність створення власного програмного продукту та його ключові функціональні та архітектурні вимоги. Для досягнення найкращих результатів пропонується сучасна розподілена Headless-архітектура, яка розділяє серверну логіку (Django REST Framework) та клієнтський інтерфейс (Next.js). Це дозволить подолати проблеми аналогів, забезпечити гнучкість, надійність та високоякісний цифровий досвід для майбутнього ювелірного бутика.

2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ РОБОТИ

2.1 Аналіз сучасного стану інструментарію на основі наукових досліджень

Вибір правильної архітектури та технологічного стеку є фундаментальним для успіху сучасних платформ електронної комерції, впливаючи на їхню швидкість, надійність, безпеку та здатність до масштабування. Щоб обґрунтувати вибір інструментарію для розробки ювелірного вебзастосунку, було проведено аналіз сучасних досліджень у галузі програмної інженерії.

Комплексне дослідження, опубліковане у 2024 році в авторитетному виданні IEEE Access (індексується в Scopus), оцінювало продуктивність REST-фреймворків та середовищ виконання. Воно показало, що розподілена (Headless) архітектура має суттєві переваги над традиційними монолітами при роботі з високо навантаженими вебзастосунками [8]. Дослідники продемонстрували, що чітке розмежування серверної бізнес-логіки та клієнтського інтерфейсу дозволяє ефективніше використовувати апаратні ресурси сервера.

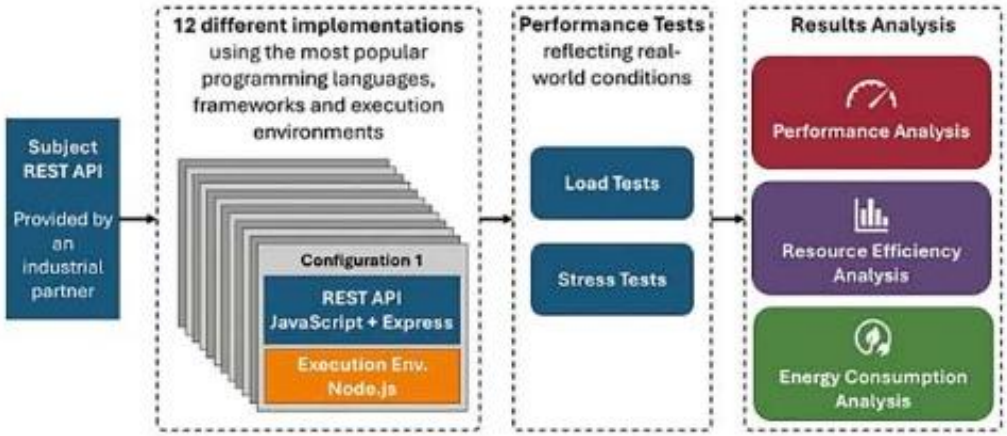

Рисунок 2.1 – Аналіз продуктивності REST API

Стаття також підтверджує, що використання екосистеми Python, зокрема Django REST Framework, для побудови API гарантує високий рівень безпеки транзакцій та стабільну обробку складних реляційних запитів. Водночас, делегування завдань з рендерингу інтерфейсу ізольованим фронтенд-фреймворкам (на базі React) значно зменшує навантаження на бекенд, підвищує

чуйність системи для кінцевого користувача та полегшує подальшу підтримку коду [8].

Спираючись на ці науково обґрунтовані дані, для розробки ювелірного інтернет-магазину було обрано стек технологій, що включає Django, Django REST Framework, Next.js (React), а також бази даних SQLite та PostgreSQL.

Клієнтська частина застосунку, що відповідає за взаємодію з покупцями, побудована на React. Це провідна JavaScript-бібліотека для створення компонентних інтерфейсів.

Основна перевага React полягає у використанні Virtual DOM, що оптимізує оновлення інтерфейсу, дозволяючи змінювати лише необхідні елементи (наприклад, динамічно перераховувати вартість у кошику) без повного рендерингу сторінки [9].

Для вирішення питань SEO-оптимізації, критично важливих для електронної комерції, застосовано фреймворк Next.js. Він реалізує серверний рендеринг (SSR), що дозволяє генерувати сторінки з каталогом ювелірних виробів на стороні сервера в момент запиту. Це забезпечує швидке завантаження, ефективне індексування пошуковими системами (зокрема, Google) та високу продуктивність на мобільних пристроях, що є пріоритетом для клієнтів преміум-сегмента [10].

У процесі розробки ювелірної платформи застосовується гібридна концепція керування базами даних. На етапі локальної розробки використовується SQLite. Це вбудована реляційна СКБД, що характеризується мінімальними вимогами до ресурсів та зберіганням даних в одному файлі [11].

Її переваги полягають у швидкості розгортання та тестування моделей, а також у спрощенні налаштування локального середовища без необхідності розгортання повноцінного сервера.

Для продакшн-середовища (реальної експлуатації) обрано PostgreSQL. Це потужна об'єктно-реляційна СКБД з відкритим вихідним кодом, вона є

оптимальним рішенням для обробки надзвичайно складної структури даних ювелірного магазину [12].

PostgreSQL забезпечує високий рівень транзакційної цілісності (властивості ACID), що є критично важливим для фінансових операцій. Додатковою перевагою є розширена підтримка формату JSONB, яка дозволяє ефективно зберігати специфічні, неструктуровані атрибути товарів (наприклад, динамічні характеристики діамантів), реалізуючи гібридний підхід до зберігання даних, що поєднує переваги реляційних та NoSQL баз даних [12].

Django – це передовий інструментарій для розробки веб-додатків на мові Python. Він створений для того, щоб прискорити процес створення складних та надійних систем, забезпечуючи при цьому високий рівень безпеки. Його архітектура побудована за принципом MVT (Model-View-Template), а філософія «все включено» означає, що багато необхідних функцій вже інтегровані [13].

Для ювелірного магазину Django виконує ключові завдання:

1) управління користувачами: вбудована система автентифікації дозволяє безпечно реєструвати та авторизувати користувачів;

2) зручне адміністрування: автоматично генерується панель адміністратора, яка значно спрощує роботу з каталогом товарів;

3) захист від загроз: Django забезпечує надійний захист від поширених веб-вразливостей, таких як SQL-ін'єкції, атаки CSRF та XSS.

Ефективна розробка та налагодження такої потужної серверної частини найчастіше відбувається у спеціалізованих середовищах, наприклад, PyCharm. Ці інструменти забезпечують тісну інтеграцію з віртуальними середовищами проєкту та надають зручні засоби для тестування.

Оскільки вебзастосунок використовує розподілену архітектуру, стандартного Django недостатньо. Серверу потрібно не просто формувати веб-сторінки, а й надавати чисті дані у форматі JSON.

Django REST Framework (DRF) – це потужна бібліотека, яка розширює можливості Django, дозволяючи створювати гнучкі та ефективні Web API [14]. У проєкті ювелірної онлайн-комерції DRF відповідає за:

1) перетворення даних: серіалізація (перетворення даних з бази даних у формат JSON) та десеріалізація (зворотне перетворення);

2) налаштування маршрутів: визначення шляхів для доступу до даних через API;

3) безпечна автентифікація: реалізація безпечного механізму автентифікації клієнтів за допомогою JWT-токенів (JSON Web Tokens). Це сучасний стандарт для односторінкових додатків (SPA), який забезпечує безпечний обмін даними без збереження стану сесії на сервері.

Таблиця 2.1 – Порівняльний аналіз технологій з альтернативами

Компонент системи	**Обрана технологія**	**Найближча альтернатива**	**Обґрунтування вибору для ювелірного магазину**
Бекенд (Серверна логіка)	Django + DRF (Python)	Express.js (Node.js)	Django має вбудовану адмін-панель «з коробки», що ідеально для керування каталогом, та надійнішу вбудовану систему безпеки транзакцій порівняно з мікро-фреймворком Express.
Фронтенд (Клієнтський інтерфейс)	Next.js (React)	Чистий React.js (Create React App)	Next.js забезпечує серверний рендеринг (SSR), що є критичною вимогою для SEO-оптимізації карток товарів, чого не може надати чистий React (CSR).
База даних (СУБД)	PostgreSQL	MongoDB (NoSQL)	Реляційна структура PostgreSQL (з підтримкою ACID) критично необхідна для безпеки фінансових операцій, тоді як MongoDB не гарантує суворої цілісності транзакцій замовлень.
Середовище розробки (База)	SQLite (локально) PostgreSQL (на сервері)	Повна розробка лише на PostgreSQL	SQLite дозволяє швидко прототипувати систему на етапі локальної розробки без розгортання важких Docker-контейнерів для БД.

2.2 Специфікація вимог до програмного забезпечення

1. Призначення та межі проєкту

1.1 Призначення системи

Вебзастосунок призначений для надання клієнтам комфортного та надійного доступу до каталогу ювелірних виробів, з можливістю здійснення покупок онлайн. Система забезпечує пошук прикрас, управління кошиком, оформлення замовлень, а також надає власникам бізнесу надійний інструмент для автоматизації продажів та управління асортиментом.

1.2 Погодження, ухвалені в програмній документації

- специфікація розроблена на основі вимог до сучасних платформ електронної комерції та стандартів безпеки передачі даних;
- ухвалено використання сучасних вебтехнологій (шаблону MVT) для забезпечення високої швидкодії та кросплатформної сумісності;
- визначено перелік основних функцій магазину, які будуть реалізовані до дедлайну.

1.3 Межі проєкту

- крайня дата завершення робіт: 15.06.2026 р.;
- проєкт охоплює розробку клієнтської частини (вітрини) та серверної інфраструктури (адмін-панелі та бази даних) інтернет-магазину;
- не включає розробку нативних мобільних застосунків (iOS, Android), але передбачає повністю адаптивний дизайн для мобільних пристроїв.

2. Загальний опис

2.1 Сфера застосування

Вебзастосунок використовується для роздрібної онлайн-торгівлі ювелірними виробами (B2C). Застосунок підтримує цілодобовий перегляд вітрини, вибір прикрас та безпечне оформлення замовлень з будь-якого комп'ютера чи смартфона.

2.2 Характеристики користувачів

– покупці (фізичні особи): клієнти, які використовують застосунок для пошуку прикрас, збереження обраного та оформлення замовлень;

– адміністратори/менеджери: персонал магазину, який має доступ до закритої частини сайту для управління товарами, цінами та статусами замовлень;

– рівень підготовки: для покупців достатньо базових навичок користування веббраузером, інтерфейс має бути максимально інтуїтивним.

2.3 Загальна структура і склад системи

– клієнтська частина: вебінтерфейс, побудований на основі фронтенд-фреймворку Next.js;

– серверна частина: побудована на мові Python із використанням фреймворків Django та Django REST Framework;

– рівень бази даних: реляційна СУБД PostgreSQL.

2.4 Загальні обмеження

– застосунок працює лише за наявності інтернет-з'єднання;

– коректне відображення візуального контенту залежить від пропускної здатності мережі клієнта (через наявність високодеталізованих фотографій прикрас).

3. Функції системи

3.1 Каталог та пошук прикрас

3.1.1 Опис функції

Відвідувачі можуть переглядати колекції, шукати вироби за типом (каблучки, сережки тощо) та користуватися системою фільтрації.

