МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА АГРЕГАТОРУ ТОВАРНОЇ ІНФОРМАЦІЇ З ІНТЕРНЕТ МАГАЗИНІВ»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.С. Литвинов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Литвинову Микиті Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка агрегатору товарної інформації з Інтернет магазинів

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка агрегатору.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	05.02.2024	
3.	Обробка методичних та теоретичних	01.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	05.04.2024	
5.	Розробка третього розділу.	13.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2024	

Студент _____ М.С. Литвинов
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка агрегатору товарної інформації з Інтернет магазинів»: 76 с., 23 рис., 14 джерел, 2 додатка.

АГРЕГАТОР, БАЗА ДАНИХ, CSS, DJANGO, HTML, JAVASCRIPT, POSTGRESQL, PYTHON, SQL.

Об’єкт дослідження – процес розробки агрегатору товарної інформації.

Мета роботи: розробити мінімалістичне функціональне програмне забезпечення для агрегації товарів з використанням фреймворку Django.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання та тестування програмного забезпечення.

Результати та їх новизна: було розроблено додаток, що дозволяє користувачам проглядати товари, що були додані магазинами, і магазинам автоматично додавати товари до системи.

Рекомендації щодо використання результатів роботи: розроблену систему агрегації товарів, з мінімальним розширенням функціональності, можна використовувати у комерційній діяльності.

Значимість роботи та висновки: дослідження виявило проблеми, з якими можуть зіткнутися розробники при створенні подібних систем. Це допоможе розробникам у розробці стратегій створення програмного продуктів.

SUMMARY

Bachelor’s qualifying paper “Development of an Aggregator of Product Information from Online Stores”: 76 pages, 23 figures, 14 references, 2 supplements.

AGGREGATOR, DATABASE, CSS, DJANGO, HTML, JAVASCRIPT, POSTGRESQL, PYTHON, SQL.

The object of the study is the process of developing a product information aggregator.

The aim of the study is to develop a minimalist functional software for product aggregation using the Django framework.

The methods of research are collecting and analyzing software requirements, methods of modeling, designing, constructing and testing software.

An application was developed that allows users to view products added by stores and allows stores to automatically add products to the system.

The developed product aggregation system can be used in commercial activities, with minimal expansion of functionality.

The study revealed the problems that developers may face when creating such systems. This will help developers in developing strategies for creating software products.

ЗМІСТ

ВСТУП

В умовах стрімкого розвитку інформаційних технологій та інтернет-торгівлі зростає потреба у зручних та ефективних інструментах для збору та обробки інформації про товари з різних джерел. Агрегатори товарної інформації, що дозволяють користувачам швидко знаходити та порівнювати продукти за різними характеристиками та цінами, стають невід'ємною частиною сучасного електронного комерційного середовища.

Агрегатор товарної інформації повинен витримувати велику кількість користувачів та одночасне оновлення бази великою кількістю товарів з різних магазинів. Для досягнення такої витривалості потрібно використовувати перевірені та надійні інструменти. Вибір мови програмування потрібно робити беручи до уваги здібності команди розробників. Мова програмування Python ідеально підходить для швидкої та надійної розробки системи малою командою. Одним з найпопулярніших фреймворків для веброзробки є Django, що надає готові рішення для роботи з базою даних та створення інтерфейсів для користувачів.

Метою даної роботи є створення мінімалістичного агрегатора товарної інформації з використанням фреймворку Django, JQuery для клієнтського коду та бази даних PostgreSQL. Для досягнення цієї мети потрібно зробити:

– проаналізувати вимоги, предметну область, проєкти конкурентів;
– спроєктувати базу даних та структуру програми;
– розробити програму;
– провести тестування продукту.

Об'єкт дослідження – процес розробки агрегатору товарної інформації.

Предмет дослідження – взаємодія між мовою Python та SQL та можливі підводні камені у реалізації цієї взаємодії.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз вимог

У цьому розділі розглядаються функціональні та технологічні вимоги до розроблюваної системи.

1.1.1 Основні вимоги

Метою проєкту є надання можливості відвідувачам проглядати товари з добіркою магазинів, що продають цей товар та мати можливість переходити на сторінку товару обраного магазина. Також проєкт має бути націленим на роботу з онлайн магазинами, задля розміщення їх пропозицій на платформі [1].

До функціональних вимог належать:

– каталогізація товарів: товари мають бути розміщені по відповідних категоріях;
– характеристика товарів: товари повинні мати відповідний опис та характеристики;
– пошук товарів: потрібна реалізація можливості пошуку товарів;
– порівняння товарів: потрібна можливість порівняння товарів для полегшення взаємодії користувачів з системою;
– додавання товарів: магазини повинні мати можливість заносити товари до системи.

Відповідно до зазначених вимог передбачена реалізація основних сторінок [2]:

– головна сторінка: головним змістом може бути як добірка рекламних банерів сайту, так каталог наявних категорій;

- сторінка виводу вмісту категорій: категорії можуть бути мати або товари, або підкатегорії;
- сторінка товару: в ній знаходиться детальна інформація про товар та список магазинів, що продають цей товар;
- сторінка профілю: магазинам треба якимось чином керувати своєю інформацією у системі, також ця сторінка знадобиться звичайним відвідувачам при подальшому розвитку системи;
- сторінка порівняння товарів: на сторінці будуть виводитися обрані товари, та їх характеристики для порівняння.

1.1.2 Аналіз існуючих систем

На ринку товарних агрегаторів України зараз домінують три найбільші сайти: E-Katalog [3], Hotline [4] та Price.ua [5]. Всі вони працюють вже більше ніж 30 років, тому аналіз їх устрою надасть важливу інформацію по тому, як має виглядати успішний товарний агрегатор. Їх устрій частково може розглядатися спільно, бо системи мають багато чого спільного.

Усі системи мають великий список категорій, від важкої побутової техніки до дитячих товарів.

Категорії з товарами розділені на низку збірних категорій. Різниця між Price.ua та іншими сайтами полягає у тому, що у інших сайтів максимальний рівень розбиття категорій дорівнює трьом, коли у Price.ua це значення варіюється, що створює неприємне відображення при виборі категорії.

На сторінці товарів категорій всі сайти мають приблизно однаковий зміст: одну чверть сторінки займає панель з фільтрами, а інший простір займає список з товарів.

Панель порівняння є в основному таблицею товарів, у яких у відповідному порядку вказані характеристики товарів.

Пошук розташован вгорі у хедері, і має функцію підказок при пошуку.

1.2 Аналіз інструментів

1.2.1 Мова програмування Python

Вибір даної мови програмування є резонним з декількох причин:

– наявність відповідних до потреб даного проєкту фреймворків;
– легкість написання коду, що буде корисно для малої команди розробників;
– великий вибір готових рішень для типових завдань розробки;
– простота інтеграції як вебінтерфейсів так і баз даних.

1.2.2 Фреймворк Django

Вибір фреймворку Django для розробки серверної частини проєкту є резонним з декількох причин [6]:

– легкість інтеграції баз даних;
– наявні готові рішення для роботи з шаблонами, адресації, обробки форм;
– висока швидкість розробки системи;
– наявні плагіни для розширення і до того великого функціоналу системи.

1.2.3 Об’єктно-реляційна система керування базами даних PostgreSQL

Використання саме реляційних СКБД було обрано через те, що модель бази даних, ще на моменті опрацювання вимог, виглядає занадто складною для

нереляційного рішення, і вкрай вірогідно буде мати багато зв’язків між моделями.

PostgreSQL [7] була обране як СКБД через:

– значну популярність цієї системи [8];
– наявність значного функціоналу, що покращить роботу системи в цілому;
– відмінну інтеграцію з Django.

1.2.4 Бібліотека JQuery

Для фронтенду буде використовуватися мова JavaScript, але для полегшення її використання додатково буде використовуватися бібліотека JQuery [9]. Це надасть можливості легше створювати AJAX запити та маніпулювати елементами розмітки HTML.

Хоча ця бібліотека і вважається застарілою, але її функціонал все ще здатен полегшити розробку, а вік бібліотеки лише вказує на велику екосистему додатків до бібліотеки, що були розроблені за цей час.

1.2.5 Керування версіями та спільна розробка з Git

Для полегшення розробки та контролю збереження версій проєкту буде використовуватися система контролю версій Git [10]. Це допоможе контролювати зміни у проєкті та захистить від можливої втрати прогресу. Також Git надасть можливість іншим розробникам безпечно підключитися до розробки.

Огляд обраних технологій формує технологічну основу для подальшого проєктування та розробки торгової онлайн-платформи, забезпечуючи необхідну продуктивність, безпеку та масштабованість.

1.3 Висновки до розділу 1

У першому розділі були детально розглянуті вимоги до системи, інструменти для розробки системи та основну структуру сайту. Також був проведений аналіз проєктів – конкурентів. В результаті було отримано розуміння того, що треба зробити та яким образом.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Формування технічного завдання

Перед початком реалізації проєкту потрібно розробити структуру системи. Це включає в себе:

– розробити моделі та схеми реалізації бази даних;
– спланувати структуру сайту;
– спланувати взаємодію користувача з системою.

2.2 Проєктування бази даних

Найголовнішою частиною системи є база даних. Тому найперше, що треба робити при розробці будь-якої інформаційної системи – це ретельно розробити базу даних.

Для того щоб розробити таблиці бази даних, треба виокремити окремі сутності, дані яких будуть максимально незалежними один від одного.

Згідно вимог, база даних має включати в себе такі сутності:

– категорія: повинна містити назву категорії та спосіб зв’язки з іншими категоріями, таким чином створюючи деревовидну структуру каталогу;
– товар: має містити базову інформацію (назву, опис, бренд), код товару та його зображення;
– бренд: один бренд може мати велику кількість товарів, тому його можно віднести до окремої сутності, що має назву;
– магазин: сервіс агрегації товарної інформації працює з великою кількістю магазинів, тому їх треба віднести до окремої сутності, що повинна мати назву, посилання на магазин, зображення логотипу (для

візуального виділення) та мати можливість завантажити у систему прайс лист з товарами;

- товар магазину: через те, що магазини можуть продавати один і той самий товар, з’явилася необхідність виділити товари один від одного, через що сутність товару буде збірною для сутностей товарів магазинів, тобто сутність товару магазину вже буде містити ціну товару, назву надану магазином, відсоток знижки на товар, посилання на товар в магазині та додатково може мати показник наявності;
- характеристика товару: їх буде багато, і кожна категорія буде мати власні характеристики, тому через спробу уніфікувати створення товарів, їх можна виділити в окрему сутність, що містить назву характеристики;
- значення характеристики товару: характеристики товарів багато, але ще більше їх значень, кожен товар може мати власні значення, тому їх теж можна виділити в окрему сутність, що містить у собі значення характеристики;
- користувач: користувач є тим, хто взаємодіє з системою, а також власник магазину має якось керувати інформацією магазину в системі, тому користувач має мати три рівні доступу (звичайний користувач, представник магазину та адміністратор), пароль, пошту та username.

Додатково ще можна виділити сутності історії товару, що уособлює у собі інформацію про товар, яку періодично збирає система та заходячи на перед – сутність заявки на підключення магазину, бо магазини мають мати можливість підключитися до системи. Та все ж метою роботи є створення мінімального агрегатору.

Для доступу до сторінок товарів та категорій потрібен зрозумілий ідентифікатор. Для цього до таблиць продукту та категорії треба додати поле slug, у якому буде зберігатися, доступна до використання в url рядку, назва сутностей.

Діаграми в подальшому будуть створюватися сервісом draw.io [11].

Врахувавши все вищеперераховане, було розроблено ER-діаграму бази даних (рис. 2.1).

ER-діаграма зображує сутності бази та їх відносини.

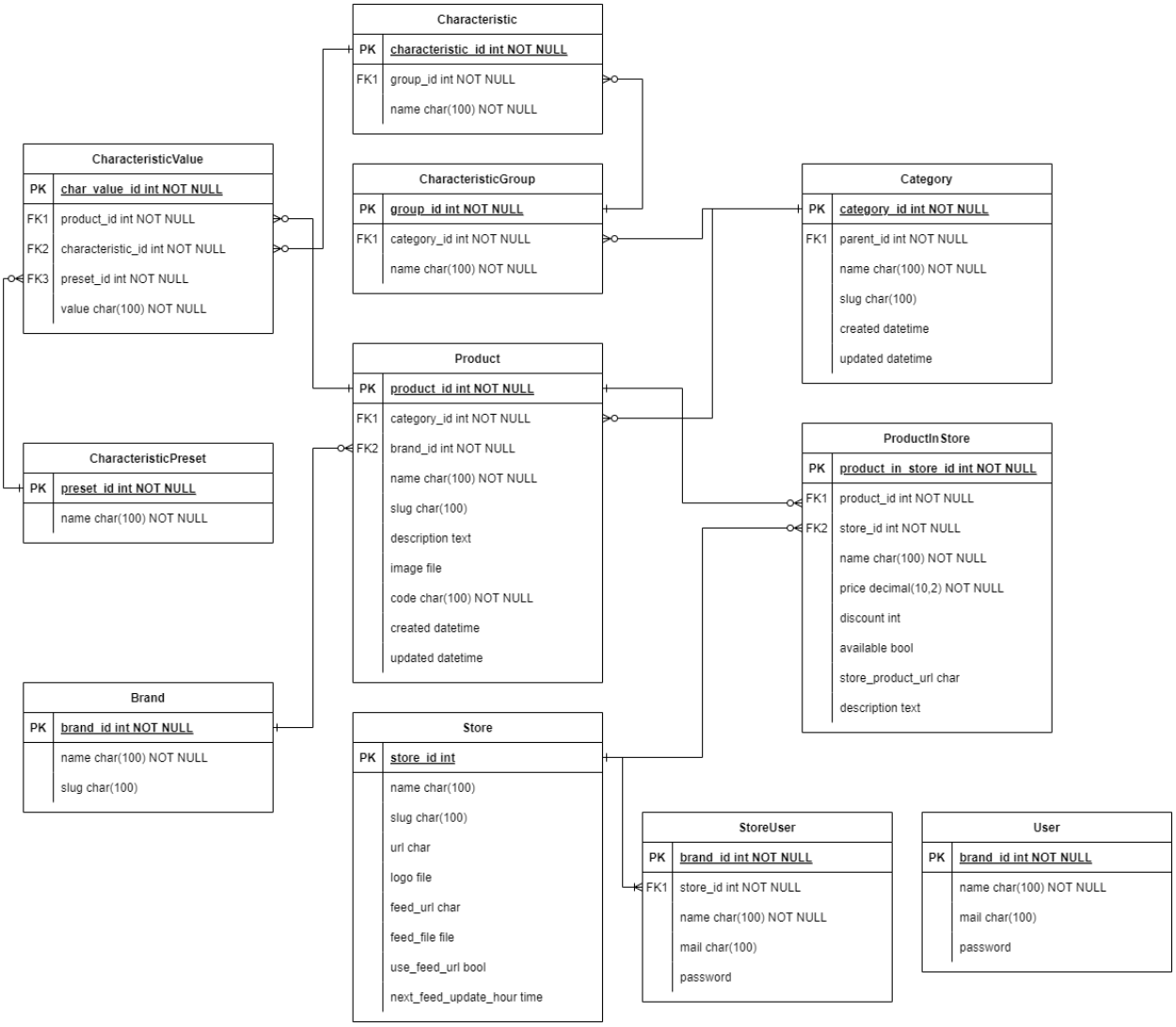

Рисунок 2.1 – ER діаграма бази даних

2.3 Проєктування сайту

Основні потреби від сайту були обговорені у першому розділі. У цьому розділі буде розроблено кінцеву структуру сайту та надано макети (мокапи) основних сторінок сайту.

