**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 МЕБЛЕВОГО МАГАЗИНУ

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

**Здобувач** _____ **Олександр ФЕДОРОВСЬКИЙ**

«__» _____ 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**

**Федоровського Олександра**

1. Тема кваліфікаційної роботи Вебзастосунок меблевого магазину 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 меблевого магазину

4. Перелік питань, що підлягають розробці:

– аналіз предметної сфери та існуючих рішень;

– пошук і обґрунтування вибору інструментів, технологій та платформ для розробки вебзастосунку;

– проєктування бази даних;

– проєктування інтерфейсу користувача;

– тестування вебзастосунку.

5. Перелік графічних матеріалів:

Презентація

Дата видачі завдання «10» лютого 2026 р.

**КАЛЕНДАРНИЙ ПЛАН**
**виконання кваліфікаційної роботи**

Тема: **Вебзастосунок меблевого магазину**

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1. | Розробка та затвердження завдання на виконання КБР | 21.11.2025 | 01.12.2025 | Виконано |
| 2. | Огляд літератури за темою роботи | 20.01.2026 | 30.01.2026 | Виконано |
| 3. | Складання календарного плану КБР | 09.02.2026 | 10.02.2026 | 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 | 06.03.2026 | 07.04.2026 | Виконано |
| 8. | Відгук керівника КБР | | | |
| 9. | Оформлення КБР та презентації | 11.03.2026 | 25.05.2026 | Виконано |
| 10. | Попередній захист | 26.05.2026 | 26.05.2026 | Виконано |
| 11. | Рецензування | | | |
| 12. | Завершення оформлення КБР та презентації | | | |
| 13. | Захист кваліфікаційної роботи | | | |

**Здобувач** _____ **Олександр ФЕДОРОВСЬКИЙ**

«10» лютого 2026 р.

**Керівник роботи**

Канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«10» лютого 2026 р.

# АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

**«Вебзастосунок меблевого магазину»**

Здобувач 408 гр.: Федоровський Олександр

Керівник: канд. техн. наук, доцент Євген Давиденко

Кваліфікаційна робота присвячується розробці вебзастосунку для меблевого магазину, що забезпечуватиме надійну, швидку та безпечну взаємодію із каталогом товарів, пошук необхідних меблів та автоматизовані процеси електронної комерції.

**Метою роботи є** розробка вебзастосунку меблевого магазину, який забезпечить зручну взаємодію користувачів із каталогом товарів, надійну обробку даних та ефективне управління вмістом сайту з боку власника (адміністратора).

Для досягнення поставленої мети необхідно виконати наступний перелік завдань:

1) дослідження предметної галузі: аналіз існуючих рішень у сфері електронної торгівлі меблів, визначення їхніх переваг та недоліків;

2) проєктування інтерфейсу користувача з урахуванням зручності використання та відповідності сучасним стандартам вебдизайну;

3) реалізація інформаційних та архітектурних моделей, проєктування бази даних та логіки backend-частини застосунку;

4) тестування вебзастосунку для перевірки його функціональності, коректної роботи бази даних та загальної продуктивності системи.

**Об’єктом роботи** є процес електронної комерції меблевого магазину.

**Предметом кваліфікаційної роботи** є інструментарій розробки вебзастосунку меблевого магазину.

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 (аналітичній частині) проводиться огляд предметної області, аналіз існуючих аналогів галузі, визначаються їхні переваги та недоліки, а також наводиться детальне обґрунтування плану виконання завдання.

У другому розділі здійснюється моделювання об’єкту та предмету роботи. Формується специфікація вимог до програмного забезпечення, розробляються функціональні та інформаційні моделі вебзастосунку, а також наводиться опис обраних методів, технологій та математичного апарату.

Третій розділ присвячено загальній архітектура, моделюванню та проєктуванню програмного забезпечення, зокрема структури бази даних, клієнтської та серверної частин вебзастосунку меблевого магазину.

У четвертому розділі здійнюється етап програмної реалізації застосунку. Описується процес тестування ПЗ. Здійснюється глибокий аналіз результатів розробки.

У висновках проводиться аналіз виконаних робіт, оцінка ступеня виконання поставлених завдань та підбиваються загальні підсумки проєкту.

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 27 рисунків.

Ключові слова: *вебзастосунок, PostgreSQL, TailwindCSS, Django, Python, REST.*

# ABSTRACT

to the qualifying bachelor's thesis

**«Furniture store web application»**

Student of 408 group: Fedorovskiy Oleksandr

Supervisor: Candidate of Technical Sciences,

Associate Professor Davydenko Yevhen

The qualification work is dedicated to the development of a web application for a furniture store that will provide reliable, fast, and secure interaction with the product catalog, search for the desired furniture, and automated e-commerce processes.

The goal of this project is to develop a web application for a furniture store that will ensure convenient user interaction with the product catalog, reliable data processing, and effective management of the website’s content by the owner (administrator).

To achieve this goal, the following tasks must be completed:

1) domain research: analysis of existing solutions in the field of furniture e-commerce, identification of their advantages and disadvantages;

2) design of the user interface with a focus on usability and compliance with modern web design standards;

3) implementation of information and architectural models, database design, and backend logic of the application;

4) testing of the web application to verify its functionality, proper database operation, and overall system performance.

The scope of this work is the e-commerce process of a furniture store.

The subject of this work is the toolkit for developing a web application for a furniture store.

The qualification work 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 thesis.

The first chapter (analytical section) provides an overview of the subject area, analyzes existing industry analogues, identifies their advantages and disadvantages, and presents a detailed justification of the task implementation plan.

The second chapter involves modeling the object and subject of the work. Software requirements are specified, functional and information models of the web application are developed, and a description of the selected methods, technologies, and mathematical framework is provided.

The third chapter is devoted to the general architecture, modeling, and design of the software, specifically the database structure and the client and server components of the furniture store's web application.

The fourth chapter covers the software implementation phase of the application. The software testing process is described. A thorough analysis of the development results is conducted.

The conclusions include an analysis of the work performed, an assessment of the degree to which the set tasks were accomplished, and a summary of the project.

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 27 figures.

*Keywords: web application, PostgreSQL, TailwindCSS, Django, Python, REST.*

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

ACID – atomicity consistency isolation durability

API – application programming interface

CRM – customer relationship management

CSRF – cross-site request forgery

CSS – cascading style sheets

DRF – Django REST Framework

HTML – hypertext markup language

HTTP/HTTPS – hypertext transfer protocol secure

JSON – JavaScript object notation

ORM – object-relation mapping

OWASP – open worldwide application security project

PBKDF2 – password-based key derivation function

REST – representational state transfer

SHA256 – secure hash algorithm 256-bit

SQL – structured query language

SSL – secure sockets layer

TLS – transport layer security

UI / UX – user interface / user experience

UML – unified modelling language

URL – uniform resource locator

XSS – cross-site scripting

ПЗ – програмне забезпечення

СКБД – система керування базами даних

# ВСТУП

З розвитком цифрових технологій та стрімким зростанням сфери електронної комерції онлайн-покупки стали невід’ємною частиною сучасного суспільства. Користувачі очікують швидкого, зручного та безпечного доступу до інформації про товари, що сприяє ефективному вирішенню задач для бізнесу та задоволенню потреб споживачів.

Науково-практичне значення розробки вебзастосунку меблевого магазину полягає в удосконаленні засобів онлайн-торгівлі через використання сучасних технологій розробки у сфері вебу, оптимізації моделей баз даних та створення інтуїтивно-зручного інтерфейсу користувача.

**Метою роботи є** розробка вебзастосунку меблевого магазину, який забезпечить зручну взаємодію користувачів із каталогом товарів, надійну обробку даних та ефективне управління вмістом сайту з боку власника (адміністратора).

Для досягнення поставленої мети необхідно виконати наступний перелік завдань:

1) дослідження предметної галузі: аналіз існуючих рішень у сфері електронної торгівлі меблів, визначення їхніх переваг та недоліків;

2) проєктування інтерфейсу користувача з урахуванням зручності використання та відповідності сучасним стандартам вебдизайну;

3) реалізація інформаційних та архітектурних моделей, проєктування бази даних та логіки backend-частини застосунку;

4) тестування вебзастосунку для перевірки його функціональності, коректної роботи бази даних та загальної продуктивності системи.

**Об’єктом роботи** є процес електронної комерції меблевого магазину.

**Предметом роботи** є технології, методи проєктування та програмний інструментарій розробки вебзастосунку для електронної комерції.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У сучасному суспільстві електронна комерція відіграє важливу роль у задоволенні потреб споживачів та функціонуванні бізнесу. Одним із найважливіших інструментів на ринку є онлайн-магазини, що забезпечують швидкий, зручний та безпечний спосіб ознайомлення із каталогом товарів і здійснення замовлень.

Ринок таких послуг, які швидко зростають з такими функціями, як інтерактивний каталог товарів, функції фільтрації, навігація по кошику, інтеграції платіжних систем, а також персоналізовані облікові записи користувачів, процвітає.

Метою цього розділу є аналіз предметної області вебзастосунків для електронної комерції (зокрема, меблевих онлайн-магазинів). Це дозволяє сформувати цілісну картину про ключові моменти, на яких сформована розробка торгових онлайн-платформ, і слугує підґрунтям для подальшої розробки власного вебзастосунку меблевого магазину.

## 1.1 Огляд предметної області вебзастосунку

Проєктування меблевого інтернет-магазину, є складним процесом, що охоплює вивчення різних галузей, пов'язаних із сучасними інформаційними технологіями та процесами торгівлі.

Цей процес спрямований на аналіз актуальних вимог у сфері веброзробки, вивчення вимог клієнтів, особливостей рішень для онлайн-комерції, а також на формування необхідних навичок для створення функціонального та безпечного програмного забезпечення.

Важливо розглянути сучасні тенденції у сфері вебзастосунків. Веброзробка постійно еволюціонує під впливом технологічних інновацій та переходу користувачів на мобільні пристрої.

Зокрема, широке застосування серверних фреймворків (таких як Django або Node.js) у поєднанні з актуальними підходами до фронтенд-розробки

(використання React, Vue.js або потужних CSS-фреймворків) значно вплинуло на можливості створення швидких, адаптивних і масштабованих платформ для електронної комерції [1]. Знання цих тенденцій дозволяє розробникам краще орієнтуватися у виборі інструментів для реалізації ефективного інтернет-магазину.

Аналіз вимог користувачів є важливим етапом під час проєктування вебзастосунку. Сучасні покупці очікують швидкого завантаження сторінок, зрозумілої навігації, якісної візуалізації товарів та надійного захисту персональних даних. Для задоволення цих вимог необхідно забезпечити роботу бази даних для обробки запитів каталогу, реалізацію авторизації через безпечні протоколи та шифрування конфіденційної інформації (зокрема, деталей замовлень).

Аналізуючи популярних меблевих продавців в інтернеті, стають зрозуміліші пріоритетні функціональні можливості: комплексна система фільтрації меблів за властивостями, легка навігація по кошику та інформативна панель адміністратора, яка керує її вмістом [2].

Важливим аспектом є усвідомлення того, як архітектурні рішення, такі як монолітний чи мікросервісний підхід, впливають на реалізацію вебзастосунків. Вибір архітектури визначає специфіку проєктування системи, методи роботи з реляційними даними та стратегії масштабування. Зокрема, для великих маркетплейсів часто застосовується мікросервісна архітектура, що дозволяє незалежно функціонувати таким компонентам, як системи оплати, управління запасами та каталогу товарів.

Проте для розробки меблевого інтернет-магазину середньої масштабованості найбільш ефективним залишається використання монолітної архітектури (наприклад, патерну Model-View-Template), яка забезпечує високу швидкість розробки, цілісність даних та легкість розгортання застосунку на сервері [3].

Таким чином, розробка вебзастосунку меблевого магазину вимагає складного підходу до аналізу тенденцій в електронній комерції, розуміння потреб цільової аудиторії та коректного вибору архітектурних рішень для забезпечення стабільної роботи системи.

## 1.2 Аналіз існуючих програмних рішень

Вебзастосунок меблевого магазину – це цифрова платформа, створена для швидкого, безпечного та зручного вибору, налаштування й купівлі товарів. Такі рішення спрощують взаємодію клієнтів з асортиментом, надаючи доступ до детальних описів, фотографій та відгуків, що є критично важливим для сучасного ринку електронної комерції.

У сучасній торгівлі, де в пріоритеті є час та зручність для клієнта, інтернет-магазини стали незамінним інструментом торгівлі. Вебзастосунок меблевого магазину має забезпечити ефективну взаємодію користувачів із каталогом товарів за допомогою актуальних та застребуваних вебтехнологій [4].

Проєкт розробки вебзастосунку для меблевого магазину націлений на створення користувацького інтерфейсу, що забезпечує легкий пошук товарів та гнучкі можливості фільтрації.

Важливими вимогами є високий рівень безпеки платіжних транзакцій та масштабованість платформи для обробки великого трафіку.

Основні технічні завдання включають реалізацію механізмів реєстрації та авторизації користувачів, оптимізацію функціоналу кошика, адаптивну верстку для мобільних пристроїв, впровадження сучасних стандартів безпеки даних (наприклад, TLS-шифрування), а також інтеграцію з корпоративними системами CRM та обліку товарних запасів [5].

На сучасному ринку представлено багато вебзастосунків для продажу меблів та товарів для дому, серед яких IKEA, JYSK та Wayfair. Вони відрізняються функціоналом і дизайном, проте всі мають спільну мету – оптимізацію продажів і забезпечення зручності для користувачів [6].

**Інтернет-магазин «IKEA»**

IKEA – це міжнародна платформа найбільшої у світі мережі з продажу меблів. Її застосунок дозволяє не лише переглядати каталог товарів, але й перевіряти їх наявність у конкретних фізичних магазинах. Застосунок має багату екосистему, яка містить інтерактивні планувальники кімнат, можливість

збереження списків покупок та детальну специфікацію кожного елемента інтер’єру (від матеріалів до інструкцій зі збірки) [7].

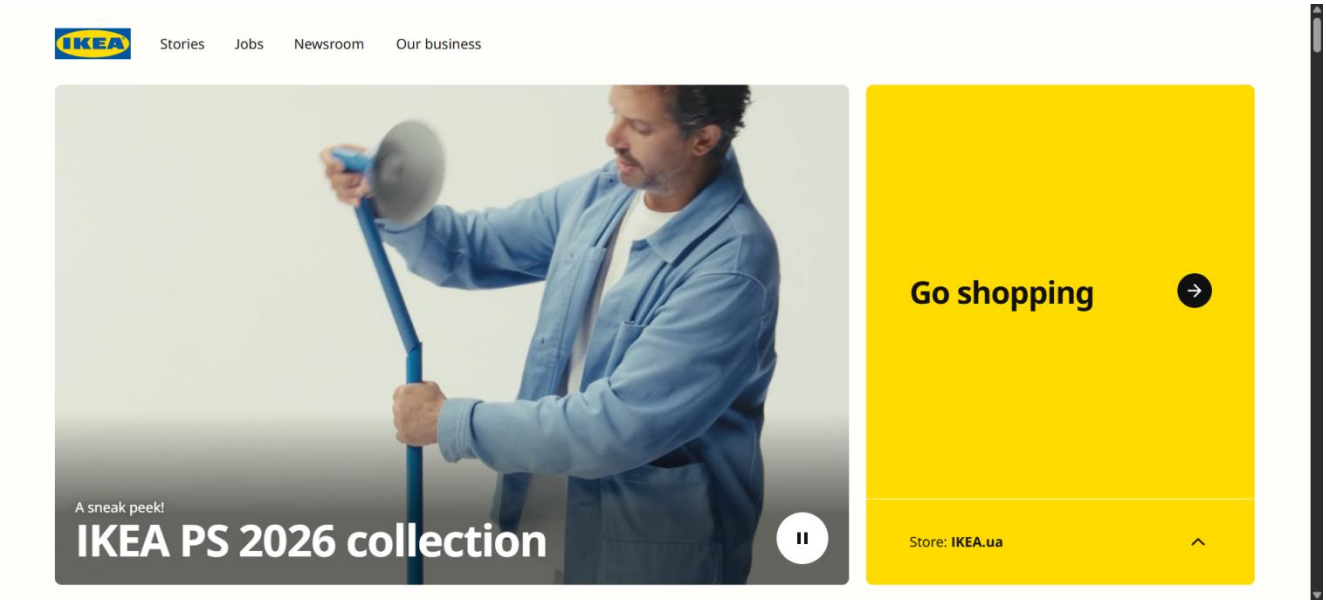


Рисунок 1.1 – Інтерфейс вебзастосунку IKEA

Розробник (дистриб’ютор): Inter IKEA Systems B.V.