3.1.2 Вхідна і вихідна інформація

– вхідна: критерії пошуку або обрані фільтри (ціна, метал, наявність каміння);

– вихідна: оновлений список карток товарів, що відповідають запиту.

3.1.3 Функціональні вимоги

– багатокритеріальна фільтрація товарів;

– миттєве оновлення результатів пошуку;

– відображення детальної інформації про кожен виріб (проба, вага, розмір).

3.2 Робота з кошиком та оформлення покупки

3.2.1 Опис функції

Клієнт має змогу додавати обрані коштовності до кошика, редагувати їх кількість та покроково оформлювати замовлення.

3.2.2 Вхідна і вихідна інформація

– вхідна: перелік обраних товарів, контактні дані покупця, адреса та спосіб доставки;

– вихідна: загальна сума до оплати, збережене в системі замовлення зі статусом очікування.

3.2.3 Функціональні вимоги

– динамічний підрахунок загальної вартості;

– перевірка наявності товару перед фіналізацією замовлення (запобігання дублюванню покупок).

3.3 Реєстрація та особистий кабінет

3.3.1 Опис функції

Користувачі можуть створювати власні акаунти для персоналізації взаємодії та збереження історії покупок.

3.3.2 Вхідна і вихідна інформація

– вхідна: електронна пошта, ім'я, пароль;

– вихідна: підтверджений профіль, безпечна сесія користувача.

3.3.3 Функціональні вимоги

– безпечне хешування паролів;

– можливість перегляду статусів виконання замовлень;

– функціонал списку обраного (Wishlist) для збереження товарів на майбутнє.

3.4 Панель адміністратора

3.4.1 Опис функції

Закрита частина сайту для працівників магазину для комплексного управління електронною комерцією.

3.4.2 Вхідна і вихідна інформація

- вхідна: фотографії прикрас, текстові описи, ціни, налаштування статусів замовлень;
- вихідна: оновлений каталог на головному сайті, актуалізована база даних.

3.4.3 Функціональні вимоги

- зручний інтерфейс додавання/редагування/видалення товарів (CRUD);
- управління складськими залишками;
- обмежений доступ лише для авторизованого персоналу.

4. Вимоги до інформаційного забезпечення

4.1 Джерела і зміст вхідної інформації

- дані користувачів (ПІБ, контакти, адреси) вводяться клієнтами під час оформлення замовлення або реєстрації;
- інформація про ювелірні вироби (описи, ціни, фото) завантажується адміністратором.

4.2 Нормативно-довідкова інформація

- довідник категорій товарів (каблучки, кольє, браслети);
- довідник матеріалів (золото, срібло, платина) та вставок (діаманти, фіаніти).

4.3 Вимоги до способів організації, збереження та ведення інформації

- уся інформація зберігається в реляційній базі даних (PostgreSQL);
- таблиці в базі тісно пов'язані між собою (замовлення жорстко прив'язане до клієнта та конкретних артикулів);
- обов'язкове приховування паролів (шифрування).

5. Вимоги до технічного забезпечення

– сервери: наявність вебсервера для безперебійної роботи Django-застосунку;

– клієнтські пристрої: стандартні ПК, ноутбуки, планшети або смартфони з підключенням до мережі Інтернет.

6. Вимоги до програмного забезпечення

6.1 Архітектура програмної системи

– монолітна клієнт-серверна система, побудована за шаблоном MTV (Model-Template-View);

– можливість обміну даними через REST API для часткового оновлення інтерфейсу.

6.2 Системне програмне забезпечення

– ОС сервера: будь-яка ОС, що підтримує середовище Python (найчастіше Linux).

6.3 Мережне програмне забезпечення

– протокол HTTPS із сертифікатом SSL/TLS для безпеки передачі даних.

6.4 Програмне забезпечення ведення інформаційної бази

– СКБД: PostgreSQL.

6.5 Мова і технологія розробки

– мови: Python, SQL, HTML, CSS, JavaScript;

– бекенд-фреймворки: Django, Django REST Framework;

– фронтенд-фреймворк: Next.js.

7. Вимоги до зовнішніх інтерфейсів

7.1 Інтерфейс користувача

– сучасний, привабливий та адаптивний дизайн (Responsive Web Design);

– сайт має однаково коректно та зручно відображатися на великому екрані монітора та на мобільному телефоні.

7.2 Апаратний інтерфейс

– не потребує спеціалізованого обладнання.

7.3 Програмний інтерфейс

– використання Django REST API для комунікації між клієнтською та серверною частинами під час фільтрації товарів.

7.4 Комунікаційний протокол

– HTTPS для всіх запитів.

8. Властивості програмного забезпечення

8.1 Доступність

– магазин має безперешкодно відкриватися у всіх сучасних браузерах (Chrome, Safari, Firefox тощо);

– цілодобовий доступ для здійснення покупок.

8.2 Супроводжуваність

– модульна архітектура коду (розділення на додатки Django) повинна дозволяти легко додавати нові функції.

8.3 Переносимість

– клієнтам не потрібно завантажувати жодного ПЗ на свій пристрій, система працює через веббраузер незалежно від ОС.

8.4 Продуктивність

– сторінки з високоякісними фотографіями прикрас мають завантажуватися максимально швидко (шляхом оптимізації та кешування).

8.5 Надійність

– гарантований захист від помилок під час покупок (система запобігає замовленню товару, якого вже немає в наявності).

8.6 Безпека

– обов'язкове хешування паролів клієнтів;

– захист від хакерських втручань (вбудований захист Django від SQL-ін'єкцій, XSS, CSRF);

– використання захищеного з'єднання для конфіденційності платіжних даних.

Висновки до розділу 2

У другому розділі детально розглянуто, обрано та обґрунтовано програмні інструменти та методи для розробки ювелірного вебзастосунку. Дослідивши актуальні тренди веброзробки, сформовано висновок про доцільність застосування розподіленої Headless-архітектури.

Вибір Django REST Framework для бекенду, Next.js для фронтенду та бази даних PostgreSQL гарантує високу продуктивність, безпеку фінансових операцій та відмінну SEO-оптимізацію каталогу.

Підсумком розділу стало формування детальної специфікації вимог до створюваного програмного забезпечення. У цьому документі було чітко зафіксовано такі пункти:

- головну мету створення та рамки проєкту;
- загальну характеристику онлайн-магазину;
- основні можливості системи (пошук прикрас, робота з кошиком, панель керування);
- критерії до збереження інформації та бази даних;
- вимоги до серверного обладнання та інфраструктури;
- перелік необхідного програмного інструментарію;
- вимоги до зовнішнього вигляду сайту (адаптивного інтерфейсу);
- базові властивості готового продукту (швидкодія, безпека, кросплатформність).

3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ ПРОДАЖУ ЮВЕЛІРНИХ ВИРОБІВ

3.1 Сценарії та діаграми використання системи

Сценарій використання системи (Use Case) – це опис того, як користувач взаємодіє з програмним продуктом, крок за кроком, щоб реалізувати певну функціональність та досягти поставленої задачі.

Use Case 1: «Реєстрація клієнта на сайті»

Область застосування (Scope): Вебзастосунок для продажу ювелірних виробів.

Рівень (Level): User-goal (орієнтований на мету користувача).

Основний актор (Primary Actor): Гість або незареєстрований користувач.

Зацікавлені сторони та їхні інтереси (Stakeholders and Interests):

1) клієнт, який зацікавлений у створенні особистого кабінету для зручних покупок;

2) бізнес компанія, що зацікавлена в отриманні електронної пошти клієнта для підтримки зв'язку.

Передумови (Preconditions): Користувач має доступ до вебсайту та володіє існуючою електронною поштою.

Гарантія успішного виконання (Success Guarantee): Обліковий запис створено, а користувач, тим часом, успішно авторизувався в системі.

Основний сценарій успіху (Main Success Scenario):

1) користувач натискає кнопку «Реєстрація»;

2) користувач вводить свою електронну пошту та придумує пароль;

3) система зберігає дані користувача в базі даних;

4) система автоматично входить у створений профіль користувача.

Розширення (Extensions): На етапі введення даних, якщо такий email вже зареєстрований, система видає помилку «Користувач з таким email вже існує».

Спеціальні вимоги (Special Requirements): Пароль користувача має містити не менше 8 символів.

Технологічні та інформаційні варіації (Technology and Data Variations): Паролі зберігаються у базі даних у хешованому (зашифрованному) вигляді.

Частота використання (Frequency of Occurrence): Близько 10% дій усіх нових відвідувачів сайту.

Додаткові положення (Miscellaneous): Після успішної реєстрації на пошту клієнта автоматично надсилається вітальний лист.

Use Case 2: «Пошук прикраси за назвою»

Область застосування (Scope): Вебзастосунок для продажу ювелірних виробів.

Рівень (Level): User-goal (орієнтований на мету користувача).

Основний актор (Primary Actor): Користувач.

Зацікавлені сторони та їхні інтереси (Stakeholders and Interests): Користувач, який зацікавлений швидко знайти потрібний товар у каталозі.

Передумови (Preconditions): У каталозі магазину є хоча б один доданий товар.

Гарантія успішного виконання (Success Guarantee): Користувач бачить на екрані список знайдених прикрас.

Основний сценарій успіху (Main Success Scenario):

1) користувач натискає на рядок пошуку у верхній частині сайту;
2) користувач вводить слово (наприклад, «каблучка»);
3) система шукає збіги у базі даних товарів;
4) система показує результати пошуку на екрані.

Розширення (Extensions): Якщо за введеним словом нічого не знайдено, система виводить напис «Шуканих товарів не знайдено».

Спеціальні вимоги (Special Requirements): Пошук повинен працювати незалежно від того, в якому регістрі введено слово.

Технологічні та інформаційні варіації (Technology and Data Variations): Використовується пошук по базі даних за допомогою SQL-запиту.

Частота використання (Frequency of Occurrence): Близько 80% усіх користувачів використовують пошук.

Додаткові положення (Miscellaneous): Відсутні.

Use Case 3: «Перегляд деталей прикраси»

Область застосування (Scope): Вебзастосунок для продажу ювелірних виробів.

Рівень (Level): User-goal (орієнтований на мету користувача).

Основний актор (Primary Actor): Користувач.

Зацікавлені сторони та їхні інтереси (Stakeholders and Interests): Користувач зацікавлений побачити якісні фотографії ювелірних виробів, опис та ціну прикраси.

Передумови (Preconditions): Користувач знаходиться у загальному каталозі сайту або на сторінці результатів пошуку.

Гарантія успішного виконання (Success Guarantee): На екрані відкрита сторінка з детальним описом конкретного товару.

Основний сценарій успіху (Main Success Scenario):

1) користувач натискає на фотографію прикраси в каталозі;
2) система завантажує сторінку обраного товару;
3) користувач переглядає фотографії, читає характеристики та бачить ціну;

Розширення (Extensions): Відсутні.

Спеціальні вимоги (Special Requirements): Фотографії прикрас мають бути високої якості для детального огляду.

Технологічні та інформаційні варіації (Technology and Data Variations): Уся інформація про виріб завантажується з таблиці товарів (Products).

Частота використання (Frequency of Occurrence): Дуже часто (основна дія при виборі товару).

Додаткові положення (Miscellaneous): На цій сторінці обов'язково розміщена кнопка «Додати до кошика».

