ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Вебсайт інтернет-магазину для продажу велосипедів з використанням багатокритеріальних фільтрів та рекомендаційної системи»

Ступінь вищої освіти – бакалавр
Спеціальність – 122 «Комп'ютерні науки»
Освітня програма «Комп'ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Андрій МІЛЕВСЬКИЙ
Керівник: викладач кафедри комп'ютерних наук
Богдан БОЙКО

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп'ютерних наук
_____Сергій МІЧКІВСЬКИЙ
«____»____20___р

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Мілевський Андрій Олександрович

Тема роботи	Вебсайт інтернет-магазину для продажу велосипедів з використанням багатокритеріальних фільтрів та рекомендаційної системи (комплексна робота)
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробити вебсайт інтернет-магазину з продажу велосипедів. На сайті реалізувати використання багатокритеріальних фільтрів, рекомендаційну систему підбору подібних або зв'язаних товарів та велосипедний конфігуратор індивідуального велосипеду під замовлення.
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	1. Семенюк А.Я. Тенденції в розробці веб-застосунків/ А.Я. Семенюк, Д.О. Балдик //Держава, регіони, підприємництво: інформаційні, суспільно-правові, соціально-економічні аспекти розвитку: матеріали VI Міжнародної наукової конференції (5-6 грудня 2024 р., м. Київ).- Київ: Університет "КРОК", 2024. - URL: https://conf.krok.edu.ua/SRE/SRE-2024/paper/view/2370 2. Шаповалов С.М., Мічківський С.М. Розробка рекомендаційної системи для підбору співрозмовника в соціальній мережі // XII Міжнародна науково-технічна конференція «Проблеми Інформатизації» (м. Київ, 13 грудня 2018 року). – К.: Державний університет телекомунікацій, 2018. - URL: https://dut.edu.ua/uploads/l_1701_65845854.pdf 3. Смірнов І. Розробка веб додатків з використанням Python і Django - URL: https://webcase.com.ua/uk/blog/razrabotka-veb-prilozhenij-s-ispolzovaniem-python-i-django/
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне, системотехнічне або експериментальне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблему в галузі комп'ютерних наук, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання 27 грудня 2024 р.

Керівник Богдан БОЙКО

Здобувач освітнього ступеня бакалавра Андрій МІЛЕВСЬКИЙ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
Підготовчий етап			
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
Основний етап			
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
Завершальний етап			
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник Богдан БОЙКО

Здобувач освітнього ступеня бакалавра Андрій МІЛЕВСЬКИЙ

Мілевський А. О. Веб-сайт інтернет-магазину з продажу велосипедів, з використанням багатокритеріальних фільтрів та рекомендаційної системи.

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 - Комп'ютерні науки (освітня програма - Комп'ютерні науки) СО Бакалавр. - ВНЗ "Університет економіки та права "КРОК", Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2025.

Описано розробку веб-застосунку інтернет-магазину з продажу велосипедів, який забезпечує візуалізацію та управління контентом інтернет-магазину, реалізує багатокритеріальні фільтри, рекомендаційну систему, а також конфігуратор для створення індивідуального велосипеда під замовлення.

Ключові слова: велосипед, веб-застосунок, інтернет-магазин, конфігуратор, велосипедні компоненти, багатокритеріальний фільтр, Django, React, REST API, MySQL.

Milevskyi A. O. Website of an Online Bicycle Store with Multi-Criteria Filters and a Recommendation System.

Explanatory note for the qualification work in the specialty 122 – Computer Science (educational program “Computer Science”) for the Bachelor’s degree. – “University of Economics and Law “KROK””, Educational and Scientific Institute of Information and Communication Technologies, Department of Computer Science, Kyiv, 2025.

The development of a web application for an online bicycle store is described, which provides visualization and management of online store content, implements multi-criteria filters, a recommendation system, and a configurator for creating a fully customized bicycle to order.

Keywords: bicycle, web application, online store, configurator, bicycle components, multi-criteria filter, Django, React, REST API, MySQL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ 6

ВСТУП 8

РОЗДІЛ 1 ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ ВЕБ-САЙТУ ІНТЕРНЕТ-МАГАЗИНУ З ПРОДАЖУ ВЕЛОСИПЕДІВ, З ВИКОРИСТАННЯМ БАГАТОКРИТЕРІАЛЬНИХ ФІЛЬТРІВ ТА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ 10

1.1 Опис предметної області 10

1.2 Аналіз потенційних конкурентних переваг 12

1.3 Постановка завдання на кваліфікаційну роботу 16

Висновки до розділу 1 20

РОЗДІЛ 2 ПРОЕКТУВАННЯ 22

2.1 Проектування структури 22

2.2 Моделювання даних 27

2.3 Моделювання процесів 30

2.4 Математична модель 34

Висновки до розділу 2 35

РОЗДІЛ 3 РЕАЛІЗАЦІЯ 37

3.1 Особливості реалізації системи 37

3.2 Конструювання системи 49

3.3 Тестування системи 51

3.4 Використання системи 53

Висновки до розділу 3 59

ВИСНОВКИ 61

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 63

ДОДАТОК А ФРАГМЕНТИ ЛІСТИНГУ 68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API (Application Programming Interface) – це набір правил, протоколів та інструментів, які дозволяють різним програмам взаємодіяти один з одним. Завдяки API, програмісти можуть створювати більш складні та функціональні програми, використовуючи вже існуючі сервіси та програми. [2]

Django – високорівневий відкритий Python-фреймворк (програмний каркас) для розробки вебсистем. Названо його було на честь джазмена Джанго Рейнхардта (відповідно до музичних смаків одного зі засновників проєкту).[3]

Frontend – клієнтська частина веб-застосунку, що відповідає за інтерфейс користувача та взаємодію з сервером.[5]

Backend – серверна частина веб-застосунку, яка обробляє запити, взаємодіє з базою даних та забезпечує бізнес-логіку.[5]

MySQL – вільна система керування реляційними базами даних, яка була розроблена компанією «ТсХ» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. [6]

REST API (Representational State Transfer Application Program Interface) – архітектурний стиль, який дозволяє одному програмному забезпеченню спілкуватися з іншим програмним забезпеченням через мережу або на одному пристрої. Найпоширенішим використанням REST API є створення веб-сервісу. [7]

Конфігуратор — програмний інтерфейс або модуль, що дозволяє користувачеві створювати індивідуальну конфігурацію товару або послуги, обираючи доступні параметри та компоненти [8].

Конфігуратор велосипедів – функціональний модуль веб-застосунку, що дозволяє користувачам створювати індивідуальні велосипеди під замовлення шляхом вибору комплектуючих.

Багатокритеріальний фільтр – механізм для пошуку товарів за декількома критеріями одночасно (наприклад, тип велосипеда, матеріал рами, діаметр коліс тощо).

XAMPP – безкоштовна багатоплатформова збірка вебсервера з відкритим початковим кодом, що містить HTTP-сервер Apache, базу даних MariaDB, MySQL й інтерпретатори скриптів для мов програмування PHP та Perl, а також додаткові бібліотеки, що дозволяють запустити повноцінний вебсервер [39].

Django ORM (Object Relational Mapper) – це інструмент, що дозволяє представляти таблиці бази даних як об'єкти Python. Завдяки Django ORM розробники можуть працювати з базою даних, використовуючи звичний синтаксис Python, замість написання складних SQL-запитів [45].

ВСТУП

Сучасні технології вплинули на електронну комерцію, перетворивши її на одну з найбільш перспективних галузей. З розвитком онлайн-торгівлі створення ефективних веб-сайтів для продажу товарів стало важливим напрямком досліджень. Ключовими аспектами є зручність, функціональність, швидкість роботи та безпека веб-ресурсів.

Актуальність теми. Усе більше споживачів віддають перевагу онлайн-покупкам, оскільки це дозволяє економити час та отримувати доступ до широкого асортименту товарів. Велосипеди є популярним засобом пересування та активного відпочинку.

Сучасні покупці цінують можливість налаштовувати товар під свої потреби. У велотехніці це означає не лише вибір готових моделей, а й створення індивідуального велосипеда. Тому розробка інтернет-магазину з повноцінним конфігуратором, що дозволяє формувати унікальні моделі під замовлення, є важливим завданням. Додатково інтеграція зручних інструментів пошуку та фільтрації покращить процес вибору для користувачів.

Мета дослідження. Метою роботи є розробка сучасного веб-сайту для інтернет-магазину, який спеціалізується на продажу велосипедів. Сайт передбачає інтеграцію різноманітних функціональних елементів, таких як багатокритеріальні фільтри для пошуку товарів, система рекомендацій для підбору велосипедів за індивідуальними вподобаннями користувача та конфігуратор для створення індивідуальних велосипедів під замовлення. Це дозволить забезпечити широкий функціонал та безпечну роботу користувачів із платформою.

Завдання дослідження. Для досягнення поставленої мети необхідно визначити основні вимоги до функціональності сайту, розробити його архітектуру та дизайн, а також реалізувати веб-застосунок із використанням сучасних технологій. Важливим етапом є створення конфігуратора для індивідуального складання велосипедів, інтеграція багатокритеріальних

фільтрів і рекомендаційної системи. Окрім цього, потрібно забезпечити безпеку та оптимізацію сайту для швидкої роботи, після чого протестувати його функціональність і оцінити ефективність.

Об’єктом дослідження є процес створення та управління веб-ресурсом для онлайн-продажу товарів, методи фільтрації товарів, алгоритми рекомендацій та принципи побудови конфігураторів.

Предметом дослідження є веб-сайт інтернет-магазину для продажу велосипедів із використанням багатокритеріальних фільтрів, рекомендаційної системи та конфігуратора для створення велосипедів під замовлення.

Методи дослідження. Використано методи аналізу, проєктування та програмної реалізації веб-застосунків, а також сучасні технології веб-розробки та серверні рішення для збереження й обробки даних.

Використання інформаційних технологій. У ході розробки сайту використовувалися стандартні та авторські програмні рішення. Основними засобами реалізації стали фреймворки для клієнтської та серверної частини, а також системи управління базами даних.

Практичне значення результатів. Дослідження сприяє розвитку електронної комерції у сфері продажу велосипедів та персоналізованих товарів. Запропонована платформа з конфігуратором, багатокритеріальними фільтрами та рекомендаційною системою може стати основою для онлайн-магазинів, орієнтованих на індивідуальний підхід до клієнтів. Розроблені методи покращують взаємодію з користувачами, підвищують продажі та лояльність клієнтів.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку посилань (50 найменувань). Пояснювальна записка містить 21 рисунок. Загальний обсяг пояснювальної записки складає 78 сторінок, основний зміст викладено на 62 сторінках.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ ВЕБ-САЙТУ ІНТЕРНЕТ-МАГАЗИНУ З ПРОДАЖУ ВЕЛОСИПЕДІВ, З ВИКОРИСТАННЯМ БАГАТОКРИТЕРІАЛЬНИХ ФІЛЬТРІВ ТА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

1.1 Опис предметної області

Інтернет-торгівля є однією з найбільш динамічних галузей сучасної економіки. В умовах цифровізації бізнес-процесів і збільшення попиту на онлайн-покупки, з’являється потреба в створенні веб-сайтів, які забезпечують високий рівень зручності, ефективності та персоналізації для користувачів. Однією з таких категорій товарів, які активно продаються через інтернет, є велосипеди.

Велосипеди, як зручний та екологічний транспортний засіб, мають широкий спектр застосування, включаючи активний відпочинок, спортивні заняття та щоденне пересування по місту. З огляду на високий попит і різноманіття моделей, вибір велосипеда може бути складним для користувача, особливо якщо він не має достатнього досвіду в цій сфері. У зв’язку з цим, важливо створити інтерфейс інтернет-магазину, який дозволяє легко вибирати велосипед, враховуючи потреби користувача, без необхідності знати всі технічні деталі.

Основними труднощами при покупці велосипеда через інтернет є:

1. Вибір серед великої кількості моделей – велосипеди можуть відрізнятися за типами (гірські, дорожні, шосейні, міські), розмірами, матеріалами рами, типами амортизації, конструкцією коліс та іншими характеристиками.
2. Необхідність індивідуального підходу – покупці часто бажають отримати велосипед, що відповідає їхнім конкретним вимогам, тому вони потребують можливості налаштування компонувальних елементів.

Для полегшення цього процесу в інтернет-магазинах використовуються рішення, які допомагають покупцям легко знаходити оптимальний товар, а саме: багатокритеріальні фільтри та рекомендаційні системи. Багатокритеріальні фільтри дозволяють користувачам обирати товар за кількома критеріями одночасно, наприклад, за типом велосипеда, розміром коліс, матеріалом рами, ціною та іншими характеристиками. Це значно прискорює процес пошуку потрібного товару.

Рекомендаційні системи на основі аналізу поведінки користувача і історії його покупок чи переглядів дозволяють персоналізувати пропозиції і надавати найбільш релевантні варіанти товарів, що відповідають інтересам покупця.

Однією з важливих складових є також конфігуратор для складання індивідуальних велосипедів під замовлення. Цей інструмент дозволяє покупцю вибирати окремі компоненти велосипеда (раму, колеса, гальма, трансмісію, тощо) і створювати персоналізований велосипед відповідно до своїх потреб. Це особливо актуально для велосипедистів, які мають специфічні вимоги до характеристик, або для тих, хто хоче створити унікальний велосипед, який відповідає їхньому стилю та призначенню.

Також важливою складовою – кілька типів користувачів, кожен з яких має свої функціональні ролі, наприклад:

- Покупець, який може переглядати каталог, користуватись фільтрами, створювати індивідуальні конфігурації, додавати товари до кошика, оформлювати замовлення, зберігати налаштування, залишати відгуки;
- Контент-менеджер, який відповідає за наповнення платформи: додає нові товари, оновлює описи, завантажує зображення, слідкує за актуальністю даних;
- Адміністратор, який має повний доступ до системи, може керувати користувачами, налаштовувати ролі, модерацію, мати доступ до статистики та логів;

- Механік, як співробітник, відповідальний за збирання велосипедів за індивідуальними конфігураціями, які надходять через платформу. Він отримує точні специфікації та перевіряє сумісність компонентів перед складанням.

Таким чином, створення ефективного інтернет-магазину для продажу велосипедів передбачає не лише сучасні функціональні рішення, як-от багатокритеріальні фільтри, рекомендаційні системи та конфігуратори, але й скоординовану роботу фахівців, кожен з яких виконує специфічні задачі у межах веб-платформи.

У рамках даного проекту передбачається розробка веб-застосунку, що включатиме всі ці функціональні можливості, що дозволить значно покращити користувацький досвід при покупці велосипедів онлайн.