2.3.1 Структура сайту

Для легкості навігації на сайті, всі основні посилання винесені у футер і хедер, які є однаковими для кожної сторінки сайту.

Весь контент сторінки знаходиться у головному елементі сторінки: тегу <main>.

Хедер є прикріпленим до верху вікна браузера, для постійного доступу до нього користувачем. Хедер наповнюють:

- лого сайту, що працює як посилання на головну сторінку;
- кнопку обрання каталогу, що відкриває спадне меню з категоріями, яке при наведенні на категорію відкриває нове спадне меню з підкатегоріями цієї категорії;
- вікно пошуку, що має спадне меню, що буде використовуватися для функції підказок;
- перемикач мови (функціонал буде опущений, бо це виходить за межі мети роботи);
- кнопка для переходу до порівняння товарів;
- кнопка профілю користувача.

Футер має на меті вміщати у собі корисні посилання, такі як посилання на соціальні мережі проєкту, довідкова інформація та форму заявки на підключення сайту.

Логіка розстановки контенту в основному елементі така: адаптивний контейнер з визначеними межами розбитий на 4 рівні частини. ¼ зліва відведена на колонку другорядного контенту, таких як фільтри, швидкі посилання, тощо.

Інші ¾ виділені на головну інформацію: добірка товарів, чи список категорій для перегляду.

Виходячи з цього, картки товарів у категоріях займають рівно ¼ сторінки, тобто 3 картки товарів на ряд.

2.3.2 Візуальне проєктування сторінок

Для розробки мокапів використовувався додаток Figma. На були розроблені мокапи основних сторінок сайту.

Головна сторінка – тут будуть розміщені рекламні банера та різні промо акції, пов’язані з проєктом, що швидче за все не буде реалізовано на кінцевому етапі розробки, бо на створення контенту треба допомога маркетологів та сам проєкт має якийсь час пропрацювати, щоб накопичити достатню кількість контенту для сторінки (див. рис. 2.2).

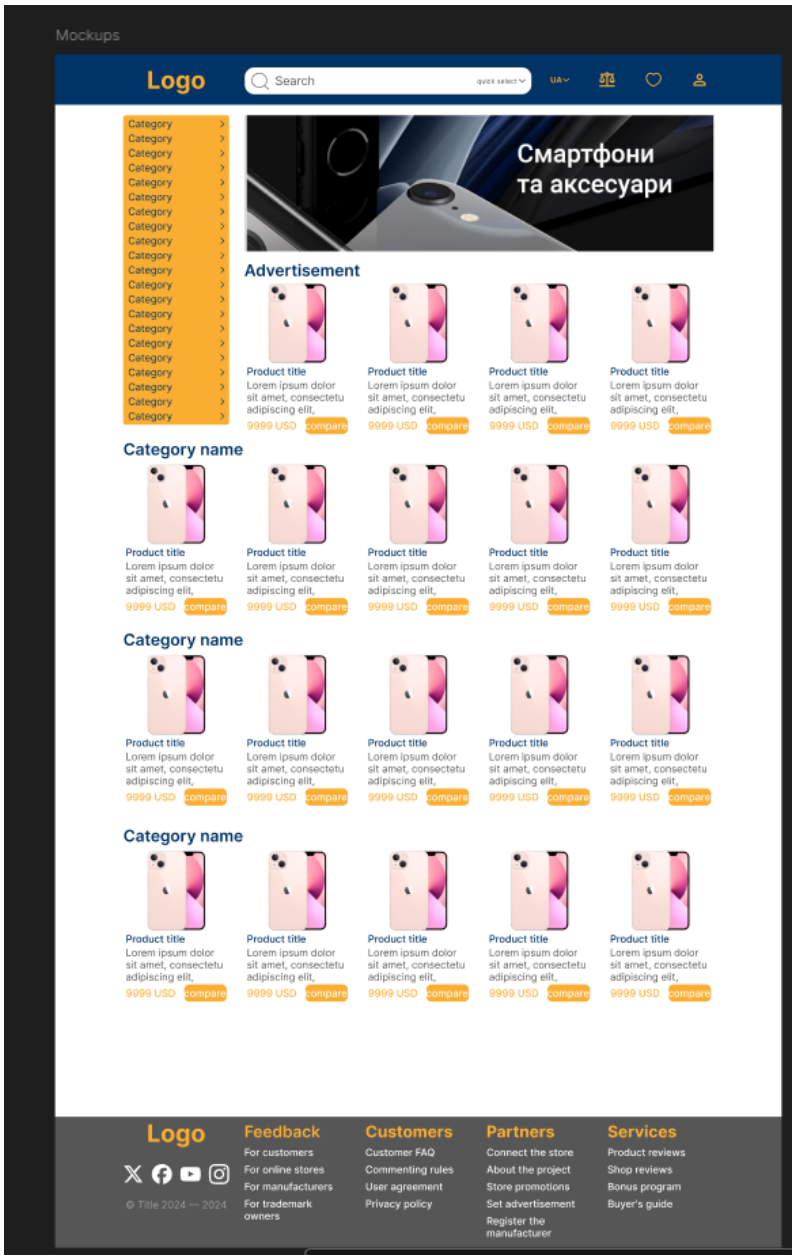

Рисунок 2.2 – Макет головної сторінки сайту

Сторінка категорії – тут будуть виводитись товари категорій, чи дерево підкатегорій, відповідно до позиції категорії у системі (див. рис. 2.3).

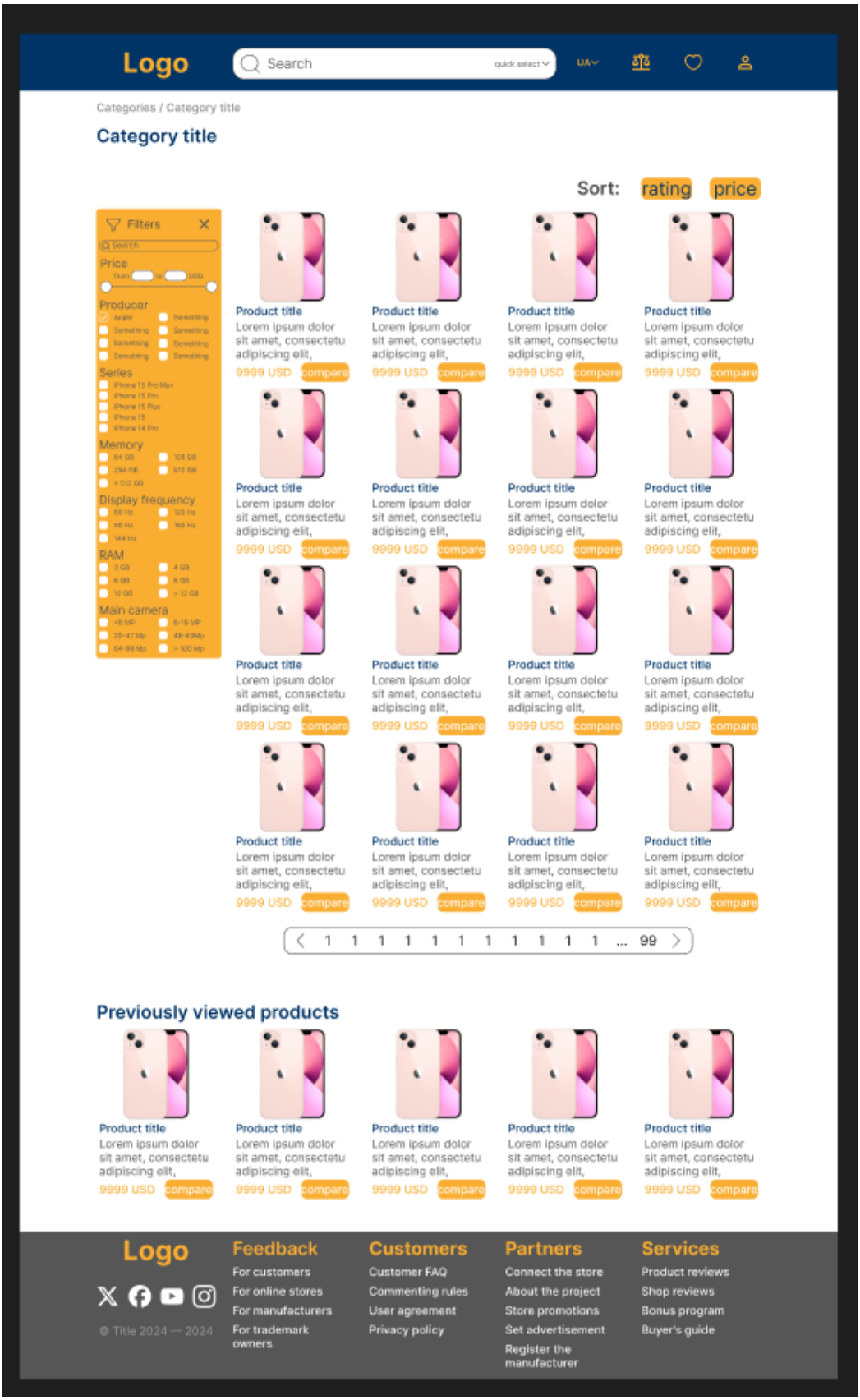

Рисунок 2.3 – Макет сторінки категорії товарів

Сторінка продукту – тут буде знаходитись опис товару, вкладки характеристик товару та магазинів, що продають даний товар (див. рис. 2.4).

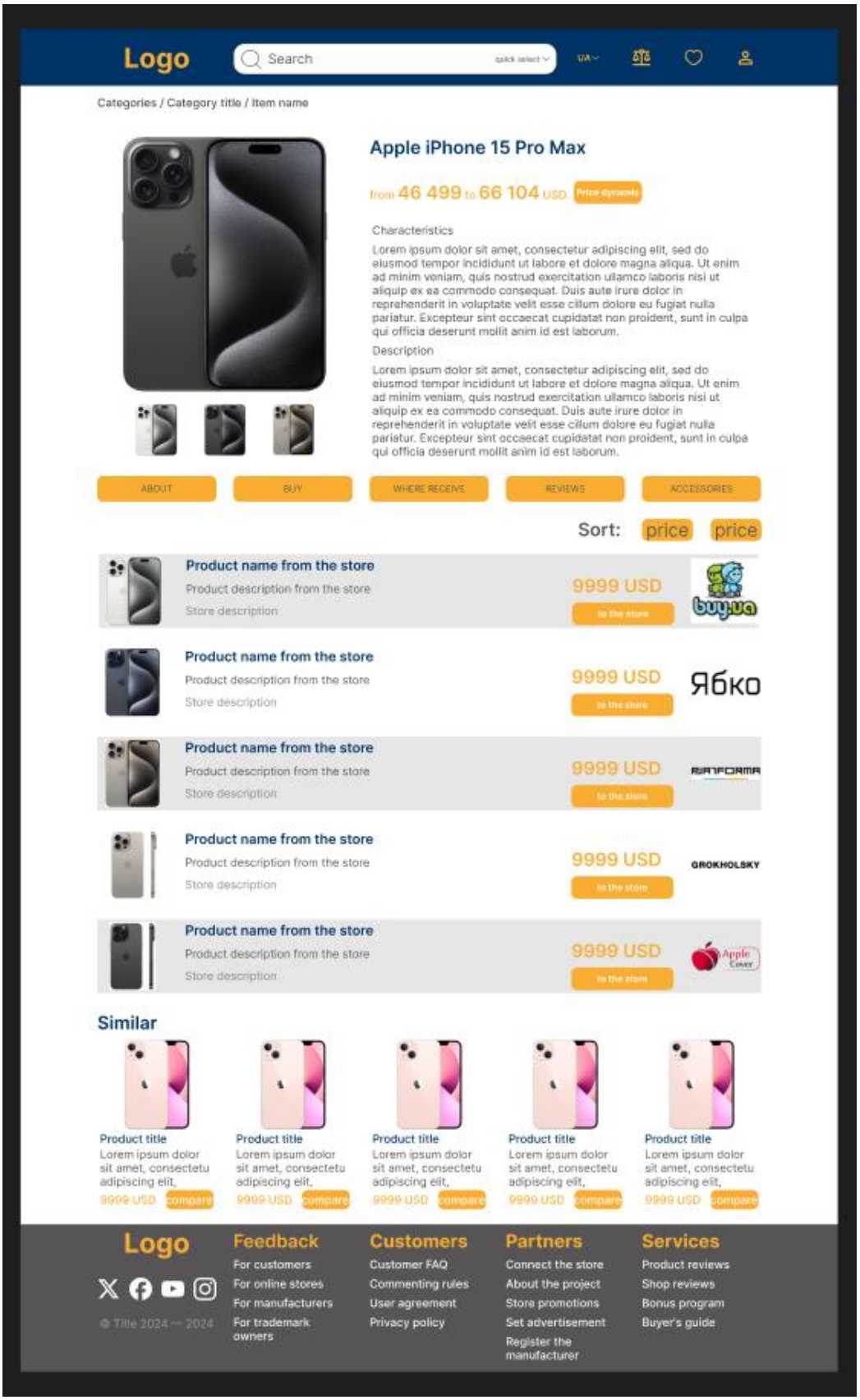

Рисунок 2.4 – Макет сторінки товару

Макети створювалися на самому початку розробки сайту, тому мають лише приблизний вміст відповідних сторінок. Це пов’язано з технічними можливостями, можливостями розробника та можливими змінами планів під час розробки системи.

2.4 Взаємодія користувача з сайтом

2.4.1 Види користувачів

Як вже було зазначено у першому розділі, у системі мають бути багато магазинів, що у свою чергу створює необхідність у наявності користувача, що буде мати можливість керувати магазином зі сторони інтернет магазина-партнера. У пункті проєктування бази даних цей користувач вже був окреслений, як окрема сутність.

Через перенасиченість інформацією у системі, потрібно буде багато персоналу для підтримки роботи сайту. Буде нерозумно надавати кожному адміністратору системи повні права. Тому потрібен адміністратор з меншими повноваженнями. Так буде мати сенс створити адміністратора та суперюзера (головного адміністратора).

2.4.2 Діаграма прецедентів

Діаграма прецедентів дозволяє легко продемонструвати взаємодію користувачів з системою. Це дозволяє зрозуміти, які дії повинні мати можливість виконувати різні користувачі.