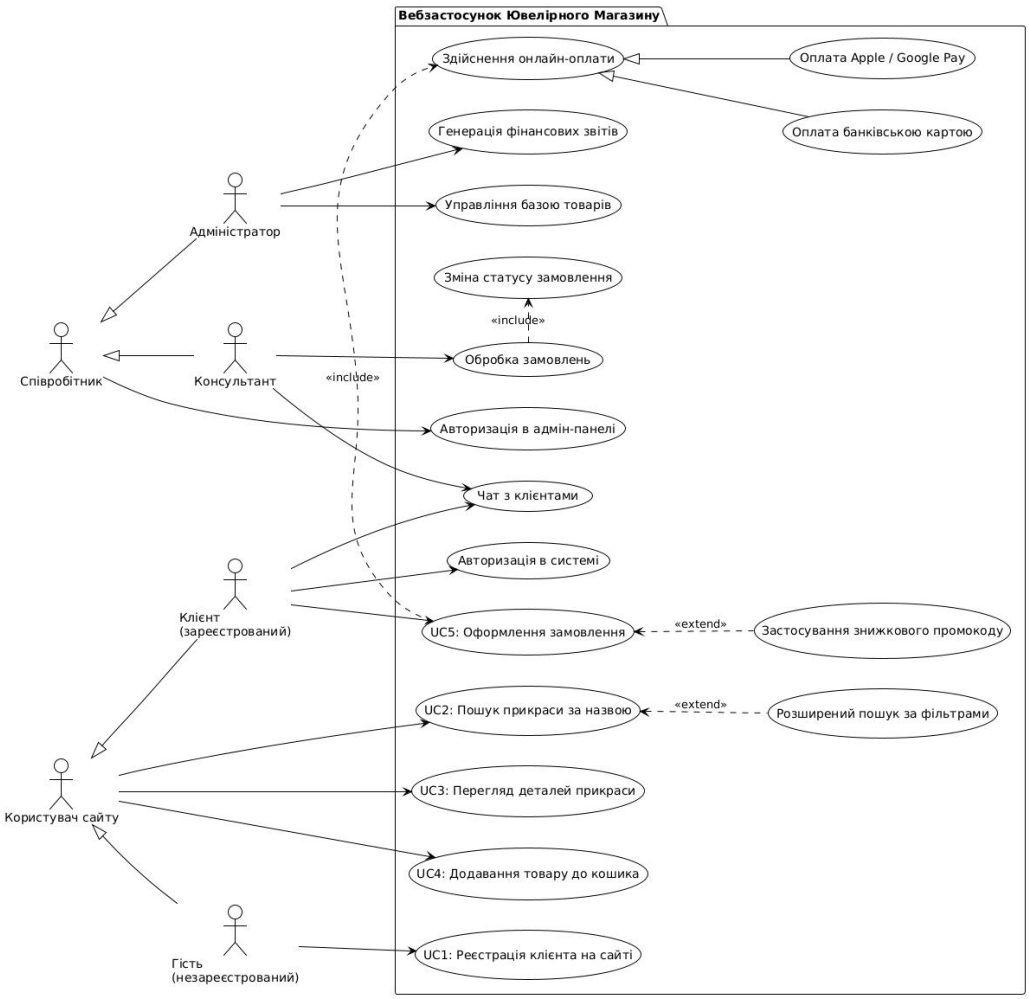

Рисунок 3.1 – Діаграма сценаріїв використання системи

Дана діаграма (рис 3.1) включає в собі таких акторів системи як: адміністратор, співробітник, консультант, клієнт (зареєстрований), користувач сайту, гість (незареєстрований). Також наявні всі можливі види зв’язків та відношень між акторами та їхніми діями:

1) спадкування та узагальнення сценаріїв (Generalization);
2) розширення (Extend);
3) включення (Include).

3.2 Створення діаграми класів

Діаграма класів (рис 3.2) демонструє архітектуру вебзастосунку для продажу ювелірних виробів. Сервіс спроєктований за принципом трьохрівневої архітектури: база даних, API та клієнтська частина, які забезпечать гнучкість, швидкість і безпеку системи.

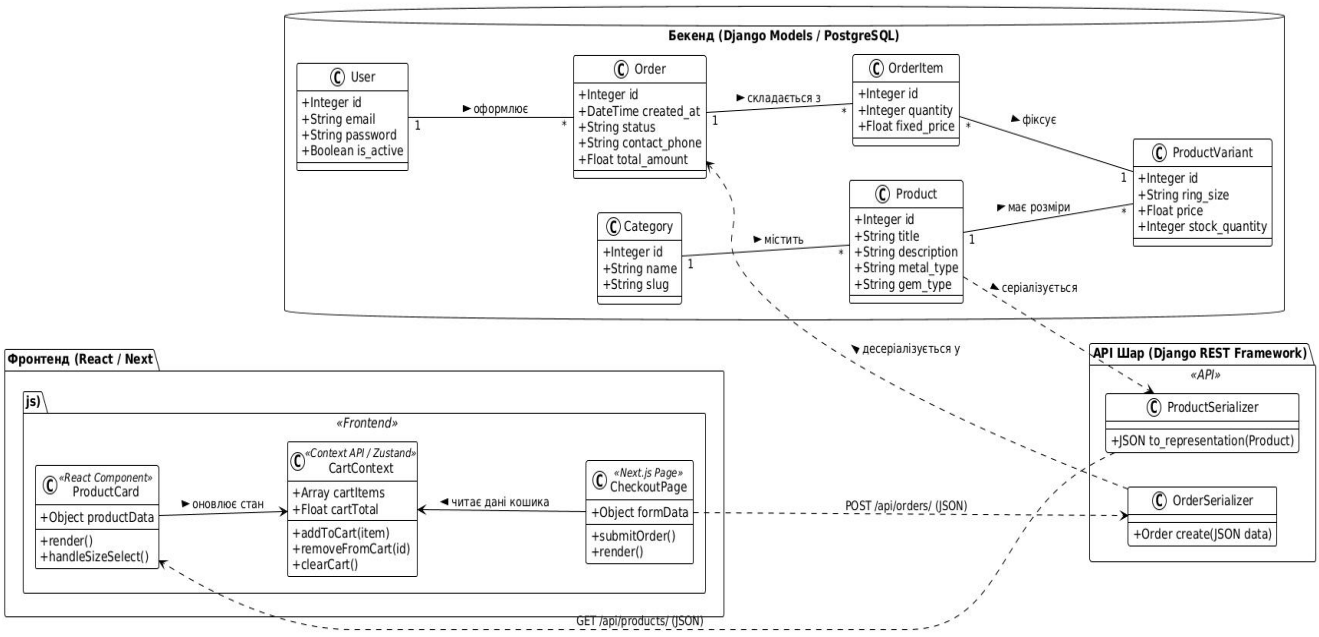

Рисунок 3.2 – Діаграма класів

Серверна частина, розроблена на Django, працює з базою даних PostgreSQL. В основі лежить каталог, де модель Product містить загальний опис товару, а ProductVariant – його конкретні варіації (розмір, вага, залишки). Замовлення обробляються через моделі Order та OrderItem, причому в OrderItem фіксується ціна на момент покупки для забезпечення фінансової точності.

API, створений на Django REST Framework, слугує посередником. Серіалізатори, як-от ProductSerializer, перетворюють дані з бази в JSON для фронтенду. OrderSerializer виконує зворотну функцію: отримує дані від клієнта, валідує їх і створює записи в базі.

React та Next.js відповідають за користувацький інтерфейс. Компонент ProductCard відображає товари, а CartContext керує станом віртуального кошика.

Процес покупки завершується на сторінці CheckoutPage, яка збирає дані та надсилає фінальний запит на сервер для оформлення замовлення.

3.3 Створення діаграм взаємодії

Діаграма взаємодії (рис. 3.3) демонструє механізм обробки запиту користувача щодо відображення певного ювелірного виробу (каблучки чи підвіски). Ключовим аспектом є забезпечення доступу до даних виключно через систему, без надання клієнту прямого доступу до бази даних.

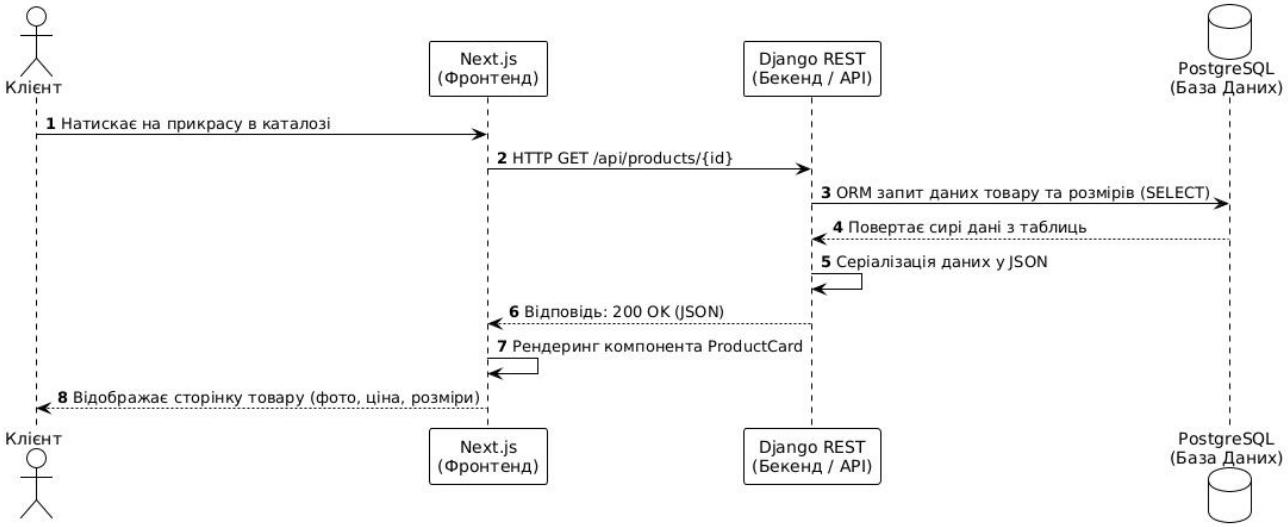

Рисунок 3.3 – Діаграма взаємодії: Перегляд деталей прикраси

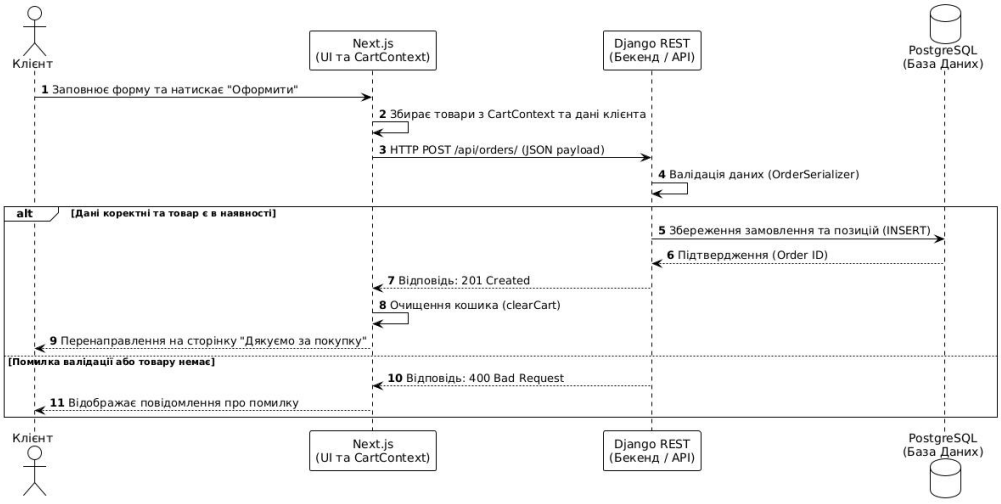

Рисунок 3.4 – Діаграма взаємодії: Оформлення замовлення

Наступний сценарій (рис 3.4) ілюструє транзакцію купівлі, в процесі якої інформація надсилається від користувача до сервера, що супроводжується обов'язковою валідацією безпеки на серверній стороні.