1.2 Аналіз потенційних конкурентних переваг

Інтернет-торгівля велосипедами є швидко зростаючим сегментом ринку, на якому конкуренція постійно посилюється. Враховуючи високу конкуренцію серед онлайн-магазинів, важливо визначити потенційні конкурентні переваги, які можуть зробити веб-сайт більш привабливим для споживачів і забезпечити його успіх на ринку.

Однією з головних переваг є зручність користування сайтом. Сучасні споживачі очікують, що процес покупки буде швидким, простим і зручним. Це особливо важливо для покупців, які шукають велосипед для активного відпочинку, спорту чи щоденної експлуатації, але не мають глибоких технічних знань. На рис 1.1 зображено веб-сайт магазину «Велопланета», в якому реалізований багатокритеріальний фільтр. За допомогою багатокритеріальних фільтрів користувач може швидко відфільтрувати моделі за різними параметрами, такими як тип велосипеда, розмір, ціна та інші характеристики. Це дозволяє легко знайти оптимальний варіант без необхідності переглядати весь асортимент.

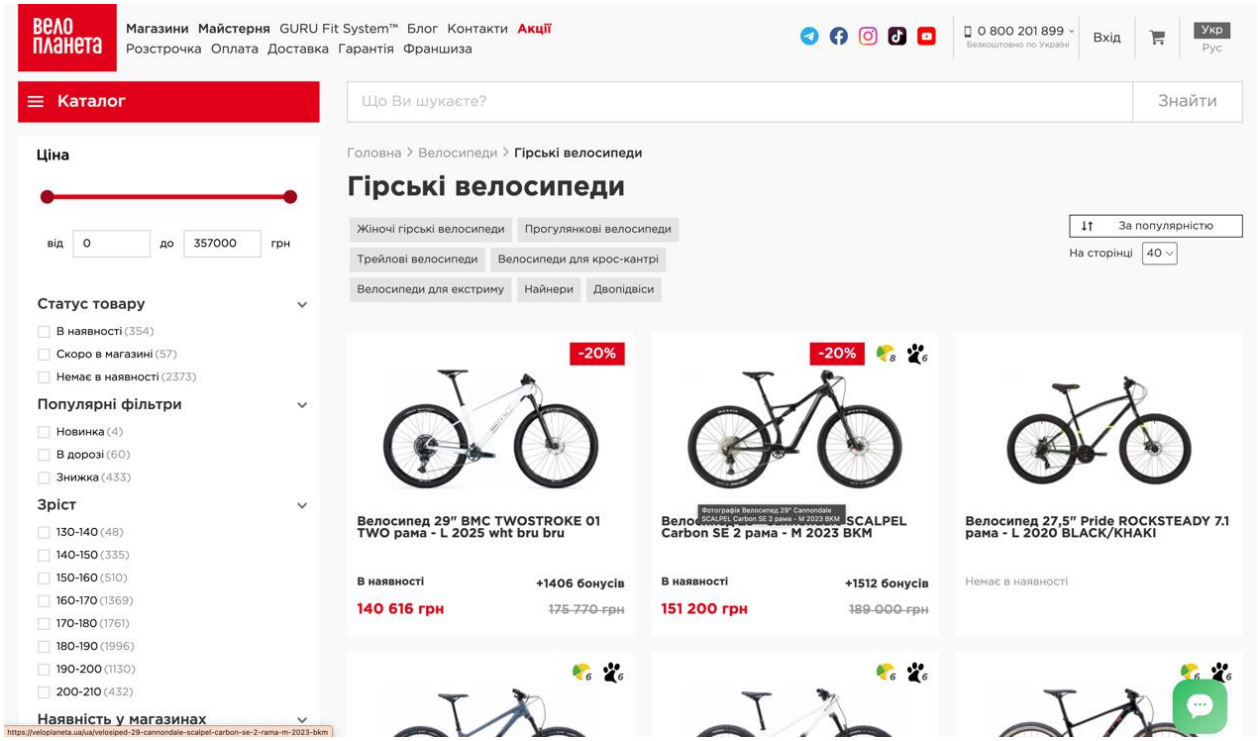


Рисунок 1.1 – Сторінка інтернет-магазину «Велопланета»

Джерело [9].

Персоналізовані рекомендації є ще однією важливою конкурентною перевагою. Рекомендаційна система, яка пропонує товари, виходячи з попередніх пошуків і поведінки користувача на сайті, може значно покращити процес покупки, зменшивши час на пошук потрібного товару. Такі системи можуть пропонувати товари або велосипеди, схожі на ті, які споживач переглядав або купував раніше, підвищуючи ймовірність успішної покупки та рівень задоволення від процесу.

Конфігуратор для створення індивідуальних велосипедів також є важливою конкурентною перевагою. Багато інтернет-магазинів пропонують тільки стандартні моделі велосипедів, не надаючи можливості налаштування індивідуальних варіантів. Конфігуратор дозволяє користувачам самостійно створювати велосипед, що повністю відповідає їхнім вимогам, вибираючи компоненти, такі як рама, підвіска, гальма, трансмісія, колеса та інші елементи.

На рис. 1.2 зображено веб-сайт Fanatik, на якому вже реалізовано конфігуратор, однак він має деякі недоліки, зокрема обмежений вибір дисциплін, що доступні для налаштування велосипедів, оскільки він орієнтований лише на гірські велосипеди. Крім того інтерфейс конфігуратора складний і незрозумілий, через те що елементи навігації та вибору параметрів компонентів розташовані незручно, подання не є зрозумілим для звичайного користувача, який не є експертом, а вибір виробників компонентів невеликий.

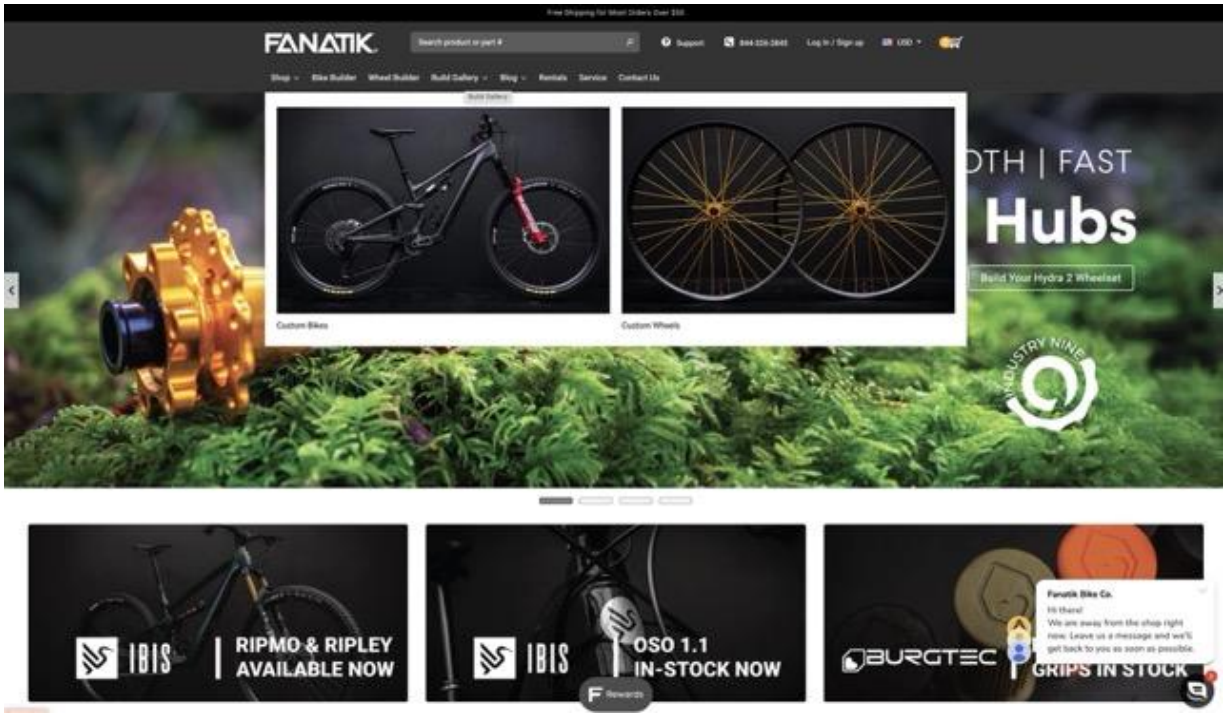


Рисунок 1.2 – Інтернет-магазин fanatikbike.com

Джерело [1].

Важливою можливістю є створення велосипеда для різних дисциплін, таких як шосейні гонки, ендуро, крос-кантрі, перегони на час, гравійні, туризм чи міська їзда. Це дає змогу як досвідченим велосипедистам, так і новачкам створювати моделі, що відповідають їхнім стилю катання, вимогам до продуктивності та умовам використання.

Ще однією важливою перевагою є система безпеки на веб-сайті, яка забезпечує захист особистих даних користувачів і безпеку їхніх транзакцій. У зв’язку з високими вимогами до конфіденційності в e-commerce, безпечна

платіжна система та захист персональних даних є обов’язковими для підтримки довіри споживачів. Веб-сайт, який надає клієнтам гарантію безпеки їхніх фінансових операцій та особистої інформації, буде більш привабливим для користувачів.

Сучасний і мінімалістичний дизайн є ще одним важливим фактором, що впливає на привабливість онлайн-магазину. Інтуїтивно зрозумілий інтерфейс із продуманим розташуванням елементів, естетичне оформлення та продумана колірна гама роблять сайт більш приємним для використання. Мінімалістичний стиль допомагає зосередитися на основному контенті, спрощує навігацію та створює відчуття преміальності. Використання сучасних UI/UX-рішень покращує взаємодію з платформою та позитивно впливає на загальне враження користувачів.

Швидкість роботи сайту та його оптимізація для мобільних пристроїв також є важливими конкурентними перевагами. Зважаючи на те, що значна частина покупок здійснюється через мобільні пристрої, веб-сайт повинен бути повністю адаптований до різних розмірів екранів і швидко завантажуватися. Це дозволяє клієнтам зручно робити покупки в будь-який час і з будь-якого пристрою.

Також, підтримка клієнтів є важливим фактором, який може значно підвищити конкурентоспроможність. Надання швидкої та ефективної допомоги наприклад через онлайн-чат, телефон або електронну пошту дозволяє швидко вирішувати питання клієнтів, підвищуючи їхню задоволеність, довіру та лояльність до бренду.

Отже, основними конкурентними перевагами інтернет-магазину є сучасний інтерфейс, персоналізовані рекомендації, можливість створення індивідуальних велосипедів для різних дисциплін через конфігуратор, мінімалістичний дизайн, швидка робота сайту, безпека даних, а також якісна підтримка клієнтів. Завдяки цим факторам магазин зможе привабити більше покупців і виділитися серед конкурентів.

1.3 Постановка завдання на кваліфікаційну роботу

Загальний опис завдання: метою роботи є розробка функціонального веб-сайту інтернет-магазину для продажу велосипедів, що містить багатокритеріальні фільтри, систему рекомендацій та повноцінний конфігуратор, який дозволяє користувачам створювати індивідуальні моделі велосипедів під власні потреби. Це забезпечить зручний вибір товарів, персоналізований підхід та покращений користувацький досвід. Конфігуратор має підтримувати різні дисципліни такі як:

- XC (Cross-Country / Крос-кантрі) – легкі та швидкі гірські велосипеди для перегонів по пересіченій місцевості.
- Enduro – велосипеди для складних спусків та технічних трас з потужною підвіскою.
- DH (Downhill / Даунхілл) – велосипеди для екстремального спуску з гори на високій швидкості, з довгохідною амортизацією.
- Road (Шосейні) – велосипеди для високошвидкісної їзди по асфальтованих дорогах.
- TT (Time Trial / Тріатлон) – велосипеди для роздільного старту та тріатлону, з аеродинамічними характеристиками.
- Gravel (Гревел) – універсальні велосипеди, що поєднують риси шосейних та МТБ, пристосовані для їзди по змішаних покриттях (асфальт, ґрунт, гравій).
- City & Touring – комфортні велосипеди для міських поїздок та велотуризму, що забезпечують зручність, ергономіку та практичність.

Сайт також матиме сучасний, мінімалістичний та адаптивний дизайн, який покращить навігацію, зменшить час пошуку товарів і зробить процес покупки максимально комфортним.

Вхідні дані:

1. Реєстрація та авторизація користувачів:

Користувачі реєструються на сайті за допомогою електронної пошти. Для цього необхідно вказати ім’я, прізвище, електронну пошту, пароль.

2. Профіль користувача:

Після реєстрації покупці можуть зберігати улюблені моделі велосипедів, конфігурації, переглядати історію покупок і налаштовувати вподобання. Змінювати або додавати аватар, завантаживши фото.

3. Конфігуратор велосипедів:

У конфігураторі користувач вводить свої особисті параметри структури тіла (зріст та внутрішня довжина ноги), потім користувач вибирає бажану дисципліну та конфігуратор підбирає йому велосипедні рами відповідно до заданих параметрів користувача і дає змогу вибирати компоненти поетапно для збору індивідуального велосипеда, відповідно до обраної дисципліни (TT, XC, Enduro, DH, Road, Gravel, City & Touring). Кожен компонент автоматично підбирається за сумісністю, і користувач може змінювати конфігурацію в реальному часі.

4. Пошук та фільтрація:

Користувачі можуть застосовувати багатокритеріальні фільтри для пошуку велосипедів. Фільтри дозволяють вибирати велосипеди за різними параметрами, такими як тип велосипеда та його цільове призначення, матеріал рами, колір рами, розмір рами, розмір коліс, тип трансмісії, тип гальм, тип амортизації, ціна тощо.

5. Рекомендаційна система:

Система аналізує попередні перегляди, покупки та активність користувача на сайті, щоб запропонувати найбільш релевантні моделі велосипедів, аксесуари та інші товари.

6. Оформлення замовлення:

Покупці можуть оформити замовлення, вибравши спосіб оплати з списку доступних платіжних методів та можуть вибрати спосіб доставки.

Після оформлення замовлення користувач може відслідковувати статус замовлення через персональний кабінет.

Вихідні дані:

1. Система рейтингу та відгуків:

Користувачі можуть оцінювати велосипеди за такими критеріями, як якість, зручність, ціна. Відгуки користувачів допомагають іншим покупцям приймати рішення щодо покупок, а також сприяють поліпшенню роботи магазину.

2. Персоналізовані рекомендації:

На основі аналізу діяльності користувача, система пропонує найбільш релевантні моделі велосипедів, аксесуари та комплектуючі, враховуючи попередні покупки, перегляди та інші параметри, а також дисципліну, яку вибирає користувач.

3. Збережені конфігурації:

Користувачі будучи авторизовані можуть зберігати створені конфігурації велосипедів і повернутися до них пізніше для завершення покупки або для зміни компонентів, базуючись на попередніх виборах.