Діаграма складається з акторів (користувачів) та прецедентів (їх дій). Також була показана наслідуваність дій даних користувачів.

На рисунку 2.5 зображено діаграму прецедентів для користувачів системи, відповідно до результатів минулої підкатегорії.

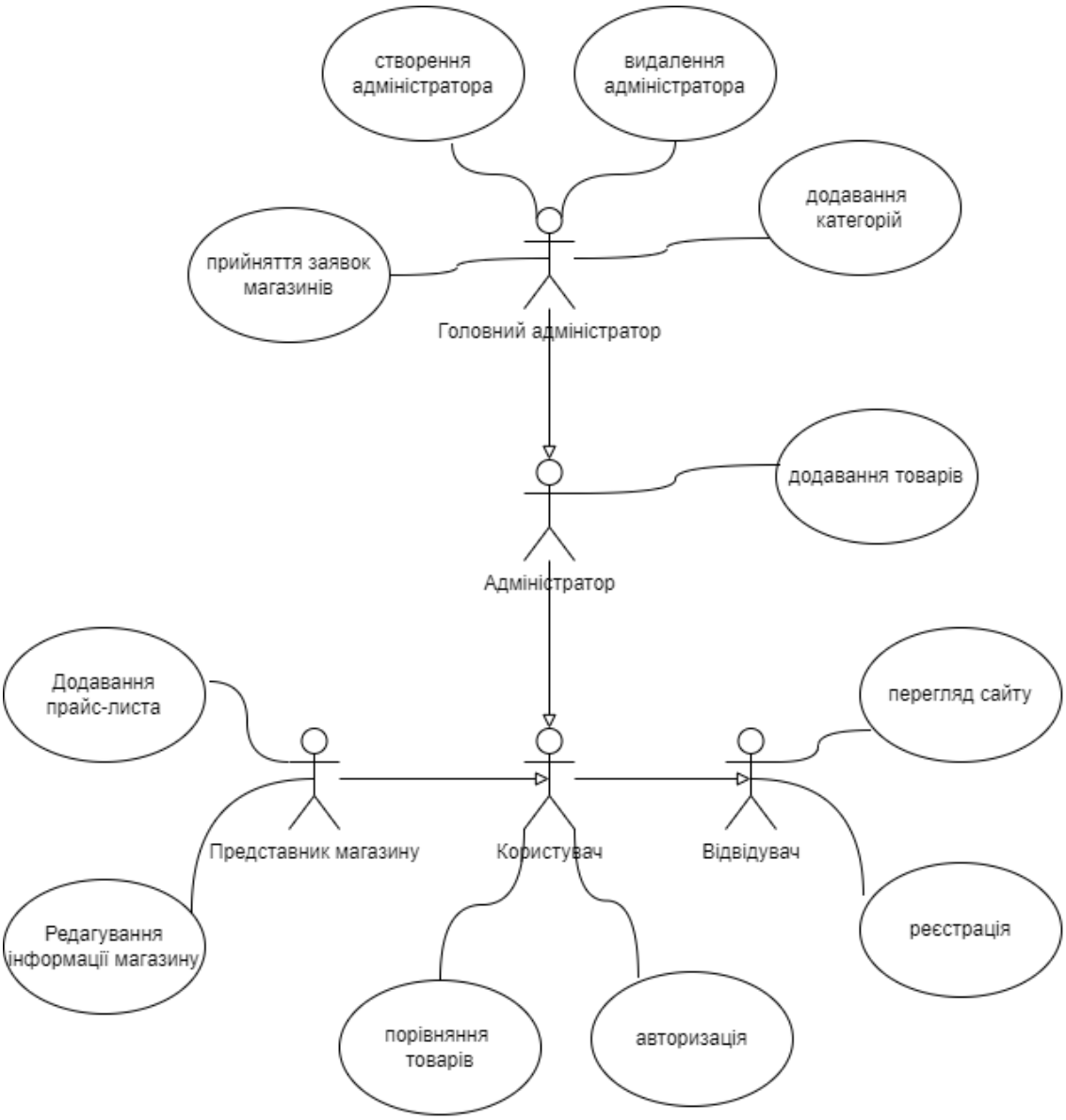

Рисунок 2.5 – Діаграма прецедентів

2.5 Висновки до розділу 2

В ході опрацювання даного розділу було розроблено базу даних системи та побудовано ER-діаграму, що уособлює взаємодію сутностей цієї бази. Було спроєктовано структуру сайту, а також її приблизний вигляд. Також були розглянуті особливості роботи з системою, що у свою чергу дало більше розуміння у тому, які типи користувачів мають бути присутні у системі.

3 РОЗРОБКА СИСТЕМИ

Після обробки вимог, опрацювання технологій, проєктування складових системи та поведінки користувачів, починається етап розробки продукту. Після розробки необхідно провести тестування функціоналу готового продукту. Через малу кількість часу, та нестачу розробників, тестування буде тільки ручне.

3.1 Опрацювання системи

Фреймворк Django працює за принципом модульних додатків, тобто дозволяє розділяти проєкт на менші додатки, що фактично має допомогти використовувати частини проєкту в інших застосунках.

За шаблон проєктування Django використовує архітектуру MVC – Model-View-Controller, або якщо точніше, то її модифіковану версію MTV – Model-Template-View. Це означає, що додаток розділяють на частини, що відповідають свою спеціалізацію.

Модель (Model) відповідає за взаємодію з базою даних: створення, видалення, зчитування та модифікацію її даних. Вид (View) відповідає за представлення цих даних, їх відображення та дизайн. Контролер (Controller) зв’язує модель з представленням та відповідає за прийняття та обробку запитів користувача. Ця архітектура полегшує розробку та надає ще більшу модульність проєкту.

У випадку з Django Вид (View) був замінений на Шаблон (Template), а в якості контролера використовується блок Вид (View). Ці зміни зумовлені назвами Django компонентів, що відповідають за відповідні маніпуляції.

Структура Django проєкту та архітектура маршруту запиту зображена на рисунку 3.1

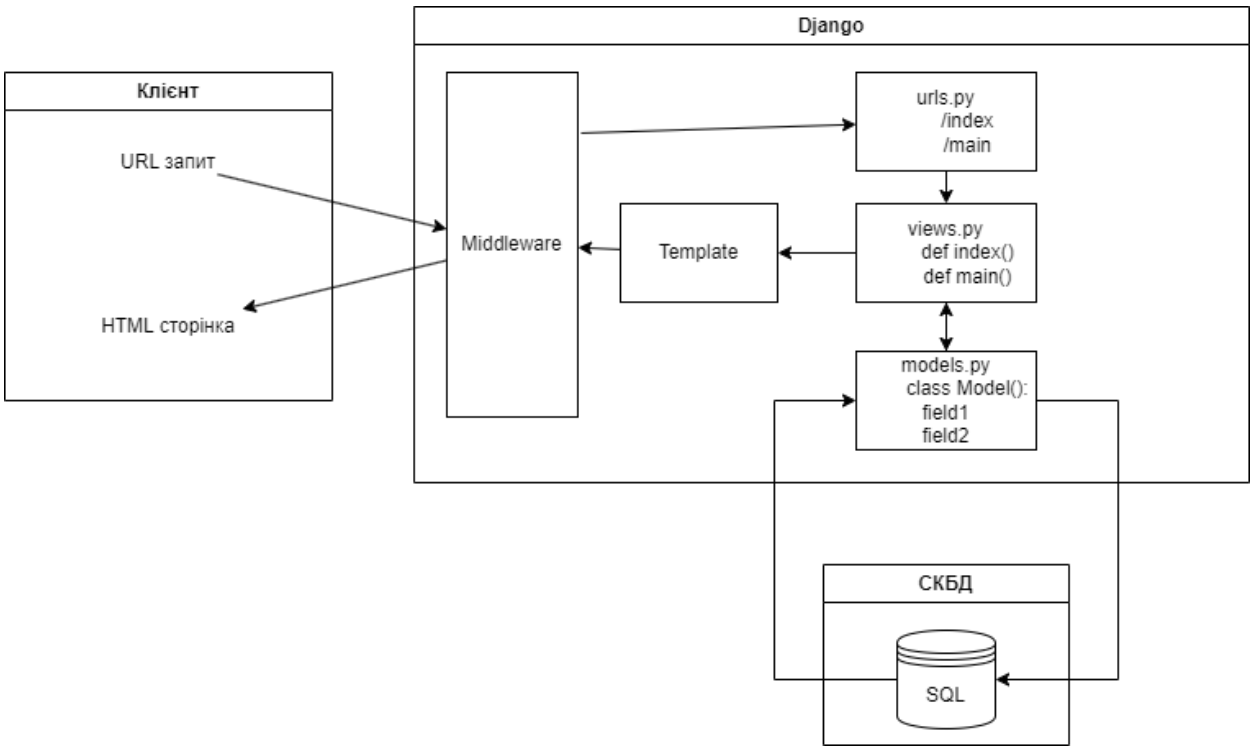

Рисунок 3.1 – Маршрут запиту у системі Django

Отже до структури додатку на Django належать такі моменти:

– middleware – це прошарок системи, що відповідає за первинну обробку запиту;
– urls.py – це маршрутизатор, що перевіряє шлях, який запитав користувач, і переключає запит на відповідну частину контролеру;
– views.py уособлює у собі контролер у MVC MTV;
– models.py – є програмним уособленням моделі у архітектурі MVC MTV;
– template – це збірне поняття представлень, які контролер повертає по запиту користувача, отже це є заключною частиною архітектури MVC.

Через значну розгалуженість результуючих файлів (див. рис. 3.2), можна дещо заплутатися, тому окреслимо основні моменти:

– теки catalog, main, users – це окремі додатки системи, де тека catalog відповідає за усе, що відноситься до роботи категорій і товарів категорій, main – відповідає за основні і другорядні сторінки, що не

відносяться до основної мети проєкту, а додаток users відповідає за користувачів, та все з ними пов’язане;

– fixtures – у цій теці зберігаються кастомні шаблонні теги (про них буде розказано у відповідному розділі);
– media зберігає зображення товарів;
– static відповідає за статичні файли, які система буде використовувати у шаблонах: код Javascript та CSS, шрифти та іконки, статичні зображення на кшталт зображення за замовчуванням;
– templates – зберігає шаблони, які використовуються для виведення представлень на запит користувача;
– PriceGator – головний додаток проєкту, який зберігає налаштування серверу та кореневий маршрутизатор.

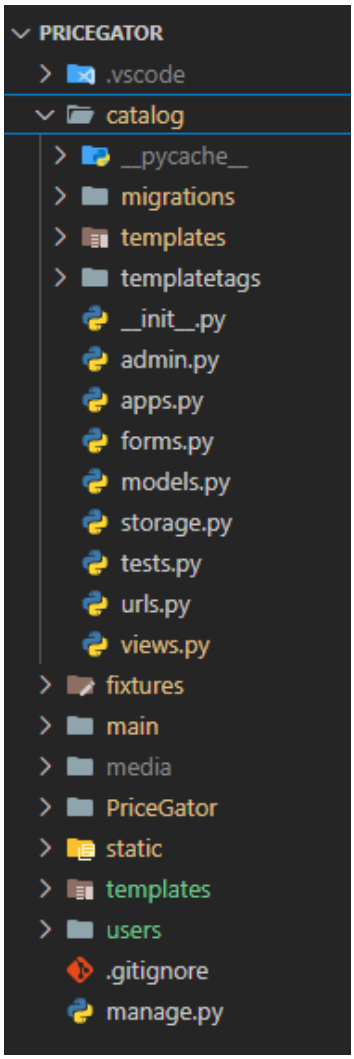

Рисунок 3.2 – Файлова структура проєкту

3.1.1 Створення моделей та бази даних

Для початку треба створити базу даних та підключити її до системи. Для створення бази у PostgreSQL буде використаний графічний інтерфейс pgAdmin 4. Для цього необхідно встановити сервер, створити супер користувача, увійти до системи та створити відповідну таблицю та користувача (з необхідними доступами), через який сервер буде зв’язуватися з базою (див. рис. 3.3).

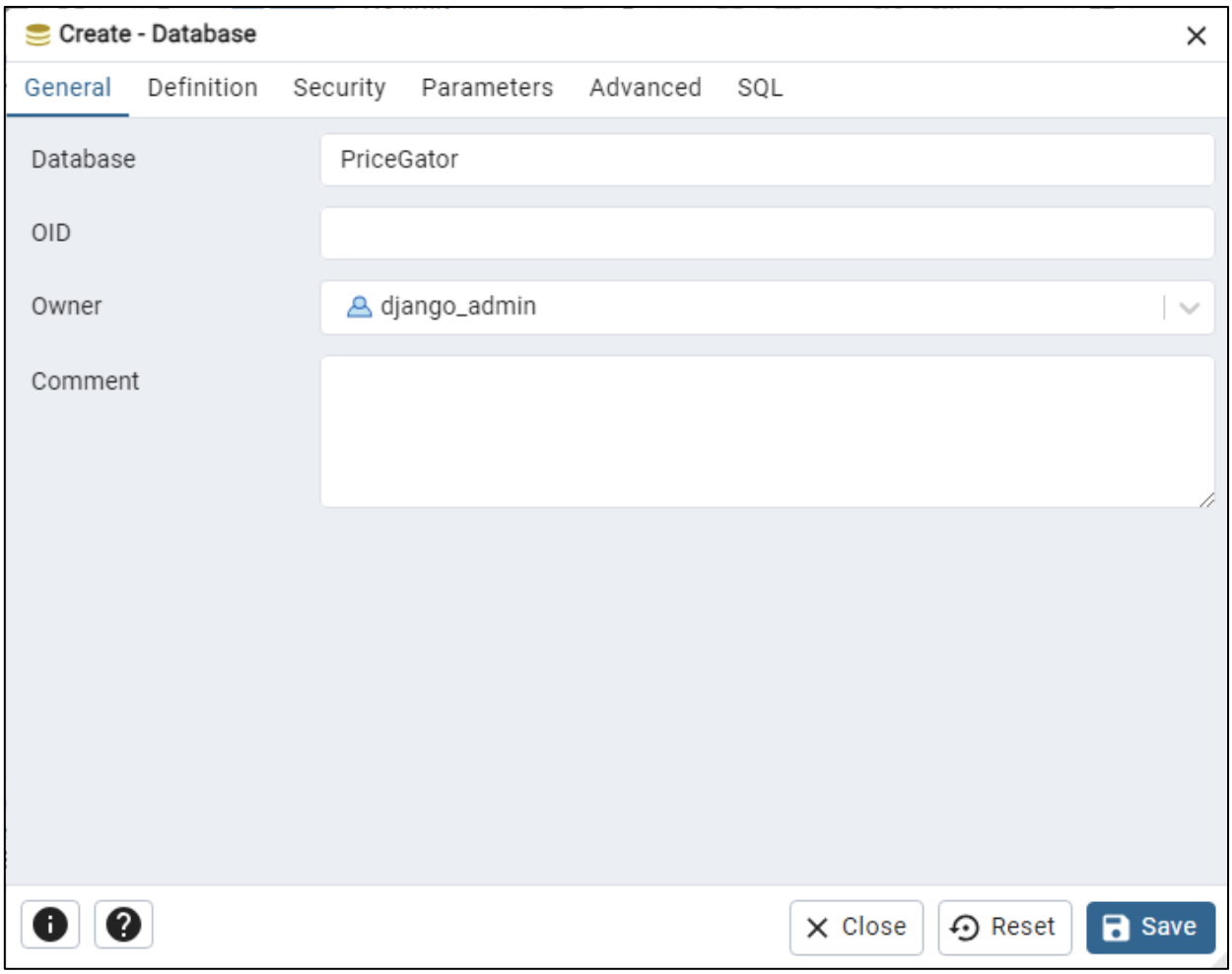

Рисунок 3.3 – Створення бази даних проєкту

Буде створена порожня база даних. Заповнювати її таблицями власноруч не потрібно. Django має власний бекенд для роботи з PostgreSQL, який створить таблиці, відповідно до моделей, що були створені. Для активації бекенду у файлі “settings.py” головного додатку, у змінній “DATABASES”

вказати який бекенд використовувати, назву бази, ім’я та пароль користувача, адресу серверу бази даних та її порт (див. рис. 3.4).