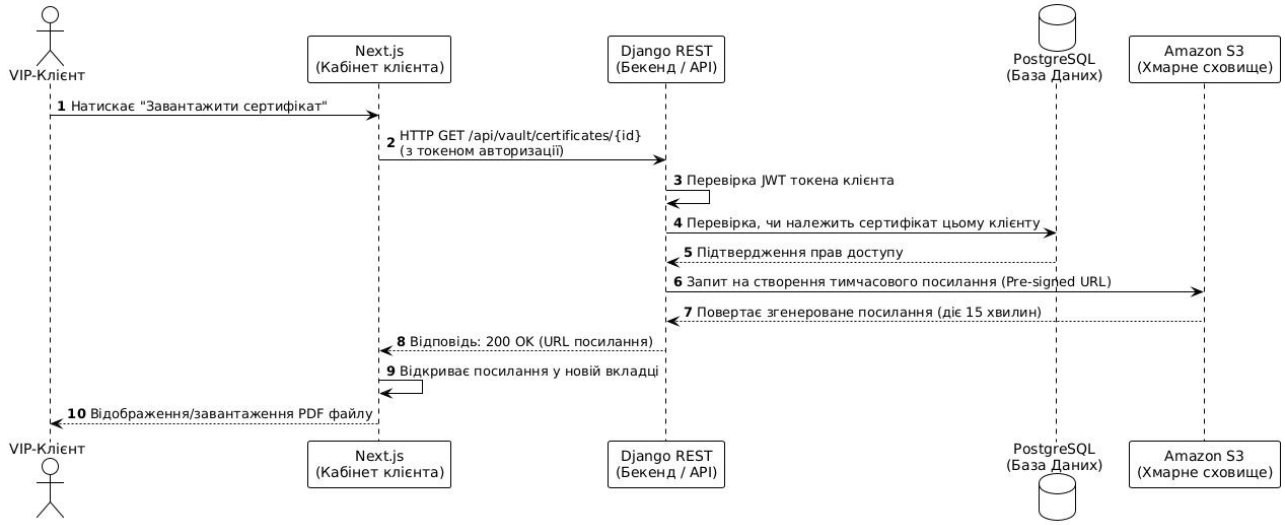

Рисунок 3.5 – Діаграма взаємодії: Отримання сертифікату

Діаграма (рис. 3.5) ілюструє механізм захисту конфіденційних даних системою. Сертифікати не зберігаються у загальнодоступному вигляді; натомість, серверна частина щоразу створює унікальне, тимчасове посилання для кожного окремого клієнта.

3.4 Створення ER-діаграм для бази даних застосунку

Спроєктована за допомогою ER-діаграми база даних (рис. 3.6) побудована за принципом реляційної моделі PostgreSQL.

Її архітектура характеризується високим ступенем нормалізації та забезпечує надійний контроль цілісності фінансових та конфіденційних даних.

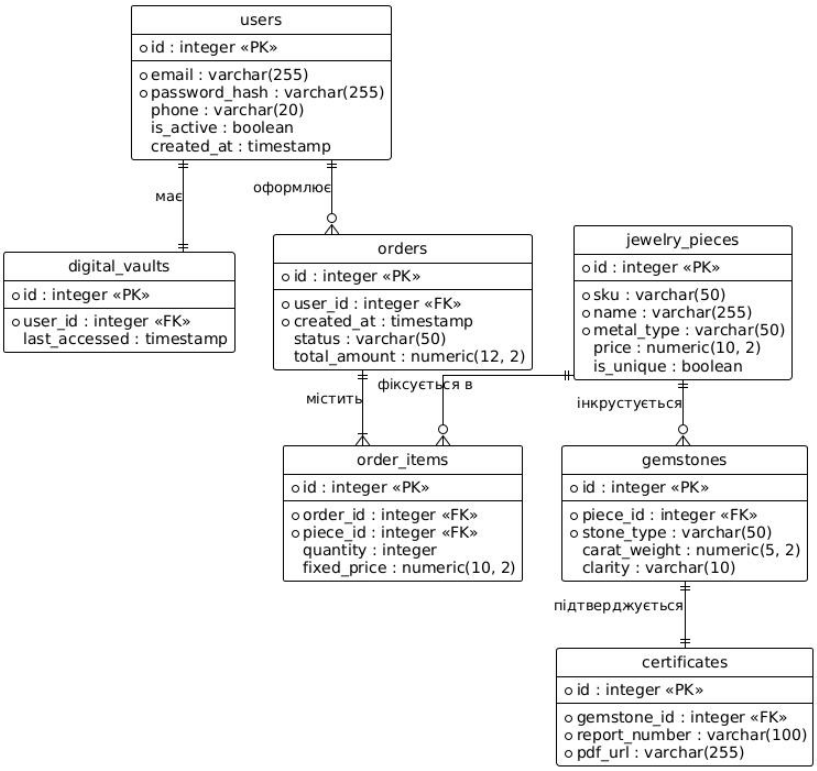

Рисунок 3.6 – ER-діаграма бази даних системи

Основні зв'язки в базі даних представлені двома типами: зв'язок «один-до-одного» (1:1)**,** який встановлює ексклюзивні відносини між сутностями (наприклад, один VIP-клієнт асоціюється з одним «Цифровим сейфом», а один дорогоцінний камінь – з одним унікальним PDF-сертифікатом), та зв'язок «один-до-багатьох» (1:N)**,** що використовується для моделювання стандартних бізнес-процесів (один користувач може мати багато замовлень, одне замовлення може містити кілька позицій, а одна прикраса може бути інкрустована кількома різними каменями).

Для забезпечення фінансової стійкості та незмінності вартості угоди, ціна товару зберігається як у таблиці товарів, так і дублюється в полі fixed_price

таблиці order_items. Це гарантує, що вартість замовлення залишається незмінною, незалежно від подальших оновлень цін у каталозі.

3.5 Створення діаграми розгортання застосунку

Фізична архітектура ювелірного вебзастосунку (рис. 3.7) побудована на принципах мікросервісної архітектури та розподілених систем.

Система не розміщена на єдиному сервері, а навпаки – логічно та фізично розподілена між незалежними компонентами. Такий підхід забезпечує високу відмовостійкість, надійний захист фінансових транзакцій та блискавичне завантаження візуального контенту.

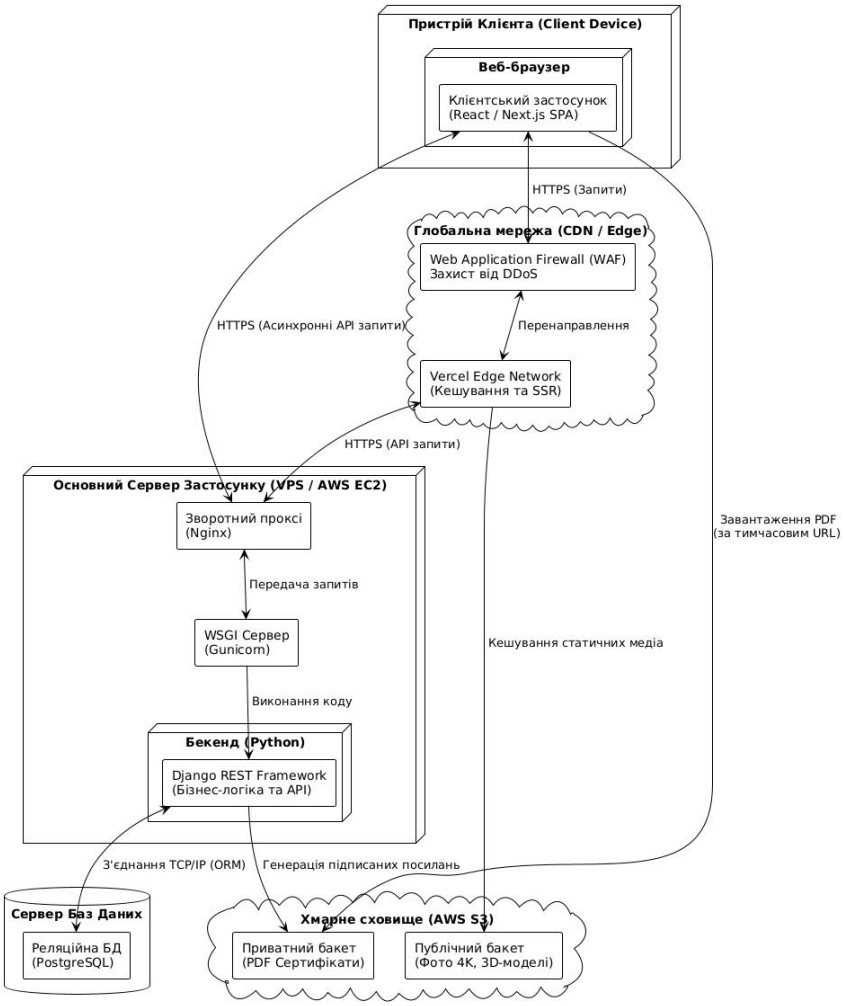

Рисунок 3.7 – Діаграма розгортання вебзастосунку

Взаємодія користувача з системою починається в браузері, в який завантажується сучасний клієнтський застосунок – Single Page Application, розроблений на React/Next.js.

Для забезпечення максимально можливої швидкості завантаження по всьому світу фронтенд розгорнуто на глобальній мережі доставки контенту (Vercel Edge Network або аналогічній). Кожен запит перед тим, як потрапити на основні сервери, проходить через Web Application Firewall. Цей захисний екран фільтрує шкідливий трафік, блокує спроби DDoS-атак і є обов’язковим елементом безпеки для преміум-платформ електронної комерції.

Вся бізнес-логіка, фінансові операції та механізми безпеки винесені на окремий виділений сервер. Вхідною точкою на сервер є веб-сервер Nginx, який виконує роль зворотного проксі: він приймає запити, проводить первинну обробку та передає їх на WSGI-сервер Gunicorn.

Саме тут працює основне ядро системи – Django REST Framework. Серверна частина не відповідає за генерацію HTML-сторінок, а функціонує виключно як API. Вона приймає запити у форматі JSON від фронтенду та повертає структуровані дані, що дозволяє фронтенду самостійно формувати інтерфейс користувача.

Реляційна база даних PostgreSQL: Тут зберігається вся текстова інформація, включаючи профілі користувачів, деталі замовлень та фінансові записи. PostgreSQL розміщена на окремому, високозахищеному сервері, що гарантує цілісність та конфіденційність цих даних.