4. Історія покупок:

Користувачі можуть переглядати історію своїх замовлень, повторно купувати товари, а також відслідковувати зміни в цінах, акціях та знижках, що робить процес покупки більш зручним.

5. Сучасний мінімалістичний дизайн:

Веб-сайт має зручний інтерфейс з зрозумілою навігацією, що дозволяє користувачам швидко знайти потрібну модель велосипеда чи аксесуар. Дизайн сайту використовує мінімалістичний стиль, щоб не відволікати користувачів від основного контенту.

Завдання для реалізації проєкту:

– аналіз предметної області – дослідження ринку онлайн-продажу велосипедів, вивчення конкурентних рішень, вивчення побудови велосипедів, дослідження технологій, нюансів, сумісності

компонентів та дослідження доступних компонентів в цілому, які в них використовуються.

- проєктування архітектури системи – визначення структури сайту, стеку технологій, моделювання бази даних, логіки взаємодії між компонентами і бізнес-логіки в цілому.
- реалізація конфігуратора велосипедів – створення механізму, який дозволяє покупцям покроково обираючи компоненти, збирати велосипед відповідно до своїх вимог.
- наповнення бази даних товарами та компонентами.
- інтеграція багатокритеріальних фільтрів – забезпечення швидкого пошуку товарів за різними параметрами.
- розробка системи рекомендацій – створення алгоритму, що пропонуватиме релевантні товари на основі вподобань та історії взаємодій.
- розробка серверної частини – керування товарами, користувачами, замовленнями та взаємодія конфігуратора з базою даних.
- оптимізація та тестування – перевірка швидкодії, безпеки та стабільності роботи сайту.

Методи виконання роботи:

1. Розробка веб-сайту інтернет магазину з продажу велосипедів
2. Розробка конфігуратора з динамічним підбором сумісних компонентів велосипеда, автоматичним розрахунком ціни та ваги.
3. Реалізація алгоритмів фільтрації та рекомендацій.
4. Автоматизоване та ручне тестування сайту.

Очікуваний результат: веб-сайт інтернет-магазину велосипедів забезпечить користувачам зручний процес вибору велосипедів, завдяки інтеграції багатокритеріальних фільтрів, системи рекомендацій та мінімалістичному інтерфейсу. Конфігуратор дозволить покупцям створювати індивідуальні моделі велосипедів, підбираючи компоненти відповідно до своїх потреб та обраної дисципліни (XC, Enduro, DH, Road, TT, Gravel, City &

Touring). Користувачі зможуть вводити свої персональні параметри, такі як зріст та внутрішня довжина ноги, щоб отримати відповідну раму та компоненти для свого велосипеда. Всі елементи конфігуратора будуть автоматично підбиратися за сумісністю, що дозволить здійснити вибір в реальному часі.

Персоналізовані рекомендації на основі попередніх переглядів та покупок значно спростять процес вибору товарів, пропонуючи найбільш релевантні варіанти велосипедів та аксесуарів. Система оцінок і відгуків дозволить покупцям дізнатися більше про якість товарів і приймати більш обґрунтовані рішення.

Крім того, користувачі зможуть зберігати створені конфігурації, що дозволить повернутися до них пізніше для покупки або подальших змін. Багатокритеріальні фільтри нададуть можливість швидко знаходити товари за різними параметрами, такими як тип застосування велосипеда, матеріал рами, ціна та інші параметри.

Висновки до розділу 1

У першому розділі було обґрунтовано актуальність розробки веб-сайту інтернет-магазину для продажу велосипедів із використанням багатокритеріальних фільтрів, рекомендаційної системи та конфігуратора. Проведено аналіз предметної області, що виявив низку ключових проблем, зокрема складність вибору серед великої кількості моделей, потребу в індивідуальному підході до підбору компонентів та необхідність підвищення зручності користування онлайн-магазинами для недосвідчених користувачів.

Особлива увага приділяється ролі сучасних інструментів, які підвищують ефективність взаємодії користувача з платформою. Зокрема, багатокритеріальні фільтри дозволяють значно скоротити час пошуку відповідного товару, забезпечуючи можливість одночасного фільтрування за різними параметрами. Рекомендаційна система покликана підвищити релевантність результатів, персоналізуючи інтерфейс та пропонуючи

найбільш підходящі моделі на основі історії переглядів, покупок та поведінкових характеристик користувача.

Було проаналізовано важливі конкурентні переваги, до яких належать: наявність мінімалістичного інтерфейсу, адаптивність до мобільних пристроїв, швидкодія сайту, система підтримки користувачів, безпека даних, а також можливість персоналізованого налаштування моделей велосипедів відповідно до дисциплін (XC, DH, Road, Gravel тощо). Інтеграція конфігуратора значно розширює функціональні можливості платформи, дозволяючи покупцю формувати велосипед з урахуванням індивідуальних фізіологічних параметрів та стилю катання.

На основі проведеного аналізу було сформульовано мету та завдання кваліфікаційної роботи. У межах розробки передбачається створення повнофункціонального веб-застосунку, який включатиме фільтрацію, рекомендації, конфігурацію, оформлення замовлень, збереження конфігурацій, персональні кабінети користувачів та інші функції, необхідні для комфортної та ефективної онлайн-купівлі велосипедів.

Таким чином, розділ 1 заклав теоретичне та методологічне підґрунтя для подальшого проєктування й реалізації системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ

2.1 Проектування структури

Проєктування структури для веб-платформи інтернет-магазину з продажу велосипедів з використанням багатокритеріальних фільтрів, рекомендаційної системи та конфігуратором індивідуального велосипеду є одним з ключових етапів у створенні інформаційної системи. На цьому етапі нам потрібно визначити основні логічні блоки системи, способи їхньої взаємодії та підходи до організації обробки даних.

Були визначені такі функціональні вимоги до системи:

1. Онлайн-конфігуратор велосипедів:

– можливість вибору користувачем окремих компонентів (рама, колеса, сідло, гальма тощо) з метою створення індивідуального велосипеда;

2. Автоматична рекомендація сумісних компонентів:

– система має враховувати залежності між компонентами та відображати лише сумісні варіанти;

3. Динамічний розрахунок ціни [14]:

– загальна вартість формується у режимі реального часу відповідно до обраних користувачем елементів;

4. Збереження та редагування конфігурації [14]:

– передбачається можливість авторизованого користувача зберігати зібрані конфігурації у профілі та змінювати їх пізніше;

5. Оформлення замовлення:

– після створення конфігурації користувач може ініціювати процес замовлення з подальшим збереженням у системі;

Нефункціональні вимоги:

1. Швидкодія та оптимізація:

– інтерфейс має завантажуватись швидко, з мінімальною затримкою, зокрема на мобільних пристроях;

2. Інтуїтивна взаємодія:

– дизайн має бути мінімалістичним, логічним, зрозумілим користувачу без потреби в додатковому навчанні;

3. Можливість розширення:

– структура системи повинна дозволяти додавання нових компонентів, типів велосипедів, функцій без суттєвої зміни вже реалізованої логіки;

4. Безпечність:

– обов’язкове шифрування облікових даних, аутентифікація через сучасні протоколи (наприклад, JWT-токени) [12];

Зважаючи на функціональну специфіку інтернет-магазину з можливістю створення індивідуальної конфігурації велосипедів, особливу увагу треба приділити забезпеченню інтерактивної взаємодії з користувачем. динамічного відображення інформації, гнучкого моделювання компонентів та паралельної обробки запитів.

Під час формування структури системи були розглянуті декілька варіантів архітектурних підходів, такі як:

– монолітна архітектура – проста у реалізації, але недостатньо масштабована в умовах динамічного контенту [10];
– класична MVC-архітектура – дозволяє розділити модель, уявлення та логіку, однак в контексті сучасної веб розробки потребує адаптації [11];
– клієнт-серверна архітектура з RESTful API – забезпечує розділення фронтенду та бекенду та незалежну розробку[12];
– мікросервісна архітектура – доречна для високонавантажених систем, але є надлишковою для поточного масштабу задачі [13];

та було виконане їх порівняння, котре відображене в таблиці 2.1, в якій зазначені переваги і недоліки кожного з підходів.

Таблиця 2.1 – Порівняння архітектурних підходів

Архітектура	**Переваги**	**Недоліки**	**Доцільність використання**
Монолітна	- Проста реалізація - Швидкий запуск	- Слабка масштабованість - Важко підтримувати при зростанні проекту	Маленькі проєкти або MVP з обмеженим функціоналом
Класична MVC	- Чітке розділення логіки, уявлення і моделі - Добра структура коду	- Менш гнучка для SPA та сучасних frontend-фреймворків - Потребує адаптації	Традиційні веб-додатки, проєкти зі статичним інтерфейсом
Клієнт-сервер з RESTful API	- Незалежна розробка клієнта і сервера - Легка інтеграція - Гнучкість	- Потребує чіткої організації API - Зростає складність тестування і відлагодження	Сучасні веб-застосунки, SPA, проєкти з активною frontend-частиною
Мікросервісна	- Висока масштабованість - Незалежне розгортання сервісів	- Висока складність реалізації та підтримки - Потребує DevOps-інфраструктури	Високонавантажені системи, корпоративні рішення, великі команди

На основі проведеного досліду та враховуючи вищеописані особливості та вимоги, доцільним видається підхід, заснований на розділенні клієнтської та серверної частин, де клієнтська частина (інтерфейс користувача) відповідає за взаємодію з користувачем, а серверна – за обробку запитів, бізнес-логіку та взаємодію з базою даних через API. Такий підхід забезпечує високу

адаптивність, модульність та підтримуваність системи, а також сприяє майбутньому масштабуванню.

Таким чином, структура майбутньої системи має забезпечувати гнучке розширення, ефективну обробку користувацьких конфігурацій та високу якість обслуговування у режимі реального часу. Обґрунтовано можна припустити, що використання технологій, які підтримують архітектуру із RESTful API, з високою ймовірністю задовольнить визначені вимоги та дозволить побудувати адаптивну, функціонуючу платформу.

Була розроблена діаграма варіантів використання (Use Case Diagram), зображена на рис. 2.1. Вона дозволяє відобразити основні функціональні можливості системи з точки зору кінцевого користувача та інших зовнішніх акторів, а також визначити їхню взаємодію з програмним продуктом [15].

Метою побудови діаграми прецедентів є:

- виявлення основних сценаріїв використання системи (тобто її функціональних вимог);
- визначення ролей користувачів (акторів), які ініціюють ті чи інші функції;
- забезпечення розуміння вимог як для розробників, так і для замовників;
- створення основи для подальшого моделювання логіки роботи системи.

У контексті платформи для онлайн-конфігурації та замовлення велосипедів – діаграма прецедентів дозволяє формалізувати взаємодію таких акторів, як Користувач (User), Контент-менеджер (Content manager), Збирач (Assembler) та Адміністратор (Admin), із відповідними функціями системи.

Користувач має можливість зареєструватися, увійти до системи, переглядати велосипеди, використовувати конфігуратор, зберігати та змінювати конфігурацію, створювати замовлення й переглядати їхній статус.

Контент-менеджер відповідає за управління і додавання велосипедів та компонентів.

Збирач має доступ до перегляду замовлень і управління компонентами.

Адміністратор взаємодіє із системою для перегляду та редагування замовлень, управління користувачами, а також зміни статусу замовлень.

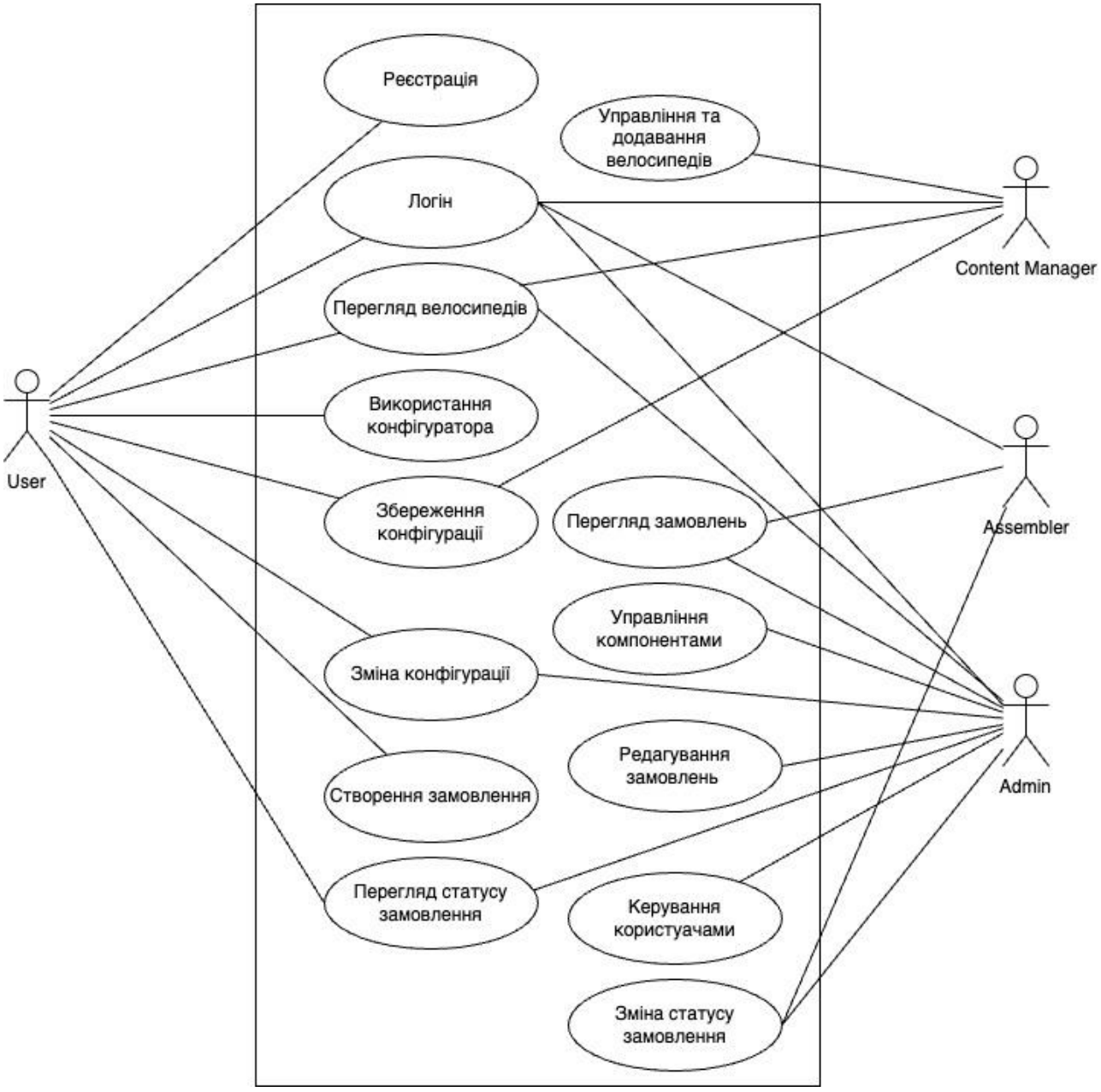


Рисунок 2.1 – Діаграма варіантів використання (Use Case Diagram)

Джерело: розроблено автором

2.2 Моделювання даних

Для представлення структури об’єктів системи та їх взаємозв’язків була побудована діаграма класів UML, яка відображає логіку взаємодії основних елементів платформи інтернет-магазину велосипедів з можливістю покупки як готових моделей, так і індивідуально зібраних конфігурацій (рис. 2.2).