Посилання: [4]

Архітектура: мікросервісна архітектура.

Технологічний стек:

– бекенд: Java, Node.JS, Python;
– фронтенд: Javascript (React), Typescript;

Перелік функцій, характеристик:

– ієрархічний каталог товарів;
– інтерактивні інструменти планування інтер'єру (3D);
– перевірка наявності товарів на складах у режимі реального часу;
– інтеграція з програмами лояльності (IKEA Family);
– розширений особистий кабінет з історією замовлень.

Переваги:

– інноваційний функціонал (візуалізація товарів в інтер'єрі);
– висока надійність та стабільність при великих навантаженнях;
– вичерпна інформація про кожен товар.

Недоліки:

– високі вимоги до інтернет-з'єднання через велику кількість важких медіафайлів;

– перевантаження інтерфейсу для користувачів, яким потрібна швидка покупка одного товару.

**Інтернет-магазин «JYSK»**

JYSK – це міжнародна мережа роздрібної торгівлі, онлайн-платформа якої спеціалізується на меблях та товарах для дому у скандинавському стилі. Вебзастосунок компанії вирізняється високою швидкістю, зручним процесом оформлення замовлень, зокрема завдяки функції «Замов та забери». Інтерфейс платформи характеризується лаконічністю, ефективною системою фільтрації та фокусом на відгуках користувачів [8].

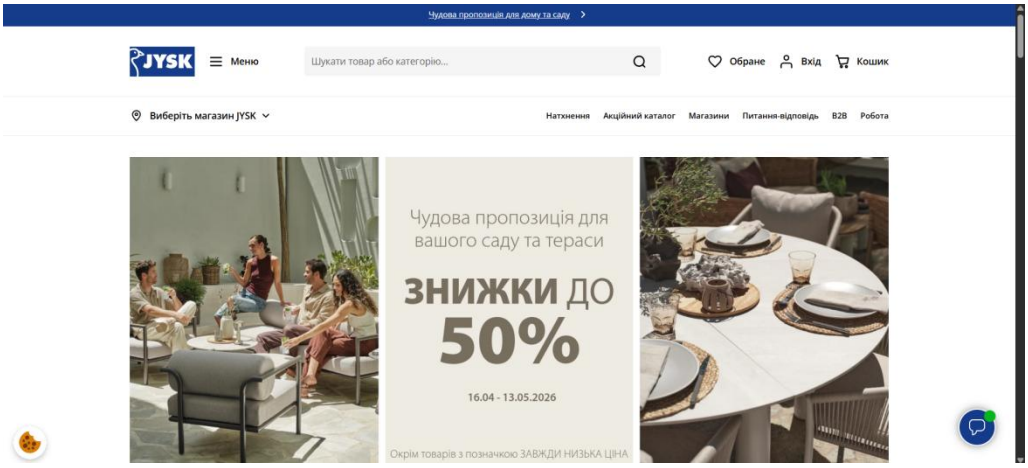


Рисунок 1.2 – Інтерфейс вебзастосунку JYSK

Розробник: JYSK A/S.

Посилання: [5]

Архітектура: клієнт-серверна архітектура.

Технологічний стек: PHP (на базі CMS рішень), JavaScript (Vue.js).

Основні характеристики:

– каталог із системою тегів та фільтрів;

– оформлення замовлення як зареєстрованим користувачем, так і гостем;

– інтегрована система відгуків та рейтингів товарів;

– пошук найближчого магазину за геолокацією;

– підтримка b2b-клієнтів.

Переваги:

– швидке завантаження сторінок та легка навігація;

– інтуїтивно зрозумілий процес оформлення покупки (checkout);

– чудова адаптація під мобільні пристрої.

Недоліки:

– відсутність інтерактивних інструментів для планування простору;

– обмежені можливості візуалізації (лише стандартні фото).

**Онлайн-маркетплейс «Wayfair»**

Wayfair – один із найбільших у світі цифрових продавців меблів та товарів для дому, який не має фізичних магазинів.

Їхній вебзастосунок є прикладом впровадження штучного інтелекту в сфері електронної комерції. Платформа пропонує детальну категоризацію товарів, потужну систему персоналізованих рекомендацій на основі історії переглядів, а також функцію візуального пошуку, що дозволяє знаходити меблі за завантаженим фото [9].

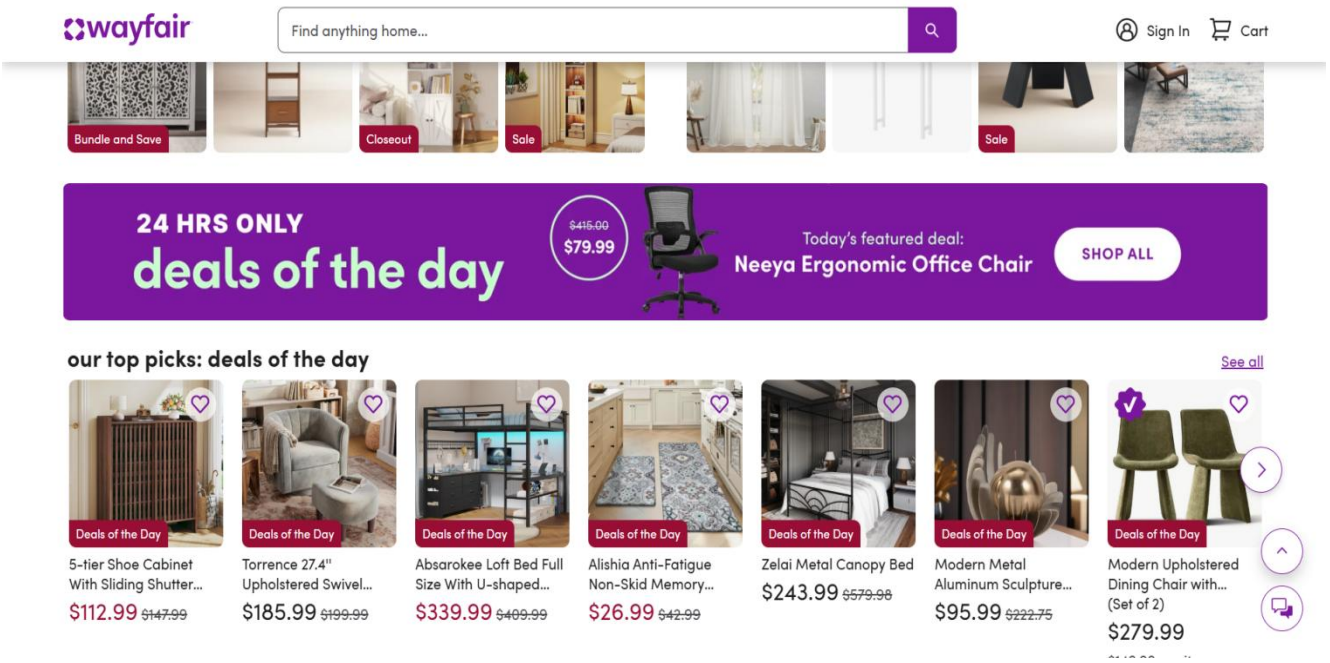


Рисунок 1.3 – Інтерфейс вебзастосунку Wayfair

Розробник (дистриб'ютор): Wayfair Inc.

Посилання: [6]

Архітектура: хмарна мікросервісна архітектура.

Технологічний стек: Python, PHP, React, GraphQL.

Основні функції:

– глибока фільтрація (за стилем, матеріалом, габаритами, кольором);
– візуальний пошук (visual search) за допомогою AI;
– персоналізована стрічка рекомендацій;
– динамічне ціноутворення та управління знижками;
– розширений кошик із підрахунком вартості доставки в реальному часі.

Переваги:

– найкращі на ринку алгоритми рекомендацій товарів;
– унікальний функціонал пошуку меблів за зображенням;
– детальні технічні специфікації до кожного товару.

Недоліки:

– інтерфейс може здаватися перевантаженим через велику кількість пропозицій;
– складна навігація через надмірно великий асортимент від різних постачальників.

### 1.3 Аналіз структурних і функціональних особливостей об’єкта роботи

Об’єктом цієї кваліфікаційної роботи є вебзастосунок для меблевого магазину, спроєктований з акцентом на швидкість розгортання, надійність та ефективне управління контентом.

Для реалізації було обрано монолітну архітектуру, що базується на високорівневому фреймворку Django (Python), шаблонізаторі Django Templates та CSS-фреймворку Tailwind CSS. Використання такого технологічного стека дозволяє забезпечити максимально швидкий цикл розробки завдяки наявності широкого спектра вбудованих інструментів для електронної комерції.

Застосунок побудовано на засадах класичної архітектури MVC (Model-View-Controller), яка в екосистемі Django реалізована через патерн MVT (Model-View-Template):

1) клієнтська частина реалізована за допомогою вбудованого шаблонізатора Django у поєднанні з фреймворком Tailwind CSS. Вона відповідає за відображення каталогу меблів, кошика та інтерфейсу користувача, забезпечуючи високу швидкість завантаження завдяки серверному рендерингу (SSR);

2) серверна частина побудована на мові Python та фреймворку Django. Вона забезпечує маршрутизацію, обробку бізнес-логіки електронної комерції, керування сесіями та автентифікацією користувачів. Важливою перевагою є використання автоматично згенерованої адміністративної панелі, що дозволяє швидко наповнювати магазин товарами;

3) рівень бази даних базується на об'єктно-реляційному відображенні (Django ORM), що дозволяє уникнути написання прямих SQL-запитів. Для етапу розробки та тестування застосовується СКБД SQLite з передбаченою архітектурною можливістю швидкої міграції на PostgreSQL для розгортання в продуктовому середовищі.

У структурі застосунку також присутні:

1) вбудована система автентифікації та захисту (захист від CSRF та SQL-ін'єкцій), що забезпечує безпечний доступ до особистого профілю клієнта;

2) модуль управління зображенням меблів;

3) система керування сесіями для збереження вмісту кошика покупок навіть для незареєстрованих відвідувачів;

4) адаптивний інтерфейс, що гарантує зручне відображення вітрини на всіх можливих пристроях.

**Функціонал вебзастосунку**

Основні можливості системи включають:

1) каталог продукції: перегляд списку меблів із детальними картками товарів, що містять описи, розміри, перелік матеріалів та фотографії;

2) пошук та навігація: фільтрація та сортування товарів за категоріями (дивани, столи, шафи тощо) та ціновим діапазоном;

3) кошик користувача: управління вибором товарів (додавання, видалення) із автоматичним розрахунком загальної вартості замовлення в режимі реального часу;

4) особистий кабінет: реєстрація, авторизація та управління профілем із доступом до історії попередніх замовлень;

5) оформлення замовлення (checkout): процес купівлі з валідацією контактних даних та інформації про доставку.

6) адміністрування: захищена панель для додавання нових товарів, оновлення цін, контролю залишків на складі та опрацювання вхідних замовлень.

**Висновки до розділу 1**

У першому розділі кваліфікаційної роботи проведено аналіз предметної області та сучасні підходи до реалізації інтернет-магазину для меблевого бізнесу.

Здійснено порівняння провідних існуючих вебзастосунків (IKEA, JYSK, Wayfair тощо), з урахуванням їх архітектури, структури, функціональності та ефективності.

Визначено головні структурні та функціональні особливості об’єкта роботи (клієнт-серверна архітектура за патерном MVT, безпечне керування сесіями й автентифікацією користувачів, а також реалізація кошика та панелі адміністратора тощо).

# 2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ

## 2.1 Моделі та методи для вирішення поставлених завдань

При створенні інтернет-магазину меблів був застосований комплексний інженерний підхід, що включає сучасні методи системного аналізу, проєктування баз даних та моделювання програмного забезпечення.

Приділяючи увагу вимогам платформ електронної комерції до продуктивності та відмовостійкості, ключовими завданнями були: забезпечення стабільності транзакцій, оптимізація швидкості завантаження вітрини товарів, гнучка масштабованість системи та розробка інтуїтивно зрозумілого користувацького інтерфейсу та досвіду (UI/UX). Архітектура системи побудована на основі найкращих галузевих практик (Best Practices) та сучасних вебтехнологій, з метою створення безперервного та логічного шляху клієнта від пошуку товару до завершення покупки.

У процесі реалізації були використані наступні методології:

1) моделювання сценаріїв використання (Use Case Modeling): для детального опису взаємодії користувачів із системою та визначення повного спектру функціональних вимог (наприклад, навігація каталогом, робота з кошиком, процес оформлення замовлення);

2) розбиття системи на незалежні функціональні модулі (каталог, управління замовленнями, профілі користувачів, адміністративна панель) для спрощення розробки та підтримки;

3) проєктування з орієнтацією на користувача (User-Centered Design): з урахуванням специфіки продажу меблів, що передбачає надання покупцям великої кількості візуальної інформації (галереї зображень) та детальних технічних характеристик.

Для забезпечення ефективності та надійності вебзастосунку його інформаційна архітектура розроблена з урахуванням принципів реляційної алгебри

та нормалізації. Це дозволяє уникнути дублювання даних, зберегти їхню цілісність та спростити управління.

База даних є центральним сховищем, яке організоване навколо ключових сутностей електронної комерції: клієнтів, категорій товарів, окремих товарних позицій, тимчасового вмісту кошика та фінальних замовлень.

Кожна сутність має чітко визначений набір властивостей. Наприклад, товар характеризується унікальним артикулом, поточною та акційною ціною, детальним описом, габаритами та статусом доступності. Зв'язки між даними реалізовані за допомогою первинних та зовнішніх ключів, що забезпечує коректне відображення відносин. Зокрема, реалізовано зв'язок «один-до-багатьох» між категоріями та товарами, а також складніший зв'язок «багато-до-багатьох» для управління кошиком та замовленнями, де один товар може бути в багатьох замовленнях, і навпаки.

Для взаємодії з базою даних застосовується система об'єктно-реляційного відображення (ORM), яка підвищує безпеку та абстрагує розробників від низькорівневих операцій з даними.

Програмна основа платформи побудована за принципами MVC (Model-View-Controller) – загальноприйнятим підходом до організації коду. У контексті фреймворку Django, ця модель має назву MVT (Model-View-Template). Така архітектура чітко розділяє завдання між різними частинами системи, що робить її легшою для підтримки та розвитку в майбутньому:

1) інтерфейс користувача (Template/Frontend): цей компонент відповідає за те, як люди бачать сайт. Він генерує сторінки безпосередньо на сервері (Server-Side Rendering). Завдяки використанню спеціальних шаблонів для розмітки (HTML) та сучасного CSS-фреймворку Tailwind CSS, створюється зрозумілий, адаптивний та швидкий інтерфейс. Це забезпечує коректне відображення магазину на екранах різного розміру та швидке завантаження контенту;

2) логіка застосунку (View/Backend): цей шар, написаний мовою Python, виконує роль «мозку» системи. Він обробляє запити до сайту, розуміє, що

користувач хоче зробити, звертається до бази даних для отримання потрібної інформації, виконує необхідні обчислення (наприклад, підраховує загальну вартість замовлення з урахуванням доставки) і передає ці дані для відображення на сторінці. Важливою частиною цього шару є автоматично створена адміністративна панель. Вона дозволяє легко керувати всіма даними магазину (додавати, переглядати, редагувати та видаляти товари, замовлення тощо) без необхідності писати додатковий код;

3) дані та їх зберігання (Model): цей компонент визначає структуру даних, які зберігаються в базі даних, та правила роботи з ними. Тут прописуються обмеження, щоб дані були коректними (наприклад, неможливо встановити від'ємну ціну на товар).