Хмарне об'єктне сховище (Amazon S3): Мультимедійний контент, що потребує швидкого доступу, винесено в хмарне сховище. Воно поділено на дві логічні зони:

1) публічний бакет: Призначений для швидкої віддачі візуального контенту високої якості, такого як 4K-фотографії ювелірних виробів та 3D-моделі;

2) приватний бакет: Містить конфіденційні документи, наприклад, PDF-сертифікати. Доступ до цього бакету суворо контролюється. Користувачі можуть

отримати доступ до файлів лише через тимчасові підписані посилання, які генеруються сервером Django після успішної перевірки прав доступу клієнта.

3.6 Створення діаграми діяльності вебзастосунку

Представлена діаграма діяльності (рис. 3.8) візуалізує один з основних бізнес-процесів, що реалізується в ювелірному вебзастосунку.

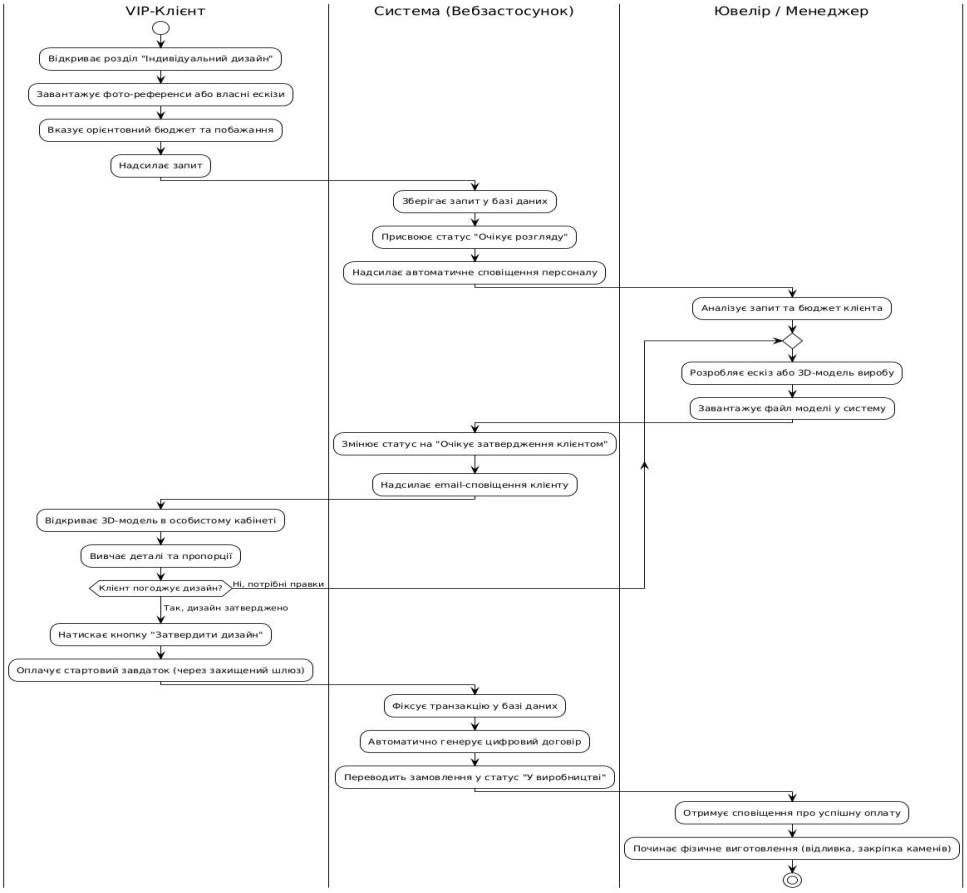

Рисунок 3.8 – Діаграма діяльності: Оформлення індивідуального замовлення

Цей процес стосується створення унікальних прикрас на замовлення клієнта. Для чіткого розмежування функціональних обов'язків, відображаються наступні етапи: взаємодія з клієнтом, автоматизовані системні операції та безпосередню роботу ювеліра (або персонального менеджера).

3.7 Створення дизайн-макетів для застосунку

Макет сайту є візуальним прототипом, що деталізує розташування контенту та інтерактивних елементів (текст, кнопки, медіа) на вебсторінці. Він слугує попереднім кресленням, що дозволяє оцінити структуру та візуальну ієрархію до етапу кодування.

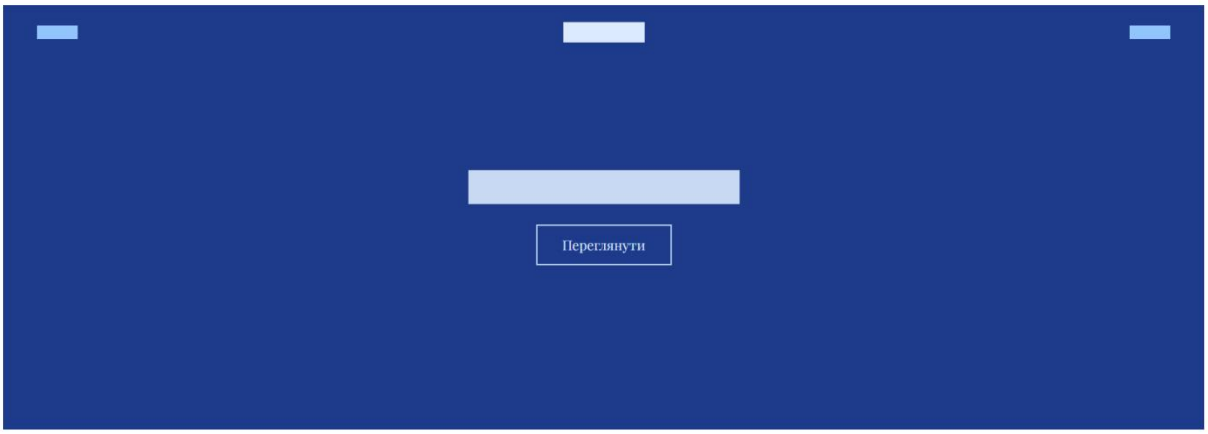

Рисунок 3.9 – Макет інтерфейсу головної сторінки

Рисунок 3.10 – Макет сторінки ексклюзивного ювелірного виробу

Сторінка ексклюзивного вибору (рис. 3.10) відмовляється від стандартного підходу інтернет-магазинів на користь презентації цінної прикраси. Замість звичайної галереї спроектована зона для макро-зуму або 3D-огляду, фокус на геммологічних характеристиках (сертифікат GIA) та кнопка бронювання.

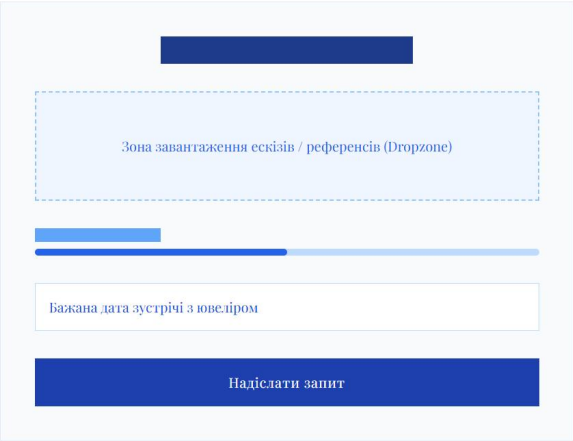

Рисунок 3.11 – Макет форми індивідуального замовлення

Інтерактивна форма для клієнтів (рис. 3.11), які бажають створити унікальну прикрасу. Містить спеціальну зону для завантаження власних ескізів чи референсів (Dropzone), повзунок орієнтовного бюджету та поле для погодження дати індивідуальної консультації з ювеліром.

Висновки до розділу 3

У третьому розділі проведено моделювання та проєктування архітектури вебзастосунку для продажу ювелірних виробів.

За допомогою інструментарію UML детально змодельовано логіку роботи та поведінку системи. Діаграми використання допомогли чітко розмежувати права доступу для різних типів користувачів (від гостя до VIP-клієнта та персоналу).

Водночас побудовані діаграми взаємодії продемонстрували, як саме система обробляє критично важливі запити – від швидкого завантаження каталогу до проведення захищених фінансових транзакцій із перевіркою даних на сервері.

Отже, спроєктована система має чіткий розподіл відповідальності між компонентами, що створює надійний фундамент і дозволяє впевнено переходити до наступного етапу – безпосередньо написання та тестування програмного коду.

4 КОДУВАННЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

4.1 Організація структури програмного проєкту

Програмний код вебзастосунку містить чітке логічне та фізичне розділення на клієнтську (frontend) та серверну (backend) частини.

Призначення ключових файлів та директорій серверної частини:

1) manage.py – це головний сценарій управління проєктом. Використовується для запуску локального сервера, виконання міграцій бази даних та створення нових застосунків;

2) requirements.txt – це файл із переліком усіх зовнішніх Python-бібліотек та їх версій, необхідних для розгортання середовища;

3) core/ – головна директорія налаштувань проєкту:

- settings.py – конфігураційний файл, що містить підключення до бази даних PostgreSQL, налаштування безпеки (CORS, JWT) та реєстрацію додатків;
- urls.py – головний маршрутизатор бекенду, який перенаправляє вхідні HTTP-запити до відповідних додатків;
- wsgi.py – точка входу для WSGI-сумісних вебсерверів (наприклад, Gunicorn) при розгортанні проєкту на реальному сервері (Production).

4) products/ (приклад типового застосунку, такі як users, orders):

- models.py – містить класи (ORM-моделі), які описують структуру таблиць бази даних для ювелірних виробів та категорій;
- serializers.py – містить класи для перетворення складних об'єктів бази даних у формат JSON (і навпаки);
- views.py – містить контролери (класи або функції), які реалізують бізнес-логіку обробки запитів (наприклад, фільтрацію каталогу);
- admin.py – файл конфігурації вбудованої панелі адміністратора для зручного управління товарами.;
- migrations/ – папка, що автоматично генерується і містить історію змін структури бази даних.

Призначення ключових файлів та директорій клієнтської частини:

1) package.json – частина проєкту, що має метадані застосунку, сценарії для запуску, збірки та список усіх залежностей (React, Next.js, Axios);

2) next.config.js – файл конфігурації мета-фреймворку Next.js, де налаштовуються параметри серверного рендерингу, змінні оточення та правила оптимізації зображень;

3) public/ – директорія для зберігання статичних ресурсів (логотипів, шрифтів, іконок), які доступні браузеру напряму без обробки Webpack;

4) app/ – головна директорія маршрутизації (App Router):

- layout.tsx – кореневий шаблон застосунку. Містить спільні елементи інтерфейсу, які відображаються на всіх сторінках (наприклад, шапка сайту Navbar та підвал Footer);
- page.tsx – головна сторінка інтернет-магазину (Landing page);
- globals.css – файл глобальних стилів;
- catalog/page.tsx – сторінка каталогу ювелірних виробів. Шлях до папки автоматично створює маршрут /catalog;
- cart/page.tsx – сторінка кошика покупця (маршрут /cart).

5) components/ – директорія для UI-компонентів, які повторно використовуються у проєкті:

- ProductCard.tsx – компонент картки одного ювелірного виробу (фото, назва, ціна, кнопка «Купити»);
- Navbar.tsx – компонент верхнього навігаційного меню.