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        "NAME": "PriceGator",
        "USER": "django_admin",
        "PASSWORD": "Ylf4Hsd0",
        "HOST": "127.0.0.1",
        "PORT": "5432",
    }
}
```

Рисунок 3.4 – Налаштування для підключення бази даних PostgreSQL

Після підключення бази даних треба створити моделі даних, які будуть реалізувати таблиці у базі даних. Для цього треба у відповідному додатку, у файлі “models.py” створити необхідні класи, що уособлюють сутності бази даних. Змінні цих класів будуть полями таблиць.

На рисунку 3.5 зображений приклад створення моделі товару у додатку каталогу товарів.

При створенні моделі клас повинен наслідувати вбудований клас моделі. Клас Meta позначає додаткові параметри моделі, такі як: які поля індексувати, як позначати множину об’єктів та один об’єкт, порядок зображення та унікальні поля. Додатково до класу можна додавати методи, до яких можна буде звертатися при використанні об’єктів моделі.

Таким чином створюються усі наступні моделі. Винятками є лише модель категорій і моделі користувачів. У випадку з моделлю категорії, наслідування класу відбувається не від звичайного класу моделей, а від класу, що надається модулем Django-MPTT. Цей модуль надає можливість створювати деревовидні моделі даних, де об’єкт моделі може мати зв’язок з іншим об’єктом цієї ж моделі. Також модуль надає методи для взаємодії з цією структурою, що вкрай полегшило роботу зі очевидно деревоподібною

структурою категорій. Для користувачів відмінність полягає лише у наслідуваному класі. Замість django.db.models.Model, вони наслідують django.contrib.auth.models.AbstractUser.

```
from django.db import models

class Product(models.Model):
    """ Product model. Stores base information about product.  It is used to gather info about the same products.
"""
    category = models.ForeignKey(Category, related_name='products',
on_delete=models.CASCADE,verbose_name='Категорія')
    brand = models.ForeignKey(Brand, related_name='products', on_delete=models.CASCADE)

    name = models.CharField(max_length=100, unique=True, db_index=True, verbose_name='Назва')
    slug = models.SlugField(max_length=100, unique=True, db_index=True, verbose_name='URL')
    description = models.TextField(blank=True, null=True, verbose_name='Опис')
    image = models.ImageField(upload_to='products/%Y/%m/%d', blank=True, null=True,
verbose_name='Зображення')
    code = models.CharField(max_length=100, unique=True,verbose_name='Код товару')
    created = models.DateTimeField(auto_now_add=True,verbose_name='Створено')
    updated = models.DateTimeField(auto_now=True,verbose_name='Оновлено')

    class Meta:
        ordering = ('name',)
        verbose_name = 'Продукт'
        verbose_name_plural = 'Продукти'
        index_together = (('id', 'slug'),)

    def __str__(self):
        return f'{self.name} Категорія: {self.category.name}'

    def get_min_price(self):
        price = self.products_in_store.aggregate(models.Min('price'))['price__min']
        if price > 1000:
            return round(price, 0)
        return price

    def get_max_price(self):
        price = self.products_in_store.aggregate(models.Max('price'))['price__max']
        if price > 1000:
            return round(price, 0)
        return price
```

Рисунок 3.5 – Приклад створення моделі товарів

Після створення моделей необхідно зробити міграцію моделей. Тут Django теж все робить автоматично. Потрібно лише у командну строку ввести:

– py manage.py makemigrations – створює файли міграції;

– py manage.py migrate – проводить саму міграцію.

Manage.py являє собою інтерфейс для роботи з проєктом та знаходиться в його корені.

На рисунку 3.6 зображено як це відбувається (міграції вже були проведені, тому ніякі операції не здійснилися).

```
PS C:\Users\megak\Desktop\unik\4 year\sem 2\DIPLOM\Try4\PriceGator> py manage.py makemigrations
No changes detected
PS C:\Users\megak\Desktop\unik\4 year\sem 2\DIPLOM\Try4\PriceGator> py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, catalog, contenttypes, sessions, users
Running migrations:
  No migrations to apply.
```

Рисунок 3.6 – Приклад проведення міграції моделей

3.1.2 Реалізація маршрутизаторів

Перед реалізацією контролерів, краще розробити схему маршрутизації, щоб зрозуміти, які взагалі контролери повинні бути присутні. На рисунку 3.7 зображено код маршрутизатора основного додатка, який відповідає за маршрутизацію по всьому проєкту. На рисунках 3.8, 3.9, 3.10 зображено код маршрутизаторів інших додатків, но яки посилається основний маршрутизатор.

У маршрутизаторі створюється список функцій, що працюють як диспетчер. При проходженні по масиву, адреса запиту звіряється з адресами, що були зазначені як перша змінна функції path(). При знаходженні збігу, викликається функція, що була вказано другою змінною цієї функції. Третьою змінною функції є назва для оберненого звернення. Першим елементом масиву

було визначено посилання на сторінку адміністрування, що працює по адресу «admin/», що буде означати «адреса сайту/admin/». Далі були зазначені адреси для запитів пошуку та автозаповнення для пошуку, що хоч і були визначені у додатку каталогу, були спеціально імпортовані в головний маршрутизатор, щоб їх адреса була мінімалістичною. У кінці були включені маршрутизатори інших додатків.

```
from django.contrib import admin
from django.urls import path
from django.urls.conf import include
from django.conf import settings
from django.conf.urls.static import static

from PriceGator import settings

from main import views
from catalog import views as catalog_views



urlpatterns = [
    path('admin/', admin.site.urls),

    path('search/', catalog_views.search, name='search'),
    path('autocomplete/', catalog_views.autocomplete, name='autocomplete'),

    path('user/', include('users.urls', namespace='users')),
    path('', include('main.urls', namespace='main')),
    path('', include('catalog.urls', namespace='catalog')),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

if settings.DEBUG:
    import debug_toolbar

    urlpatterns = [
        path('__debug__/', include(debug_toolbar.urls)),
    ] + urlpatterns
```

Рисунок 3.7 – Основний маршрутизатор системи

```
from django.urls import path

from catalog import views

app_name = 'catalog'

urlpatterns = [
    path('<slug:category_slug>/', views.category, name='category-page'),
    path("<slug:category_slug>/<slug:product_slug>/", views.product, name="product-page"),
]
```

Рисунок 3.8 – Маршрутизатор додатку каталогу

```
from django.urls import path

from main import views

app_name = 'main'

urlpatterns = [
    path('', views.index, name='index'),
    path("about/", views.about, name="about-page"),
    path("contacts/", views.contacts, name="contacts-page"),
]
```

Рисунок 3.9 – Маршрутизатор додатку для основних сторінок

```
from django.urls import path

from users import views

app_name = 'users'

urlpatterns = [
    path("", views.profile, name="profile"),
    path('login/', views.login, name='login'),
    path("register/", views.register, name="register"),
    path("logout/", views.logout, name="logout"),
]
```

Рисунок 3.10 – Маршрутизатор додатку користувачів

У маршрутизаторах додатків зазначені маршрути на відповідні контролери додатку, а самим маршрутизаторам присвоєні імена, для роботи зворотного пошуку адрес.

У маршрутизаторі каталогу зустрічається подібна конструкція у маршруті: <slug:category_slug>. Ця структура використовується для захоплення значень. Якщо вказати конкретний конвертер, то буде захоплене тільки відповідне значення. У цьому випадку захоплюється slug каталогу, що потім передається окремим значенням у контролер для обробки. В проєкті це використовується лише для сторінок товарів та їх каталогів.

3.1.3 Реалізація контролерів

Контролери створюються для кожного додатку окремо, та являються функціями, чи класами, на які посилаються маршрутизатори, і яки приймають запит та додаткову інформацію, що надійшла з цим запитом.

У додатку Б наведено приклад реалізації контролерів на прикладі додатку каталога.

У даному контролері були реалізовані функції для повернення сторінок категорій за їх слагом, продукту за своїм слагом та слагом його категорії та повернення пошуку з пропозиціями пошуку. Також тут була реалізована функція що формує хлібні крихти для сторінок товарів та категорій по категорії та товару.

Контролери, після обробки, викликають функцію рендеру сторінки, що приймає запит, посилання на шаблон, контекст для шаблона та інші змінні. Функція рендеру повертає об’єкт HttpResponse, що являє собою вже готову до повернення сторінку. Тому контролер по суті повертає вже готову сторінку користувачу. Також контролер може повернути об’єкт JsonResponse, який є серіалізованим Json об’єктом. В даному випадку він використовується у контролері автопідсказок для пошуку, і викликається AJAX запитом при заповненні поля пошуку.

3.1.4 Реалізація шаблонів

Шаблон – це спосіб вебфреймворка створювати динамічну HTML розмітку. Шаблон включає в себе статичну розмітку та спеціальний синтакс, що описує як динамічний контент буде влаштовуватися.

Django має власний шаблонізатор «Django template language» [12], але має підтримку інших популярних шаблонізаторів, таких як Jinja2.

Django визначає стандартний API для завантаження та відтворення шаблонів незалежно від серверної частини. Завантаження складається з пошуку шаблону для заданого ідентифікатора та його попередньої обробки, зазвичай компілюючи його для представлення в пам’яті. Рендеринг означає інтерполяцію шаблону контекстними даними та повернення отриманого рядка.

Синтаксис даної мови шаблонування є наступним:

– змінні записуються у подвійних фігурних дужках: {{ змінна }}, змінні надаються контекстом, що у свою чергу створюється контролером;
– теги – уособлюють додаткову логіку під час рендеренгу, позначаються фігурними дужками зі знаками відсотків всередині: {% логіка %};
– фільтри – записуються після знаку « | » у тегу змінної, і використовується для трансформації значень та змінних тегів;
– коментарі – використовуються для коментарів та виключення частини коду з рендеренгу.

Приклад реалізації шаблонів зазначено на рисунку 3.11.

У даному прикладі були використані теги {% if %}, {% else %}, {% endif %}, що є логічними тегами, що мають схожі представлення у мові Python.

Також було використано структурні теги {% block content %}{% endblock content %}, що має на увазі те, що між цими тегами буде розташовано те, що наслідуваному шаблоні буде вставлено в такий самий тег, тобто шаблони можна наслідувати, а цей шаблон є наслідуваним.

Ще тут представлений тег циклу {% for crumb in breadcrumbs %} та {% endfor %}.