На відміну від систем, які потребують постійного зв'язку (як месенджери чи сервіси потокового відео, що використовують протокол WebSocket), електронна комерція найкраще працює за моделлю, де кожен запит є самостійним. Взаємодія між браузером та сервером магазину відбувається за допомогою стандартного протоколу HTTP/HTTPS. Це означає, що браузер надсилає запит, а сервер надсилає відповідь, після чого з'єднання розривається до наступного запиту.

Для забезпечення сучасного рівня інтерактивності (без необхідності повного перезавантаження сторінки під час кожної дії користувача), комунікаційна модель архітектурним стилем REST. Це дозволяє реалізувати такі функції, як додавання товару до кошика або миттєва фільтрація каталогу: браузер відправляє фоновий запит на сервер, який за допомогою Django REST Framework серіалізує результати пошуку у формат JSON та повертає їх для динамічного оновлення на стороні клієнта. Така гібридна комунікаційна модель (поєднання серверного рендерингу для головних сторінок та API-запитів для інтерактивних елементів) суттєво знижує навантаження на мережу та пришвидшує відгук інтерфейсу.

Особливості роботи інтернет-магазинів вимагають опрацювання чутливої інформації, такої як персональні дані клієнтів, історія їх покупок і платіжні реквізити. Через це гарантування конфіденційності, цілісності та доступності цих

даних є ключовим завданням для створеного застосунку. Безпека забезпечується багаторівневою системою захисту, яка інтегрована у основу фреймворку Django:

1) криптографічний захист автентифікації: для збереження облікових записів застосовується незворотне хешування паролів за допомогою алгоритму PBKDF2 із застосуванням криптографічної солі (salt) та численних ітерацій хешування (SHA256). Це виключає можливість розшифрування паролів навіть у випадку несанкціонованого доступу до бази даних користувачів;

2) захист від типових вразливостей: системна архітектура автоматично блокуватиме найпоширеніші типи атак, визначені стандартами OWASP. Використання ORM захищає від SQL-ін’єкцій за рахунок параметризації та екранування запитів до бази даних. Запобігання міжсайтовому скриптингу (XSS) реалізоване через вбудовані механізми шаблонизатора, який примусово перетворює користувацькі скрипти у безпечний текст;

3) захист сесій та транзакцій: для протидії атакам CSRF система створює унікальні, приховані криптографічні токени для кожної форми (реєстрація, редагування даних, оформлення замовлення). Сервер відмовляє у виконанні будь-яких змін, якщо такий токен відсутній або є недійсним;

4) шифрування на транспортному рівні: десь обмін інформацією між користувачем і сервером відбувається виключно через захищений протокол HTTPS. Використання SSL/TLS сертифікатів забезпечує безпечний канал передачі, захищений від атак типу «людина посередині», що є необхідною умовою для безпечної роботи інтернет-магазину.

## 2.2 Аналіз інструментарію майбутнього програмного забезпечення

Розробка вебзастосунку інтернет-магазину меблів вимагає ретельного аналізу сучасних технологій для проєктування архітектури, що забезпечить високу продуктивність каталогу, стабільність під час обробки замовлень та надійний захист персональних і платіжних даних та транзакцій клієнтів. Вибір технологій обумовлений специфікою системи: потребою в роботі з ієрархією товарів,

забезпеченням швидкої фільтрації бази даних, а також створенням зручного, адаптивного та інтуїтивно зрозумілого інтерфейсу для покупців.

Згідно з сучасними дослідженнями у сфері програмної інженерії, одним із найважливіших завдань при створенні вебзастосунків є дотримання принципу DRY (Don't Repeat Yourself), який передбачає оптимізацію роботи розробника через використання готових фреймворків [10]. Для обґрунтування вибору серверного інструментарію було проаналізовано наукову статтю, присвячену порівняльному аналізу продуктивності найпопулярніших фреймворків на мові Python – Django та Flask.

Аналіз даних, наведених у дослідженні, довів, що використання Django дозволяє значно скоротити час на розробку платформи. Реалізація базового функціоналу (автентифікація, панель адміністратора, робота з профілями) у Django займає значно менше часу завдяки наявності вбудованих інструментів, тоді як мікрофреймворк Flask вимагає написання більшої кількості коду вручну [10].

Крім того, дослідження виявило, що розмір згенерованих HTML-файлів у Django є меншим, оскільки Flask зберігає на сторінках більше інформації про поля форм, додаючи зайві теги.

Менший обсяг коду сторінки безпосередньо впливає на швидкість завантаження вітрини магазину в браузері клієнта. Виходячи з результатів цього наукового дослідження, для розробки серверної частини меблевого магазину було обрано фреймворк Django як найбільш продуктивний та надійний інструмент.

**Серверний рівень: Python, Django та DRF**

Python – це високорівнева мова програмування загального призначення, яка відзначається простотою синтаксису, зрозумілим для читання кодом та наявністю величезної кількості бібліотек, що робить її одним із лідерів у сфері серверної розробки.

Django виступає основним вебфреймворком серверної частини застосунку. Він побудований за архітектурним шаблоном MVT (Model-View-Template) і створений для швидкої розробки безпечних та масштабованих вебзастосунків.

Django ідеально підходить для електронної комерції, оскільки містить систему керування реляційними базами даних (ORM), надійні механізми автентифікації, захист від поширених вразливостей (CSRF, SQL-ін'єкції) та автоматично генеровану панель адміністратора, яка є критично необхідною для управління асортиментом меблів [11].

Django REST Framework (DRF) – це потужна та гнучка бібліотека для створення Web API в екосистемі стандартного Django. У межах розроблюваного застосунку DRF використовується для серіалізації складних об'єктів бази даних (наприклад, моделей товарів чи категорій) у формат JSON. Це дозволяє легко реалізувати асинхронні запити для фільтрації меблів без перезавантаження сторінки та створює основу для майбутнього масштабування проєкту (наприклад, розробки окремого мобільного застосунку) [12].

**Рівень бази даних: PostgreSQL**

PostgreSQL виступає основною системою керування реляційними базами даних (СКБД) для продуктового середовища вебзастосунку.

Електронна комерція вимагає абсолютної точності у збереженні транзакцій, замовлень та фінансових даних. PostgreSQL гарантує повну відповідність принципам ACID (атомарність, узгодженість, ізольованість, довговічність), що унеможливлює втрату даних при одночасних покупках кількома користувачами [13]. Вона чудово інтегрується з Django ORM та ефективно працює зі складними зв'язками між таблицями (товари, категорії, атрибути, відгуки).

**Клієнтський рівень: Django Templates та Tailwind CSS**

Для реалізації клієнтської частини (Frontend) замість створення відокремленого SPA-застосунку було обрано підхід рендерингу сторінок на стороні сервера (Server-Side Rendering) за допомогою Django Templates. Це рішення забезпечує максимальну швидкість розробки та найкращу пошукову оптимізацію (SEO), оскільки пошукові роботи миттєво отримують готовий HTML-код із назвами та описами меблів.

Tailwind CSS був обраний як основний інструмент для стилізації інтерфейсу [14]. На відміну від класичних фреймворків (таких як Bootstrap), Tailwind є утилітарним (utility-first) CSS-фреймворком. Він дозволяє створювати унікальний, сучасний та адаптивний дизайн безпосередньо в HTML-шаблонах шляхом комбінування низькорівневих класів.

Tailwind CSS забезпечує високу швидкість верстки карток товарів, навігаційних меню та кошика, гарантуючи при цьому, що підсумковий файл стилів матиме мінімальний розмір, оскільки міститиме лише ті класи, які фактично використані у проєкті.

## 2.3 Специфікація вимог до програмного забезпечення

**Цільове призначення проєкту:** головною метою є створення спеціалізованої платформи електронної комерції (інтернет-магазину меблів), яка забезпечить зручний пошук товарних позицій, безперебійне оформлення замовлень та управління каталогом через автоматизовану адміністративну панель.

**Обмеження та рамки виконання:**

1) граничний термін здачі в експлуатацію: 20.06.2026 р.;

2) технологічний стек: мова Python, фреймворки Django та Django REST Framework для серверної логіки, СКБД PostgreSQL для збереження даних, HTML/JS та Tailwind CSS для розробки клієнтського інтерфейсу.

**Галузь експлуатації:** програмний продукт орієнтований на сферу роздрібної онлайн-торгівлі предметами інтер'єру. Він надає покупцям цілодобовий та зручний доступ до вітрини магазину з будь-яких пристроїв без необхідності встановлення додаткового програмного забезпечення.

**Функціональні можливості системи:**

1) навігація каталогом та система фільтрації:

– опис: відвідувачі мають змогу переглядати асортимент меблів, сортувати їх за категоріями, вартістю чи характеристиками;

– вхідні дані: параметри пошуку або обрані критерії фільтрації;

– вихідні дані: відсортований перелік карток товарів, що відповідає запиту.

2) управління кошиком та фіналізація покупки:

– опис: користувач може додавати обрані предмети до кошика, редагувати їх кількість, бачити динамічний підрахунок вартості та формувати кінцеве замовлення;

– вхідні дані: ідентифікатори товарів, кількість, контактна інформація та адреса доставки;

– вихідні дані: згенероване в системі замовлення зі статусом «Очікує обробки».

3) реєстрація та управління профілем клієнта:

– опис: можливість створення особистого кабінету для збереження історії покупок та прискорення процесу оформлення наступних замовлень;

– вхідні дані: електронна пошта, пароль, ПІБ;

– вихідні дані: активна автентифікована сесія або згенерований JWT-токен.

4) адміністрування контенту:

– опис: закритий модуль для менеджерів магазину, що дозволяє додавати нові моделі диванів чи шаф, змінювати ціни та обробляти вхідні заявки від клієнтів;

– вхідні дані: медіафайли (фотографії), текстові описи, числові значення (ціна, залишок);

– вихідні дані: оновлена структура бази даних та актуалізована вітрина сайту.

**Вимоги до інформаційного забезпечення:**

1) джерела даних: уся інформація про клієнтів, товарну номенклатуру та транзакції централізовано зберігається у реляційній базі даних PostgreSQL;

2) обробка даних: система повинна забезпечувати строгу зв'язність реляційних таблиць (наприклад, зв'язок замовлення з конкретним користувачем та набором товарів).

**Вимоги до технічної інфраструктури:**

1) операційне середовище: серверна частина сумісна з будь-якою ОС (переважно Linux-дистрибутиви), що має підтримку середовища Python;

2) ресурси: наявність виділеного або хмарного сервера для розгортання WSGI-сервера (наприклад, Gunicorn/Nginx) та запуску СКБД PostgreSQL.

**Вимоги до програмної архітектури:**

1) тип архітектури: монолітна система за патерном MTV (Model-View-Template) із розширенням до REST API для гнучкого обміну даними;

2) інструментарій: Python (Django, DRF) для бекенду, PostgreSQL для збереження даних.

**Вимоги до зовнішніх інтерфейсів (UI):**

Візуальна частина: графічний інтерфейс має бути адаптивним (Responsive Web Design), забезпечувати коректне відображення як на мобільних телефонах, так і на широкоформатних моніторах. Для стилізації використовується утилітарний CSS-фреймворк Tailwind CSS.

**Нефункціональні властивості платформи:**

1) кросплатформність: безперешкодний доступ через усі сучасні веббраузери (Chrome, Safari, Edge) незалежно від операційної системи покупця.

2) масштабованість: архітектура повинна витримувати збільшення товарного асортименту до десятків тисяч позицій без деградації швидкості бази даних.

3) швидкодія: сторінки каталогу мають завантажуватися миттєво завдяки серверному рендерингу (SSR) та оптимізації SQL-запитів через ORM.

4) відмовостійкість: гарантія транзикційної цілісності (ACID) – це унеможливлення ситуацій, коли товар списується зі складу, але замовлення не фіксується.

5) безпека: надійне хешування паролів (PBKDF2), захист від вразливостей XSS та CSRF, а також маршрутизація всього трафіку через захищений протокол HTTPS.

## Висновки до розділу 2

У другій частині кваліфікаційної роботи був сформований теоретичний фундамент для розробки меблевого інтернет-магазину. Спроєктовано базові моделі архітектури, а також визначено ключові механізми захисту персональних даних покупців під час здійснення онлайн-платежів. На основі огляду фахових наукових джерел здійснено обґрунтований вибір технологічного стека та інструментарію майбутнього програмного забезпечення, довівши доцільність використання зв'язки Python та Django у колаборації з базою даних PostgreSQL для забезпечення максимальної швидкості створення надійного продукту електронної комерації. Підсумком розділу стало формування детальної специфікації вимог до програмного забезпечення. У цьому документі чітко виявлені межі та мету проєкту, базовий функціонал, перелік нефункціональних властивостей системи, вимоги до технічної інфраструктури, програмної архітектури та вимоги до інформаційного забезпечення.

# 3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Даний розділ присвячено розробці проєктних рішень, що забезпечують реалізацію специфікації вимог до ПЗ (моделювання функцій та інформаційних потоків об’єкту).

## 3.1 Реалізація сценаріїв та діаграми використання системи

Use-Case є важливим інструментом для проєктування та розробки програмного забезпечення, так як вони надають допомогу в розумінні потреб користувачів і визначають як система має поводитися в різних ситуаціях.

Таблиця 3.1 – Use Case 1: «Авторизація користувача (Вхід)»

| **Характеристика** | **Опис** |
|---|---|
| **Область застосування (Scope)** | Вебзастосунок меблевого магазину. |
| **Рівень (Level)** | User-goal. |
| **Основний актор (Primary Actor)** | Зареєстрований клієнт. |
| **Зацікавлені сторони (Stakeholders)** | 1) клієнт: хоче увійти в свій профіль. |
| **Передумови (Preconditions)** | Користувач вже має створений акаунт. |
| **Гарантія успіху (Success Guarantee)** | Користувач увійшов у систему і бачить свій кабінет. |
| **Основний сценарій (Main Scenario)** | 1) користувач натискає кнопку «Увійти»;<br>2) вводить свій email та пароль;<br>3) система перевіряє дані та відкриває кабінет. |
| **Розширення (Extensions)** | Якщо пароль невірний, система пише помилку. |
| **Спеціальні вимоги** | Блокування акаунту після 5 невдалих спроб входу. |
| **Технологічні варіації** | Використовуються захищені сесії браузера. |
| **Частота використання** | Часто. |
| **Додаткові положення** | Є кнопка «Забули пароль?». |

Таблиця 3.2 – Use Case 2: «Перегляд каталогу меблів»

| **Характеристика** | **Опис** |
|---|---|
| **Область застосування (Scope)** | Вебзастосунок меблевого магазину. |
| **Рівень (Level)** | User-goal. |
| **Основний актор (Primary Actor)** | Користувач. |
| **Зацікавлені сторони (Stakeholders)** | 1) користувач: хоче подивитися асортимент магазину. |
| **Передумови (Preconditions)** | Немає. |
| **Гарантія успіху (Success Guarantee)** | Користувач бачить список товарів обраної категорії. |
| **Основний сценарій (Main Scenario)** | 1) користувач відкриває головне меню сайту;<br>2) обирає категорію, наприклад, «Дивани»;<br>3) система завантажує сторінку зі списком диванів. |
| **Розширення (Extensions)** | Немає. |
| **Спеціальні вимоги** | Швидке завантаження зображень меблів. |
| **Технологічні варіації** | Використовується поділ на сторінки (пагінація). |
| **Частота використання** | Дуже часто (основна дія відвідувачів). |
| **Додаткові положення** | Немає. |

Сценарії використання допомагають узагальнити вимоги користувачів та визначити функціональність та працездатність системи з боку їхнього використання.