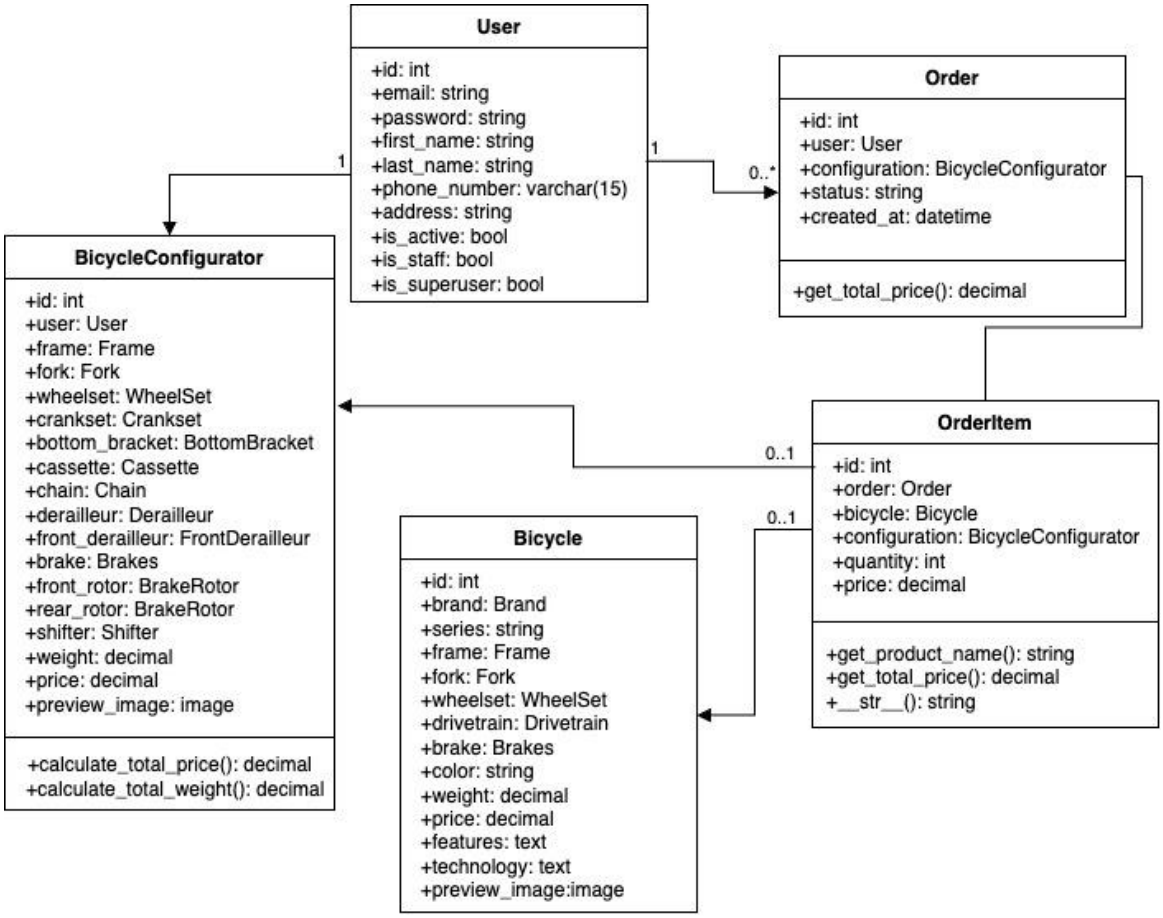


Рисунок 2.2 – Діаграма класів (Class diagram)

Джерело: розроблено автором

Дана модель реалізує об’єктно-орієнтований підхід до проектування, де кожна сутність системи відображена у вигляді класу з відповідними атрибутами та асоціаціями.

Основні класи системи:

– User — представляє зареєстрованого користувача. Містить атрибути: email, password, first_name, last_name,

shipping_address, is_staff, is_superuser. Користувач може мати зв’язки типу один-до-багатьох з класами Order та BicycleConfigurator.

- Bicycle — описує готову модель велосипеда з каталогу. Ключові атрибути включають: brand, series, color, frame, fork, transmission, brakes, weight, price, image_url. Використовується у замовленнях.
- BicycleConfigurator — клас, що зберігає індивідуально зібраний велосипед. Конфігурація формується з набору компонентів, а також зберігає загальні параметри: total_price, total_weight, image_preview. Має зв’язок з користувачем та використовується в замовленнях.
- Order — відображає оформлене замовлення. Має асоціацію з класом User та включає список об’єктів класу OrderItem (композиція або агрегація).
- OrderItem — представляє окрему позицію в замовленні. Має атрибути: quantity, price, product_type, product_id, а також зв’язок з Order. Залежно від product_type (наприклад, bicycle або config), посилається або на Bicycle, або на BicycleConfigurator.

Класи пов’язані між собою через відношення асоціації, зокрема:

- Один User може створити багато Order та багато BicycleConfigurator.
- Один Order складається з багатьох OrderItem.
- Кожен OrderItem містить посилання на один об’єкт типу Bicycle або BicycleConfigurator.

Об’єкти взаємодіють між собою згідно з бізнес-логікою платформи, з урахуванням наступних бізнес-вимог:

- зберігання готових велосипедів у каталозі;
- створення індивідуальних конфігурацій з окремих компонентів;
- можливість одного замовлення містити декілька як готових моделей, так і конфігурацій;
- прив’язка конфігурацій та замовлень до конкретного користувача;
- динамічний розрахунок ціни та ваги.

Також було побудовано ER-діаграму (Entity-Relationship Diagram), яка відображає логічну модель даних системи [16] (рис. 2.3), яка включає основні сутності: User, Order, OrderItem, Bicycle та BicycleConfigurator.

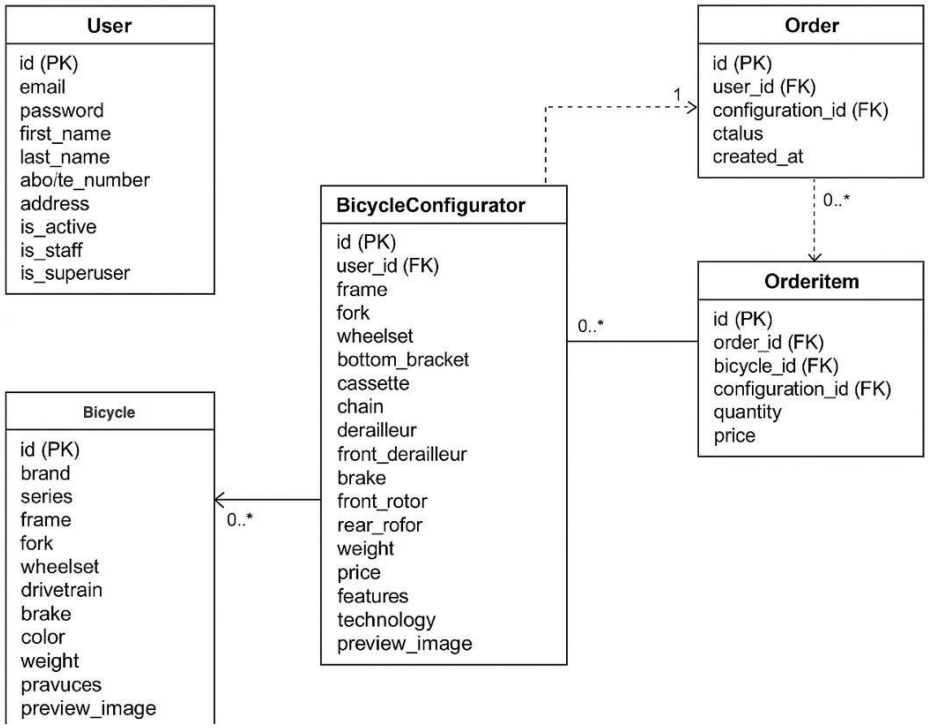


Рисунок 2.3 – ER-Діаграма

Джерело: розроблено автором

Основні сутності моделі:

• **User** – зареєстрований користувач платформи. Має атрибути: email, пароль, ім’я, прізвище, адреса доставки тощо. Один користувач може створювати багато замовлень та зберігати індивідуальні конфігурації. Роль користувача визначається за допомогою атрибутів is_staff та is_superuser, які регулюють рівень доступу до адміністративної частини платформи.

• **Bicycle** – готова модель велосипеда з каталогу. Містить характеристики: бренд, серія, колір, рама, вилка, трансмісія, гальма, вага, ціна, зображення тощо.

• **BicycleConfigurator** – індивідуальна конфігурація велосипеда, що формується користувачем із набору компонентів. Кожна конфігурація також зберігає вагу, ціну та зображення. Може бути використана у замовленні.

• **Order** – оформлене замовлення користувача. Має зв’язок з користувачем та включає один або більше елементів (позицій) через проміжну сутність OrderItem.

• **OrderItem** – окрема позиція у замовленні. Може посилатися або на готовий велосипед (Bicycle), або на конфігурацію (BicycleConfigurator). Має атрибути: кількість, ціна, ідентифікатор товару. Відповідно до бізнес-логіки, одна позиція може містити лише один тип товару.

2.3 Моделювання процесів

Для моделювання динаміки процесів використано діаграму діяльності, яка описує внутрішній процес прийняття рішень і виконання дій, а також умови переходів між окремими етапами.

У даному випадку розглянуто сценарій взаємодії користувача з системою при замовленні велосипеда. Діаграма діяльності користувача, яка зображена на рисунку 2.4, відображає два основних шляхи взаємодії користувача з системою:

1. Замовлення з каталогу: користувач заходить на сайт, переглядає перелік доступних моделей велосипедів, і за наявності прийнятного варіанту, оформлює замовлення без необхідності додаткової конфігурації.

2. Використання конфігуратора: у випадку, якщо доступні моделі не відповідають вимогам користувача, він переходить до конфігуратора. На початковому етапі вводяться базові дані про параметри велосипедиста (зріст, висота ноги), після чого користувач обирає цільове призначення велосипеду (дисципліну, наприклад: гірський, шосейний, міський велосипед, гравійний, тощо). Система на основі введених параметрів автоматично підбирає відповідну раму, після чого відкривається можливість поетапного обирання та налаштування всіх компонентів велосипеда (колеса, гальма, трансмісія тощо).

У завершальній фазі користувач має змогу зберегти конфігурацію або одразу оформити замовлення.

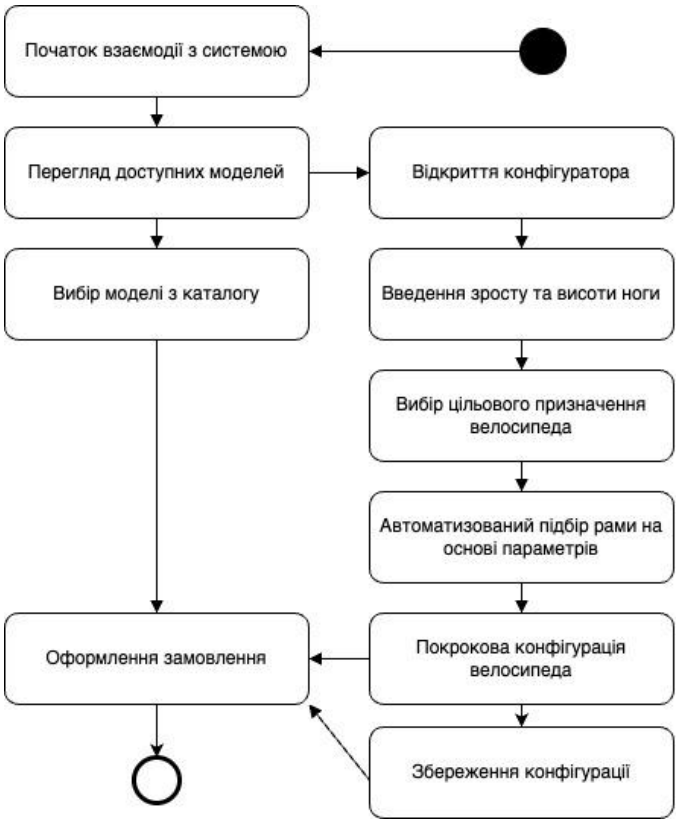


Рисунок 2.4 – Діаграма діяльності користувача (Activity Diagram)

Джерело: розроблено автором

Також було розглянуто сценарій взаємодії адміністратора. Діаграма діяльності адміністратора, яка зображена на рисунку 2.5 – відображає два основні напрями його роботи: керування замовленнями та керування користувачами, а також керування компонентами системи.

1. Керування замовленнями: після авторизації в панелі адміністратора – адміністратор може перейти до модуля управління замовленнями, де здійснюється перегляд замовлень, зміна їх статусу та редагування деталей кожного окремого замовлення. Це дозволяє оперативно реагувати на запити клієнтів і швидко усувати можливі проблеми, пов’язані з виконанням замовлень.

2. Керування компонентами: адміністратор має можливість переглядати список компонентів системи, редагувати їх і зберігати налаштування. У свою чергу, завдяки цьому зменшується ймовірність помилок при замовленні, зменшується час на вирішення проблем та спрощується підтримка системи.

3. Керування користувачами: адміністратор може відкривати модуль керування обліковими записами, де відображається список зареєстрованих користувачів, доступна функція редагування або видалення записів, а також призначення ролей. Це дозволяє підтримувати безпеку системи та ефективно розподіляти права доступу, швидко реагуючи на порушення або технічні збої.