6) services/ – директорія для допоміжних сервісів та інтеграцій:

- api.ts – файл конфігурації HTTP-клієнта, який містить базову адресу бекенду та логіку додавання JWT-токенів до заголовків запитів.

4.2 Розробка frontend-частини застосунку

Клієнтська частина вебзастосунку побудована на базі React та фреймворку Next.js. Головне завдання на цьому етапі полягало у створенні продуктивного, адаптивного та зрозумілого інтерфейсу, який відповідає найвищим стандартам електронної комерції.

Застосовано компонентний підхід для розробки візуальної частини. Це означає, що весь користувацький інтерфейс розбито на незалежні модулі. Такий підхід дозволив:

1) уникнути дублювання коду, роблячи його більш компактним та ефективним;

2) спростити подальшу підтримку та розширення функціоналу.

Серед ключових розроблених компонентів:

1) ProductCard (Картка товару): цей компонент відповідає за відображення кожного ювелірного виробу. Він містить оптимізоване зображення, назву, актуальну ціну та зручну кнопку для додавання товару до кошика;

2) FilterSidebar (Панель фільтрації): це складний інтерактивний елемент, який дозволяє користувачам задавати бажані параметри пошуку (наприклад, матеріал, ціновий діапазон, категорію). На основі цих даних формується запит до сервера;

3) Navbar (Навігаційна панель): цей глобальний компонент забезпечує легкий доступ до всіх основних розділів магазину та відображає поточний стан кошика користувача.

Для локального управління даними всередині компонентів використано стандартні хуки React:

1) useState: цей хук допомагає керувати динамічними елементами інтерфейсу, такими як відкриття/закриття вікон, збереження даних з форм авторизації та тимчасове зберігання обраних фільтрів;

2) useEffect: цей хук використовується для обробки побічних ефектів, зокрема для синхронізації даних профілю користувача з локальним сховищем браузера.

Навігація по вебзастосунку реалізована за допомогою Next.js App Router. Для максимальної SEO-оптимізації карток товарів застосовано серверний рендеринг (SSR – Server-Side Rendering).

Коли користувач переходить на сторінку каталогу, сервер Next.js попередньо звертається до API бекенду, отримує дані про товари, формує готову HTML-сторінку з цим контентом і відправляє її браузеру. Це забезпечує:

1) кращу індексацію пошуковими системами;

2) кначно швидший час першого відображення контенту, особливо на мобільних пристроях.

Взаємодія фронтенду з бекендом (розробленим на Django REST Framework) винесена в окремий сервісний шар. Налаштовано клієнт для виконання асинхронних HTTP-запитів, який обробляє обмін даними у форматі JSON.

При кожному запиті до захищених розділів (наприклад, оформлення замовлення або перегляд історії покупок) клієнтський застосунок автоматично отримує JWT-токен авторизації з пам'яті та додає його до заголовка Authorization: Bearer. Якщо термін дії токена закінчується (що призводить до помилки 401 Unauthorized), фронтенд автоматично запускає процес оновлення токена (Refresh Token), не перериваючи роботу користувача. Це забезпечує безперебійний доступ до функцій, що вимагають автентифікації, та підтримує високий рівень безпеки.

4.3 Фізична реалізація бази даних та керування даними застосунку

Система зберігання даних для вебзастосунку використовує гнучкий та гібридний підхід. Це означає, що застососвано різні системи керування базами даних (СУБД) залежно від того, на якому етапі знаходиться розробка програмного продукту.

Такий метод є загальноприйнятою практикою в сучасній розробці вебзастосунків, оскільки він дозволяє оптимізувати процес написання коду та забезпечує стабільну роботу системи в реальних умовах:

1) локальна розробка та прототипування: на початкових етапах розробки та створення прототипів використано реляційну базу даних SQLite. Її перевага в тому, що вона не потребує встановлення окремого сервера, а всі дані зберігаються в одному файлі на локальному комп'ютері (db.sqlite3). Це значно прискорює тестування ідей, створення нових структур даних та внесення змін до бази даних, не витрачаючи час на налаштування інфраструктури;

2) продакшн-середовище: для запуску системи в реальному робочому середовищі (production) підключено до більш потужної бази даних PostgreSQL. Вибір PostgreSQL пояснюється високими вимогами інтернет-магазину до надійності та цілісності транзакцій. PostgreSQL використовує сучасний механізм паралельного доступу, який дозволяє одночасно обробляти велику кількість запитів на оформлення замовлень, не викликаючи блокування бази даних. Перемикання між цими двома базами даних відбувається автоматично на рівні конфігурації фреймворку Django, керованої через змінні оточення.

Взаємодію серверної частини застосунку з базою даних від прямого написання SQL-запитів абстраговано, використовуючи технологію ORM (Object-Relational Mapping), яка інтегрована у фреймворк Django. Раніше розроблена логічна модель даних (рис. 3.9) реалізується у вигляді класів мови Python. Кожен клас (модель) відповідає за певну таблицю в базі даних, а його атрибути – за стовпці цієї таблиці з відповідними типами даних.

Використання ORM забезпечує два ключові переваги:

1) безпека: усі запити до бази даних автоматично обробляються таким чином, щоб унеможливити атаки типу SQL-ін'єкцій. Це робить нашу систему максимально захищеною;

2) керування версіями бази даних: будь-які зміни в структурі таблиць (наприклад, додавання нових характеристик для ювелірних виробів) фіксуються у вигляді файлів міграцій. Це дозволяє відстежувати історію змін та безпечно впроваджувати оновлення системи.

Лістинг коду 4.1 – Конфігурація БД зі змінними оточення

```
import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

# Перевірка середовища (Локальне чи Production)
IS_PRODUCTION = os.getenv('PRODUCTION', 'False') == 'True'

if IS_PRODUCTION:
    # Налаштування для PostgreSQL (Production)
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql',
            'NAME': os.getenv('DB_NAME'),
            'USER': os.getenv('DB_USER'),
            'PASSWORD': os.getenv('DB_PASSWORD'),
            'HOST': os.getenv('DB_HOST', 'localhost'),
            'PORT': os.getenv('DB_PORT', '5432'),
        }
    }
else:
    # Налаштування для SQLite (Локальна розробка)
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': BASE_DIR / 'db.sqlite3',
        }
    }
```

Лістинг коду 4.2 – Фізична реалізація таблиць БД за допомогою Django ORM

```
class Category(models.Model):
    name = models.CharField("Назва", max_length=100)
    slug = models.SlugField(unique=True, max_length=120)

    class Meta:
        verbose_name = "Категорія"
        verbose_name_plural = "Категорії товарів"
        ordering = ["name"]

    def __str__(self):
        return self.name

    def save(self, *args, **kwargs):
```

```
if not self.slug:
    self.slug = slugify(self.name)[:120] or "category"
super().save(*args, **kwargs)
```

	id	name	slug
1	1	Каблучки	kabliuchky
2	2	Кольє	kolia
3	3	Браслети	braslety

Рисунок 4.1 – Створена таблиця БД Category

На основі створеної таблиці категорії товару (рис. 4.1), було також створено такі моделі як: формування замовлення, список бажаного (wishlist), види каміння тощо.

4.4 Реалізація backend-частини застосунку

Серверна частина вебзастосунку побудована на мові програмування Python, використовуючи потужний фреймворк Django та його розширення Django REST Framework (DRF).

Оскільки архітектура системи є розділеною (Headless), бекенд не відповідає за формування візуальної частини вебсторінок (HTML). Його основна роль полягає в обробці запитів, що надходять від клієнта через протокол HTTP, виконанні необхідних бізнес-операцій та обміні даними з клієнтським застосунком у зручному форматі JSON.

Дані, які зберігаються в реляційній базі даних PostgreSQL, мають складну структуру, що не дозволяє безпосередньо надсилати їх мережею.

Щоб вирішити цю проблему, використано механізм серіалізації. Спеціальні класи, які називаються серіалізаторами (Serializers), виконують двостороннє перетворення. Вони беруть складні об'єкти Python (наприклад, результати запитів до бази даних або екземпляри моделей) і перетворюють їх на прості типи даних, які легко передаються у форматі JSON. Крім того, серіалізатори виконують

десеріалізацію – перевірку та валідацію даних, що надходять від клієнта, перед їх збереженням у базі даних.

Для того, щоб уникнути повторення коду та дотримуватися принципу «Не повторюй самого себе» (DRY), використано інструменти DRF – набори представлень (ModelViewSet).

Це дозволило автоматично створити базові операції для роботи з основними елементами каталогу: додавання, перегляд, редагування та видалення (CRUD). Для більш складних завдань, таких як розрахунок вартості товарів у кошику або застосування фільтрів за кількома критеріями, ми модифікували стандартні методи цих представлень.

Оскільки клієнтська та серверна частини повністю відокремлені, традиційні серверні сесії (з використанням файлів cookie) не можуть бути використані. Тому для ідентифікації користувачів та захисту конфіденційних даних (наприклад, історії замовлень) впроваджено систему безстанової автентифікації на основі JSON Web Tokens (JWT).

Після успішного входу користувача сервер генерує два криптографічно підписаних токени:

1) access_token: короткотривалий токен для доступу до захищених ресурсів;

2) refresh_token: токен для безпечного поновлення доступу, коли access_token закінчується.

Цей підхід значно зменшує навантаження на базу даних, оскільки серверу не потрібно зберігати інформацію про стан сесії кожного користувача. Вся необхідна інформація для перевірки автентичності міститься безпосередньо в самому токені.

Лістинг 4.3 – Серіалізатор для моделі ювелірних товарів

```
from rest_framework import serializers
from .models import Product, Category

class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = ['id', 'name', 'slug']
```

```
    class ProductSerializer(serializers.ModelSerializer):
        # Відображення назви категорії замість простого ID
        category_name = serializers.StringRelatedField(source='category',
read_only=True)
        # Повернення URL-адреси зображення
        image_url = serializers.SerializerMethodField()

        class Meta:
            model = Product
            fields = [
                'id', 'category_name', 'name', 'description',
                'price', 'stock', 'image_url', 'available'
            ]

        def get_image_url(self, obj):
            # Формування повного шляху до медіафайлу
            if obj.image:
                request = self.context.get('request')
                if request:
                    return request.build_absolute_uri(obj.image.url)

            return None
```

У лістингу 4.3 наведено приклад серіалізатора для моделі ювелірного виробу. Використано серіалізатор StringRelatedField для коректного відображення назви категорії замість її числового ідентифікатора.

У сфері електронної комерції стандартних операцій, які надає REST, часто не вистачає для реалізації всіх необхідних функцій. Наприклад, такі дії, як оновлення статусу замовлення або додавання ювелірних виробів до списку бажань, є складними процесами. Вони потребують спеціальних перевірок дозволів та додаткової логіки, що виходить за рамки простих операцій.

Щоб впоратися з цим, впроваджено декоратор @action з фреймворку DRF. Цей механізм дозволяє додавати нові, спеціалізовані маршрути (ендпоінти) до існуючих наборів представлень (ViewSets). Ці нові маршрути узагальнюють унікальні бізнес-процеси, роблячи систему більш гнучкою.

Крім того, для гарантування безпеки та цілісності даних на сервері, модифіковано стандартні методи збереження даних. Зокрема, метод perform_create у контролері замовлень перехоплює процес створення нового замовлення. Він автоматично пов'язує його з поточним користувачем, отримуючи його ідентифікатор безпосередньо з JWT-токена. Це робить неможливим для клієнта підробку ідентифікатора покупця.