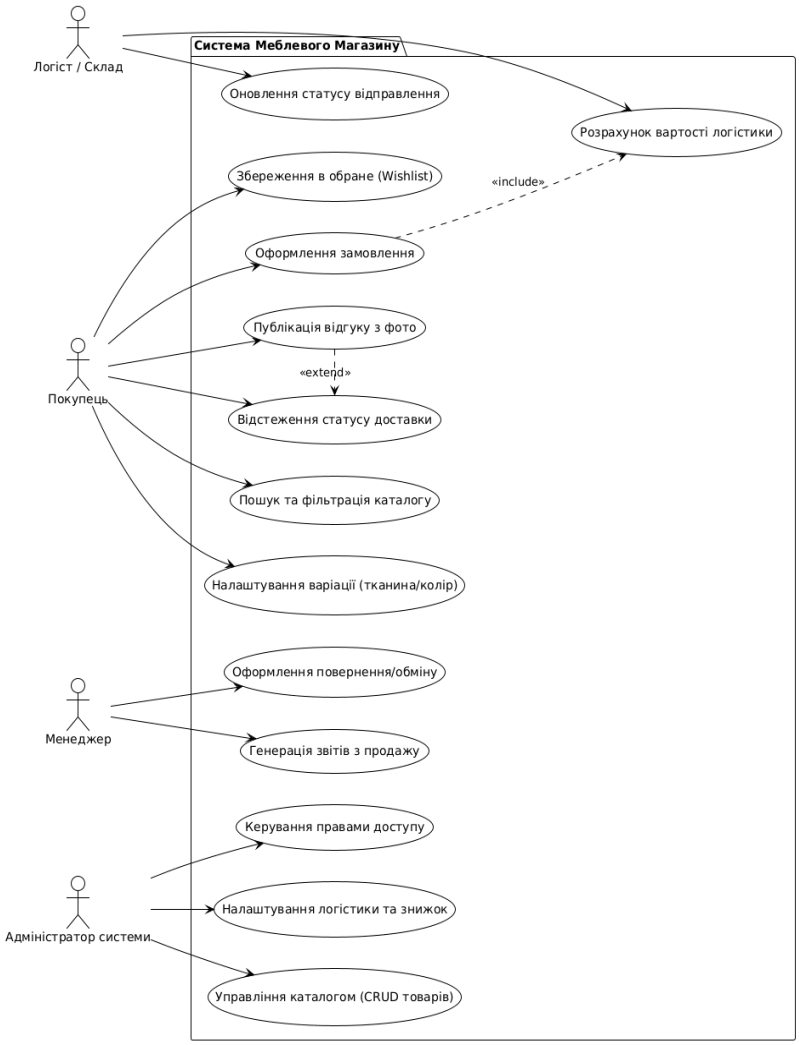


Рисунок 3.1 – Діаграма варіантів використання

Діаграма (рис. 3.1) містить в собі наступних акторів: логіста, що керує складом, покупця, менеджера та адміністратора системи.

Сценарії використання включають в собі: управління каталогом товарів, налаштування логістики та знижок, керування правами доступу для адміністратора системи; генерація звітів з продажу, оформлення повернення або обміну для менеджера; налаштування варіацій тканин або кольору для меблів, пошук та фільтрація каталогу, відстеження статусу доставки, оформлення замовлення, додавання до списку бажаного та публікація відгуку для покупця; оновлення статусу відправлення та розрахунок вартості логістики для керівника складом.

Під час створення діаграми використані наступні види відношень та зв’язків: unidirectional association, extend relationship, include relationship.

## 3.2 Реалізація діаграм діяльності

Діаграма (рис. 3.2) демонструє як споживач шукає товар у каталозі, конфігурує його (обирає тканину, колір мебелі) та як вебзастосунок реагує на ці дії (налаштовує ціну, перевіряє наявність на складі).

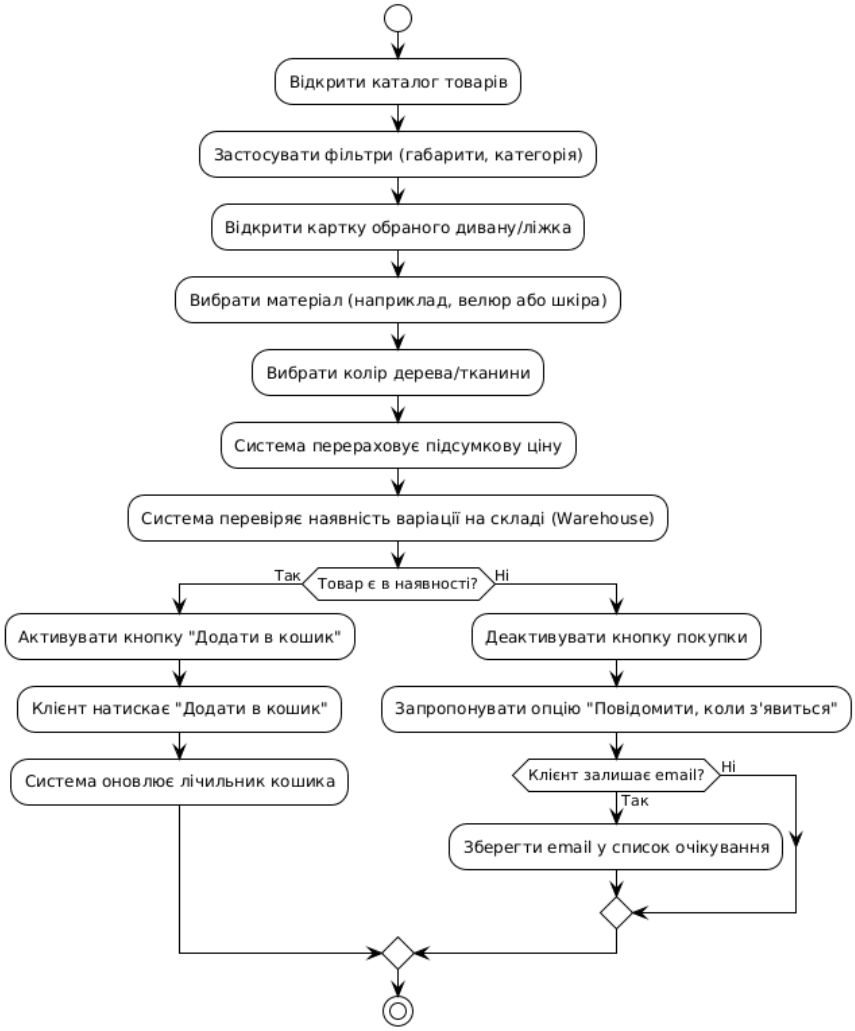


Рисунок 3.2 – Діаграма діяльності: Вибір та конфігурація меблів

Наступна діаграма (рис. 3.3) показує специфічні кроки чекауту, розрахунку вартості доставки меблів залежно від поверху та наявності ліфта.

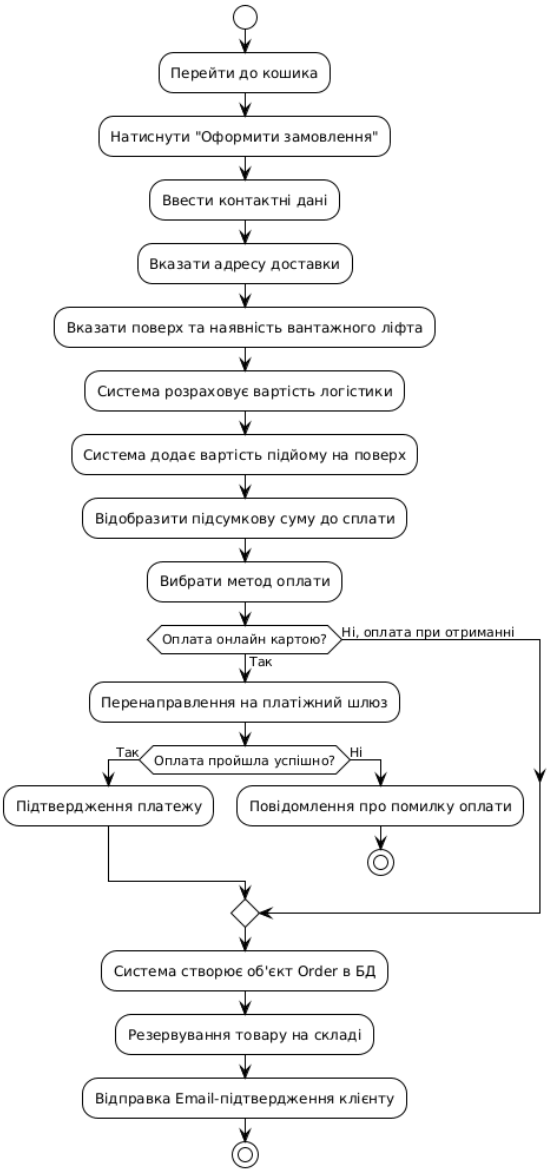


Рисунок 3.3 – Діаграма діяльності: Процес розрахунку логістики

Отже, діаграми діяльності активно використовують в сферах програмування, аналітики бізнесу, проєктуванні програмного забезпечення, моделюванні бізнес-процесів та інших областях для візуалізації та розуміння порядку виконання дій.

Вони дозволяють влучно продемонструвати послідовність дій у різних сценаріях та виявити можливі проблеми та оптимізувати процеси [15].

## 3.3 Реалізація діаграм взаємодії

Діаграми взаємодії в UML використовуються для встановлення зв'язків між об'єктами. Вони не маніпулюють даними, пов'язаними з конкретним шляхом зв'язку, а зосереджені на передачі повідомлень та на тому, як ці повідомлення формують певну функціональність системи. Діаграми взаємодії призначені для відображення того, як об'єкти реалізують конкретні системні вимоги. Ключовими компонентами таких діаграм є лінія життя та повідомлення [15].

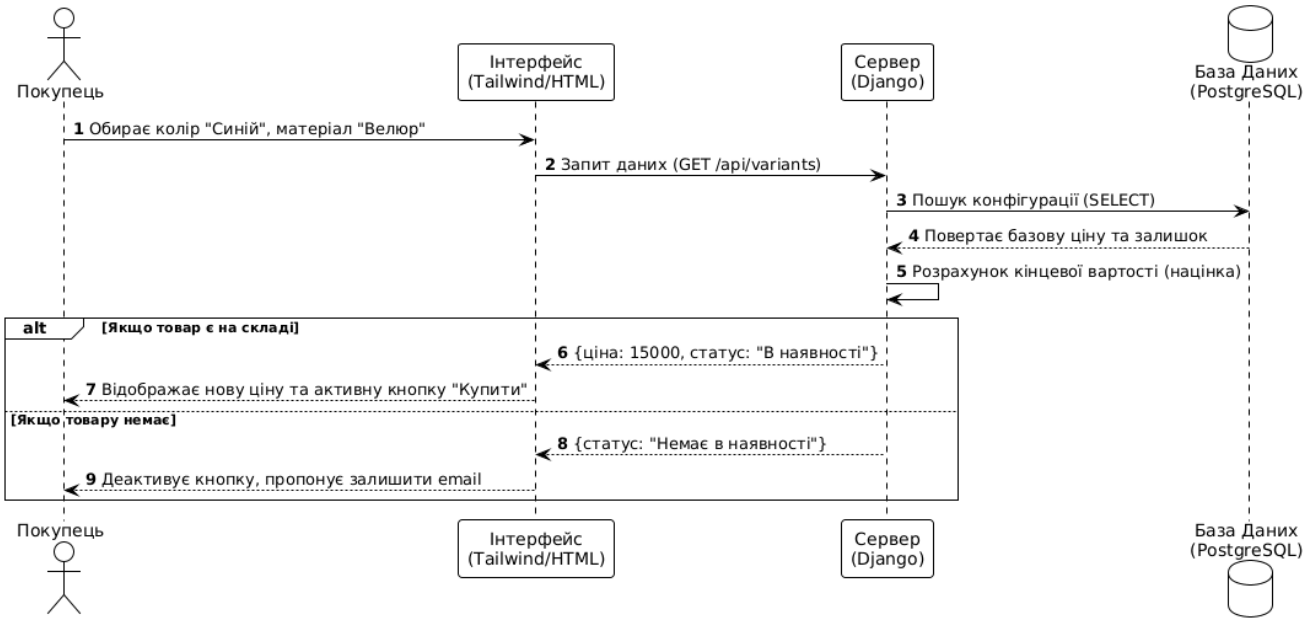


Рисунок 3.4 – Діаграма взаємодії: Вибір варіації меблів

На даній діаграмі (рис. 3.4) продемонстровано синхронний обмін повідомленнями під час вибору варіації меблів.

Клієнт відправляє запит на Django-сервер, що звертається до бази даних PostgreSQL, для перевірки наявності конкретної конфігурації меблів на складі.

Залежно від відповіді бази даних, сервер формує відповідь, а клієнтський інтерфейс змінює свій стан: блокує або активує опцію покупки.

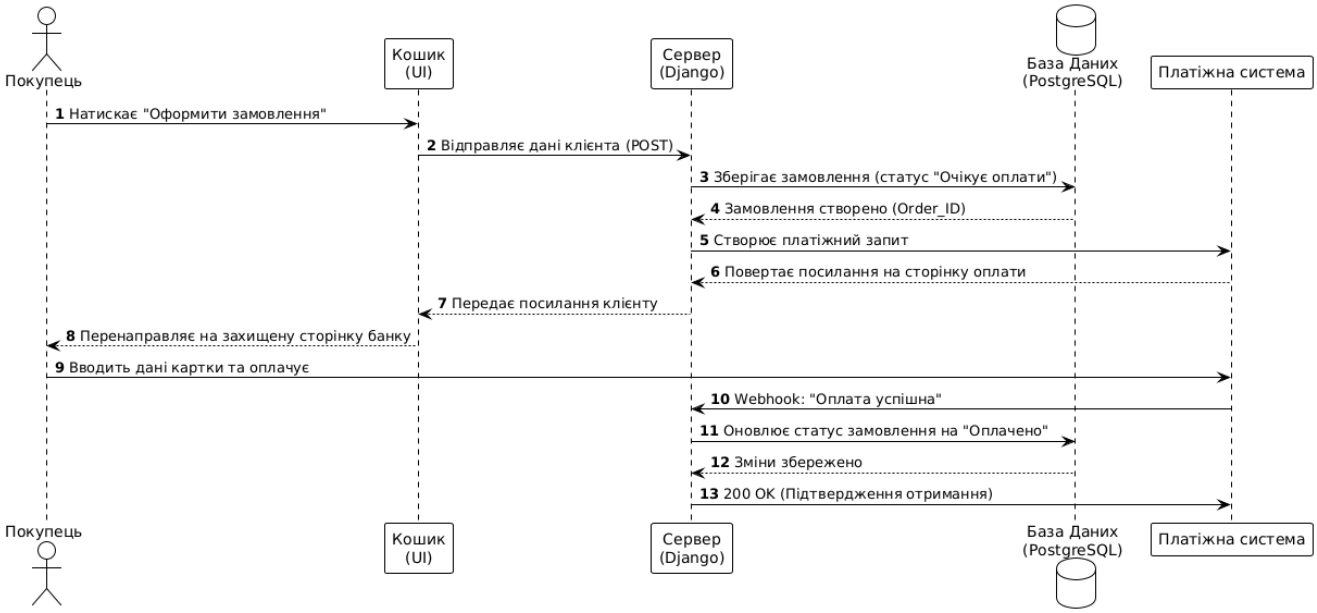


Рисунок 3.5 – Діаграма взаємодії: Оформлення та оплата замовлення

На наступній діаграмі (рис. 3.5) зображено процес чекауту, додається об’єкт зовнішньої платіжної системи, з якої взаємодіятиме сервер.