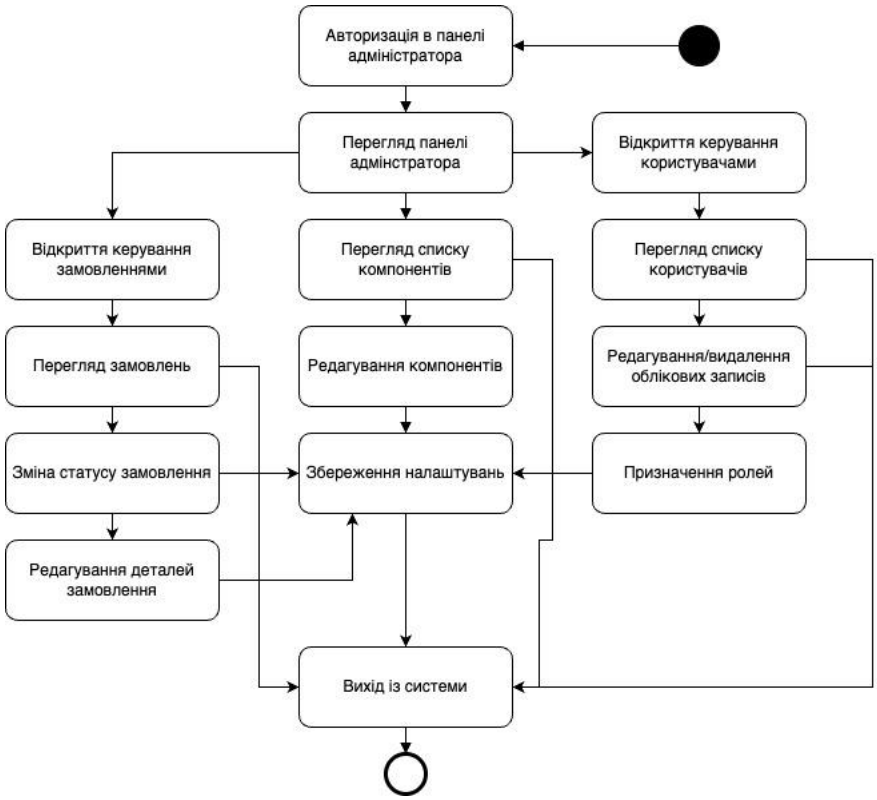


Рисунок 2.5 – Діаграма діяльності користувача (Activity Diagram)

Джерело: розроблено автором

На основі цих сценаріїв можна окреслити основні етапи логіки взаємодії, відобразити послідовність переходів між станами, врахувати можливі розгалуження та альтернативні шляхи виконання дій, а також, це спрощує подальше програмне моделювання.

Наступним етапом була розроблена діаграма послідовності, яка відображає хронологію обміну повідомленнями між учасниками системи під час виконання певного сценарію (рис. 2.6).

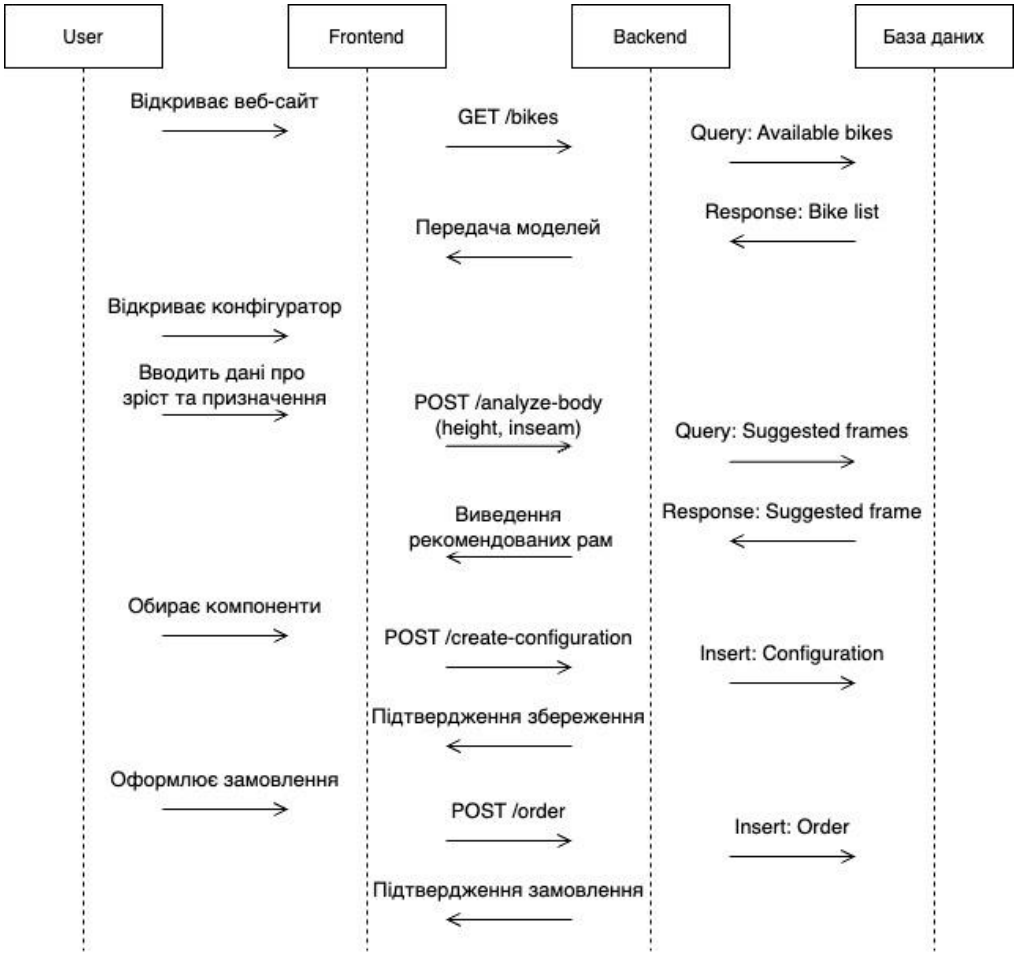


Рисунок 2.6 – Діаграма послідовності (Sequence Diagram)

Джерело: розроблено автором

У розроблюваній платформі ключовим є сценарій створення користувацького замовлення – як на основі вже готової моделі велосипеда, так і шляхом використання конфігуратора для складання індивідуального варіанту. Для опису цього сценарію доцільним є подання взаємодії таких об’єктів: користувач (User), інтерфейс користувача (Frontend), серверна частина (Backend) та база даних.

Діаграма включає такі основні етапи:

1. Початковий запит: користувач заходить на сайт. Інтерфейс надсилає запит на сервер для отримання списку доступних велосипедів. Сервер у свою чергу звертається до бази даних, отримує інформацію та повертає її на інтерфейс.

2. Робота з конфігуратором (за необхідності): якщо користувач не знаходить потрібної моделі, він відкриває конфігуратор, вводить свої фізіологічні параметри (зріст, висота ноги). Дані надсилаються на сервер, який звертається до бази даних для підбору відповідної рами. Результат передається назад на інтерфейс.

3. Формування конфігурації: користувач обирає бажані компоненти велосипеда. Інтерфейс надсилає сформовану конфігурацію на сервер, де вона зберігається в базі даних.

4. Оформлення замовлення: користувач підтверджує замовлення. Запит надсилається на сервер, який створює запис про замовлення у базі даних та повертає підтвердження.

2.4 Математична модель

Для визначення ефективного алгоритму рекомендаційної системи, яка функціонуватиме на головній сторінці платформи інтернет-магазину велосипедів, було проведено порівняння трьох основних підходів: контентно-орієнтованого фільтрування, колаборативного фільтрування та гібридного підходу. Кожен із них має свої переваги, недоліки та специфіку реалізації.

Контентно-орієнтований підхід (Content-Based Filtering) ґрунтується на аналізі властивостей товарів, з якими взаємодіє користувач. Наприклад, якщо користувач переглядав гірські велосипеди з алюмінієвою рамою та дисковими гальмами, система рекомендуватиме інші моделі з подібними характеристиками. Формально, для кожного товару формується вектор ознак, а рекомендації генеруються на основі метрик схожості (наприклад, косинусної подібності) між векторами [47][48].

Колаборативне фільтрування (Collaborative Filtering) базується на аналізі поведінки інших користувачів. Воно працює за принципом: «користувачі, схожі на вас, також купували». Такий підхід може бути реалізований у вигляді user-based або item-based фільтрації. Його перевага полягає в здатності виявляти нові, несподівані рекомендації, однак головним недоліком є проблема «холодного старту» — відсутність даних про нових користувачів або товари.

Гібридний підхід (Hybrid Filtering) поєднує в собі елементи обох методів, компенсуючи їхні обмеження. Наприклад, система може одночасно враховувати історію переглядів (content-based) та поведінку схожих користувачів (collaborative), формуючи рекомендаційний список із врахуванням обох джерел [49].

З урахуванням того, що веб-платформа перебуває на етапі запуску та накопичення користувацької активності є обмеженим, оптимальним вибором є контентно-орієнтований підхід. Він дозволяє формувати рекомендації на основі метаданих велосипедів (тип, дисципліна, вага, матеріал рами, тип гальм), незалежно від наявності історії переглядів або покупок інших користувачів. Надалі можлива інтеграція гібридної моделі для підвищення точності та різноманіття рекомендацій при збільшенні обсягу даних [50].

Висновки до розділу 2

У другому розділі було детально проаналізовано та спроєктовано архітектуру веб-платформи інтернет-магазину з продажу велосипедів, яка поєднує можливості багатокритеріального пошуку, рекомендаційної системи та онлайн-конфігуратора індивідуальних велосипедів. Основою запропонованого рішення стала клієнт-серверна архітектура з використанням RESTful API, що забезпечує гнучкість, модульність та високу масштабованість системи, а також полегшує її подальшу підтримку та розвиток.

У процесі проєктування було визначено функціональні та нефункціональні вимоги до системи, які охоплюють як технічні

характеристики, так і особливості користувацької взаємодії. Проведений аналіз архітектурних підходів дозволив обґрунтовано обрати оптимальну модель побудови системи, яка відповідає сучасним вимогам до веб-застосунків.

Узгоджене використання різних видів моделювання дало змогу сформувати цілісне уявлення про логіку роботи системи. Побудовані діаграми відобразили як структуру даних, так і динаміку процесів. Зокрема, діаграми прецедентів, діяльності та послідовності дозволили формалізувати поведінку користувача в системі, в той час як ER-діаграма та діаграма класів відобразили логічну й об’єктно-орієнтовану структуру даних, що лягає в основу реалізації.

Окрему увагу приділено рекомендаційній системі, для якої на основі порівняльного аналізу обрано контентно-орієнтований підхід. Такий алгоритм дозволяє аналізувати характеристики переглянутих товарів (тип велосипеда, бренд, дисципліна, гальма, трансмісія тощо) і знаходити подібні за властивостями моделі, які можуть зацікавити користувача. Це рішення є ефективним на початкових етапах наповнення системи та не потребує великого обсягу історичних даних, забезпечуючи персоналізовані рекомендації вже з перших сеансів взаємодії користувача з платформою.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ

3.1 Особливості реалізації системи

Реалізація веб-платформи інтернет-магазину з продажу велосипедів з конфігуратором індивідуального велосипеду здійснювалася з використанням сучасного стеку веб-технологій, який поєднує зручність, масштабованість та широкі можливості для інтеграції. Основний акцент зроблено на використанні Django як фреймворку серверної частини, що забезпечує швидку розробку, безпеку та чітку структуру коду, а також Django REST framework для побудови API [22][23].

Обрана версія – Django 5.2, оскільки вона є актуальною на момент розробки та має статус LTS (Long-Term Support), що гарантує тривалу підтримку, стабільність і оновлення безпеки. Крім того, ця версія містить покращену продуктивність, підтримку асинхронного виконання, сучасні API-можливості та сумісність із останніми версіями Python [40].

Вибір саме Django зумовлений кількома ключовими факторами:

– вбудована ORM-система дозволяє ефективно працювати з базою даних, що особливо важливо для реалізації логіки конфігуратора, де зберігаються тисячі варіантів компонентів і зв’язків сумісності між ними;
– швидкість розробки: завдяки генерації типових компонентів (моделей, форм, панелей адміністрування) Django значно скорочує час створення повноцінного веб-застосунку;
– безпека з коробки: фреймворк пропонує механізми захисту від основних типів атак (XSS, CSRF, SQL-ін’єкції тощо) без необхідності додаткової реалізації;
– масштабованість і розширюваність: завдяки модульній архітектурі Django підходить як для MVP, так і для розширених комерційних рішень з великою кількістю користувачів;

– велика спільнота та документація, що спрощує підтримку і подальший розвиток платформи.

Для реалізації RESTful API використано Django REST Framework (DRF), який є розширенням до Django і спрощує створення API. DRF надає інструменти для серіалізації моделей у формат JSON (і навпаки), має вбудовану систему аутентифікації та авторизації, а також підтримує функції, такі як пагінація, фільтрація, обмеження доступу і контроль навантаження. Це дозволяє ефективно реалізувати веб-сервіси на основі архітектури REST.

API забезпечує взаємодію між клієнтською частиною та сервером, дозволяючи отримувати, створювати, редагувати та видаляти дані відповідно до прав доступу користувача. Прикладом реалізації наведемо ендпоінт для отримання гірського велосипеду з каталогу:

– GET /api/mtb/27/

Призначення: отримати перелік даних гірського велосипеду з каталогу за ідентифікатором для детального перегляду.

Приклад відповіді:


```
[{
  "id": 27,
  "brand": "Cube",
  "frame": "Cube Reaction Pro (L, 470mm) - Hardtail",
  "fork": "RockShox Judy Silver TK 29\" 100mm/air",
  "wheelset": "Cube ZX20 / 29\" TLR:False",
  "drivetrain": "Shimano M8100 Drivetrain",
  "brake": "Shimano XT BR-M8100 hydraulic, 2 pistons",
  "handlebar": "Cube Flat Race Bar 720mm (31.8mmmm)",
  "detailed_images": [
    {
      "id": 1,
      "image_url":
"http://127.0.0.1:8000/media/bicycles/detailed/cube-reaction-slx-29-
slateblack-n-black-94920-02.jpg"
    }
  ],
  "series": "Reaction SLX 2025",
  "color": "slateblack´n´black",
  "weight": "13.300",
  "price": "1199.00",
  "features": "The CUBE Reaction SLX transforms every trail
adventure into a highlight. With precise Shimano XT 1x12-speed gearing
and the smooth RockShox Judy fork, this bike offers exceptional
```

```
control and riding pleasure. The powerful Shimano XT brakes ensure
outstanding safety and an unforgettable riding experience.",
       "technology": "",
       "preview_image":
"http://127.0.0.1:8000/media/bicycles/48d9b987-3cb8-4897-9ccc-
037944475702.jpg"
     }]
```


Ще одним прикладом виділимо створення замовлення:

– POST /api/orders/

Призначення: Створення нового замовлення.

Авторизація: Обов’язкова.