Лістинг 4.4 – Реалізація кастомної дії для списку бажаного у контролері товарів

```
    @action(detail=True, methods=['post'], permission_classes=[permissions.IsAuthenticated])
    def toggle_wishlist(self, request, pk=None):
        """
        Кастомна дія для управління списком бажаного.
        Доступна лише для авторизованих користувачів за адресою:
        POST /api/products/{id}/toggle_wishlist/
        """
        product = self.get_object()
        user = request.user

        # Перевіряємо, чи товар вже є у вішлісті користувача
        if user in product.wishlisted_by.all():
            product.wishlisted_by.remove(user)
            return Response(
                {'status': 'success', 'message': 'Ювелірний виріб видалено з вішлісту'},
                status=status.HTTP_200_OK
            )
        else:
            product.wishlisted_by.add(user)
            return Response(
                {'status': 'success', 'message': 'Ювелірний виріб додано до вішлісту'},
                status=status.HTTP_200_OK
            )
```

4.5 Тестування застосунку за допомогою інструментів PyCharm

Тест (рис. 4.2) демонструє здатність системи генерувати JWT-токени при передачі коректних облікових даних користувача.

Рисунок 4.2 – Тестування механізму автентифікації

Отриманий статус 200 OK підтверджує, що механізм авторизації працює безпомилково: сервер успішно валідував логін та пароль і повернув пару криптографічних токенів для подальшої безпечної взаємодії із захищеними маршрутами системи.

Рисунок 4.3 – Тестування додавання товару до списку бажаних товарів

На скриншоті (рис. 4.3) виконується POST-запит до захищеного маршруту із передачею Access-токена в заголовку та ID товару в тілі запиту. Статус 201 Created підтверджує успішне проходження авторизації та коректне відпрацювання бізнес-логіки: сервер ідентифікував користувача і створив новий запис у базі даних, прив'язавши обраний ювелірний виріб до його профілю.

Рисунок 4.4 – Тестування логіки додавання товару в кошик

В даному тесті (рис. 4.4) виконується POST-запит на маршрут для додавання вказаної кількості обраного товару до кошика. Статус 200 OK гарантує, що транзакція пройшла успішно: система не лише створила або оновила позицію в кошику, але й автоматично перерахувала фінальну вартість замовлення на рівні бази даних, повернувши оновлену структуру кошика клієнту.

4.6 Демонстрація роботи вебзастосунку продажу ювелірних виробів

Даний підрозділ продемонструє ключові функції розробленого вебзастосунку продажу ювелірних виробів. Графічний інтерфейс розроблено в темному стилі з урахуванням застребуваних тенденцій у сфері вебдизайну та досвіду користувача.

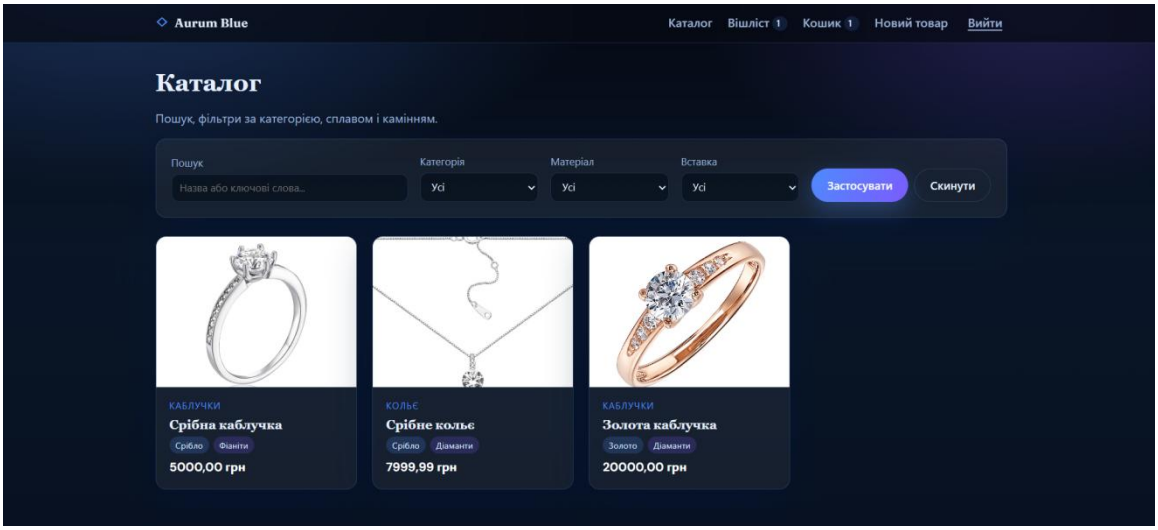

Рисунок 4.5 – Каталог онлайн-платформи

Основна зона для пошуку потрібних виробів (рис. 4.5) де присутній потужний набір фільтрів, що допоможуть відібрати товари за ключовими словами, типом, матеріалом (наприклад, срібло чи золото) та видом інкрустації. Результати відображаються у вигляді гнучкої сітки з картками товарів.

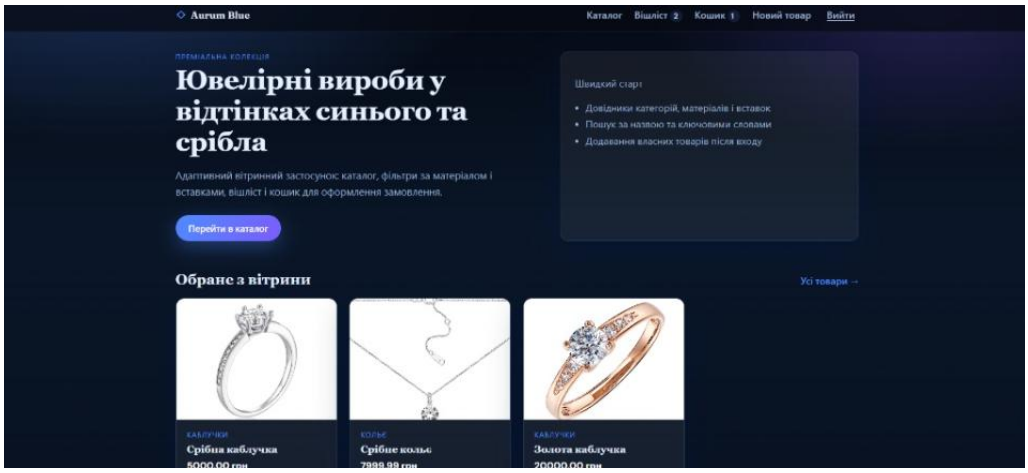

Рисунок 4.6 – Головна сторінка вебсайту

На скриншоті (рис. 4.6) зображена головна сторінка, що слугує вітриною. Тут розташований головний банер із закликом до дії, блок для швидкого старту та секція «Обране», яка представляє ключові товари.

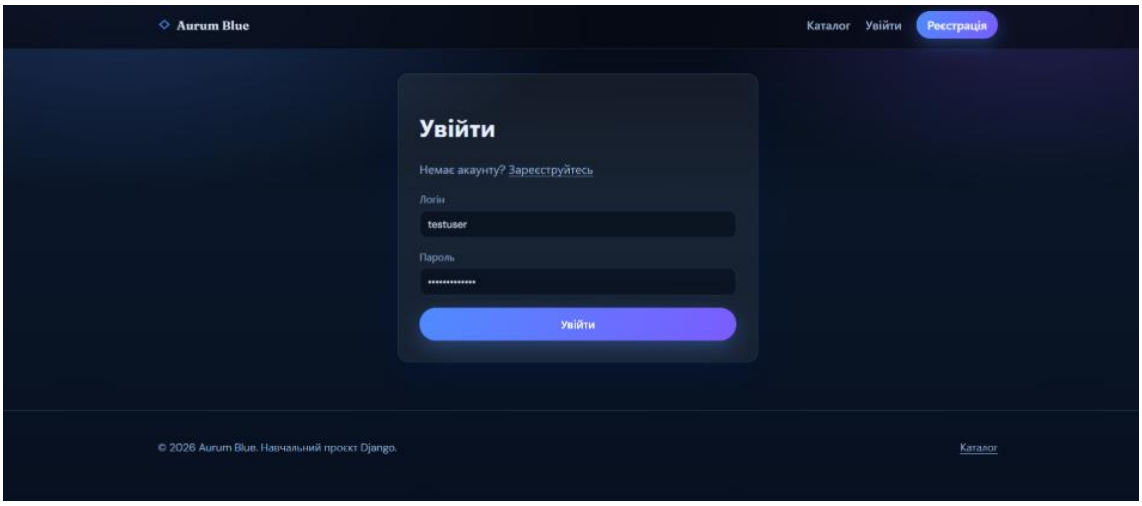

Рисунок 4.7 – Блок авторизації

Система входу (рис. 4.7), що забезпечує безпечний доступ до захищених розділів (список бажаного, кошик). Має просту форму для введення даних та можливість перейти до створення нового облікового запису.

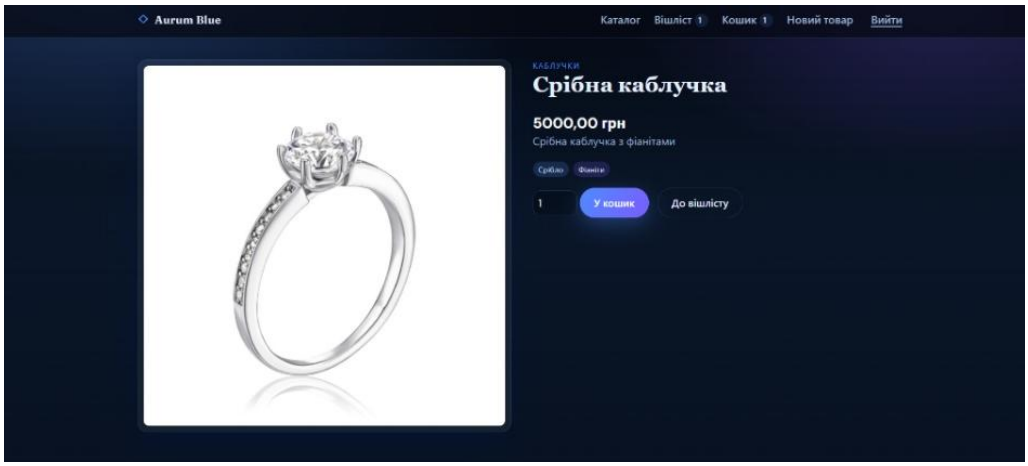

Рисунок 4.8 – Сторінка товару

На сторінці товару (рис. 4.8) є детальна інформація про обраний ювелірний виріб. Основний акцент зроблено на високоякісних фотографіях, ціні та технічних характеристиках. Інтерфейс містить елементи для здійснення покупки: вибір кількості та кнопки додавання до кошика або списку бажаного.