Проілюстровано процес оформлення замовлення з впровадженням платіжних транзакцій.

Після отримання підтвердження від платіжної системи, сервер самостійно оновлює статус замовлення в базі даних.

## 3.4 Реалізація діаграми класів

Одним із найпопулярніших типів UML-діаграм є діаграма класів. Інженери використовують її для документування архітектури програмного забезпечення. Оскільки такі діаграми визначають склад системи, що моделюється, вони належать до структурних діаграм [15].

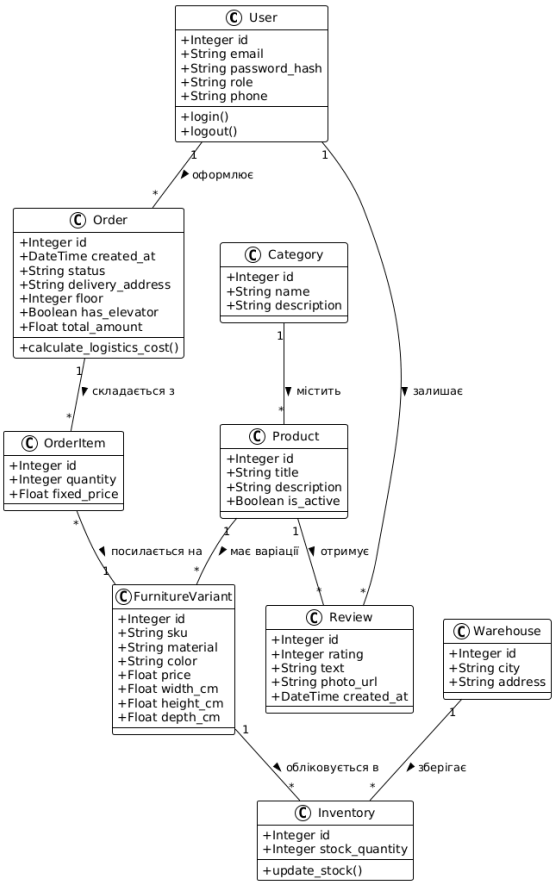


Рисунок 3.6 – Діаграма класів

Для забезпечення стабільної роботи та цілісності даних, архітектура нашої системи побудована на основі логічних модулів. Кожен клас у діаграмі відображає ключові бізнес-процеси, специфічні для продажу меблів.

Центральним елементом системи є клас User, який слугує єдиною точкою автентифікації. Замість створення окремих сутностей для клієнтів і співробітників, ми використовуємо атрибут role. Цей підхід дозволяє системі миттєво ідентифікувати тип користувача (клієнт, менеджер, адміністратор) і застосовувати відповідні права доступу, що значно спрощує адміністрування.

Специфіка меблів, що полягає у великій кількості варіацій однієї моделі, вимагала багаторівневої організації каталогу для уникнення надлишковості даних:

1) Category (Категорія): Верхній рівень ієрархії для навігації (напр., «М'які меблі»);

2) Product (Продукт): Абстрактна сутність, що містить загальну інформацію про модель (наприклад, назва «Диван Лофт», опис);

3) FurnitureVariant (Варіація товару): Ключова сутність, що представляє конкретну конфігурацію товару (наприклад, диван з велюровою оббивкою смарагдового кольору). Саме на цьому рівні визначаються унікальний артикул (SKU) та кінцева ціна.

Для ефективного управління габаритними товарами реалізовано наступні класи:

1) Warehouse (Склад): Зберігає дані про фізичні локації зберігання товару;

2) Inventory (Складські запаси): Асоціативний клас, що пов'язує конкретну варіацію товару (FurnitureVariant) з певним складом (Warehouse) та відстежує точну кількість одиниць (stock_quantity) у кожній локації.

Процес покупки реалізовано через два основні класи:

1) Order (Замовлення): Агрегує загальну інформацію про транзакцію, включаючи специфічні для меблів логістичні параметри (поверх, наявність вантажного ліфта). Метод calculate_logistics_cost() автоматично розраховує вартість доставки на основі цих даних;

2) OrderItem (Позиція замовлення): Деталізує склад кошика. Важливим архітектурним рішенням є поле fixed_price, яке фіксує вартість товару на момент покупки. Це гарантує коректність фінансових звітів та історії замовлень, незалежно від майбутніх змін цін у каталозі.

Клас Review (Відгук) дозволяє користувачам (User) залишати зворотній зв'язок щодо продукту (Product).

Можливість додавати фотографії (photo_url) підвищує довіру потенційних клієнтів, демонструючи реальний вигляд меблів в інтер'єрі.

### 3.5 Реалізація діаграми розгортання

Діаграма розгортання – це візуальна модель в UML, що демонструє фізичну архітектуру системи. Вона показує, на яких саме обчислювальних вузлах (серверах, комп'ютерах) будуть розміщені та виконуватимуться програмні компоненти та об'єкти [15].

Ключові відмінності від діаграми компонентів:

1) діаграма розгортання фокусується на фізичному розміщенні та взаємодії робочих екземплярів компонентів у реальному середовищі;

2) діаграма компонентів описує логічну структуру та зв'язки між типами компонентів, незалежно від того, де вони будуть виконуватися.

**Основні елементи діаграми:**

На діаграмі розгортання вузли зображуються як тривимірні прямокутники. Всередині них розташовуються артефакти (виконувані файли, бібліотеки), представлені звичайними прямокутниками. Вузли можуть бути вкладеними, а один логічний вузол може символізувати групу фізичних ресурсів, як-от кластер баз даних.

**Типи вузлів:**

1) вузол пристрою (Device Node): Це фізичний обчислювальний ресурс, що має власну пам'ять та процесорні потужності для виконання програм. Приклади: персональний комп'ютер, мобільний телефон, сервер;

2) вузол середовища виконання (Execution Environment Node): Це програмний ресурс, який функціонує всередині вузла-пристрою і створює середовище для виконання інших програмних елементів. Приклади: операційна система, віртуальна машина Java (JVM), веб-сервер.

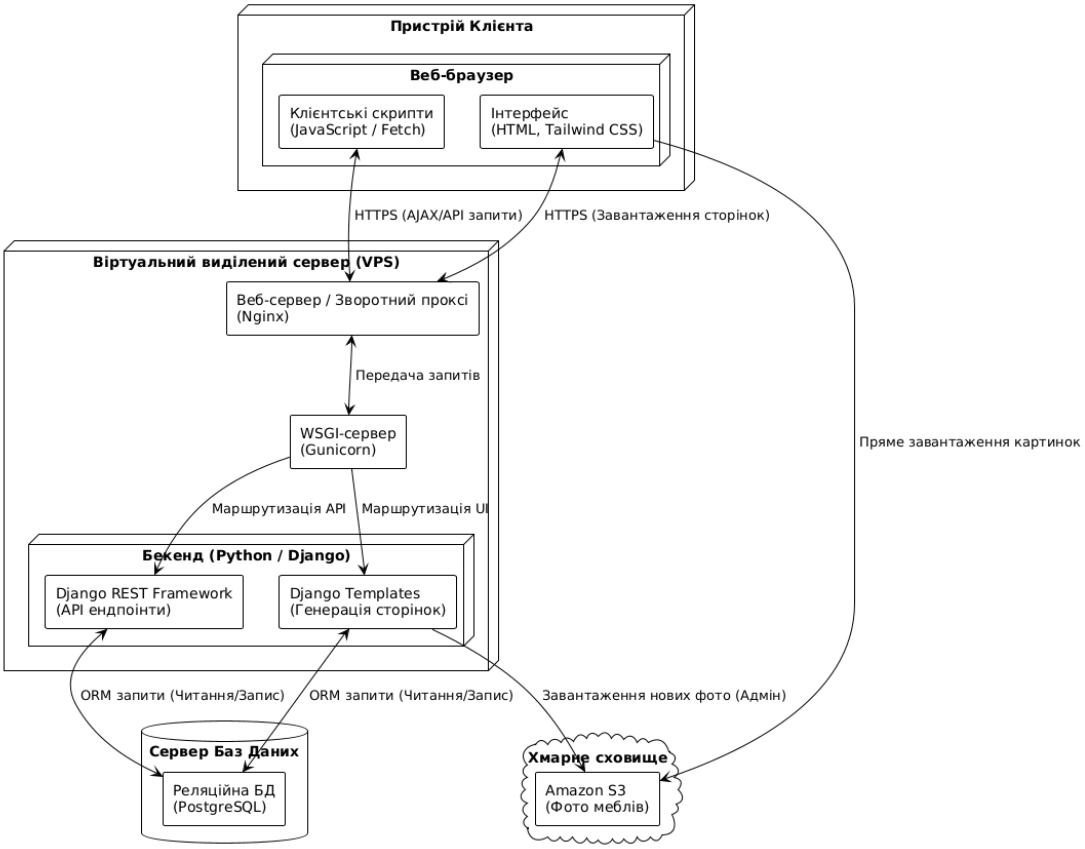


Рисунок 3.7 – Діаграма розгортання застосунку

**Структура та розгортання системи меблевого вебзастосунку**

Для забезпечення стабільної та ефективної роботи нашого меблевого вебзастосунку ми застосували класичну трирівневу архітектуру, яка включає взаємодію між клієнтом, сервером застосунку та базою даних. Додатково, для оптимізації відображення контенту, використовується гібридний підхід до рендерингу.

Основні компоненти системи та їх розміщення:

**Клієнтський рівень (Інтерфейс користувача)**

Як це працює: користувачі взаємодіють із системою через свої веббраузери.

Підходи до відображення:

1) статична частина (HTML + Tailwind CSS): Відповідає за базову структуру сторінок та їх візуальне оформлення. Це дозволяє швидко завантажувати каталог товарів, що є важливим для пошукових систем (SEO);

2) динамічна частина (JavaScript): Забезпечує інтерактивність. За допомогою асинхронних запитів (AJAX) користувачі можуть додавати товари до кошика, фільтрувати їх або оновлювати інформацію про доставку без необхідності повного перезавантаження сторінки.

**Серверний рівень (Віртуальний виділений сервер – VPS)**

Де це розміщено: основна логіка системи працює на орендованому віртуальному сервері (VPS) під управлінням операційної системи Linux.

Обробка запитів:

1) Nginx: Вебсервер, який першим приймає запити від користувачів. Він також виступає як зворотний проксі (Reverse Proxy), керує SSL-сертифікатами та передає запити далі;

2) Gunicorn: WSGI-сервер, який обробляє запити, що надходять від Nginx, і взаємодіє з додатком Django.

Бізнес-логіка (Django) розділена на два ключові модулі:

1) Django Templates (Серверний рендеринг): Генерує HTML-сторінки на стороні сервера. Відповідає за відображення головної сторінки, статичних розділів та детальної інформації про товари;

2) Django REST Framework (DRF) (API): Надає інтерфейс для взаємодії з клієнтським JavaScript у форматі JSON. Цей модуль обробляє динамічні операції, такі як оновлення цін, перевірка наявності товарів та керування вмістом кошика.

Рівень даних (Зберігання інформації):

1) як дані зберігаються: Для ефективного зберігання та швидкого доступу до інформації, дані розділені на текстові та мультимедійні;

2) основна база даних (PostgreSQL): Реляційна база даних, яка надійно зберігає інформацію про користувачів, структуру категорій товарів, деталі замовлень та дані про складські запаси. Django використовує вбудований механізм ORM для взаємодії з цією базою;

3) хмарне сховище для медіа (Amazon S3): Оскільки меблевий каталог містить багато високоякісних зображень, вони зберігаються на спеціалізованому

хмарному сервісі Amazon S3. Це значно зменшує навантаження на основний сервер, дозволяючи браузерам користувачів завантажувати зображення безпосередньо з хмари.

### 3.6 Реалізація мокапів для вебзастосунку

Мокапи є візуальними репрезентаціями, що дозволяють дизайнерам та розробникам симулювати кінцевий вигляд своїх робіт у реальному середовищі. Вони надають можливість продемонструвати інтеграцію елементів дизайну, таких як логотипи на текстилі чи брендинг на пакувальних матеріалах, до етапу фінального виробництва чи публікації. По суті, мокапи слугують інструментом для попередньої візуальної оцінки проекту зацікавленими сторонами.

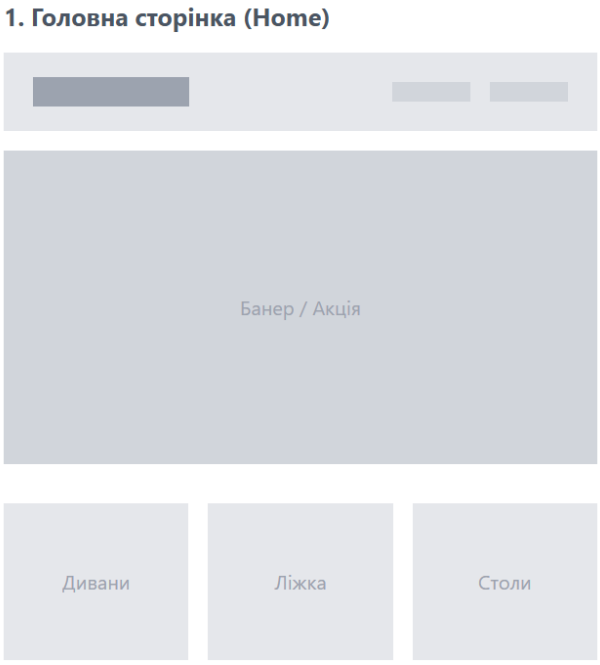


Рисунок 3.8 – Інтерфейс головної сторінки

Головна сторінка (Home Page) (рис 3.8): Функціональний елемент інтерфейсу, призначений для первинного залучення користувача та оптимізації процесу вибору товару.


Рисунок 3.9 – Інтерфейс каталогу товарів


Рисунок 3.10 – Інтерфейс деталей про продукт

В каталозі товарів (рис. 3.9) можна легко знайти потрібне серед великого вибору. Зліва – фільтри за ціною, розміром, матеріалом. По центру – картки товарів з коротким описом та ціною.

В інтерфейсі картки товару (рис 3.10) можливо детально ознайомитися з кожним предметом меблів. Переглядати фотографії в галереї, читати опис та експериментувати з різними варіантами оббивки, кольору та розміру.

## Висновки до розділу 3

Третій розділ кваліфікаційної роботи присвячений розробці детальних архітектурних та проєктних рішень для програмного забезпечення інтернет-магазину меблів. Ці рішення спрямовані на задоволення всіх структурних та функціональних вимог.

Для наочного представлення ключових бізнес-процесів вебзастосунку було розроблено комплекс UML-діаграм (прецедентів, класів, послідовності, діяльності та розгортання). Вони формалізують сценарії взаємодії клієнтів, включаючи пошук товарів за різними критеріями (габарити, матеріали, стиль), додавання до кошика, а також процеси обробки замовлень та управління асортиментом з боку адміністратора.

Детально охарактеризовано класи та компоненти, що формують програмну архітектуру, а також структуру бази даних та взаємодію модулів, відповідальних за зберігання даних про категорії товарів, складські запаси, профілі користувачів та історію транзакцій.

Графічний інтерфейс застосунку проілюстровано за допомогою mockup-макетів ключових сторінок (головна вітрина, каталог, картка товару, кошик, оформлення замовлення), що забезпечують високий рівень зручності користувацького досвіду (UX) та інтуїтивно зрозумілу навігацію.