Тіло запиту:


```
{
  "user": 5,
  "items": [
    {
      "product_type": "config",
      "product_id": 12,
      "quantity": 1
    },
    {
      "product_type": "bicycle",
      "product_id": 3,
      "quantity": 2
    }
  ]
}
```


Приклад відповіді:


```
{
  "order_id": 154,
  "status": "created",
  "total_price": 32999,
  "created_at": "2025-04-18T12:34:56Z"
}
```


Усі запити та відповіді реалізовано у форматі JSON. Для авторизованих запитів використовується JWT або токен-аутентифікація, залежно від налаштувань сервера.

Було проведено порівняння кількох популярних систем керування базами даних (СУБД), зокрема MySQL, Oracle SQL та SQLite, з урахуванням таких критеріїв, як продуктивність, підтримка Django, масштабованість, доступність на хостинг-платформах і простота адміністрування (табл 3.1).

Таблиця 3.1 – Порівняння баз даних за критеріями

Критерій	**MySQL**	**Oracle SQL**	**SQLite**
Сумісність з Django ORM	Так	Часткова (через сторонні драйвери)	Так
Підтримка транзакцій	Так	Так	Так
Масштабованість	Висока	Висока	Низька (призначена для локальних застосунків)
Продуктивність при великому обсязі даних	Висока	Висока	Низька
Підтримка на більшості хостингів	Так	Рідко	Так, але здебільшого в локальних середовищах
Підтримка CTE, JSON, window-функцій	Так (з версії 8.0)	Так	Частково (з обмеженнями)
Вартість використання	Безкоштовна (з відкритим кодом)	Платна (ліцензія Oracle)	Безкоштовна (вбудована)
Простота налаштування	Середня	Складна	Висока
Підтримка реплікації	Так	Так	Ні

Для аналізу було враховано також поширеність СУБД на хостингових платформах, MySQL пропонується як стандартне рішення для веб-додатків і є більш популярним [25][26][27][44]. Це свідчить про значно ширшу підтримку та готову інфраструктуру під MySQL порівняно з Oracle SQL чи SQLite.

На основі результатів порівняння для реалізації системи обрано MySQL – надійну реляційну СУБД, що підтримується Django і забезпечує ефективну роботу з великим обсягом структурованих даних [28][29][41]. Вибір саме MySQL обумовлений кількома факторами. По-перше, це одна з найпопулярніших систем керування базами даних у світі, яка має широку підтримку в більшості хостинг-провайдерів, зокрема для розгортання на віртуальних або VPS-серверах. По-друге, Django має повну сумісність із MySQL, включаючи підтримку міграцій, транзакцій і оптимізовану роботу ORM. По-третє, MySQL добре масштабується, забезпечує високу

продуктивність при читанні, і підходить для рішень, які передбачають часту взаємодію з великими обсягами даних, як-от конфігурації компонентів велосипеда.

У проєкті використовується версія MySQL 8.0, яка є актуальною на момент розробки [42], має статус стабільного релізу та включає значні покращення щодо продуктивності, безпеки, підтримки JSON, CTE-запитів, window-функцій, а також сумісна з Django 5.2 [43]. Це дозволяє ефективно використовувати сучасні можливості реляційного підходу до зберігання даних у поєднанні з ORM Django.

Таблиця 3.2 – Порівняння локальних веб-серверів

Параметр	**XAMPP**	**WAMP**	**MAMP**
Платформи	Windows, Linux, macOS	Windows	macOS, Windows (MAMP Pro для Linux)
Склад компонентів	Apache, MySQL, PHP, Perl, FTP, phpMyAdmin	Apache, MySQL, PHP, phpMyAdmin	Apache, MySQL, PHP, phpMyAdmin
Легкість налаштування	Дуже проста в налаштуванні, один крок для всіх платформ	Легко налаштувати, тільки для Windows	Просте налаштування для macOS, обмежено на Windows
Можливості	Підтримка Perl, FTP, підтримка багатьох платформ	Зручний інтерфейс для користувачів Windows	Оптимізовано для macOS, підтримка MAMP Pro для розширених функцій
Продуктивність	Висока продуктивність на всіх платформах	Продуктивність добре працює на Windows	Стабільно працює на macOS, MAMP Pro покращує продуктивність
Призначення	Універсальне середовище для розробки на всіх платформах	Локальний сервер для Windows	Розробка локальних серверів на macOS, з платною версією для розширених можливостей

На основі проведеного аналізу в таблиці 3.2 порівняння популярних локальних веб-серверів – XAMPP, WAMP і MAMP, можна зробити висновок,

що XAMPP є оптимальним вибором для адміністрування бази даних у даному проекті. Це середовище забезпечує підтримку всіх основних компонентів для розробки веб-застосунків, таких як Apache, MySQL, PHP, Perl та інші сервіси.

Однією з ключових переваг XAMPP є його багатоплатформність – він підтримує не лише Windows, але й Linux та macOS, що робить його універсальним для різних операційних систем. Це особливо важливо для тестування на різних платформах. Крім того, XAMPP дозволяє швидко налаштувати та запустити локальний сервер, що забезпечує зручне середовище для розробки.

Для адміністрування бази даних у XAMPP використовується phpMyAdmin – зручний веб-інтерфейс, що дозволяє виконувати всі необхідні операції з базою даних: перегляд, редагування, виконання SQL-запитів, експорт/імпорт даних та аналіз структури таблиць [30]. Завдяки цьому phpMyAdmin стає потужним інструментом для роботи з базами даних, що спрощує процес розробки та тестування.

На рисунку 3.1 наведено діаграму взаємодії використаних технологій.

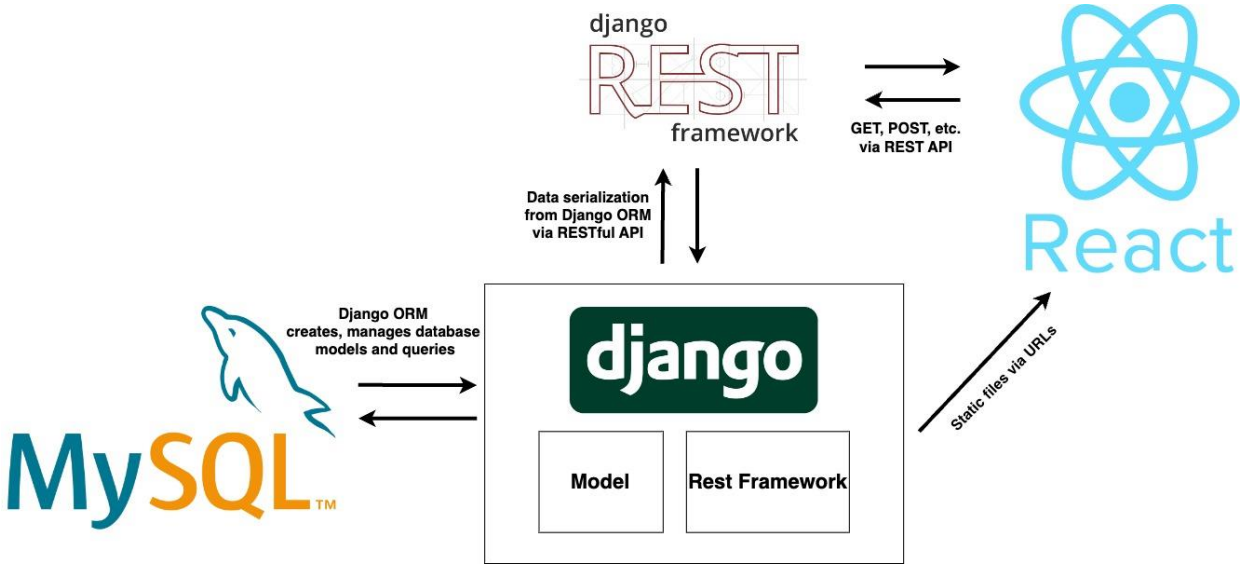


Рисунок 3.1 – Діаграма взаємодії технологій

Джерело: розроблено автором

Середовище розробки включає текстові редактори Visual Studio Code та PyCharm для написання й налагодження коду [33][34], а також XAMPP для розгортання локального серверного середовища та керування MySQL. У подальшому передбачається використання Docker для контейнеризації та розгортання проєкту на хостингових платформах, сервері або хмарному середовищі [35].

Також використовувалася система контролю версій Git у поєднанні з платформою GitHub, що забезпечила керування змінами в кодовій базі, спільну роботу над окремими компонентами застосунку, зберігання історії комітів та можливість створення резервних копій. GitHub також слугував як середовище для код-рев'ю, обговорення задач, створення pull-запитів та виправлення помилок, що суттєво покращило організацію командної роботи та підтримку проєкту в актуальному стані [46].

Обраний стек технологій є відкритим, безкоштовним і має велику підтримку спільноти. Django спрощує роботу з базами даних та реалізацію бізнес-логіки, а MySQL у поєднанні з XAMPP та phpMyAdmin забезпечує просту й ефективну роботу з даними без потреби в окремому серверному налаштуванні. React.js дає змогу реалізувати зручний та адаптивний інтерфейс, а Docker забезпечить можливість стабільного розгортання застосунку.

База даних моделюється засобами Django ORM, а фізично реалізується у MySQL [36][38]. Взаємозв'язки між сутностями реалізовано через зовнішні ключі, що забезпечує цілісність і узгодженість даних [37]. За допомогою phpMyAdmin можна переглядати структуру таблиць, виконувати запити й експортувати дані. Усі основні сутності – користувачі, конфігурації, замовлення, готові велосипеди, індивідуальні велосипеди та їхні компоненти – реалізовані окремими таблицями, що дозволяє легко масштабувати систему та керувати даними. На рисунку 3.2 зображено частину схеми бази даних для готового гірського велосипеду з каталогу.

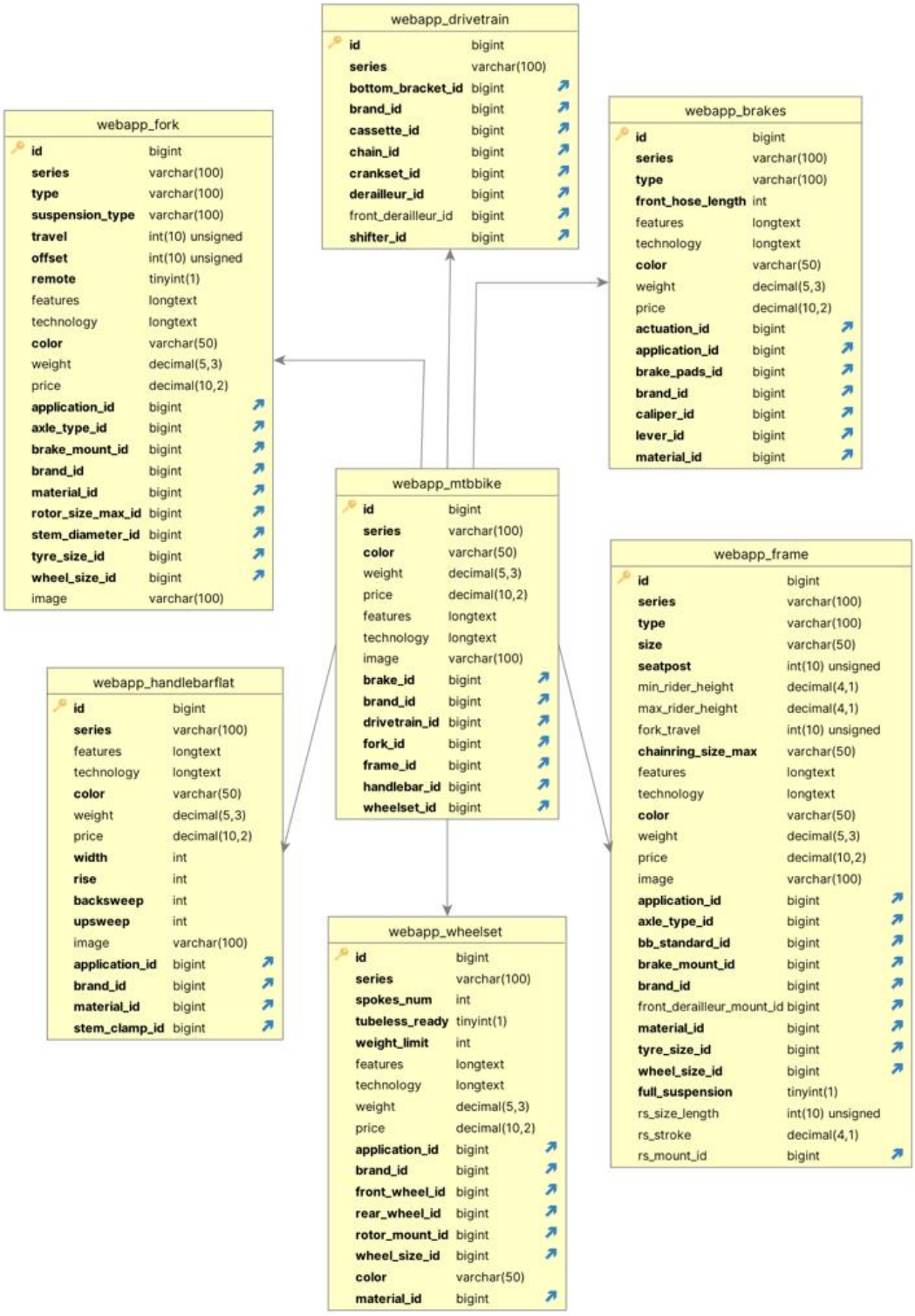


Рисунок 3.2 – Схема бази даних гірського велосипеду з каталогу

Джерело: розроблено автором

Представлений фрагмент бази даних описує структуру модулів гірських велосипедів та їхніх основних компонентів. Центральною є таблиця webapp_mtbike, що містить загальну інформацію про велосипед (серія, колір, вага, ціна) та встановлює зв’язки з основними складовими через зовнішні ключі. Кожен компонент велосипеда представлений окремою таблицею:

- webapp_fork – дані про передню вилку (тип, хід підвіски, офсет, технології виробництва, вага, матеріали).
- webapp_frame – параметри рами (розміри, тип підвіски, вага, матеріал, сумісність з компонентами).
- webapp_brakes – характеристики гальмівної системи (тип, довжина шлангів, вага, тип кріплення, матеріали).
- webapp_drivetrain – компоненти трансмісії (шатуни, касета, передні та задні перемикачі, шифтер).
- webapp_handlebarflat – специфікації керма (тип, розмір, колір, вага).
- webapp_wheelset – опис колісного комплекту (розмір, кількість спиць, вага, тип гальмівного кріплення, матеріал).

Між таблицями встановлені логічні зв’язки за допомогою зовнішніх ключів, що забезпечує цілісність даних [22] та дозволяє відображати повну конфігурацію велосипеда. Така структура бази даних підтримує гнучке керування компонентами та їх характеристиками, що необхідно для формування різних моделей велосипедів на основі змінних специфікацій. Таким самим чином розроблено таблиці для інших типів велосипеду відповідно.