```
<main id="main" class="container-xxl">
 <nav aria-label="breadcrumb" class="mt-3">
  <ol class="breadcrumb">
   {% for crumb in breadcrumbs %}
    {% if crumb.url %}
     <li class="breadcrumb-item">{% if crumb.dropdown %}{{ crumb.dropdown|safe }}{% else %}<a href="{{ crumb.url }}">{{ crumb.name }}</a>{% endif %}</li>
    {% else %}
     <li class="breadcrumb-item active" aria-current="page">{{ crumb.name }}</li>
    {% endif %}
   {% endfor %}
  </ol>
 </nav>
{% block content %}{% endblock content %}
</main>
```

Рисунок 3.11 – Реалізація виведення контенту сторінки та хлібних крихт над ним

3.1.5 Демонстрація інтерфейсу

У цьому підрозділі буде показана верстка сайту та деякі його функції.

Наповнення сайту невелике, через велику кількість компонентів для заповнення та нестачу часу. Графічна розробка відбувалася за допомогою CSS бібліотеки Bootstrap 5 [13].

Далі представлені графічні представлення шаблонів (див. рис. 3.12 – 3.18).

Для реалізації пошуку використовувався Javascript [14] з бібліотекою JQuery для AJAX запитів.

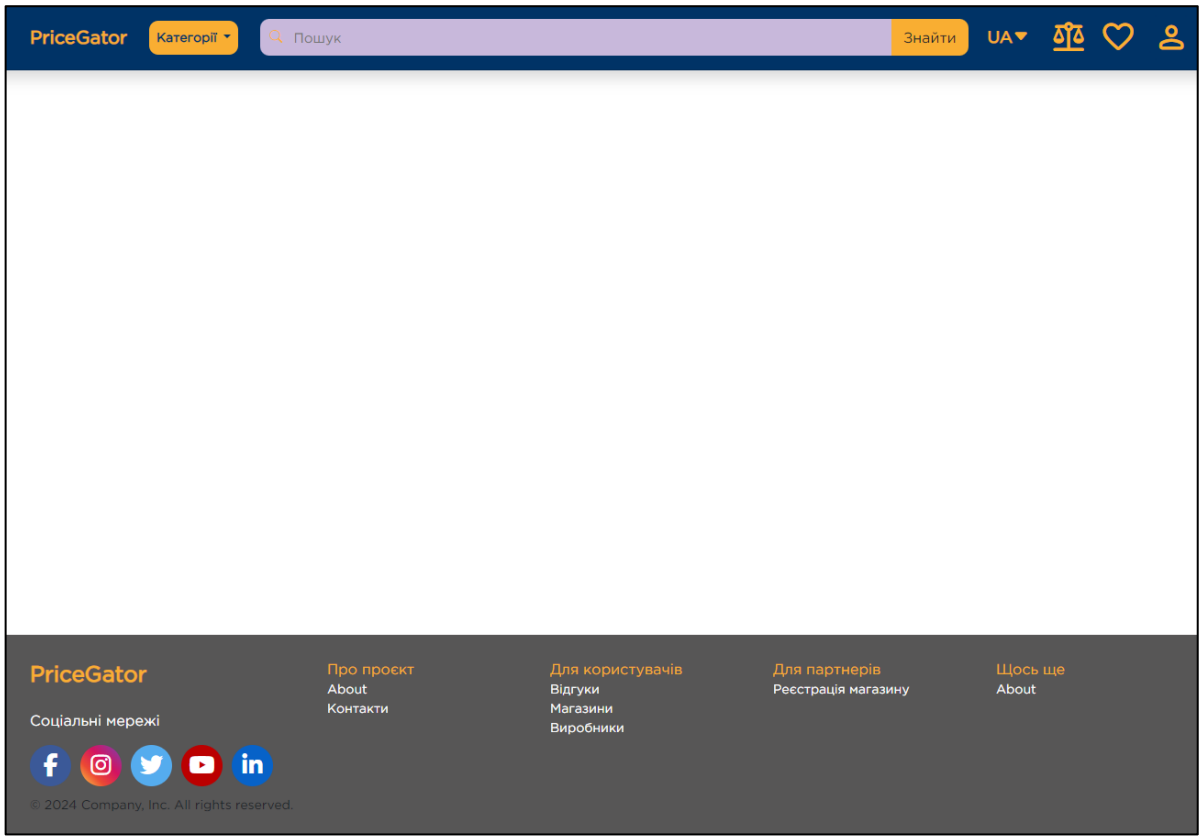

Рисунок 3.12 – приклад основного шаблону, від якого наслідуються інші сторінки

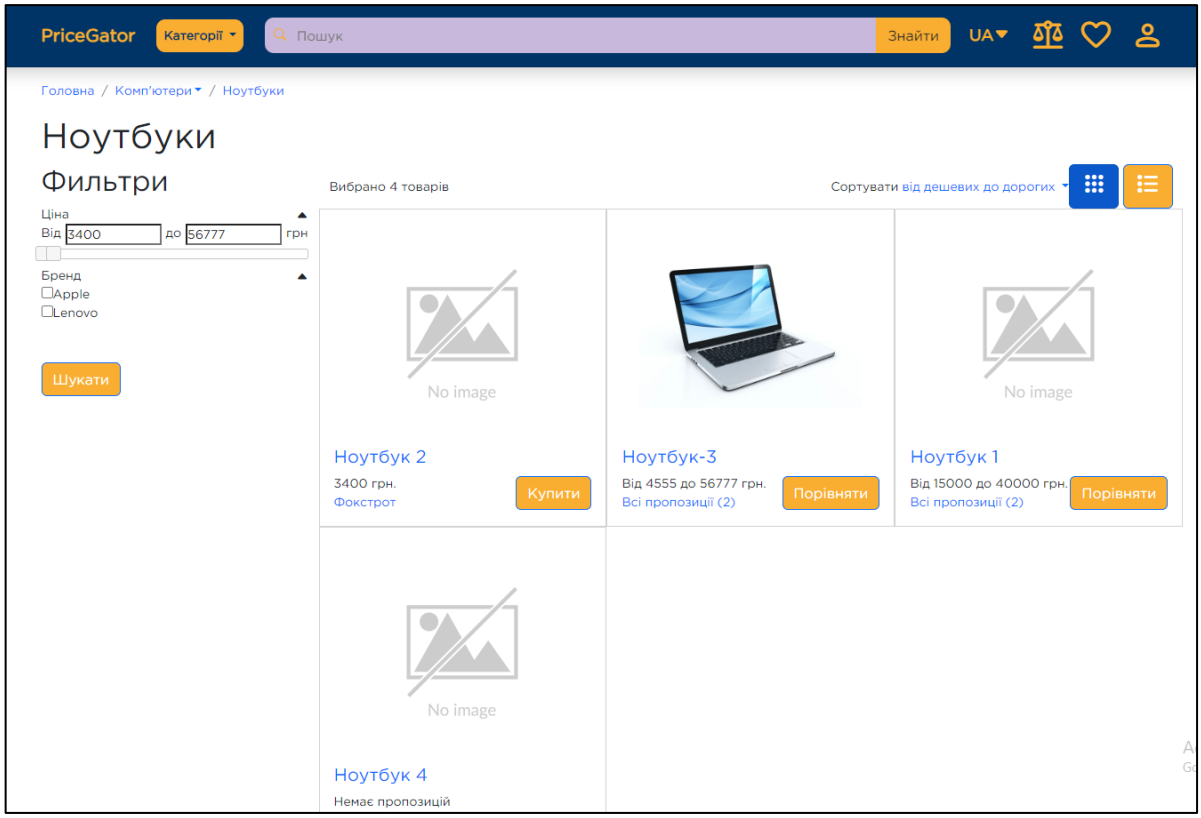

Рисунок 3.13 – Приклад сторінки категорії

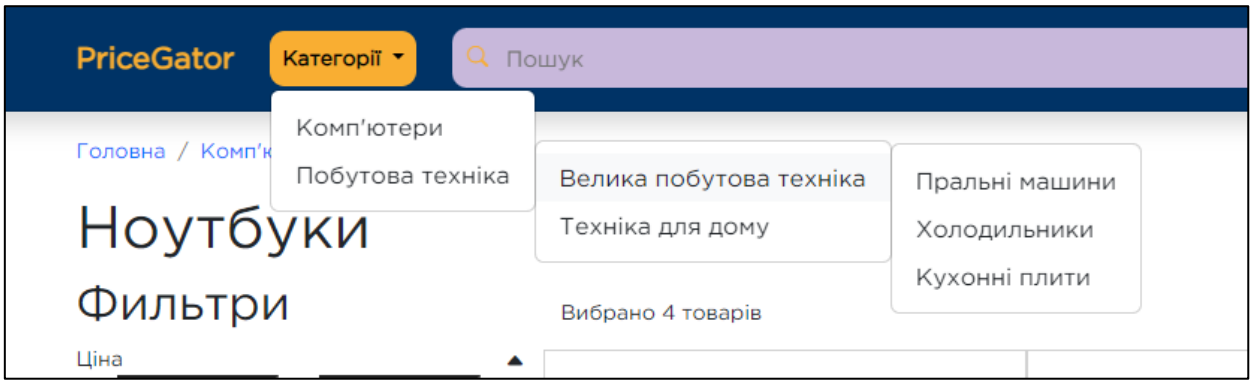

Рисунок 3.14 – Приклад спадного меню з вибором категорій

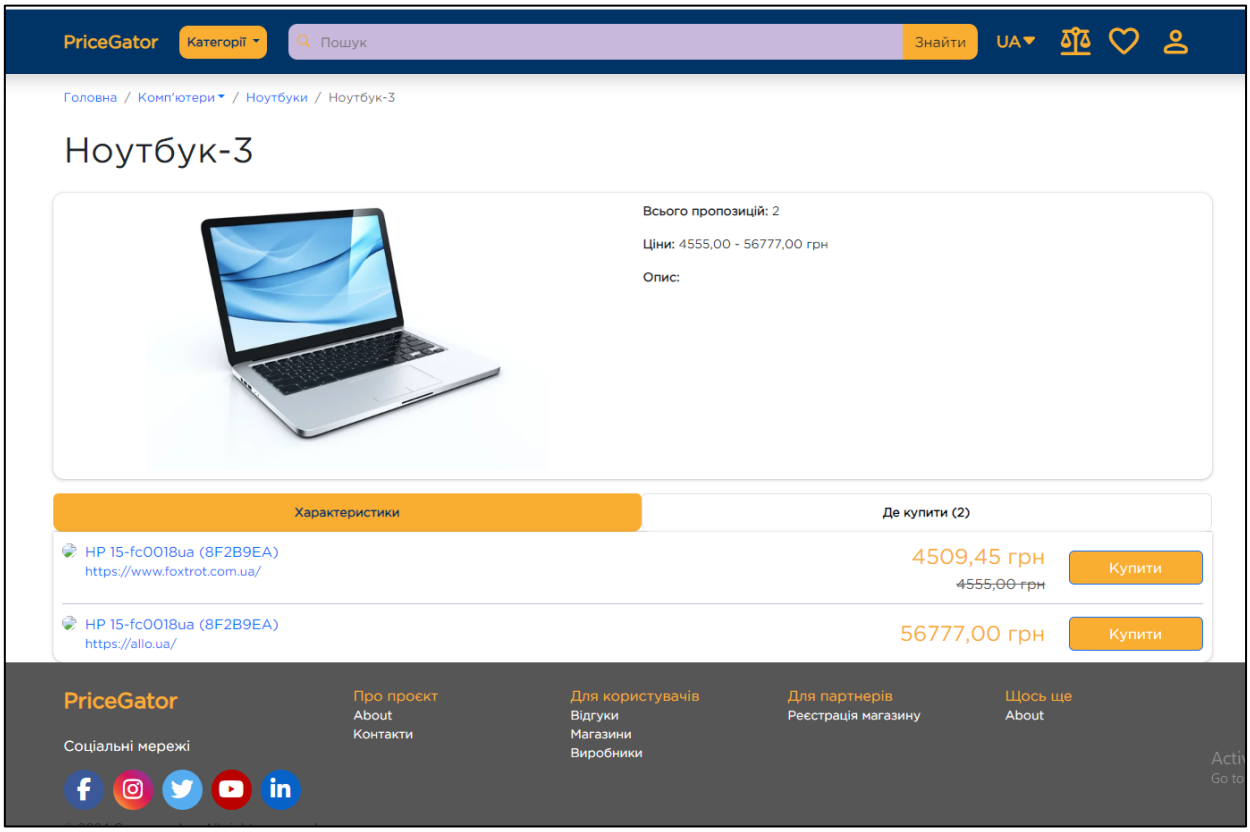

Рисунок 3.15 – Приклад сторінки товару з магазинами, що його продають

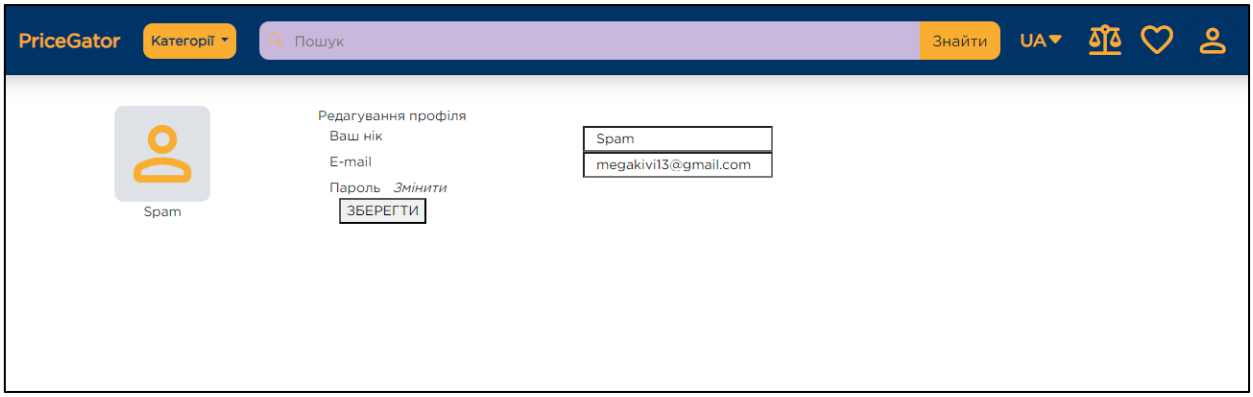

Рисунок 3.16 – Приклад сторінки користувача

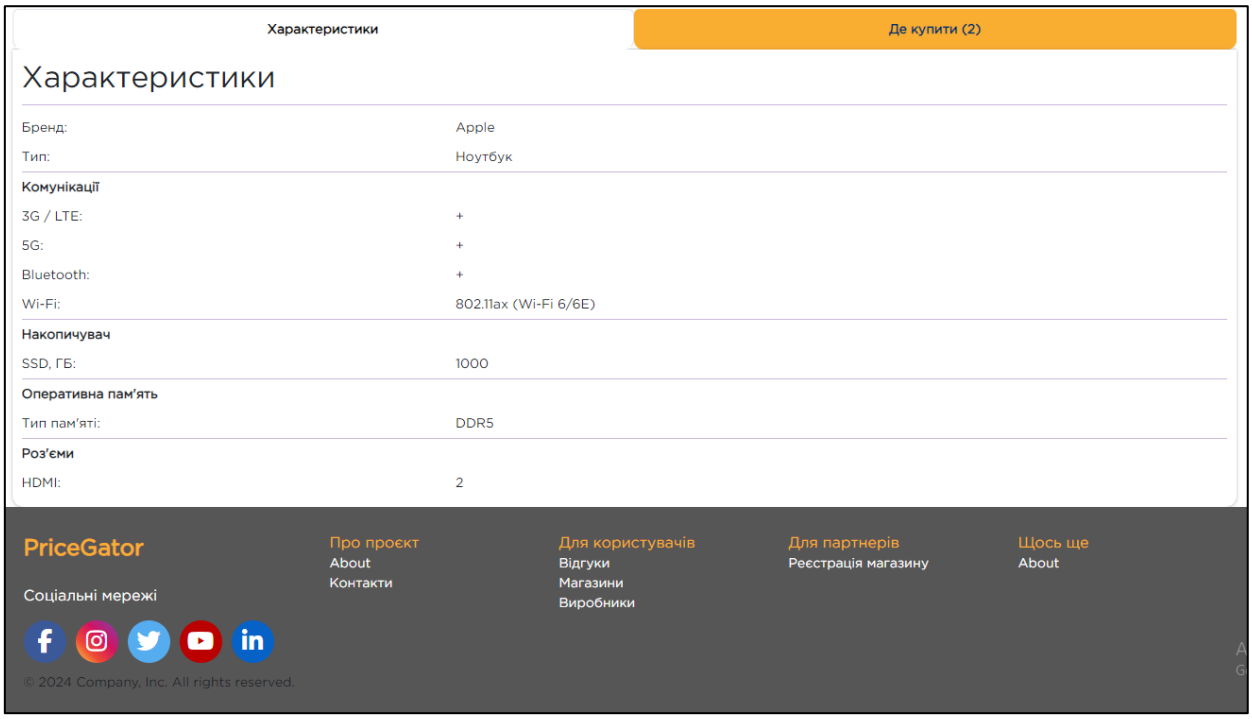

Рисунок 3.17 – Приклад виводу характеристик товару

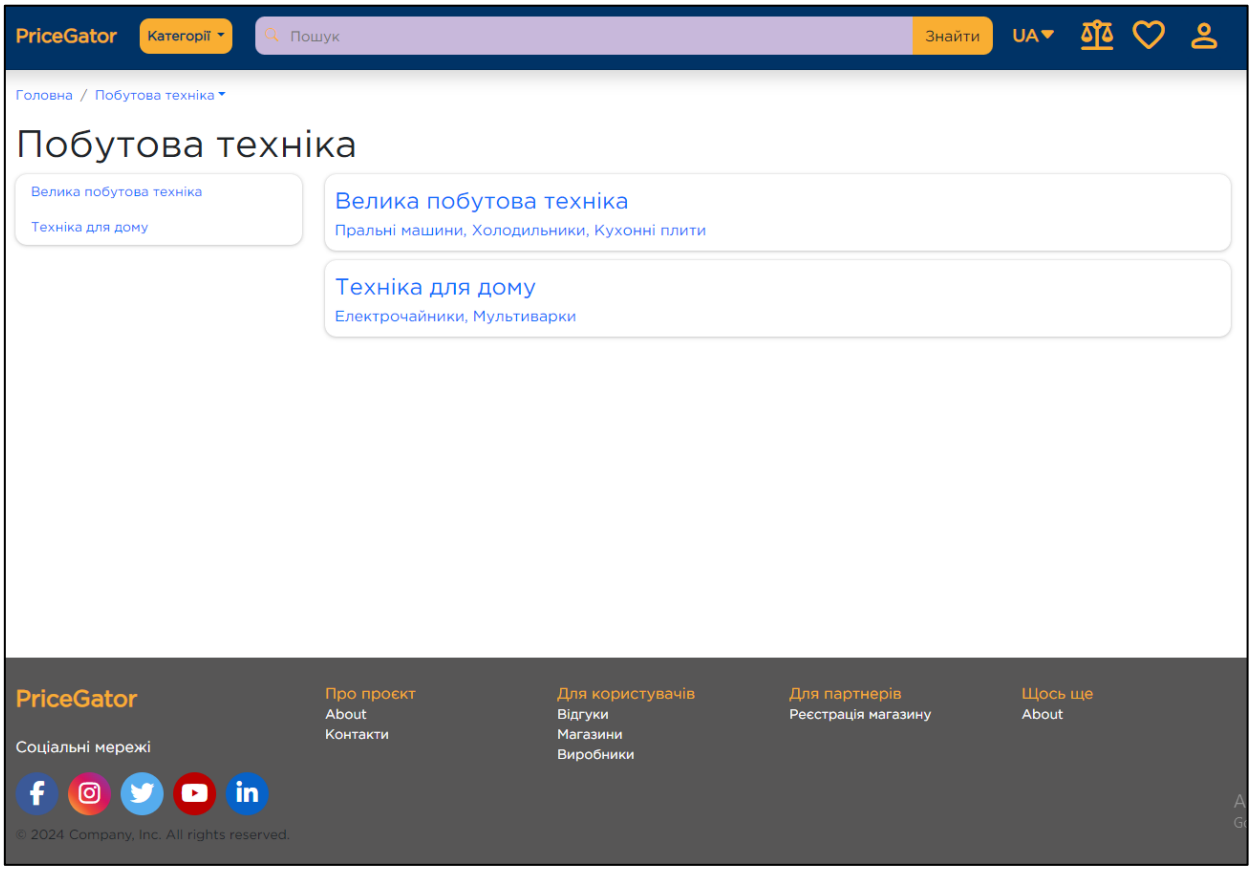

Рисунок 3.18 – Приклад виводу дерева кетегорій

3.2 Висновки до розділу 3

У третьому розділі було описано основні кроки розробки модульної системи додатків. Було реалізовано архітектуру MVC у стилі Django, а саме:

– були розроблені моделі даних;
– розроблено маршрутизатори додатків;
– створено контролери цих додатків;
– шаблони представлення сторінок.

Також було продемонстровано вигляд основних сторінок готового сайту.

ВИСНОВКИ

В даній кваліфікаційній роботі було розроблено робочий агрегатор товарної інформації, який отримав лаконічну назву PriceGator. Також було розглянуто принцип розробки з використанням фреймворку Django, мови програмування Python.

Під час роботи було ретельно проаналізовано основні вимогу та існуючих конкурентів. Було розроблене та виконане технічне завдання. Також було визначено інструментарій для розробки системи. Вибором технологій стала мова програмування Python, фреймворк Django, через простоту та швидкість роботи з ними. За базу даних було обрано систему керування базами даних PostgreSQL, через його розвинутий функціонал та відому відмінність інтеграції її з Django.

Також було сплановано структуру даних, що будуть зберігатись у базі та як вони будуть залежати один від одного. Розроблено приблизні макети головних сторінок сайту. Ще було сплановано взаємодію між користувачами та системою, що дало відповідь на те, яка кількість типів користувачів повинна бути у системі.

Після цього було розпочато реалізацію системи. Через модульну особливість фреймворку Django, проєкт поділився на 4 частини:

- головний додаток, що має у собі налаштування системи та головний маршрутизатор;
- додаток зі статичними сторінками;
- додаток з реалізацією каталогу з категоріями товарів;
- додаток з реалізацією користувачів.

При реалізації додатків, для задоволення вимог архітектури MVC, для кожного з них були розроблені модулі моделей, контролерів, роутерів та необхідні шаблони мови шаблонів Django, що додає динамічності до розмітки HTML.

Результатом став мінімалістичний сайт, що здатен демонструвати відвідувачам товари з розцінками від різних магазинів. Товари яких є акуратно відсортовані за категоріями. Система відповідає вимогам, що були закладені на етапі аналізу предметної області.

ПЕРЕЛІК ПОСИЛАНЬ

1. What is an eCommerce Aggregator? *Geomotiv*. URL: https://geomotiv.com/blog/how-to-develop-ecommerce-aggregators/ (дата звернена: 02.04.2024).
2. How eCommerce Aggregators are Changing the Game. *Debutify*. URL: https://debutify.com/blog/ecommerce-aggregators/ (дата звернення: 01.04.2024).
3. Україна – Всі ціни Інтернет-Магазинів. *E-Katalog*. URL: https://ek.ua/ua/ (дата звернення: 01.02.2024).
4. Порівняти ціни в інтернет-магазинах України. *Hotline*. URL: https://hotline.ua/ (дата звернення: 01.02.2024).
5. Порівняння цін інтернет-магазинів в Україні. *Price.ua*. URL: https://price.ua/ua (дата звернення: 01.02.2024).
6. The web framework for perfectionists with deadlines. *Django*. URL: https://www.djangoproject.com/ (дата звернення: 30.02.2024).
7. The world’s most advanced open source database. *PostgreSQL*. URL: https://www.postgresql.org/ (дата звернення: 05.03.2024).
8. What Is a Database? *Oracle Cloud Infrastructure*. URL: https://www.oracle.com/database/what-is-database/ (дата звернення: 05.03.2024).
9. The Write Less, Do More, JavaScript Library. *jQuery*. URL: https://jquery.com/ (дата звернення: 05.04.2024).
10. Git. URL: https://www.git-scm.com/ (дата звернення: 02.03.2024).
11. Draw.io. URL: https://app.diagrams.net/ (дата звернення: 10.03.2024).
12. The Django template language. *Django*. URL: https://docs.djangoproject.com/en/5.0/ref/templates/language/ (дата звернення: 24.04.2024).

13. The most popular HTML, CSS, and JS library in the World. *Bootstrap*. URL: https://getbootstrap.com/ (дата звернення: 25.04.2024).
14. Сучасний підручник з JavaScript. *JavaScript.Info*. URL: https://uk.javascript.info/ (дата звернення: 20.03.2024).

ДОДАТОК А

Лістинг коду

А.1 Основний шаблон main.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en" data-bs-theme="light">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}{% endblock %}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
    <link href="https://fonts.googleapis.com/css2?family=Material+Symbols+Rounded" rel="stylesheet" />
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <link rel="stylesheet" href="https://code.jquery.com/ui/1.13.3/themes/base/jquery-ui.css">
    <link rel="stylesheet" href="{% static 'css/main.css' %}">
    <link rel="stylesheet" href="{% static 'fontawesome/css/fontawesome.css' %}">
    <link rel="stylesheet" href="{% static 'fontawesome/css/brands.css' %}">
    <link rel="stylesheet" href="{% static 'fontawesome/css/solid.css' %}">
    {% block custom_css %}
    <style>
```

```
    .profile ul li a{
      width:100%
    }
    </style>
    {% endblock custom_css %}
    {% block custom_js %}{% endblock custom_js %}
</head>
<body class="d-flex flex-column">
  <header class="navbar p-0 navbar-expand-lg sticky-top header-nav" >
    <nav class="container-xxl bd-gutter flex-wrap flex-lg-nowrap head-nav">
      <a class="navbar-brand p-0 logo" href="/">
        PriceGator
      </a>
      <div class="categories">
        <ul class="navbar-nav">
          <!-- Dropdown -->
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle me-4" href="#" id="navbarDropdownMenuLink" role="button"data-bs-toggle="dropdown" aria-expanded="false">
              Категорії
            </a>
            <ul class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
              {% for cat in categories %}
                <li>
                  <a class="dropdown-item" href="{% url 'catalog:category-page' cat.slug %}">{{ cat.name }}</a>
                  {% if cat.children.exists %}
                    <ul class="dropdown-menu dropdown-submenu">
                      {% include 'main/category_dropdown.html' with categories=cat.children.all %}
                    </ul>
                  {% endif %}
                </li>
              {% endfor %}
            </ul>
          </li>
```

```
    </ul>
   </div>
   <div class="search-container">
    <form class="search-form" action="{% url 'search' %}" method="get">
     <div class="search-form-inner">
      <i class="bi bi-search"></i>
      <input id="search-input" type="text" name="q" placeholder="Пошук" class="search-form__input" autocomplete="off" aria-label="Пошук"></input>
     </div>
     <button class="btn search-form__submit">Знайти</button>
    </form>
   </div>
   <ul class="header-buttons">
    <li class="dropdown-center header-buttons-item header-buttons-item-lang">
     <button id="lang-btn" class="icon btn" data-bs-toggle="dropdown">UA<i class="bi bi-caret-down-fill"></i></button>
     <ul class="dropdown-menu dropdown-menu-end lang-dropdown">
      <li><button class="dropdown-item" type="button">UA</button></li>
      <li><button class="dropdown-item" type="button">EN</button></li>
     </ul>
    </li>
    <li class="header-buttons-item header-buttons-item-comparison d-flex p-1 standalone">
     <span class="icon material-symbols-rounded">
      balance
     </span>
    </li>
    <li class="header-buttons-item header-buttons-item-wishlist d-flex p-1 standalone">
     <span class="icon material-symbols-rounded">
     favorite
     </span>
    </li>
    <li class="header-buttons-item header-buttons-item-user d-flex p-1 standalone">
     <!--<span class="icon material-symbols-rounded">
      person
     </span>-->
```

```
        <div class="dropdown profile">
          <button class="profile-button p-1 d-flex" type="button" data-bs-toggle="dropdown" aria-expanded="false">
            <span class="icon material-symbols-rounded">
              person
            </span>
          </button>
          <ul class="dropdown-menu dropdown-menu-end">
            <li><a class="dropdown-item" href="{% url "users:profile" %}">Кабінет</a></li>
            <li><a class="dropdown-item" href="{% url "users:logout" %}">Вийти</a></li>
          </ul>
        </div>
      </li>
    </ul>
  </nav>
 </header>
 </div>
 <!--main content-->
 <main id="main" class="container-xxl">
  <nav aria-label="breadcrumb" class="mt-3">
    <ol class="breadcrumb">
      {% for crumb in breadcrumbs %}
        {% if crumb.url %}
          <li class="breadcrumb-item">{% if crumb.dropdown %}{{ crumb.dropdown|safe }}{% else %}<a href="{{ crumb.url }}">{{ crumb.name }}</a>{% endif %}</li>
        {% else %}
          <li class="breadcrumb-item active" aria-current="page">{{ crumb.name }}</li>
        {% endif %}
      {% endfor %}
    </ol>
  </nav>
 {% block content %}{% endblock content %}
 </main>
 <!---->
 <!--<footer>-->
```

```
<footer id="footer" class="/*fixed-bottom*/ mt-auto">
  <div class="container-xxl  bg-color-3">
    <div class="row flex-wrap">
      <div class="col-xs-12 col-lg-3">
        <div class="footer-main text-lg">
          <a class="navbar-brand" href="{% url "main:index" %}">PriceGator</a>
        </div>
        <div class="footer-buttons">
          <div class="footer-socials">
            <div class="footer-socials-title text-md">
              Соціальні мережі
            </div>
            <div class="footer-socials-list d-flex flex-wrap">
              <a href="/" rel="noopener" class="fa-brands fa-facebook-f d-flex justify-content-center align-items-center"></a>
              <a href="/" rel="noopener" class="fa-brands fa-instagram d-flex justify-content-center align-items-center"></a>
              <a href="/" rel="noopener" class="fa-brands fa-twitter d-flex justify-content-center align-items-center"></a>
              <a href="/" rel="noopener" class="fa-brands fa-youtube d-flex justify-content-center align-items-center"></a>
              <a href="/" rel="noopener" class="fa-brands fa-linkedin-in d-flex justify-content-center align-items-center"></a>
            </div>
          </div>
          <div class="footer-light-switch"></div>
        </div>
      </div>
      <div class="col-xs-12 col-lg-9">
        <div class="row flex-wrap">
          <div class="col-xs-12 col-lg-3 flex-column footer-category">
            <div class="footer-category-title d-flex">
              <span class="text-md">Про проєкт</span>
              <span class="hidden-lg icon"><i class="bi bi-caret-down-fill"></i></span>
            </div>
```

```
  <div class="footer-category-list ">
    <div class="footer-category-item">
      <a href="{% url "main:about-page" %}" rel="nofollow">About</a>
    </div>
    <div class="footer-category-item">
      <a href="{% url "main:contacts-page" %}" rel="nofollow">Контакти</a>
    </div>
  </div>
</div>
<div class="col-xs-12 col-lg-3 flex-column footer-category">
  <div class="footer-category-title d-flex">
    <span class="text-md">Для користувачів</span>
    <span class="hidden-lg icon"><i class="bi bi-caret-down-fill"></i></span>
  </div>
  <div class="footer-category-list ">
    <div class="footer-category-item">
      <a href="/" rel="nofollow">Відгуки</a>
    </div>
    <div class="footer-category-item">
      <a href="/" rel="nofollow">Магазини</a>
    </div>
    <div class="footer-category-item">
      <a href="/" rel="nofollow">Виробники</a>
    </div>
  </div>
</div>
<div class="col-xs-12 col-lg-3 flex-column footer-category">
  <div class="footer-category-title d-flex">
    <span class="text-md">Для партнерів</span>
    <span class="hidden-lg icon"><i class="bi bi-caret-down-fill"></i></span>
  </div>
  <div class="footer-category-list ">
    <div class="footer-category-item">
      <a href="/" rel="nofollow">Реєстрація магазину</a>
    </div>
```

```
            </div>
          </div>
          <div class="col-xs-12 col-lg-3 flex-column footer-category">
            <div class="footer-category-title d-flex">
              <span class="text-md">Щось ще</span>
              <span class="hidden-lg icon"><i class="bi bi-caret-down-fill"></i></span>
            </div>
            <div class="footer-category-list ">
              <div class="footer-category-item">
                <a href="{% url "main:about-page" %}" rel="nofollow">About</a>
              </div>
            </div>
          </div>
        </div>
      </div>
      <div class="footer-copyright">
        <p>© 2024 Company, Inc. All rights reserved.</p>
      </div>
    </footer>
    <!---->
    </div>
<script src="{% static 'js/custom.js' %}"></script>
<script>
  $('#search-input').on('input', function() {
    var query = $(this).val();
    if (query.length > 2) {
        $.ajax({
            url: '{% url "autocomplete" %}',
            data: { q: query },
            success: function(data) {
                var suggestions = '';
                if (data.products.length > 0) {
                    suggestions += '<div class="autocomplete-suggestions">';
                    $.each(data.products, function(index, product) {
```

```
                    suggestions += '<div class="autocomplete-suggestion" data-url="' + product.url +
'">' + product.name + '</div>';
                });
                suggestions += '</div>';
            }
            if (data.categories.length > 0) {
                suggestions += '<div class="autocomplete-suggestions">';
                $.each(data.categories, function(index, category) {
                    suggestions += '<div class="autocomplete-suggestion" data-url="' + category.url
+ '">' + category.name + '</div>';
                });
                suggestions += '</div>';
            }
            $('.search-form').next('.autocomplete-suggestions').remove();
            $('.search-form').after(suggestions);
        }
    });
  } else {
    $('.search-form').next('.autocomplete-suggestions').remove();
  }
 });
 $(document).on('click', '.autocomplete-suggestion', function() {
   var url = $(this).data('url');
   window.location.href = url;
 });
 $("form").on("submit", function (e) {
  if($('#search-input').val().length < 3){
   e.preventDefault();
  }
 });
 </script>
</body>
</html>
```