# 4 КОДУВАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Цей розділ присвячено реалізації програмних та архітектурних рішень для вебзастосунку меблевого магазину, а також проведенню тестування окремих елементів логіки проєкту.

## 4.1 Створення моделей бази даних

Структура бази даних для вебзастосунку меблевого магазину реалізовано за допомогою інструментарію Django ORM. Django ORM (Django Object-Relational Mapping) – це технологія, що дозволяє працювати з базою даних за допомогою об’єктів у мові Python, а не через написання SQL-запитів [11].

```python
class Product(models.Model):  15 usages
    name = models.CharField( verbose_name: "Назва", max_length=120)
    category = models.ForeignKey(
        Category,
        on_delete=models.PROTECT,
        related_name="products",
        verbose_name="Категорія",
    )
    description = models.TextField("Опис")
    price = models.DecimalField( verbose_name: "Ціна, грн", max_digits=10, decimal_places=2)
    image_url = models.URLField( verbose_name: "Посилання на фото", blank=True)
    in_stock = models.BooleanField( verbose_name: "В наявності", default=True)
    material = models.CharField( verbose_name: "Матеріал", max_length=80, blank=True)
    color = models.CharField( verbose_name: "Колір", max_length=60, blank=True)

    class Meta:
        ordering = ("name",)

    def __str__(self) -> str:
        return self.name
```

Рисунок 4.1 – Лістинг коду моделі Product

Архітектура моделей (рис. 4.1) меблевого магазину поділена на три блоки: керування каталогом товарів, кошик та фіксація замовлень.

У каталозі основою слугують моделі Category та Product, між якими встановлено правило цілісності PROTECT, що не дозволяє видаляти категорії, якщо в них ще є товари. Управління активними покупками здійснюється через моделі Cart та CartItem: Cart має особистий зв’язок OneToOne з профілем користувача, а в базі даних за допомогою обмеження UniqueConstraint

запобігається появі однакових товарів у межах одного кошика. Ключовим архітектурним моментом є фінансовий блок, представлений моделями Order і OrderLine, який забезпечує абсолютну незмінність історії операцій: система фіксує ціну одиниці товару у полі unit_price безпосередньо на момент оформлення замовлення та не допускає видалення придбаних позицій з бази даних. Це надійно захищає фінансову звітність від будь-яких подальших змін цін у каталозі.

Для забезпечення цілісності даних та коректної роботи системи також передбачено детальне логування всіх ключових подій, пов’язаних з обробкою замовлень і актуалізацією кошиків користувачів. Це дає змогу відстежувати статуси транзакцій, проводити аудити та вчасно реагувати на потенційні помилки або невідповідності. Крім того, архітектура підтримує масштабування за рахунок чіткої розмежованості відповідальності між компонентами: каталог відповідає за підтримку актуальної інформації про товари й категорії, корзина – за управління тимчасовими покупками, а фінансовий блок – за надійне збереження історії замовлень і операцій над ними.

Для підвищення продуктивності передбачено використання оптимізованих індексів, особливо у таблицях CartItem та OrderLine, що суттєво пришвидшує пошук і агрегацію даних при формуванні звітів і виведенні інформації користувачам. Враховуючи потребу у масштабованості, застосовано кешування часто запитуваних даних товарів, що мінімізує навантаження на базу даних і скорочує час відповіді системи. Наразі також розглядається можливість реалізації шардінгу для розподілу навантаження при значному зростанні кількості користувачів та замовлень.

З боку безпеки впроваджено багаторівневу модель авторизації і автентифікації, яка гарантує, що доступ до управління кошиком, оформлення замовлень та перегляду особистої інформації мають лише уповноважені користувачі. Паролі зберігаються у захищеному вигляді із застосуванням сучасних алгоритмів хешування, а всі транзакції проходять через захищені канали зв’язку.

Реалізовано механізми захисту від атак типу CSRF та SQL-ін’єкцій, що підвищує стійкість системи до зовнішніх загроз.

Планується інтеграція з платіжними шлюзами для автоматизації процесу оплати й обробки фінансових результатів, а також впровадження системи аналітики на базі історичних даних замовлень, яка допоможе оптимізувати асортимент, ціноутворення та маркетингові стратегії. Такий підхід надає широкі можливості для розвитку системи, зберігаючи при цьому базову стабільність і захищеність даних.

### 4.2 Створення контролерів-представлень застосунку

Частину коду, яка відповідає за обробку конкретного запиту, у Django називають представленням (аналог контролерів у моделі MVC) [11]. Зазвичай вони розміщуються у файлі views.py. Саме тут здійснюється взаємодія з даними, надісланими користувачем, а також з базою даних. Проте взаємодія з моделлю не є обов'язковою.

Програмна реалізація контролерів (views) базується на принципах максимальної продуктивності, надійності та захисту від зловживань з боку користувачів.

Для оптимізації взаємодії з базою даних та усунення неефективності, відомої як проблема «N+1 запитів», застосовуються методи select_related та prefetch_related.

Ці методи дозволяють ефективно витягувати пов'язані об'єкти (наприклад, інформацію про категорії товарів) за допомогою мінімальної кількості оптимізованих SQL-запитів.

Управління складним каталогом включає безпечне відображення параметрів сортування та динамічну генерацію фільтрів, що повністю унеможливлює будь-які спроби клієнтських маніпуляцій із запитами.

Всі операції, що призводять до зміни стану системи (такі як додавання або видалення товарів з кошика), суворо контролюються декоратором @require_POST.

Це забезпечує надійний захист від CSRF-атак та запобігає випадковим діям, спричиненим переходом за посиланням.

Введені користувачем дані (наприклад, цінові обмеження) проходять обов'язкову перевірку та очищення за допомогою спеціальних утиліт для безпечного оброблення числових значень.

Фінальним етапом бізнес-логіки є процес оформлення замовлення, який реалізований як атомарна транзакція. Система безпомилково переміщує товари з тимчасового кошика до незмінних рядків чека, надійно фіксуючи актуальну ціну кожної позиції. Після цього віртуальний кошик безпечно очищується, готуючи його до наступних покупок авторизованого клієнта.

```
/cart/add/{pk}/
@login_required  1 usage
@require_POST
def cart_add(request, pk):
    product = get_object_or_404(Product, pk=pk)
    if not product.in_stock:
        messages.warning(request, message: "Цей товар наразі недоступний.")
        return redirect( to: "product-detail", pk=pk)
    try:
        qty = max(1, int(request.POST.get( key: "quantity",  default: 1)))
    except (TypeError, ValueError):
        qty = 1
    cart = get_or_create_cart(request.user)
    item, created = CartItem.objects.get_or_create(cart=cart, product=product, defaults={"quantity": qty})
    if not created:
        item.quantity += qty
        item.save(update_fields=["quantity"])
    messages.success(request,  message: "Товар додано до кошика.")
    nxt = request.POST.get("next")
    if nxt:
        return redirect(nxt)
    return redirect("cart")
```

Рисунок 4.2 – Лістинг коду представлення додавання товару в кошик

Розроблена архітектура має дві основні сильні сторони: оптимізацію навантаження на сервер та інтуїтивно-зручний користувацький досвід.

Для забезпечення миттєвого зворотного зв'язку без дублювання інформації при оновленні сторінок, використано механізм одноразових сповіщень (Flash Messages), реалізований на базі сесій.

Автентифікація побудована за принципом «безшовного» доступу: система автоматично створює сесію одразу після реєстрації. Захищені маршрути (Protected

Routes), тим часом, перенаправляють неавторизованих користувачів, зберігаючи при цьому інформацію про кінцеву точку, до якої вони намагалися отримати доступ.

З боку оптимізації ресурсів, ключовим технічним рішенням стало застосування механізму «лінивого обчислення» (Lazy Evaluation) при роботі з об'єктами Django QuerySets. Цей підхід дозволяє формувати складні багаторівневі ланцюжки фільтрів для каталогу виключно в оперативній пам'яті сервера. Фактичний SQL-запит до бази даних виконується лише один раз, безпосередньо перед рендерингом фінального HTML-шаблону. Це призводить до критичного зниження кількості звернень до бази даних та забезпечує високу пропускну здатність платформи, навіть при значному масштабуванні асортименту.

### 4.3 Реєстрація моделей та створення контекстного процесору

Зображений код (рис. 4.3) покращує вбудовану панель адміністратора Django, перетворюючи її на інструмент для ефективного управління вебзастосунком меблевого магазину.

```python
@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ("name", "slug")
    prepopulated_fields = {"slug": ("name",)}
    search_fields = ("name",)


@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ("name", "category", "material", "color", "price", "in_stock")
    list_filter = ("category", "in_stock", "material", "color")
    search_fields = ("name", "description", "material", "color")
    autocomplete_fields = ("category",)


@admin.register(Cart)
class CartAdmin(admin.ModelAdmin):
    list_display = ("id", "user")
    inlines = [CartItemInline]
    search_fields = ("user__username",)
```

Рисунок 4.3 – Зареєстровані в адмін-панелі моделі бази даних

Роботу з асортиментом товарів значно спрощує те, що система автоматично генерує URL-адреси (завдяки prepopulated_fields) та забезпечує швидкий пошук категорій в режимі реального часу (autocomplete_fields). Це значно прискорює роботу менеджера, особливо при великій кількості товарів, запобігаючи повільному завантаженню сторінок.

Для керування продажами обрано рішення з використанням вбудованих таблиць (TabularInline). Це дозволяє переглядати та редагувати вміст кошика та деталі замовлень безпосередньо в картці клієнта або чека. Додатково, параметр extra = 0 покращує користувацький інтерфейс, прибираючи зайві порожні поля.

```python
from decimal import Decimal

from .models import Cart


def cart_summary(request):  1 usage
    zero = Decimal("0.00")
    ctx = {"cart_item_count": 0, "cart_subtotal": zero}
    if not request.user.is_authenticated:
        return ctx
    try:
        cart = request.user.cart
    except Cart.DoesNotExist:
        return ctx
    items = list(cart.items.select_related("product"))
    ctx["cart_item_count"] = sum(i.quantity for i in items)
    ctx["cart_subtotal"] = sum((i.line_total() for i in items), zero)
    return ctx
```

Рисунок 4.5 – Контекстний процесор кошику

Наведений програмний модуль (рис 4.4) виконує функції контекстного процесора (Context Processor) у фреймворку Django, мета якого полягає в глобальному впровадженні актуальних даних про стан кошика (кількість товарів та загальну суму) в усі HTML-шаблони вебзастосунку без необхідності їх повторення в кожному контролері окремо.

Алгоритм розроблено з підвищеною відмовостійкістю: він розпочинається з встановлення безпечних нульових значень та проводить перевірку стану авторизації користувача, після чого обробляє можливий виняток Cart.DoesNotExist,

що забезпечує надійне відображення інтерфейсу (наприклад, меню навігації) навіть для неавторизованих відвідувачів сайту.

З огляду на продуктивність, важливим рішенням є застосування методу select_related(«product») у поєднанні з перетворенням результату на список list(), що дає можливість миттєво завантажити всі деталі кошика в пам'ять сервера через один комплексний SQL-запит (JOIN), ефективно уникаючи поширеної проблеми надмірного навантаження бази даних «N+1».

Для досягнення ідеальної фінансової точності усі розрахунки підсумкової вартості здійснюються тільки за використанням спеціалізованого типу даних Decimal, що є строгим стандартом для розробки платформ електронної комерції і повністю уникає можливості виникнення апаратних помилок округлення чисел із рухомою комою під час формування чека.

### 4.4 Налаштування маршрутизації проєкту

Файл налаштувань маршрутів (urls.py) (рис. 4.4) слугує центральним координатором вебзастосунку, який перехоплює HTTP-запити від клієнта і точно їх направляє до відповідних серверних контролерів. Архітектура реалізована на основі семантичного (Clean URL) і зрозумілого дизайну.

Це свідчить про те, що адреси зрозумілі як для кінцевих користувачів, так і для пошукових систем, чітко демонструючи ієрархію онлайн-магазину: від головної сторінки до детального перегляду каталогу (/catalog/) та інформаційних розділів.

Основною технічною перевагою впровадженої системи є застосування типізованих динамічних параметрів, наприклад, для унікального ідентифікатора товару або для певної позиції у кошику. Цей механізм функціонує як перший рівень захисту вебзастосунку. До того, як запит потрапить до бізнес-логіки контролера, маршрутизатор Django автоматично перевіряє тип отриманих даних. Якщо система отримує текстовий рядок замість очікуваного числа (наприклад, /products/abc/),

диспетчер негайно відхилить запит з помилкою 404 (Not Found), ефективно захищаючи сервер від обробки невірних даних та помилок перетворення типів.

```
urlpatterns = [
    /
    path( route: "", views.home, name="home"),
    /catalog/
    path( route: "catalog/", views.catalog, name="catalog"),
    /products/{pk}/
    path( route: "products/<int:pk>/", views.product_detail, name="product-detail"),
    /cart/
    path( route: "cart/", views.cart_view, name="cart"),
    /cart/add/{pk}/
    path( route: "cart/add/<int:pk>/", views.cart_add, name="cart-add"),
    /cart/item/{item_id}/update/
    path( route: "cart/item/<int:item_id>/update/", views.cart_update, name="cart-update"),
    /cart/item/{item_id}/remove/
    path( route: "cart/item/<int:item_id>/remove/", views.cart_remove, name="cart-remove"),
    /cart/checkout/
    path( route: "cart/checkout/", views.checkout, name="checkout"),
    /orders/{pk}/thanks/
    path( route: "orders/<int:pk>/thanks/", views.order_thanks, name="order-thanks"),
    /about/
    path( route: "about/", views.about, name="about"),
    /contacts/
    path( route: "contacts/", views.contacts, name="contacts"),
    /signup/
    path( route: "signup/", views.signup, name="signup"),
    /profile/
    path( route: "profile/", views.profile, name="profile"),
]
```

Рисунок 4.5 – Лістинг коду urls.py

Абсолютно кожен маршрут у системі має свій унікальний ідентифікатор завдяки параметру name. Наприклад, name=«cart-add» або name=«checkout». Це рішення базується на принципі DRY – не повторюй себе. Воно дозволяє використовувати механізм зворотного вирішення URL. Завдяки цьому більше не використовуються жорстко закодовані посилання в HTML-шаблонах та коді контролерів.

Натомість система генерує посилання динамічно на основі їхніх імен. Це означає, що якщо буде вирішено змінити фізичну структуру адрес у майбутньому, наприклад, якщо перейменувати /catalog/ на /store/, потрібно буде змінити лише один рядок у файлі urls.py. І всі навігаційні елементи сайту будуть автоматично оновлені без будь-якого ризику виникнення «битих» посилань.

Усі ендпоінти точно поділені за бізнес-областями. Маршрути для публічного доступу, як-от вітрина та контакти, чітко виділені. Ті маршрути, які мають відношення до керування профілем, наприклад, реєстрація та профіль, також точно відокремлені.

А щодо персональних операцій з кошиком і замовленнями, то вони теж мають свої окремі маршрути. Саме ті маршрути, які безпосередньо впливають на стан бази даних, наприклад, видалення товару з кошика, мають свої окремі адреси. Це робить усе дуже зрозумілим і передбачуваним, а також підготовленим до подальшого розширення.