Таблиця конфігурації побудована за аналогією до структури повністю укомплектованого велосипеда. Вона містить повний набір основних компонентів, необхідних для формування готового виробу. Водночас на відміну від таблиці готових велосипедів, у ній відсутні атрибути, що стосуються бренду та серії, що дозволяє використовувати конфігурацію як універсальну модель для подальшого налаштування або індивідуальної збірки велосипеда відповідно до заданих параметрів.

На рисунку 3.3 зображена схема бази даних основного фунціоналу веб-сайту, дана частина бази даних реалізує основну функціональність системи управління користувачами, замовленнями, готовими велосипедами та їх індивідуальними конфігураціями.

Рисунок 3.3 – Схема бази даних основного функціоналу веб-застосунку інтернет-магазину з конфігуратором

Джерело: розроблено автором

Структура охоплює такі основні сутності:

- webapp_user — таблиця користувачів, яка містить дані про облікові записи клієнтів (ім’я, прізвище, електронна пошта, адреса доставки, права доступу та ін.). Права користувачів додатково деталізуються через зв’язні таблиці webapp_user_groups та webapp_user_user_permissions.
- webapp_mtbike — таблиця готових моделей велосипедів, що зберігає характеристики виробів, такі як серія, колір, вага, ціна, опис особливостей і технологій, а також посилання на відповідні компоненти (рама, вилка, гальма тощо).
- webapp_mtbicycleconfiguration — таблиця конфігурацій велосипедів, що дозволяє зберігати індивідуально сформовані клієнтом варіанти зборок. На відміну від таблиці готових велосипедів, ця таблиця не містить атрибутів бренду чи серії, але включає повний набір необхідних компонентів для створення повноцінного велосипеда. Завдяки цьому забезпечується висока гнучкість системи конфігурування.
- webapp_order — таблиця замовлень, яка містить інформацію про дату створення замовлення, його статус, загальну вартість, адресу доставки та ідентифікатор користувача.
- webapp_orderitem — проміжна таблиця для конкретизації позицій замовлення, де зберігаються відомості про кількість товару, ціну, ідентифікатор замовленого велосипеда або індивідуальної конфігурації.

Зв’язки між цими таблицями побудовані за допомогою зовнішніх ключів, що забезпечує цілісність даних, спрощує доступ до інформації та дає змогу реалізувати процеси замовлення як готових велосипедів, так і індивідуальних моделей.

Таким чином, дана структура бази даних дозволяє реалізувати як продаж стандартних моделей, так і індивідуальне формування велосипедів під

замовлення, що значно розширює можливості системи для кінцевих користувачів.

Було реалізовано рекомендаційну систему для головної сторінки веб-платформи (Рис. 3.4), що формує персоналізовані пропозиції для користувача на основі його попередніх дій. Якщо користувач авторизований, система аналізує історію переглядів, збережених конфігурацій та замовлень, формуючи добірку велосипедів або компонентів, що найчастіше відповідають його інтересам. У випадку неавторизованих користувачів використовуються загальні патерни поведінки (наприклад, найпопулярніші товари за останній тиждень або сезонні рекомендації).

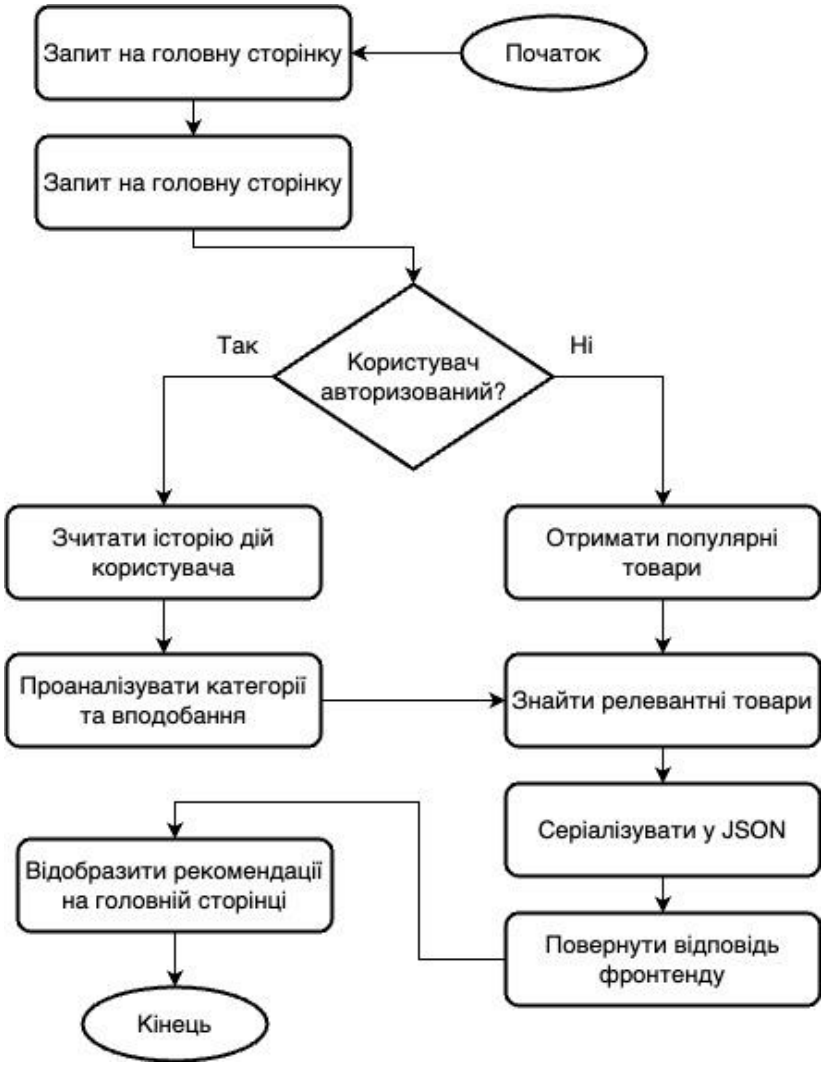


Рисунок 3.4 – Блок-схема рекомендаційної системи

Джерело: розроблено автором

Алгоритм включає кілька основних кроків:

1. Збір даних: система отримує інформацію з бази даних про дії користувача (перегляди, додавання до кошика, покупки).

2. Обробка: відбувається фільтрація та агрегація даних для виявлення переваг користувача.

3. Вивід: згенеровані рекомендації передаються у вигляді JSON через REST API та динамічно відображаються на головній сторінці у секції «Рекомендоване для вас».

3.2 Конструювання системи

Онлайн-конфігуратор є центральним функціональним елементом платформи. Його логіка передбачає можливість користувача поступово обирати окремі компоненти велосипеда, зокрема: раму, вилку, колесо, гальма, трансмісію, шифтер тощо. Компоненти зберігаються у відповідних таблицях бази даних і пов’язані через зовнішні ключі.

Підбір компонентів реалізовано за допомогою перевірки технічної та конструктивної сумісності між елементами. Наприклад, при виборі рами система автоматично обмежує вибір лише до тих вилок, які відповідають за типом кріплення (наприклад, tapered або прямий шток), розміром коліс (26”, 27.5”, 29”) та шириною та стандартом осі, а також ходом вилки. Підбір вилки за ходом є важливим для збереження правильної геометрії рами та поведінки велосипеда — наприклад, трейлові велосипеди можуть потребувати вилку з ходом 120–140 мм, у той час як для крос-кантрі потрібний хід 80–100 мм. Аналогічно, під час вибору трансмісії (наприклад, системи шатунів чи заднього перемикача), враховується кількість швидкостей, тип каретки та сумісність з рамою.

Ця логіка реалізується через набір правил сумісності, що зберігаються в базі даних і застосовуються до кожного компонента, який користувач додає до конфігурації. Таким чином, виключається можливість некоректного підбору несумісних частин велосипеду.

Крім того, система в реальному часі виконує динамічне обчислення загальної вартості та ваги зібраного велосипеда. Інформація оновлюється щоразу при зміні будь-якого компонента, спираючись на актуальні характеристики та ціни з бази даних.

Для авторизованих користувачів доступна функція збереження конфігурації в особистому кабінеті. Це дає змогу повернутися до зібраного велосипеда пізніше та оформити замовлення.

Користувач може переглядати перелік доступних готових велосипедів у каталозі. Для цього реалізовано API-ендпоінт, який фільтрує велосипеди за різними критеріями (тип, призначення, бренд тощо). У Django для цього використано модуль django-filter, що дозволяє будувати динамічні запити на основі GET-параметрів [17].

Після вибору або збирання велосипеда, користувач може ініціювати оформлення замовлення. На стороні серверної частини реалізовано механізм створення запису у таблицях Order та OrderItem, з обробкою конфігурації або готового велосипеда. Після підтвердження замовлення інформація зберігається у базі даних, а користувач отримує відповідь із деталями.

Аутентифікація в системі реалізована за допомогою JWT (JSON Web Token), що забезпечує безпечну роботу з токенами доступу. Для цього використовується бібліотека Simple JWT, яка дозволяє генерувати access та refresh токени при вході користувача в систему [18]. Після авторизації користувач отримує токен, який передається з кожним запитом у заголовку Authorization.

На стороні клієнта реалізовано компоненти інтерфейсу, які взаємодіють з API: форма конфігуратора, каталог товарів, особистий кабінет користувача, модуль оформлення замовлення. Всі запити до сервера виконуються через Axios, а отримані дані обробляються у вигляді станів у React-компонентах [19].

Таким чином, конструювання системи охоплює повну реалізацію функціональної логіки від взаємодії з користувачем до збереження даних у

базі. Усі модулі побудовані з дотриманням принципів модульності, повторного використання та безпеки, що дозволяє надалі масштабувати систему та розширювати її функціональні можливості.

3.3 Тестування системи

Тестування веб-платформи проводилося з метою перевірки коректності функціонування основних модулів системи, а також для виявлення можливих помилок на різних етапах взаємодії користувача із платформою.

Був застосований підхід функціонального тестування, що передбачає перевірку відповідності фактичної поведінки системи її вимогам і очікуваній логіці роботи. Цей підхід відповідає сучасним підходам до тестування веб-додатків [20][21].

Основними напрямами тестування стали:

1. *Тестування інтерфейсу*

   Було перевірено правильність відображення основних елементів інтерфейсу користувача:

   – коректне завантаження головної сторінки, каталогу велосипедів та сторінки конфігуратора;
   – правильність функціонування навігаційних елементів (кнопки переходів, форма авторизації);
   – адаптивність інтерфейсу для різних розмірів екранів (десктопні та мобільні пристрої).

2. *Тестування реєстрації та авторизації користувача*

Було протестовано процес реєстрації нового користувача із перевіркою заповнення обов’язкових полів, обробки помилкових даних (наприклад, спроба реєстрації з уже існуючим email) та отримання токенів доступу через JWT після успішної реєстрації.

У межах тестування авторизації було перевірено:

– можливість входу в систему з правильними обліковими даними;
– обробку помилок при введенні неправильного логіну або пароля;

– автоматичне збереження стану сесії користувача.

3. Тестування фільтрації велосипедів у каталозі

Було протестовано багатокритеріальні фільтри для каталогу велосипедів:

– фільтрація за брендом, призначенням, матеріалом рами, кольором та ціною;
– коректність оновлення переліку велосипедів відповідно до вибраних фільтрів;
– обробка випадків, коли не знайдено результатів за заданими параметрами.

Тестування показало, що система адекватно відображає тільки ті велосипеди, які відповідають критеріям фільтрації.

4. Тестування працездатності конфігуратора велосипедів

Особливу увагу було приділено тестуванню конфігуратора, зокрема:

– правильність підбору сумісних компонентів (рама, вилка, колеса, трансмісія тощо) залежно від введених параметрів користувача (зріст, висота ноги, дисципліна);
– перевірка логіки динамічного оновлення ціни та ваги конфігурації в реальному часі;
– коректність збереження обраних параметрів при переходах між етапами конфігурації.

Було виявлено, що система правильно обмежує вибір компонентів згідно із сумісністю та безпомилково проводить розрахунок вартості.

5. Тестування відповідності збереження конфігурації

Було протестовано можливість збереження створеної конфігурації в обліковому записі користувача. Перевірено такі аспекти:

– збереження повної інформації про обрані компоненти;
– можливість перегляду раніше збережених конфігурацій;
– редагування збереженої конфігурації із подальшим оновленням.

Було підтверджено, що після реєстрації або авторизації користувач має змогу зберігати свої конфігурації та повертатися до них у будь-який момент.

6. *Тестування замовлення та додавання товарів до кошика*

Перевірено роботу модуля оформлення замовлення:

– додавання велосипеда або індивідуальної конфігурації до кошика;
– коректну передачу інформації про замовлення на сервер та збереження в базі даних;
– обробку введення платіжної інформації та адреси доставки;

Результати тестування підтвердили, що замовлення успішно формуються і зберігаються у відповідних таблицях бази даних.

3.4 Використання системи

Перехід за посиланням на сторінку веб-сайту є першим кроком. Після чого користувач потрапляє на головну сторінку інтернет магазину де йому одразу представлений каталог велосипедів (рис. 3.5).