А.2 Шаблон сторінки категорії з товарами category_products.html

```
{% extends 'main.html' %}
{% load static %}
{% load category_tags %}
{% block title %}Товари в категорії {{ category.name }}{% endblock %}
{% block custom_css %}
<style>
.catalog-settings{
    align-items: flex-end;
}
.selected-label{
    align-self: flex-end;
    margin-top: 0;
    margin-bottom: 0;
}
.catalog-settings-sort{
    margin-left: auto;
}
.card-body .card-image-wrapper{

    height: 100%;
    width: 100%;
    max-width: 250px;
    max-height: 250px;
}
.card-body .card-image-wrapper img{
    object-fit: contain;
    max-height: 100%;
    max-width: 100%;
}
.card-body .card-properties-wrapper .btn{
    margin-left: auto;
}
@media (min-width: 576px) and (max-width: 1200px) {
```

```
    .card-body .card-properties-wrapper{
        flex-direction: column;
    }
    .card-body .card-properties-wrapper .btn {
        width: 100%;
    }
}
@media (max-width: 1024.98px) {
    aside {
        visibility: hidden;
        width: 0!important;
        margin: 0!important;
        padding: 0!important;
        box-shadow: none!important;
    }
}
.row .row{
    padding: 0;
    margin: 0;
}
.price-range>:not(:last-child) {
    margin-right: 5px;
}
.filter-header-title{
    margin-right: auto;
}
.filter:has(.collapse.show) > .filter-header .bi-caret-down-fill{
    transform: rotate(180deg);
}
.btn-primary{
    background-color: var(--secondary1-color);
}
.card-text{
    margin:0;
}
```

```
.card{
   border-radius:0;
}
.card-body{
   flex-grow: 0 !important;
}
</style>
{% endblock %}
{% block content %}
<div class="page-title">
   <h1>{{ category.name }}</h1>
</div>
<div class="row">
   <aside class="col-md-3">
      <h2>Фильтри</h2>
      <form id="filter-form" method="get">
         <input type="hidden" name="order_" value="pop">
         <div class="mb-2 filter">
            <div class="filter-header d-flex flex-row" data-bs-toggle="collapse" data-bs-
target="#price-filter-body">
               <label class="filter-header-title">Ціна</label>
               <i class="bi bi-caret-down-fill"></i>
            </div>
            <div class="collapse price-body show" id="price-filter-body">
               <div  class="d-flex flex-row price-range mb-1">
                  <span>Від</span>
                  <input type="number" name="price_min" value="7677" class="form-range"
id="id_price_0">
                  <span>до</span>
                  <input type="number" name="price_max" value="77777" class="form-range"
id="id_price_1">
                  <span>грн</span>
               </div>
               <div id="price-slider" class="mb-2"></div>
            </div>
```

```
            <!--<label for="id_price_0">Минимальная цена:</label>
            <input type="number" id="id_price_0" name="price_min" class="form-control"
min="0" step="0.01" value="{{ current_min_price }}">
            <label for="id_price_1">Максимальная цена:</label>
            <input type="number" id="id_price_1" name="price_max" class="form-control"
min="0" step="0.01" value="{{ current_max_price }}">-->
        </div>
        <div class="mb-2 filter">
            <div class="filter-header d-flex flex-row" data-bs-toggle="collapse" data-bs-
target="#id_brand">
                <label class="filter-header-title">Бренд</label>
                <i class="bi bi-caret-down-fill"></i>
            </div>
            <div class="collapse show" id="id_brand">
                <div>
                    <label for="id_brand_0"><input type="checkbox" name="brand" value="1"
id="id_brand_0">Apple</label>
                </div>
                <div>
                    <label for="id_brand_1"><input type="checkbox" name="brand" value="2"
id="id_brand_1">Lenovo</label>
                </div>
            </div>
        </div>
        <div class="mb-3">
            <label>{{ field.label }}</label>
            {{ field }}
        </div>
        {% for field in filter.form %}
            <div class="mb-3">
                <label>{{ field.label }}</label>
                {{ field }}
            </div>
        {% endfor %}
        <button type="submit" class="btn btn-primary">Шукати</button>
```

```
    </form>
  </aside>
  <div class="flex-grow-1 col-md-9 row">
    <div class=" catalog-settings d-flex flex-row ">
      <div class="catalog-selected-filters d-flex align-self-center">
        <div class="d-flex flex-row">
        <p class="selected-label">Вибрано  товарів</p>
        <ul class="selected-filters d-flex flex-row">
        </ul>
      </div>
      </div>
      <div class="catalog-settings-sort d-flex align-self-center">
        <span class="pe-1">Сортувати</span>
        <div class="dropdown">
          <a class="dropdown-toggle" href="#" role="button" sort data-bs-
toggle="dropdown" aria-expanded="false">
            {% if request.GET.sort == "priceup"%}від дорогих до дешевих{% else %}від
дешевих до дорогих{% endif %}
          </a>
          <ul class="dropdown-menu">
            <li><a class="dropdown-item" sort-val="pricedown"
href="?sort=pricedown">від дешевих до дорогих</a></li>
            <li><a class="dropdown-item" sort-val="priceup" href="?sort=priceup">від
дорогих до дешевих</a></li>
          </ul>
        </div>
      </div>
      <div class="catalog-settings-display  ">
        <a href="/" class="btn btn-primary active"><span class="material-symbols-rounded
icon-md-30">apps</span></a>
        <a href="/" class="btn btn-primary "><span class="material-symbols-rounded icon-
md-30">format_list_bulleted</span></a>
      </div>
    </div>
    <div class="row catalog-products">
```

```
        {% if not products %}
        <h2 class="mt-4">Товари відсутні</h2>
        {% endif %}
        {% for product in products %}
          <!-- Карта товара -->
              <div class="col-lg-4 col-sm-6 col-xs-12 card card-body d-flex flex-column">
                <a class="card-image-wrapper d-flex justify-content-center mx-auto" href="{% url "catalog:product-page" category.slug product.slug %}">
                  {% if product.image %}
                  <img class="col-12" src="{{ product.image.url }}" width="250" height="250" alt="{{ product.name }}">
                  {% else %}
                  <img class="col-12" src="{% static "img/no_image.png" %}" width="250" height="250">
                  {% endif %}
                </a>
                <div class="card-title-wrapper d-flex">
                  <h5 class="card-title"><a href="{% url "catalog:product-page" category.slug product.slug %}">{{ product.name|truncatechars:100 }}</a></h5>
                </div>
                <div class="card-properties-wrapper d-flex align-items-center mt-auto">
                  {% if product.num_items == 1 %}
                  <div class="d-flex flex-column">
                    <p class="card-text">{{ product.get_min_price }} грн.</p>
                    <a class="card-text" href="{{product.products_in_store.first.store_product_url}}">{{ product.products_in_store.first.store.name }}</a>
                  </div>
                  <a href="{{product.products_in_store.first.store_product_url}}" class="btn btn-primary">Купити</a>
                  {% else %}
                  <div class="d-flex flex-column">
                    <p class="card-text">Від {{ product.get_min_price }} до {{ product.get_max_price }} грн.</p>
                    <a class="card-text" href="{% url "catalog:product-page" category.slug
```

```
product.slug %}">Всі пропозиції ({{ product.num_items }})</a>
                        </div>
                        <a href="{% url "catalog:product-page" category.slug product.slug %}"
class="btn btn-primary">Порівняти</a>
                        {% endif %}


                    </div>
                </div>
            {% endfor %}
            {% for product in empty_products %}
              <!-- Карта товара без пропозицій-->
                <div class="col-lg-4 col-sm-6 col-xs-12 card card-body d-flex flex-column ">
                  <a class="card-image-wrapper d-flex justify-content-center mx-auto" href="{%
url "catalog:product-page" category.slug product.slug %}">
                    {% if product.image %}
                    <img class="col-12" src="{{ product.image.url }}" width="250"
height="250" alt="{{ product.name }}">
                    {% else %}
                    <img class="col-12" src="{% static "img/no_image.png" %}" width="250"
height="250">
                    {% endif %}
                  </a>
                  <div class="card-title-wrapper d-flex">
                    <h5 class="card-title"><a href="{% url "catalog:product-page" category.slug
product.slug %}">{{ product.name|truncatechars:100 }}</a></h5>
                  </div>
                  <div class="card-properties-wrapper d-flex align-items-center mt-auto">
                    <p class="card-text">Немає пропозицій</p>
                  </div>
                </div>
            {% endfor %}
            {% for product in filter.qs %}
              <div class="col-md-4">
                <div class="card mb-4">
                  <div class="card-body">
```

```
                    <h5 class="card-title">{{ product.name }}</h5>
                    <p class="card-text">{{ product.get_min_price }} - {{ product.get_max_price
}} грн.</p>
                    <a href="{% url 'product_detail' product.id %}" class="btn btn-
primary">Порівняти</a>
                  </div>
                </div>
              </div>
            {% endfor %}
        </div>
        <nav aria-label="Page navigation example">
          <ul class="pagination justify-content-center">
            <li class="page-item {% if not products.has_previous %}disabled{% endif %} ">
              <a class="page-link" href="{% if products.has_previous %}?{% change_params
page=products.previous_page_number %}{% else %}#{% endif %}">Previous</a>
            </li>
            {% for page in products.paginator.page_range %}
            {% if page >= products.number|add:-3 and page <= products.number|add:3 %}
            <li class="page-item {% if products.number == page %}active{% endif %}"><a
class="page-link" href="?{% change_params page=page %}">{{page}}</a></li>
            {% endif %}
            {% endfor %}

            <li class="page-item {% if not products.has_next %}disabled{% endif %} ">
              <a class="page-link" href="{% if products.has_next %}?{% change_params
page=products.next_page_number %}{% else %}#{% endif %}">Next</a>
            </li>
          </ul>
        </nav>
    </div>
</div>
{% block extra_js %}
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
<script>
    $(function() {
```

```
        var minPrice = parseFloat("{{ min_price }}");
        var maxPrice = parseFloat("{{ max_price }}");
        var currentMinPrice = parseFloat("{{ current_min_price }}");
        var currentMaxPrice = parseFloat("{{ current_max_price }}");
        console.log(minPrice);
        console.log(maxPrice);
        console.log(currentMinPrice);
        console.log(currentMaxPrice);
        $("#id_price_0").val({{ current_min_price }});
        $("#id_price_1").val({{ current_max_price }});
        $("#price-slider").slider({
            range: true,
            min: minPrice,
            max: maxPrice,
            values: [currentMinPrice, currentMaxPrice],
            slide: function(event, ui) {
                $("#price-range").val(ui.values[0] + " - " + ui.values[1] + " грн");
                //$("#price_min").val(ui.values[0]);
                //$("#price_max").val(ui.values[1]);
                $("#id_price_0").val(ui.values[0]);
                $("#id_price_1").val(ui.values[1]);
            }
        });
        $("#price-range").val($("#price-slider").slider("values", 0) + " - " + $("#price-
slider").slider("values", 1) + " грн");
        //$("#price_min").val($("#price-slider").slider("values", 0));
        //$("#price_max").val($("#price-slider").slider("values", 1));
        $("#id_price_0").val($("#price-slider").slider("values", 0));
        $("#id_price_1").val($("#price-slider").slider("values", 1));
        $("#id_price_0").attr("type", "number");
        $("#id_price_1").attr("type", "number");
        $("#filter-form").on('submit', function(e) {
            e.preventDefault();
            updateProducts();
        });
```

```
        function updateProducts() {
            var form = $("#filter-form");
            $.ajax({
                url: form.attr('action'),
                data: form.serialize(),
                dataType: 'json',
                success: function(data) {
                    $("#product-list").html(data.html);
                }
            });
        };
        const searchParams = new URLSearchParams(window.location.search);
        let sort_value = searchParams.get('sort'); // "some_value"
        if (sort_value){
            $("a[sort]").text($("a[sort-val="+ sort_value +"]").text())
        }
    });
</script>
{% endblock %}
{% endblock %}
```