### 4.5 Розробка клієнтського інтерфейсу застосунку

Клієнтська частина вебзастосунку зроблена на основі вбудованого рушія Django Templates. Цей рушій використовує класичну модель серверного рендерингу, щоб миттєво віддавати готовий HTML-код і забезпечувати бездоганну SEO-оптимізацію нашої вітрини магазину.

```
{% extends "base.html" %}

{% block content %}
<h2>Реєстрація</h2>
<form method="post" class="auth-form">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Створити акаунт</button>
</form>
<p>Вже маєте акаунт? <a href="{% url 'login' %}">Увійти</a></p>
{% endblock %}
```

Рисунок 4.6 – Лістинг коду шаблону реєстрації користувача

```
{% extends "base.html" %}

{% block content %}
<section>
    <h2>Затишні меблі для вашого дому</h2>
    <p>Обирайте сучасні дивани, ліжка, шафи та столи за доступними цінами.</p>
</section>

<section>
    <h3>Популярні товари</h3>
    <div class="grid">
        {% for product in featured_products %}
            <article class="card">
                {% if product.image_url %}
                    <img src="{{ product.image_url }}" alt="{{ product.name }}" class="product-image">
                {% else %}
                    <div class="product-image product-image-placeholder">Немає фото</div>
                {% endif %}
                <h4>{{ product.name }}</h4>
                <p>{{ product.category.name }}</p>
                <p>{{ product.description|truncatewords:20 }}</p>
                <p class="price">{{ product.price }} грн</p>
                <a href="{% url 'product-detail' product.pk %}" class="details-link">Детальніше</a>
            </article>
        {% empty %}
            <p>Поки що немає товарів. Додайте їх через адмінку.</p>
        {% endfor %}
    </div>
</section>
{% endblock %}
```

Рисунок 4.7 – Лістинг коду шаблону домашньої сторінки застосунку

Архітектура візуальної частини застосунку організована за суворим модульним принципом. Вона спирається на патерн успадкування шаблонів. Основним ядром нашого застосунку є глобальний макет base.html. Цей макет містить усі наскрізні елементи інтерфейсу, наприклад навігаційне меню, підключення базових стилів і футер.

Функціональні сторінки застосунку логічно ізольовані у відповідних директоріях системи. Наприклад, папка registration містить усі процеси керування користувацьким доступом, наприклад реєстрацію, авторизацію і особистий профіль. Директорія store відповідає виключно за клієнтську бізнес-логіку нашої електронної комерції, тобто головну сторінку, деталізований каталог, кошик і сторінку успішного замовлення.

Таким чином, повністю виключено дублювання коду, спрощено майбутнє масштабування дизайну і гарантовано ідеальну візуальну узгодженість на всіх етапах взаємодії клієнта з платформою.

## 4.6 Реалізація тестування вебзастосунку

Програмна реалізація модульного тестування (Unit Testing) грунтується на вбудованому класі django.test.TestCase, який перед кожним запуском тестів автоматично генерує ізольовану віртуальну базу даних, що забезпечує повноцінну безпечну перевірку бізнес-логіки без ризику пошкодження чи витоку реальних даних інтернет-магазину.

```
def test_add_to_cart_business_logic(self):
    """Перевірка механіки додавання товару до кошика через POST-запит"""
    self.client.login(username='testuser', password='securepassword123')

    # Імітуємо натискання кнопки "Додати в кошик"
    response = self.client.post(
        reverse( viewname: 'cart-add', args=[self.product.pk]),
        data: {'quantity': 1}
    )

    # Перевіряємо, що після додавання відбувся редиректт (в кошик або назад)
    self.assertEqual(response.status_code, second: 302)

    # Головна перевірка: чи з'явився запис фізично в базі даних
    cart_exists = CartItem.objects.filter(
        product=self.product,
        cart__user=self.user
    ).exists()
    self.assertTrue(cart_exists)
```

Рисунок 4.8 – Лістинг коду тестування механіки додавання товару в кошик

Створений тестовий набір всебічно охоплює всі важливі рівні архітектури: на рівні бази даних перевіряється правильність активації тригерів моделей (наприклад, автоматичне створення поля slug для категорій) та забезпечується математика безпомилковості фінансових розрахунків методу line_total() завдяки застосуванню строго типу Decimal. Інтеграційну частину тестування реалізовано за допомогою об'єкта Client, який імітує поведінку справжнього браузера, та механізму reverse(), що динамічно визначає маршрути. Особливу увагу в тестах приділено аудитам безпеки: система автоматично перевіряє роботу декораторів @login_required, підтверджуючи, що контролери успішно блокують неавторизовані HTTP-запити (статус 302 Redirect), одночасно бездоганно обробляючи POST-запити від автентифікованих сеансів під час додавання товарів до кошика з подальшою фіксацією цих транзакцій у реляційній БД.

```
(.venv) D:\Python-course\django_projects\furniture_store>python manage.py test
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
Ran 5 tests in 5.705s

OK
Destroying test database for alias 'default'...
```

Рисунок 4.9 – Результати тестування

Наведений уривок консольного виводу (рис. 4.9) ілюструє успішний результат виконання автоматизованих модульних тестів за допомогою команди python manage.py test.

Система виявила та ефективно виконала створений набір із 5 тестів за 5.7 секунди. Фінальний маркер OK та наступне коректне очищення пам'яті шляхом видалення тестового середовища (Знищення тестової бази даних...) є явним підтвердженням стабільності написаного коду, відсутності синтаксичних конфліктів та повної працездатності основної бізнес-логіки вебзастосунку.

### 4.7 Демонстрація роботи вебзастосунку

Сторінка авторизації (рис. 4.10) надає можливість ввести дані для валідації користувача та увійти за допомогою відповідної кнопки. Також наявна альтернативна опція «Зареєструватися» для нових користувачів. Вгорі сторінки присутня панель навігації по сервісу, але ця функція доступна лише авторизованим користувачам.

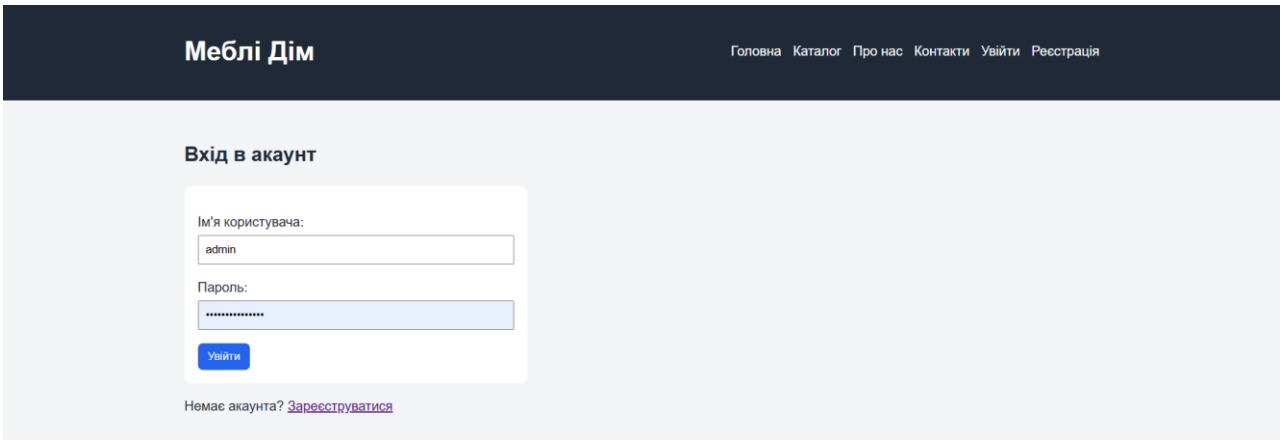


Рисунок 4.10 – Сторінка авторизації користувача

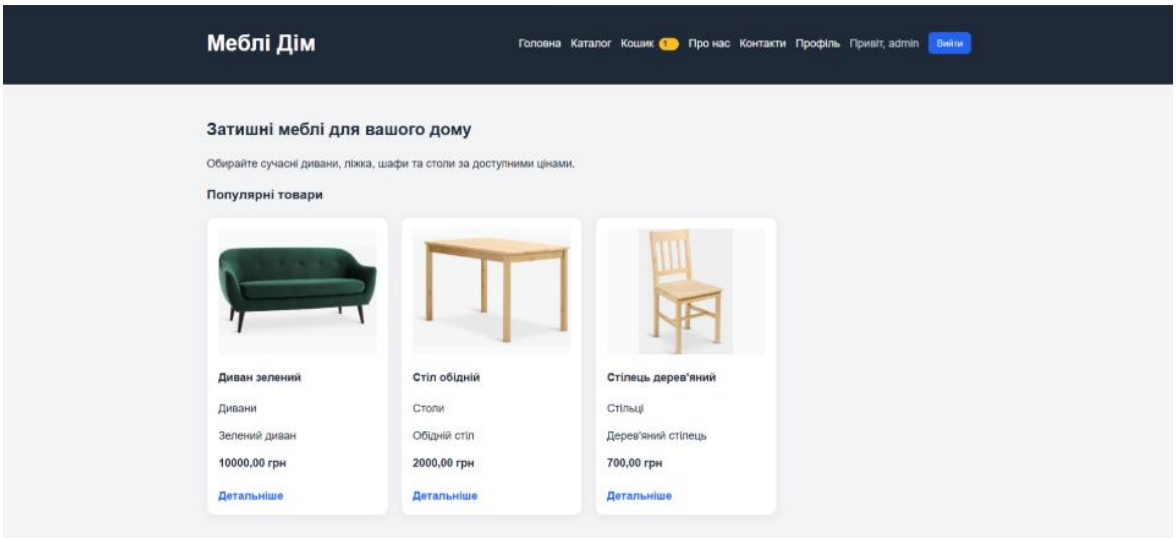


Рисунок 4.11 – Головна вітрина магазину

Головна вітрина вебзастосунку (рис. 4.11) демонструє перелік популярних серед споживачів товарів, з можливістю подивитись деталі кожної меблі, а також панель навігації, яка вже стає доступною для авторизованого користувача.

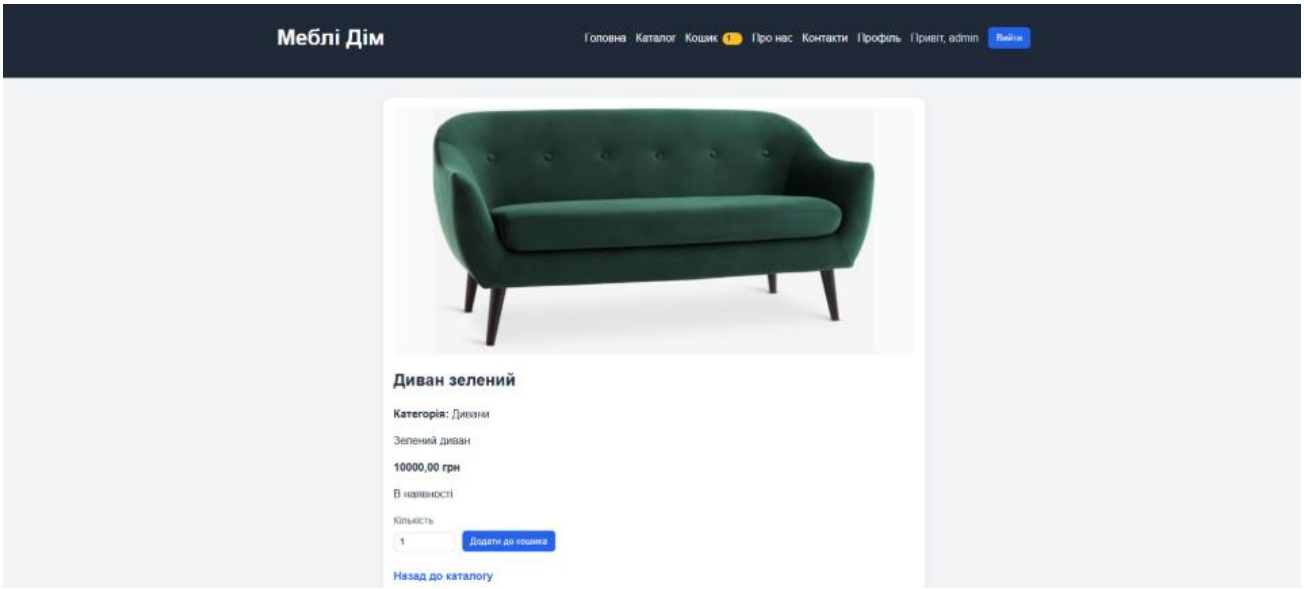


Рисунок 4.12 – Детальний опис товару

Сторінка детального опису товару (рис. 4.12) демонструє його назву, категорію, вербальний опис, ціну меблі, чи є вона наявною, кількість для формування майбутнього замовлення, опція додавання в кошика та повернення назад до каталогу.

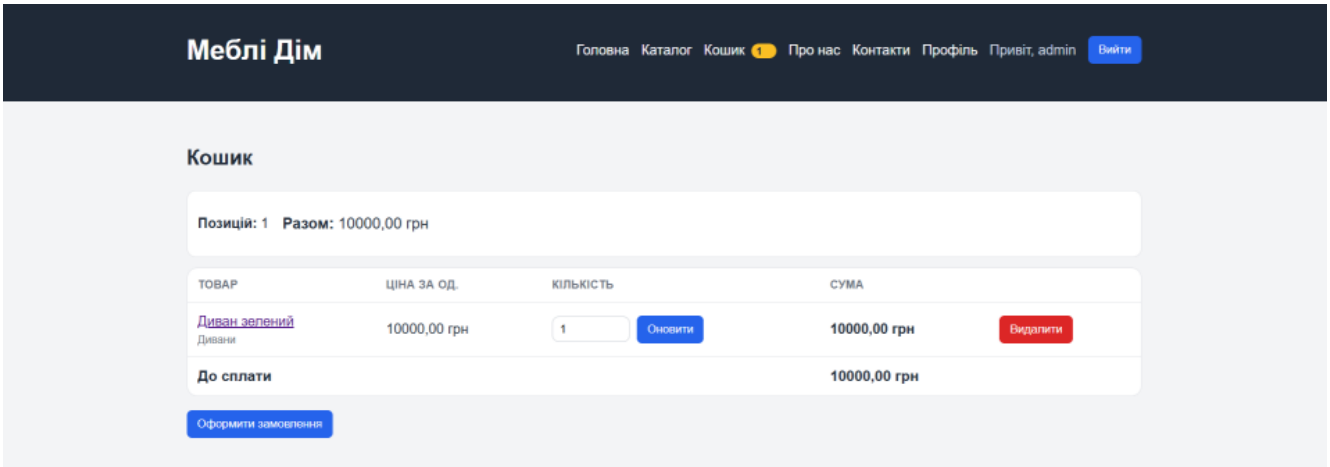


Рисунок 4.13 – Формування замовлення в кошику

Сторінка кошику (рис 4.13) реалізована у вигляді зведеної таблиці, що надає користувачу повний контроль над майбутнім замовленням.

Тут інтегровано функціонал для динамічного оновлення кількості товарів або його повного видалення з чека.