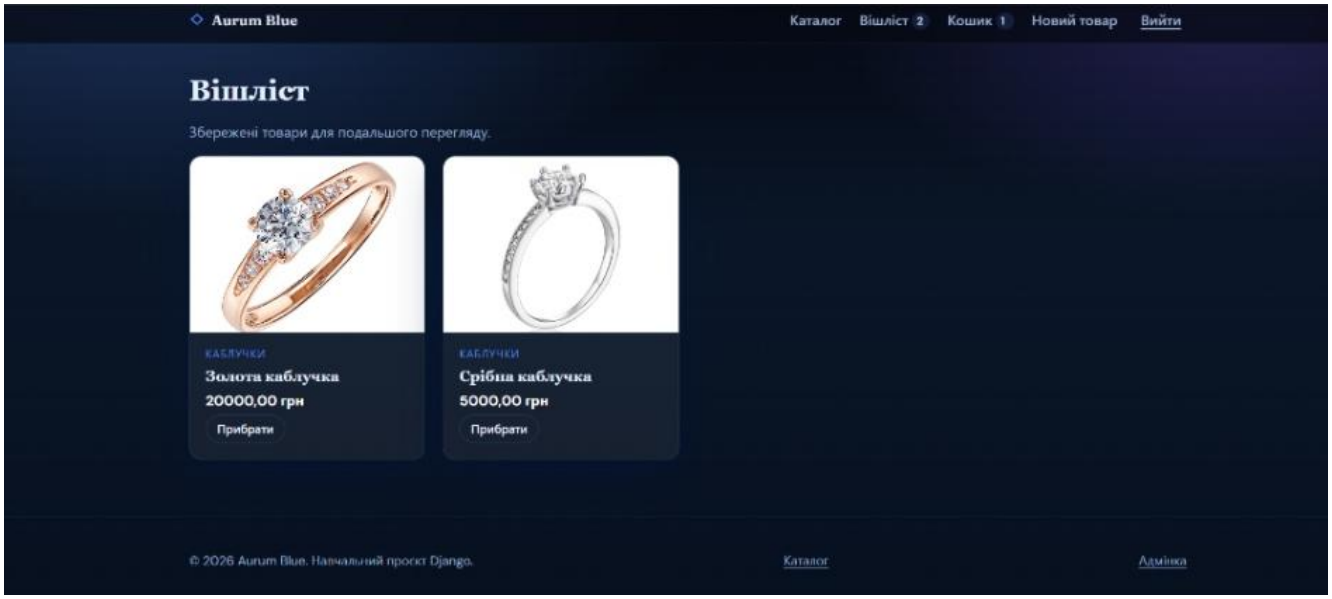

Рисунок 4.9 – Список бажаних товарів

На сторінці вішлісту (рис. 4.9) наявна персональна колекція відкладених товарів для зареєстрованих користувачів. Дозволяє легко переглядати збережені позиції та видаляти ті, що втратили актуальність.

Висновки до розділу 4

Четвертий розділ роботи детально описує практичну розробку вебмагазину для продажу ювелірних виробів. Застосована архітектура забезпечила гнучкість та масштабованість, розділивши логіку обробки даних від їх відображення.

Серверна частина (backend) створена на Python з використанням Django та Django REST Framework. Для роботи з базами даних застосовано SQLite на етапі розробки та PostgreSQL для продакшену, що гарантує цілісність даних, безпеку та швидкість обробки інформації про товари, замовлення та клієнтів.

Клієнтський інтерфейс (frontend) розроблено за допомогою Next.js. Завдяки серверному рендерингу (SSR) та адаптивному дизайну, застосунок є швидким, зручним та оптимізованим для пошукових систем. Особливості ювелірної галузі враховано через преміальний дизайн, розширену фільтрацію, список бажань та віртуальний кошик.

Практична частина завершилася успішним тестуванням. Інтеграція фронтенду та бекенду через REST API з використанням JWT-автентифікації підтвердила надійність системи. Результати свідчать про стабільність, відповідність вимогам та готовність магазину до використання в електронній комерції.

ВИСНОВКИ

Перший розділ присвячений глибокому дослідженню специфіки онлайн-продажу ювелірних виробів. Цей преміальний ринок потребує від програмних рішень не просто базових функцій для здійснення покупок, а й виняткової візуалізації товарів, ефективного управління детальними характеристиками виробів та найвищого рівня безпеки даних.

У другому розділі детально розглянуто, обрано та обґрунтовано програмні інструменти та методи для розробки ювелірного вебзастосунку. Дослідивши актуальні тренди веброзробки, сформовано висновок про доцільність застосування розподіленої Headless-архітектури.

У третьому розділі проведено моделювання та проєктування архітектури вебзастосунку для продажу ювелірних виробів.

За допомогою інструментарію UML детально змодельовано логіку роботи та поведінку системи. Діаграми використання допомогли чітко розмежувати права доступу для різних типів користувачів (від гостя до VIP-клієнта та персоналу).

Водночас побудовані діаграми взаємодії продемонстрували, як саме система обробляє критично важливі запити – від швидкого завантаження каталогу до проведення захищених фінансових транзакцій із перевіркою даних на сервері.

Отже, спроєктована система має чіткий розподіл відповідальності між компонентами, що створює надійний фундамент і дозволяє впевнено переходити до наступного етапу – безпосередньо написання та тестування програмного коду.

У четвертому розділі представлено повну практичну розробку вебзастосунку продажу ювелірних виробів, побудованого на розподіленій архітектурі. Серверна частина, написана на Python з використанням Django та DRF, працює з реляційними базами даних, забезпечуючи надійне зберігання та обробку даних. Клієнтський вебінтерфейс, створений на Next.js, використовує серверний рендеринг та адаптивний дизайн для зручного користування каталогом, списком бажань та кошиком.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The State of Fashion: Watches and Jewellery. *McKinsey & Company*. 2021. URL: https://www.mckinsey.com/industries/retail/our-insights/state-of-fashion-watches-and-jewellery (last accessed: 22.04.2026).

2. What to look for when adopting 3D & AR in the Luxury Sector. *Wearitar*. 2023. URL: https://blog.wearitar.com/what-to-look-for-when-adopting-3d-ar-in-the-luxury-sector/ (last accessed: 22.04.2026).

3. Headless Commerce: How a Headless + Jamstack Architecture Yields High Performance. *Netlify Whitepaper*. URL: https://www.netlify.com/pdf/netlify-headless-commerce-whitepaper.pdf (last accessed: 22.04.2026).

4. What is Headless Commerce? The Ultimate Guide. *Salesforce*. URL: https://www.salesforce.com/commerce/headless/guide/ (last accessed: 23.04.2026).

5. Офіційний сайт інтернет-магазину Pandora. URL: https://uk.pandora.net/ (дата звернення: 24.04.2026).

6. Офіційний сайт компанії Blue Nile. URL: https://www.bluenile.com/ (дата звернення: 24.04.2026).

7. Офіційний сайт компанії Brilliant Earth. URL: https://www.brilliantearth.com/ (дата звернення: 24.04.2026).

8. Di Meglio S., Starace L. L. L. Evaluating Performance and Resource Consumption of REST Frameworks and Execution Environments: Insights and Guidelines for Developers and Companies. *IEEE Access*. 2024. URL: https://ieeexplore.ieee.org/document/10741246 (last accessed: 26.04.2026).

9. Freeman A. Pro React 16. Berkeley : Apress, 2019. 764 p.

10. Next.js Documentation. Routing: Pages and Layouts. *Vercel*. URL: https://nextjs.org/docs/pages/building-your-application/routing (last accessed: 26.04.2026).

11. SQLite documentation. When to use SQLite. URL: https://www.sqlite.org/whentouse.html (last accessed: 26.04.2026).

12. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/ (last accessed: 26.04.2026).

13. Obe R. O., Hsu L. S. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 3rd ed. O'Reilly Media, 2017. 372 p.

14. Django documentation. Security in Django. URL: https://docs.djangoproject.com/en/stable/topics/security/ (last accessed: 27.04.2026).

15. Django REST Framework: Web APIs for Django. URL: https://www.django-rest-framework.org/ (last accessed: 27.04.2026).

ДОДАТОК А

Лістинг коду серіалайзерів вебзастосунку

```
from rest_framework import serializers

from .models import Product, Material, Stone, WishlistItem, Cart, CartLine

class MaterialSerializer(serializers.ModelSerializer):
    class Meta:
        model = Material
        fields = ['id', 'name', 'slug']

class StoneSerializer(serializers.ModelSerializer):
    class Meta:
        model = Stone
        fields = ['id', 'name', 'slug']

class ProductSerializer(serializers.ModelSerializer):
    category_name = serializers.ReadOnlyField(source='category.name')
    materials = MaterialSerializer(many=True, read_only=True)
    stones = StoneSerializer(many=True, read_only=True)

    class Meta:
        model = Product
        fields = [
            'id', 'title', 'slug', 'description', 'price',
            'category', 'category_name', 'materials', 'stones', 'image',
'created_at'
        ]

class WishlistItemSerializer(serializers.ModelSerializer):
    product = ProductSerializer(read_only=True)
    product_id = serializers.PrimaryKeyRelatedField(
        queryset=Product.objects.all(), source='product', write_only=True
    )

    class Meta:
        model = WishlistItem
        fields = ['id', 'product', 'product_id', 'created_at']

class CartLineSerializer(serializers.ModelSerializer):
    product = ProductSerializer(read_only=True)
    line_total = serializers.ReadOnlyField()

    class Meta:
        model = CartLine
        fields = ['id', 'product', 'quantity', 'line_total']

class CartSerializer(serializers.ModelSerializer):
    lines = CartLineSerializer(many=True, read_only=True)
    total = serializers.ReadOnlyField()

    class Meta:
        model = Cart
        fields = ['id', 'lines', 'total', 'updated_at']
```

ДОДАТОК Б

Лістинг коду тестів за допомогою середовища PyCharm

```
### Отримання JWT токена
POST http://127.0.0.1:8000/api/token/
Content-Type: application/json

{
  "username": "testuser",
  "password": "testuser12345"
}

### 1.1. Переглянути вішліст
GET {{host}}/wishlist/
Authorization: Bearer {{token}}
Accept: application/json

### 1.2. Додати товар до вішлісту
POST {{host}}/wishlist/
Authorization: Bearer {{token}}
Content-Type: application/json

{
  "product_id": 1
}

### 1.3. Перевірити вішліст після додавання
GET {{host}}/wishlist/
Authorization: Bearer {{token}}
Accept: application/json

### 1.4. Видалити товар з вішлісту
DELETE {{host}}/wishlist/1/
Authorization: Bearer {{token}}

### 2.1. Переглянути стан кошика
GET {{host}}/cart/my_cart/
Authorization: Bearer {{token}}
Accept: application/json

### 2.2. Додати 2 одиниці товару (з ID 1) у кошик
POST {{host}}/cart/add_item/
Authorization: Bearer {{token}}
Content-Type: application/json

{
  "product_id": 1,
  "quantity": 2
}

### 2.3. Додати ще 1 одиницю ТОГО Ж товару
POST {{host}}/cart/add_item/
Authorization: Bearer {{token}}
Content-Type: application/json
```

```
{
  "product_id": 1,
  "quantity": 1
}

### 2.4. Додати ІНШИЙ товар (з ID 2) у кошик
POST {{host}}/cart/add_item/
Authorization: Bearer {{token}}
Content-Type: application/json

{
  "product_id": 2,
  "quantity": 1
}

### 2.5. Фінальна перевірка кошика
GET {{host}}/cart/my_cart/
Authorization: Bearer {{token}}
Accept: application/json
```