А.3 Шаблон сторінки категорії без товарів category_list.html

```
{% extends "main.html" %}
{% load static %}

{% block title %}Категорії{% endblock %}
{% block custom_css %}
<style>
    .category-navigation-item a:hover{
        color: var(--secondary1-color);
    }
    .category-navigation-item:hover{
```

```
        background-color: #f6f6f6;
    }
    .category-navigation-item:first-child{
        border-radius: 12px 12px 0 0;
    }
    .category-navigation-item:last-child{
        border-radius: 0 0 12px 12px;
    }
.subcategory-childs a:not(:last-child){
    position: relative;
}
.subcategory-childs a:not(:last-child)::after {
    content: ',';
}
.subcategory{
    padding: 10px;
}
.subcategory:not(:first-child){
    margin-top:10px;
}
.subcategory a:hover{
    color: var(--secondary1-color);
}
.box{
    /*border: solid 1px;*/
    border-radius: 12px;
    /*background-color: var(--secondary2-color);*/
    box-shadow: var(--box-shadow)
}
</style>
{% endblock custom_css %}

{% block content %}
```

```
<div class="page-title">
  <h1>{{category.name}}</h1>
</div>
<div class="sub-categories row flex-wrap">
   <aside class="col-xs-12 col-lg-3">
      <ul class="category-navigation box">
         {% for child in category.children.all %}
         <li class="category-navigation-item px-3 py-2">
            <a href="#{{child.slug}}" class="">
            {{child.name}}
            </a>
         </li>
         {% endfor %}
      </ul>
   </aside>
   <div class="col-xs-12 col-lg-9">
      {% for child in category.children.all %}
      <div class="subcategory box">
         <a class="col-xs-12 text-lg" id="{{child.slug}}" href="{% url 'catalog:category-page'
child.slug %}">
            {{child.name}}
         </a>
         <div class="subcategory-childs text-md" >
            {% for child_of in child.children.all %}
            <a href="{% url 'catalog:category-page' child_of.slug %}" >{{child_of.name}}</a>
            {% endfor %}
         </div>
      </div>
      {% endfor %}
   </div>
</div>

{% endblock %}
```