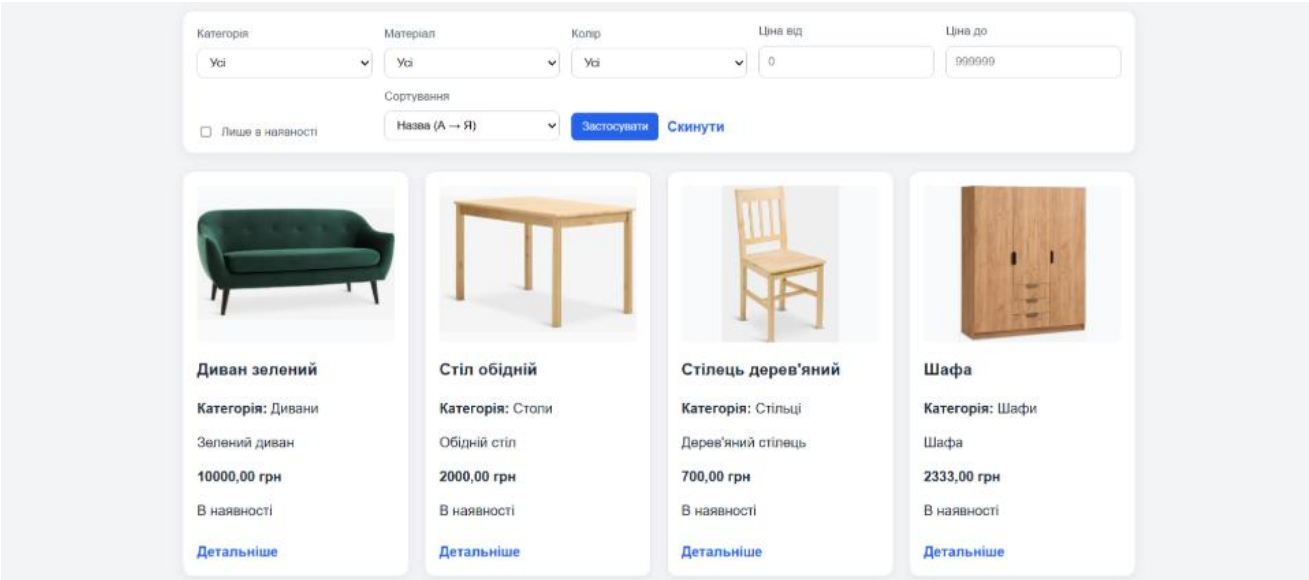


Рисунок 4.14 – Каталог меблів

Сторінка каталогу (рис. 4.14) оснащена інструментом фільтрації та сортування товарів за їх категорією, матеріалом, кольором тощо.

Нижче розташована адаптивна сітка меблів, яка чітко відображає результати пошуку відповідно до застосованих користувачем параметрів.

## Висновки до розділу 4

Підбиваючи підсумки етапу розробки та тестування програмного забезпечення, можна зауважити вдале створення стабільної, безпечної та масштабованої архітектури вебзастосунку меблевого магазину.

Використання Django дало змогу поєднати основні переваги серверного рендерингу (швидка завантажуваність і глибока SEO-індексація каталогу) з високою інтерактивністю користувацького інтерфейсу.

Проєктування реляційної бази даних за допомогою механізмів ORM забезпечило повну цілісність фінансових транзакцій та інтегрований захист від вразливостей.

Модульна система шаблонів, семантична маршрутизація і оптимізовані контролери забезпечили безпечний і зручний досвід користувача (UX).

Останнім етапом стало впровадження системи автоматизованого модульного тестування, що в ізольованому середовищі успішно підтвердила критичну бізнес-логіку платформи, підтвердивши її відмовостійкість, правильність роботи алгоритмів кошика та повну технічну готовність інтернет-магазину до етапу експлуатації.

# ВИСНОВКИ

У першому розділі кваліфікаційної роботи проведено аналіз предметної області та сучасні підходи до реалізації інтернет-магазину для меблевого бізнесу.

Здійснено порівняння провідних існуючих вебзастосунків (IKEA, JYSK, Wayfair тощо), з урахуванням їх архітектури, структури, функціональності та ефективності.

Визначено головні структурні та функціональні особливості об’єкта роботи (клієнт-серверна архітектура за патерном MVT, безпечне керування сесіями й автентифікацією користувачів, а також реалізація кошика та панелі адміністратора тощо).

У другій частині кваліфікаційної роботи був сформований теоретичний фундамент для розробки меблевого інтернет-магазину. Спроєктовано базові моделі архітектури (інформаційну, комунікаційну та інфраструктурну), а також визначено ключові механізми захисту персональних даних покупців під час здійснення онлайн-платежів.

На основі огляду фахових наукових джерел здійснено обґрунтований вибір технологічного стека, довівши доцільність використання зв'язки Python та Django у колаборації з базою даних PostgreSQL для забезпечення максимальної швидкості створення надійного продукту електронної комерації.

Підсумком розділу стало формування детальної специфікації вимог до програмного забезпечення. У цьому документі чітко виявлені межі та мету проєкту, базовий функціонал, а також перелік нефункціональних властивостей системи.

У третьому розділі кваліфікаційної роботи розроблено проєктні рішення, що забезпечать виконання структурних та функціональних особливостей програмного забезпечення, що забезпечується.

Реалізовано низку UML-діаграм, таких як: діаграма використаних, класів, взаємодії, діяльності, розгортання, що описують роботу вебзастосунку.

Описано характеристику класів та компонентів, архітектуру що використовуються при розробці програми.

Проілюстровано зовнішній вигляд форми застосунку, за допомогою mockup-макетів.

Підбиваючи підсумки етапу розробки та тестування програмного забезпечення, можна зауважити вдале створення стабільної, безпечної та масштабованої архітектури вебзастосунку меблевого магазину. Використання Django дало змогу поєднати основні переваги серверного рендерингу (швидка завантажуваність і глибока SEO-індексація каталогу) з високою інтерактивністю користувацького інтерфейсу. Проєктування реляційної бази даних за допомогою механізмів ORM забезпечило повну цілісність фінансових транзакцій та інтегрований захист від вразливостей. Модульна система шаблонів, семантична маршрутизація і оптимізовані контролери забезпечили безпечний і зручний досвід користувача (UX). Останнім етапом стало впровадження системи автоматизованого модульного тестування, що в ізольованому середовищі успішно підтвердила критичну бізнес-логіку платформи, підтвердивши її відмовостійкість, правильність роботи алгоритмів кошика та повну технічну готовність інтернет-магазину до етапу експлуатації.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кравченко О. В., Кушнір М. О. Особливості проєктування та розробки інтернет-магазинів на базі сучасних фреймворків. *Сучасні інформаційні системи*. 2023. Т. 7, № 1. С. 34–41.

2. Мельник О. А. Інтеграція систем електронної комерції з CRM та забезпечення безпеки даних. *Вісник Національного університету «Львівська політехніка»*. 2021. № 4. С. 112–118.

3. Коваленко І. В., Ткаченко В. М. Аналіз UI/UX рішень у сфері електронної комерції. *Сучасні інформаційні системи*. 2022. Т. 6, № 2. С. 45–52.

4. Офіційний сайт компанії IKEA. URL: https://www.ikea.com (дата звернення: 22.04.2026).

5. Офіційний сайт інтернет-магазину JYSK. URL: https://jysk.ua (дата звернення: 22.04.2026).

6. Офіційний сайт онлайн-маркетплейсу Wayfair. URL: https://www.wayfair.com (дата звернення: 22.04.2026).

7. Говорущенко Т. О., Осадча Т. В. Порівняльний аналіз вибраних фреймворків для створення веб-додатків. *Вісник Хмельницького національного університету. Технічні науки*. 2023. № 1(317). С. 156–161.

8. Django documentation. Django Software Foundation. URL: https://docs.djangoproject.com/ (Last accessed: 27.04.2026).

9. Django REST Framework documentation. Encode OSS. URL: https://www.django-rest-framework.org/ (Last accessed: 27.04.2026).

10. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/docs/ (Last accessed: 27.04.2026).

11. Tailwind CSS – Rapidly build modern websites without ever leaving your HTML. Tailwind Labs. URL: https://tailwindcss.com/docs (Last accessed: 27.04.2026).

12. Гнатюк С. О., Шевчук О. В. Моделювання програмного забезпечення: навч. посібник. – Київ: КНУ імені Тараса Шевченка, 2017. 128 с.

# ДОДАТОК А

## Лістинг коду представлень (views) вебзастосунку меблевого магазину

```
from decimal import Decimal, InvalidOperation

from django.contrib import messages
from django.contrib.auth import login
from django.contrib.auth.decorators import login_required
from django.shortcuts import get_object_or_404, redirect, render
from django.views.decorators.http import require_POST

from .forms import SignUpForm
from .models import Cart, CartItem, Category, Order, OrderLine, Product


def _parse_decimal(value):
    if value is None:
        return None
    if isinstance(value, str) and not value.strip():
        return None
    try:
        return Decimal(str(value).strip().replace(",", "."))
    except (InvalidOperation, ValueError):
        return None


def get_or_create_cart(user):
    cart, _ = Cart.objects.get_or_create(user=user)
    return cart


@login_required
def home(request):
    featured_products =
Product.objects.filter(in_stock=True).select_related("category")[:3]
    return render(
        request,
        "store/home.html",
        {"featured_products": featured_products},
    )


@login_required
def catalog(request):
    qs = Product.objects.select_related("category")

    category_id = request.GET.get("category")
    selected_category = None
    if category_id:
        try:
            selected_category = int(category_id)
        except (TypeError, ValueError):
            selected_category = None
        if selected_category is not None:
```

```
        qs = qs.filter(category_id=selected_category)

material = (request.GET.get("material") or "").strip()
if material:
    qs = qs.filter(material__iexact=material)

color = (request.GET.get("color") or "").strip()
if color:
    qs = qs.filter(color__iexact=color)

pmin = _parse_decimal(request.GET.get("price_min"))
pmax = _parse_decimal(request.GET.get("price_max"))
if pmin is not None:
    qs = qs.filter(price__gte=pmin)
if pmax is not None:
    qs = qs.filter(price__lte=pmax)

if request.GET.get("in_stock") == "1":
    qs = qs.filter(in_stock=True)

sort = (request.GET.get("sort") or "name_asc").strip()
ordering_map = {
    "name_asc": ("name",),
    "name_desc": ("-name",),
    "price_asc": ("price", "name"),
    "price_desc": ("-price", "name"),
    "category_asc": ("category__name", "name"),
    "material_asc": ("material", "name"),
    "color_asc": ("color", "name"),
}
qs = qs.order_by(*ordering_map.get(sort, ("name",)))

materials = (
    Product.objects.exclude(material="")
    .values_list("material", flat=True)
    .distinct()
    .order_by("material")
)
colors = (
    Product.objects.exclude(color="")
    .values_list("color", flat=True)
    .distinct()
    .order_by("color")
)

return render(
    request,
    "store/catalog.html",
    {
        "products": qs,
        "categories": Category.objects.all(),
        "materials": materials,
        "colors": colors,
        "current_filters": {
            "category": category_id or "",
            "material": material,
            "color": color,
```

```
                "price_min": request.GET.get("price_min") or "",
                "price_max": request.GET.get("price_max") or "",
                "in_stock": request.GET.get("in_stock") == "1",
                "sort": sort,
            },
            "selected_category": selected_category,
        },
    )


@login_required
def product_detail(request, pk):
    product = get_object_or_404(Product.objects.select_related("category"),
pk=pk)
    return render(request, "store/product_detail.html", {"product":
product})


@login_required
def about(request):
    return render(request, "store/about.html")


@login_required
def contacts(request):
    return render(request, "store/contacts.html")


def signup(request):
    if request.method == "POST":
        form = SignUpForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect("home")
    else:
        form = SignUpForm()
    return render(request, "registration/signup.html", {"form": form})


@login_required
def profile(request):
    return render(request, "registration/profile.html")


@login_required
@require_POST
def cart_add(request, pk):
    product = get_object_or_404(Product, pk=pk)
    if not product.in_stock:
        messages.warning(request, "Цей товар наразі недоступний.")
        return redirect("product-detail", pk=pk)
    try:
        qty = max(1, int(request.POST.get("quantity", 1)))
    except (TypeError, ValueError):
        qty = 1
    cart = get_or_create_cart(request.user)
```

```
    item, created = CartItem.objects.get_or_create(cart=cart,
product=product, defaults={"quantity": qty})
    if not created:
        item.quantity += qty
        item.save(update_fields=["quantity"])
    messages.success(request, "Товар додано до кошика.")
    nxt = request.POST.get("next")
    if nxt:
        return redirect(nxt)
    return redirect("cart")


@login_required
def cart_view(request):
    cart = get_or_create_cart(request.user)
    items = cart.items.select_related("product", "product__category")
    subtotal = sum((line.line_total() for line in items), Decimal("0.00"))
    return render(
        request,
        "store/cart.html",
        {"cart": cart, "items": items, "subtotal": subtotal},
    )


@login_required
@require_POST
def cart_update(request, item_id):
    cart = get_or_create_cart(request.user)
    item = get_object_or_404(CartItem, pk=item_id, cart=cart)
    try:
        qty = int(request.POST.get("quantity", 1))
    except (TypeError, ValueError):
        qty = 1
    if qty < 1:
        item.delete()
        messages.info(request, "Позицію видалено з кошика.")
    else:
        item.quantity = qty
        item.save(update_fields=["quantity"])
        messages.success(request, "Кількість оновлено.")
    return redirect("cart")


@login_required
@require_POST
def cart_remove(request, item_id):
    cart = get_or_create_cart(request.user)
    CartItem.objects.filter(pk=item_id, cart=cart).delete()
    messages.info(request, "Позицію видалено.")
    return redirect("cart")


@login_required
@require_POST
def checkout(request):
    cart = get_or_create_cart(request.user)
    items = list(cart.items.select_related("product"))
```

```
    if not items:
        messages.warning(request, "Кошик порожній — немає що оформлювати.")
        return redirect("cart")
    total = sum((line.line_total() for line in items), Decimal("0.00"))
    order = Order.objects.create(user=request.user, total=total)
    for line in items:
        OrderLine.objects.create(
            order=order,
            product=line.product,
            quantity=line.quantity,
            unit_price=line.product.price,
        )
    cart.items.all().delete()
    messages.success(request, f"Замовлення №{order.pk} успішно
сформовано.")
    return redirect("order-thanks", pk=order.pk)


@login_required
def order_thanks(request, pk):
    order =
get_object_or_404(Order.objects.prefetch_related("lines__product"), pk=pk,
user=request.user)
    return render(request, "store/order_thanks.html", {"order": order})
```

# ДОДАТОК Б

## Лістинг коду моделей бази даних вебзастосунку меблевого застосунку

```
from django.conf import settings
from django.db import models
from django.utils.text import slugify


class Category(models.Model):
    name = models.CharField("Назва", max_length=120)
    slug = models.SlugField(max_length=140, unique=True)

    class Meta:
        verbose_name = "Категорія"
        verbose_name_plural = "Категорії"
        ordering = ("name",)

    def __str__(self) -> str:
        return self.name

    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = (slugify(self.name) or "category")[:140]
        super().save(*args, **kwargs)


class Product(models.Model):
    name = models.CharField("Назва", max_length=120)
    category = models.ForeignKey(
        Category,
        on_delete=models.PROTECT,
        related_name="products",
        verbose_name="Категорія",
    )
    description = models.TextField("Опис")
    price = models.DecimalField("Ціна, грн", max_digits=10,
decimal_places=2)
    image_url = models.URLField("Посилання на фото", blank=True)
    in_stock = models.BooleanField("В наявності", default=True)
    material = models.CharField("Матеріал", max_length=80, blank=True)
    color = models.CharField("Колір", max_length=60, blank=True)

    class Meta:
        ordering = ("name",)

    def __str__(self) -> str:
        return self.name


class Cart(models.Model):
    user = models.OneToOneField(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name="cart",
    )
```

```
    class Meta:
        verbose_name = "Кошик"
        verbose_name_plural = "Кошики"


class CartItem(models.Model):
    cart = models.ForeignKey(Cart, on_delete=models.CASCADE,
related_name="items")
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)

    class Meta:
        constraints = [
            models.UniqueConstraint(fields=["cart", "product"],
name="unique_cart_product"),
        ]

    def line_total(self):
        return self.product.price * self.quantity


class Order(models.Model):
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name="orders",
    )
    created_at = models.DateTimeField(auto_now_add=True)
    total = models.DecimalField(max_digits=12, decimal_places=2, default=0)

    class Meta:
        ordering = ("-created_at",)


class OrderLine(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE,
related_name="lines")
    product = models.ForeignKey(Product, on_delete=models.PROTECT)
    quantity = models.PositiveIntegerField()
    unit_price = models.DecimalField(max_digits=10, decimal_places=2)

    def line_total(self):
        return self.unit_price * self.quantity
```