А.4 Шаблон сторінки товару product_list.html

```
{% extends "main.html" %}
{% load static %}
{% block custom_css %}
<style>
.content{
    padding: 10px;
}
.content img{
    max-height: 100%;
    max-width: calc(100% - 50px);
}
.card-image-wrapper{
    height: 100%;
    width: 100%;
    max-width: 450px;
    max-height: 450px;
}
.card-image-wrapper img{
    object-fit: contain;
    max-height: 100%;
    max-width: 100%;
}
.btn{
    width: fit-content;
}
.box{
    border-radius: 12px;
    box-shadow: var(--box-shadow);
    padding: 10px;
}
.tab-content{
    border-top: 0;
    border-top-left-radius: 0;
```

```
    border-top-right-radius: 0;
}
.tabs ul li button{
    width: 100%;
    padding: 10px;
}
.store-info{
    flex-shrink: 0;
}
.delivery-info{
    width: 100%;
}
.price-info{
    flex-shrink: 0;
}
table{
    width: 100%;
}
.tab-header{
    border-bottom: 1px solid var(--secondary4-color);
    padding-bottom: 10px;
}
.group-name{
    border-top: 1px solid var(--secondary4-color);
}
.char-table td {
    padding: 5px 0;
}
.store-logo img{
    height:auto;
    max-width: 100px;
    border-radius: 12px;
}
.price-info{
    /*width: 150px;*/
```

```
}
.price-info-vals{
    text-align: end;
}
.price-info-vals .price{
    padding-right:0!important;
}
.price{
    color: var(--secondary1-color);
}
.old-price{
    text-decoration: line-through;
    color: var(--secondary2-color);
}
.btn-primary{
    background-color: var(--secondary1-color);
}
.btn-buy{
    width:100%;
}
.buy-wrapper{
    width: 150px;
}
.shop-product-card:not(:last-child){
    padding-bottom: 10px;
    margin-bottom: 10px;
    border-bottom: 1px solid var(--secondary4-color);
}
.shop-product-card div:not(:last-child){
    padding-right: 10px;
}
</style>
{% endblock custom_css %}
{% block title %}{{ product.name }}{% endblock %}
{% block content %}
```

```
<div class="mt-4 title">
   <h1> {{ product.name }}</h1>
</div>
<div class="mt-4 content-main ">
   <div class="row d-flex flex-row box">
      <div class="col-xs-12 col-md-6">
         <div class="card-image-wrapper d-flex justify-content-center mx-auto">
         {% if product.image %}
         <img class="col-12" src="{{ product.image.url }}"  alt="{{ product.name }}">
         {% else %}
         <img class="col-12" src="{% static "img/no_image.png" %}" >
         {% endif %}
         </div>
      </div>
      <div class="col-xs-12 col-md-6 d-flex flex-column">
         {% if products_in_stores|length > 1  %}
         <p><b>Всього пропозицій:</b> {{products_in_stores|length}} </p>
         <p><b>Ціни:</b> {{ min_price }} - {{ max_price }} грн</p>
         {% elif products_in_stores|length == 1 %}
         <p><b>Ціна:</b> {{ min_price }} грн</p>
         <p><b>Магазин:</b> {{ products_in_stores.first.store.name }} </p>
         <a class="btn btn-primary" href="{{products_in_stores.first.url}}">Купити</a>
         {% else %}
         <p><b>Немає пропозицій</b></p>
         {% endif %}
         <div class="desc">
            <p><b>Опис:</b></p>
            <p>{{ product.description }}</P>
         </div>
      </div>
   </div>
   <div class="tabs row mt-3">
      <ul class="nav nav-tabs p-0" id="productTabs" role="tablist">
         <li class="nav-item flex-grow-1 " role="presentation">
            <button class="nav-link {% if not products_in_stores %} active {% endif %}"
```

```
id="characteristics-tab" data-bs-toggle="tab" data-bs-target="#characteristics" type="button"
role="tab" aria-controls="characteristics" aria-selected="true">Характеристики</button>
        </li>
        {% if products_in_stores %}
        <li class="nav-item flex-grow-1 " role="presentation">
          <button class="nav-link {% if products_in_stores %} active {% endif %}" id="stores-
tab" data-bs-toggle="tab" data-bs-target="#stores" type="button" role="tab" aria-
controls="stores" aria-selected="false">Де купити ({{products_in_stores.count}}) </button>
        </li>
        {% endif %}
      </ul>
   <div class="tab-content box" id="productTabsContent">
      <div class="tab-pane fade {% if not products_in_stores %} show active {% endif %}"
id="characteristics" role="tabpanel" aria-labelledby="characteristics-tab">
        <h2 class="tab-header">Характеристики</h2>
          <table class="char-table">
          {% for group, characteristics in characteristics_groups.items %}
          {% if group.name != "Основне" %}
          <tr><td class="group-name" colspan="2"><b>{{ group.name}}</b></td></tr>
          {% else %}
          <tr>
            <td>Бренд:</td>
            <td>{{ product.brand.name }}</td>
          </tr>
          {% endif %}
            {% for characteristic in characteristics %}
            <tr>
              <td>{{ characteristic.characteristic.name }}:</td>
              <td>{{ characteristic.value }}</td>
            </tr>
            {% endfor %}
          {% endfor %}
          </table>
          {% comment %}
          {% for characteristic in characteristics %}
```

```
                <li>{{ characteristic.characteristic.name }}: {{ characteristic.value }}</li>
            {% endfor %}
            {% endcomment %}
        </div>
        {% if products_in_stores %}
        <div class="tab-pane fade show active" id="stores" role="tabpanel" aria-
labelledby="stores-tab">
            <div class="shop-cards">
                {% for prod in products_in_stores %}
                <div class="shop-product-card d-flex flex-row ">
                    <div class="store-logo">
                        <a href="{{ prod.store_product_url }}"><img
src="{{prod.store.logo.url}}"></a>
                    </div>
                    <div class="store-info">
                        <a class="text-md" href="{{ prod.store_product_url
}}">{{prod.nameInStore}}</a><br>
                        <a class="text-sm" href="{{ prod.store_product_url }}">{{
prod.store.url}}</a><br>
                    </div>
                    <div class="delivery-info">
                        {% if prod.guarantee %}
                        <p class="text-md">Гарантія</p>
                        <span>{{prod.guarantee}}</span>
                        {% endif %}
                    </div>
                    <div class="price-info d-flex flex-row align-items-center">
                        <div class="price-info-vals me-3">
                            {% if prod.discount %}
                            <div class="discount">
                                <div class="price text-lg">
                                {{prod.sell_price}} грн
                                </div>
                                <div class="old-price text-md">
                                    {{prod.price}} грн
```

```
                    </div>
                </div>
                {% else %}
                <div class="price text-lg">
                    {{ prod.sell_price }} грн
                </div>
                {% endif %}
            </div>
            <div class="buy-wrapper">
                <a class="btn btn-primary btn-buy">Купити</a>
            </div>
        </div>
    </div>
    {% endfor %}
  </div>
 </div>
 {% endif %}
 </div>
</div>
{% endblock %}
```

A.5 основний файл з кодом Javascript custom.js

```
function toActive1()
{
   document.querySelectorAll(".menu-categories__item").forEach(el => {
      el.classList.remove("active")
      el.querySelector(".category-content").classList.remove("active")
   })
   this.classList.add("active");
   this.querySelector(".category-content").classList.add("active");
}
let elementsArray = document.querySelectorAll(".menu-categories__item");
```

```
elementsArray.forEach(function(elem) {
    elem.addEventListener("mouseover", toActive1);
});
function toActive2()
{
    document.querySelectorAll(".menu-categories__item1").forEach(el => {
        el.classList.remove("active")
        el.querySelector(".category-content1").classList.remove("active")
    })
    this.classList.add("active");
    this.querySelector(".category-content1").classList.add("active");
}
let elementsArray1 = document.querySelectorAll(".menu-categories__item1");
elementsArray1.forEach(function(elem) {
    elem.addEventListener("mouseover", toActive2);
});
function toActive3()
{
    document.querySelectorAll(".menu-categories__item2").forEach(el => {
        el.classList.remove("active")
    })
    this.classList.add("active");
}
let elementsArray2 = document.querySelectorAll(".menu-categories__item2");
elementsArray2.forEach(function(elem) {
    elem.addEventListener("mouseover", toActive3);
});
function changeI(){
    if (this.classList.contains("off")){
        this.classList.remove("off")
        this.classList.add("on")
        this.querySelector("i").classList.replace("bi-caret-up-fill", "bi-caret-down-fill")
    }else{
        this.classList.remove("on")
        this.classList.add("off")
```

```
        this.querySelector("i").classList.replace("bi-caret-down-fill", "bi-caret-up-fill")
    }
}
function changeIcon(){
    if (this.classList.contains("active")){
        this.classList.remove("active")
    }else{
        this.classList.add("active")
    }
}
document.getElementById("lang-btn").addEventListener("click", changeI)
document.querySelectorAll(".footer-category-title").forEach(function(elem) {
    elem.addEventListener("click", changeIcon);
});
let that_one_carete = document.getElementById("category-caret")
let modal1 = document.getElementById("category-list")
let observer = new MutationObserver(function(mutations) {
    for (let mutation of mutations) {
      if (mutation.type === 'attributes') {
        if (modal1.getAttribute("style") == 'display: none;') {
          that_one_carete.classList.replace("bi-caret-up-fill", "bi-caret-down-fill");
        }
          else{
              that_one_carete.classList.replace("bi-caret-down-fill", "bi-caret-up-fill");
          }
      }
    }
  }
);
console.log("doneload")
```

ДОДАТОК Б

Контролер додатку каталогу

```
from django.shortcuts import get_object_or_404, render
from django.http import JsonResponse
from django.urls import reverse, reverse_lazy
from django.utils.html import format_html
from django.core.paginator import Paginator
from django.db.models import Q, Max, Min, Count
from django.db.models import Exists, OuterRef

from .models import Category, Product, ProductInStore, Characteristic, CharacteristicValue,
CharacteristicGroup, Store, CharacteristicPreset, Brand
from django.db.models import Min, Max

def category(request, category_slug):
    category = get_object_or_404(Category, slug=category_slug)
    context = {
        'category': category,
    }

    breadcrumbs = get_breadcrumbs(category)
    context['breadcrumbs'] = breadcrumbs

    if category.children.exists():
        context['children'] = category.name
        return render(request, 'catalog/category_list.html', context)
    else:
        page = request.GET.get('page', 1)
        sort = request.GET.get('sort', 'pricedown')

        sorted_items = category.products.annotate(min_price=Min('products_in_store__price'),
max_price=Max('products_in_store__price'), num_items=Count('products_in_store'))
```

```
        print(sorted_items)
        empty_items = sorted_items.filter(num_items=0)
        sorted_items = sorted_items.exclude(num_items=0)
        if sort == 'pricedown':
            sorted_items = sorted_items.order_by('min_price', 'max_price')
        else:
            sorted_items = sorted_items.order_by('-max_price','min_price' )
        paginator = Paginator(sorted_items, 24)
        current_page = paginator.page(page)
        context['products'] = current_page
        context['empty_products'] = empty_items
    return render(request, 'catalog/category_products.html', context)


def product(request, category_slug, product_slug):
    product = get_object_or_404(Product, slug=product_slug)
    context = {
        'product': product,
    }
    in_stores = ProductInStore.objects.filter(product=product)
    context['products_in_stores'] = in_stores

    min_price =
ProductInStore.objects.filter(product=product).aggregate(Min('price'))['price__min']
    max_price =
ProductInStore.objects.filter(product=product).aggregate(Max('price'))['price__max']
    context['min_price'] = min_price
    context['max_price'] = max_price

    breadcrumbs = get_breadcrumbs(product.category,product)
    context['breadcrumbs'] = breadcrumbs

    char_dict = {}
    characteristics_groups =
CharacteristicGroup.objects.filter(category=product.category).filter(name='Основне')
```

```
    for group in characteristics_groups:
        characteristic_values = CharacteristicValue.objects.filter(product=product,
characteristic__group=group)
        if len(characteristic_values) > 0:
            char_dict[group] = characteristic_values
    characteristics_groups =
CharacteristicGroup.objects.filter(category=product.category).exclude(name='Основне')
    for group in characteristics_groups:
        characteristic_values = CharacteristicValue.objects.filter(product=product,
characteristic__group=group)
        if len(characteristic_values) > 0:
            char_dict[group] = characteristic_values
    context['characteristics_groups'] = char_dict

    return render(request, 'catalog/product.html', context)

def get_breadcrumbs(category=None,item=None):
    breadcrumbs = [{'name': 'Головна', 'url': reverse('main:index'), 'dropdown': None}]

    if category:
        ancestors = category.get_ancestors(include_self=True)
        for ancestor in ancestors:
            siblings = ancestor.get_siblings(include_self=True)
            dropdown = None
            if siblings.count() > 1:
                dropdown = format_html(
                    '<div class="dropdown d-inline"><a href="{}" class="breadcrumb-item dropdown-
toggle" id="breadcrumb-{}" data-bs-toggle="dropdown" aria-expanded="false">{}</a><ul
class="dropdown-menu" aria-labelledby="breadcrumb-{}">',
                    reverse('catalog:category-page', args=[ancestor.slug]), ancestor.slug, ancestor.name,
ancestor.slug
                )
                for sibling in siblings:
                    dropdown += format_html(
                        '<li><a class="dropdown-item" href="{}">{}</a></li>',
```

```
                    reverse('catalog:category-page', args=[sibling.slug]), sibling.name
                )
            dropdown += '</ul></div>'
        breadcrumbs.append({'name': ancestor.name, 'url': reverse('catalog:category-page', args=[ancestor.slug]), 'dropdown': dropdown})

    if item:
        breadcrumbs.append({'name': item.name, 'url': None, 'dropdown': None})

    return breadcrumbs

def search(request):
    query = request.GET.get('q')
    products = Product.objects.filter(Q(name__icontains=query))
    brands = Brand.objects.filter(Q(name__icontains=query))
    stores = Store.objects.filter(Q(name__icontains=query))
    categories = Category.objects.filter(Q(name__icontains=query))
    breadcrumbs = [{'name': 'Головна', 'url': reverse('main:index')}, {'name': 'Пошук', 'url': '#'}]
    return render(request, 'catalog/search_results.html', {
        'products': products,
        'brands': brands,
        'stores': stores,
        'categories': categories,
        'query': query,
        'breadcrumbs': breadcrumbs,
    })

def autocomplete(request):
    query = request.GET.get('q', '')
    product_results = Product.objects.filter(name__icontains=query)[:5]
    category_results = Category.objects.filter(name__icontains=query)[:5]

    results = {
        'products': [{'id': product.id, 'name': product.name, 'url': reverse('catalog:product-page', args=[product.category.slug , product.slug])} for product in product_results],
```

```
        'categories': [{'id': category.id, 'name': category.name, 'url': reverse('catalog:category-page', args=[category.slug])} for category in category_results],
    }
    return JsonResponse(results)
```