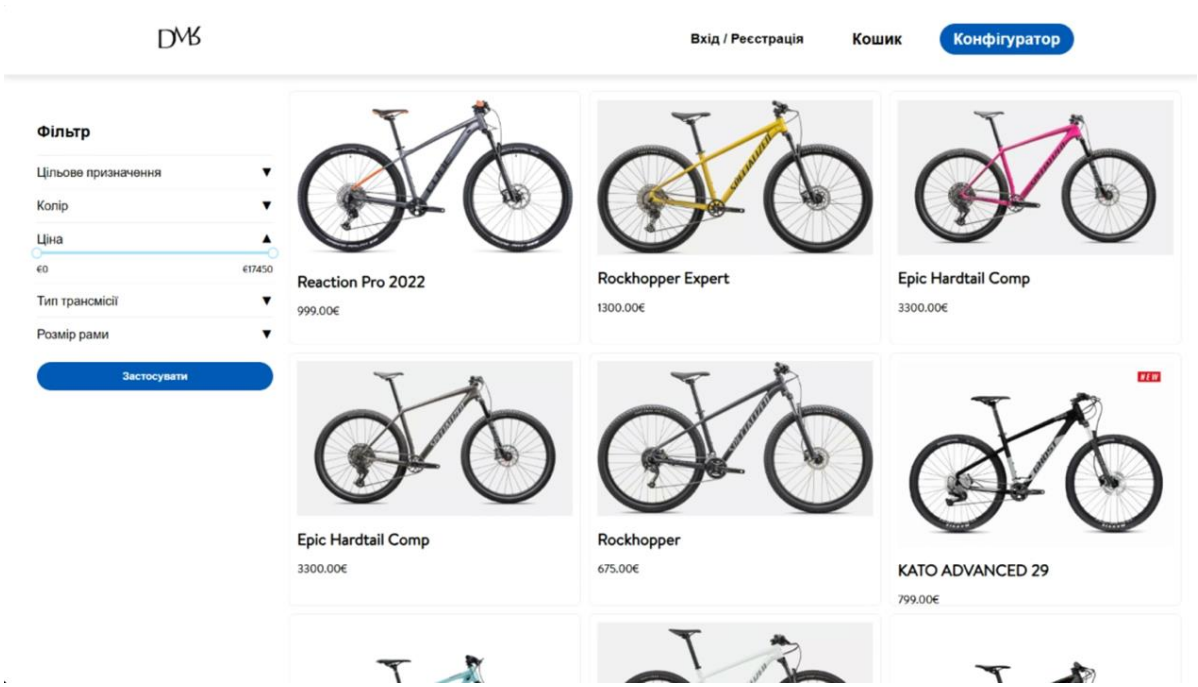


Рисунок 3.5 – Головна сторінка веб-сайту

Джерело: розроблено автором

На головній сторінці представлені елементи такі як:

- навігаційна панель, на котрій елементами є логотип при натисканні якого виконується перехід на головну сторінку, а також кнопка «Конфігуратор», при натисканні якої відбувається перехід на сторінку конфігуратора індивідуального велосипеду під замовлення;
- картки попереднього перегляду готових моделей велосипедів з відображенням фотографії велосипеду, ціни та назви моделі, при натисканні на картку – виконується перехід на сторінку детального перегляду велосипеду;
- фільтр надає можливість сортувати за особливостями, кольором та ціною надаючи можливість пошуку велосипеду за багатьма параметрами одночасно. Після натискання на кнопку «Застосувати» сайт оновлюється та відображає моделі відповідно за тими параметрами, які були вибрані користувачем.

Для отримання детальної інформації про велосипед потрібно натиснути на картку попереднього перегляду, після чого відбудеться перехід на сторінку детального перегляду вибраної моделі велосипеду (рис. 3.6). На цій сторінці конкретного велосипеду розташована детальна інформація про велосипед, а саме: назва, модель, рік випуску, ціна, колір, встановлені компоненти та цільове призначення.

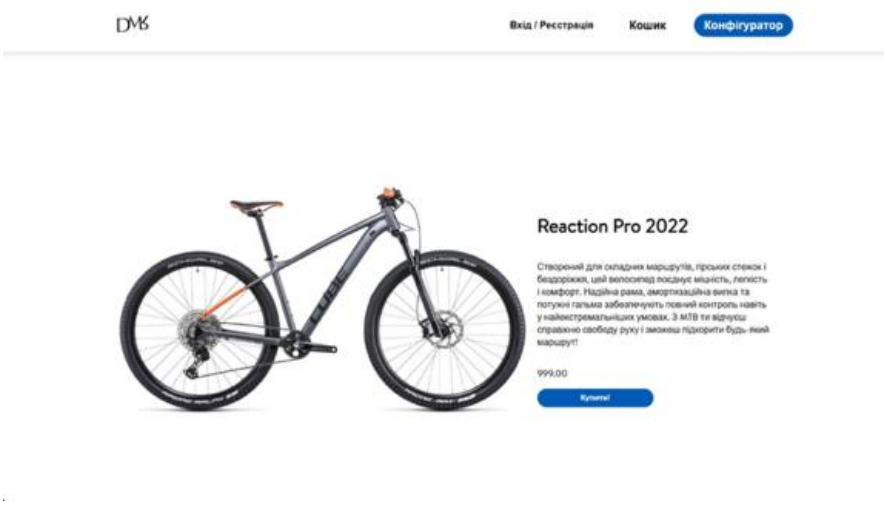


Рисунок 3.6 – Сторінка детального перегляду конкретного велосипеду

Джерело: розроблено автором

Щоб зробити замовлення треба натиснути на кнопку «Купити», після чого велосипед додається до кошику, і виконається переадресація на сторінку кошику (рис. 3.7), де користувач може ввести дані про адресу, платіжну інформацію та замовити велосипед. Також користувач може обрати спосіб доставки: безпосередньо до відділення магазину або до пункту видачі партнера, звідки зможе самостійно забрати велосипед.

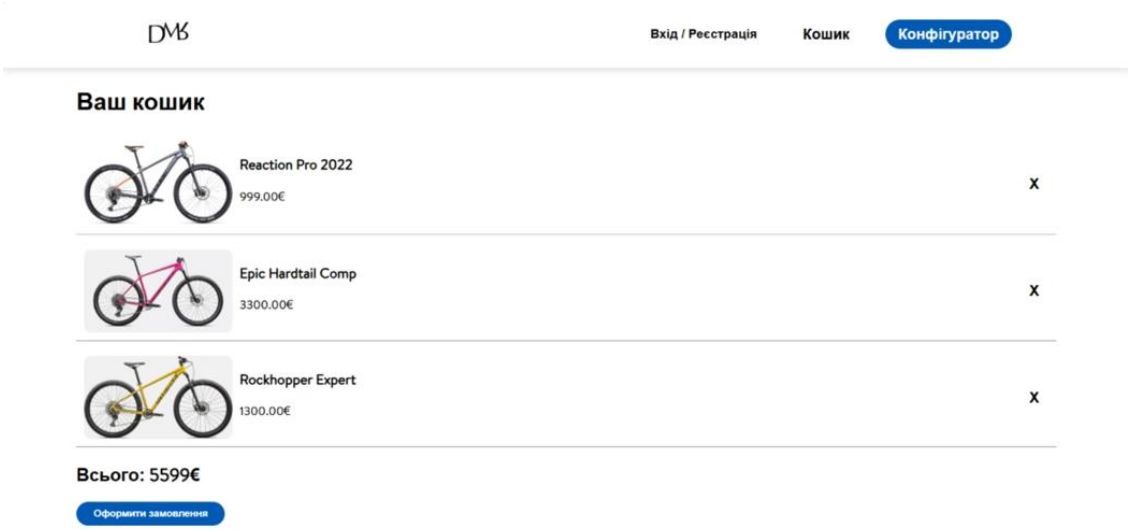


Рисунок 3.7 – Сторінка кошику доданих товарів

Джерело: розроблено автором

Для створення велосипеду під замовлення в правому верхньому куту розташована кнопка «Конфігуратор», після натискання на котру відкривається початкова сторінка конфігуратора індивідуального велосипеду під замовлення (рис. 3.8). Першим етапом якого є введення користувачем інформації про зріст та довжину ноги.

Після коректного введення інформації про зріст та довжину ноги користувача і натискання на кнопку «Далі» відбувається перехід на другу сторінку конфігуратора (рис. 3.9), на якій другим етапом є вибір цільового призначення майбутнього велосипеду. При натисканні на фотографію здійснюється вибір дисципліни.

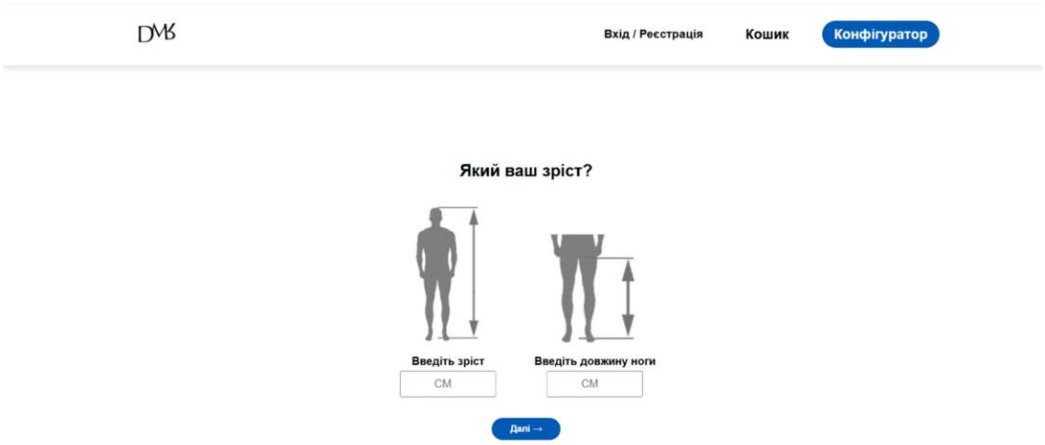


Рисунок 3.8 – Перша сторінка конфігуратора

Джерело: розроблено автором

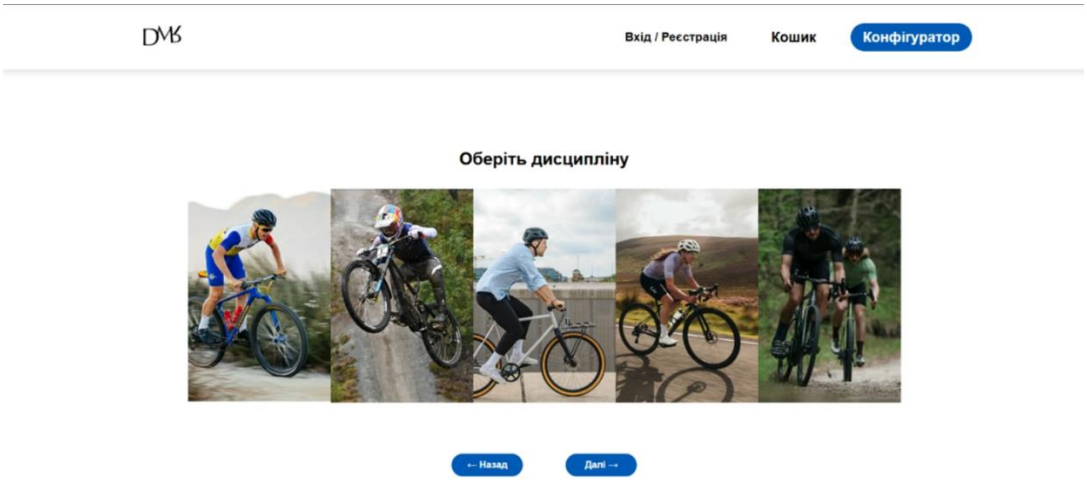


Рисунок 3.9 – Друга сторінка конфігуратора «Цільове призначення»

Джерело: розроблено автором

Після натискання на кнопку «Далі» виконується автоматичний підбір підходящих рам на основі введених даних та обраної дисципліни. Після успіху відбувається перехід до сторінки конструктору (рис. 3.10), першим етапом якого користувач обирає раму велосипеду і натискає кнопку далі. Таким самим чином на другому етапі користувач обирає вилку велосипеду і на всіх наступних етапах робить відповідно вибір для всіх компонентів велосипеду. Усі компоненти автоматично підбираються за сумісністю. Після чого

користувач може, або одразу зробити замовлення, або зберегти конфігурацію будучи авторизованим користувачем. Якщо користувач не є авторизованим, то при спробі збереження конфігурації – система запропонує користувачу авторизуватись (рис.3.11). У випадку якщо у користувача ще немає аккаунту він може його створити натиснувши кнопку «Зареєструватись». Після введення даних та натисканням на кнопку реєстрації – конфігурація автоматично буде збережена, і користувач в будь-який момент зможе повернутись і зробити замовлення.

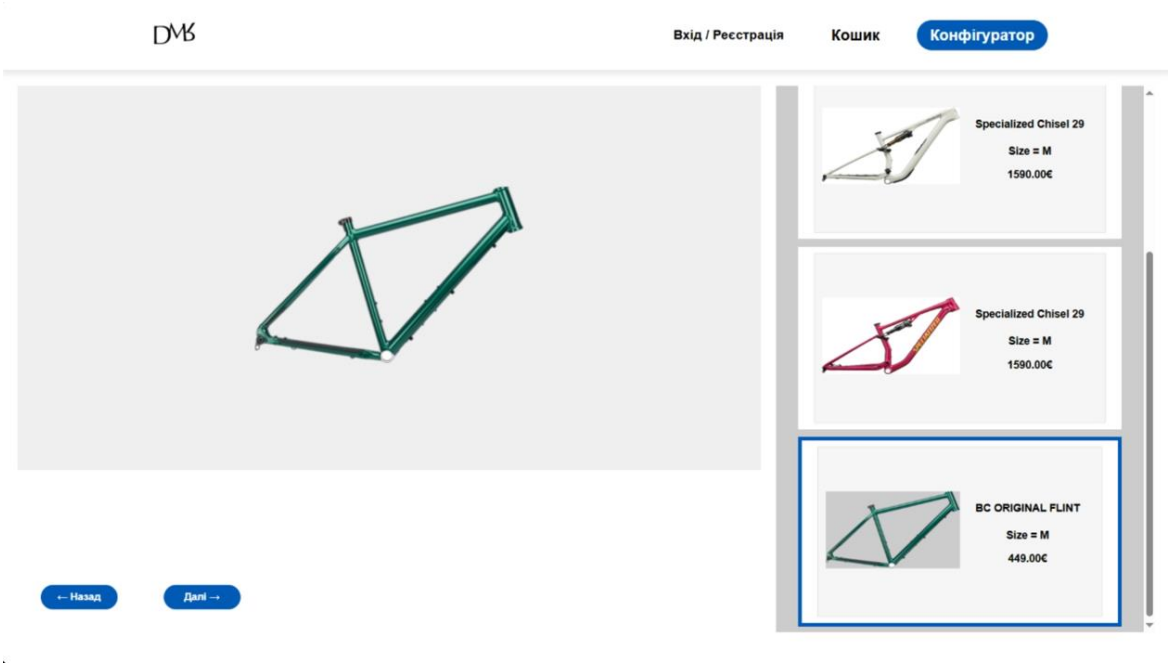


Рисунок 3.10 – Сторінка конфігуратора «Конструктор»

Джерело: розроблено автором

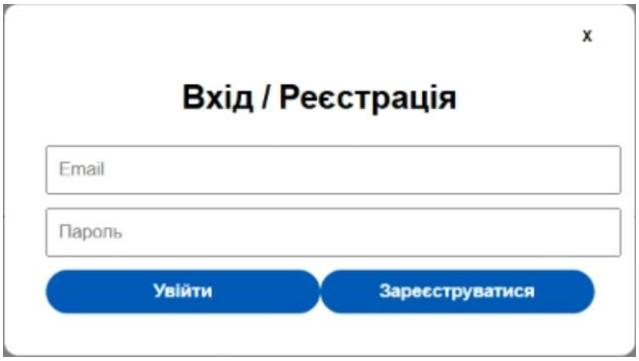


Рисунок 3.11 – Вікно авторизації

Джерело: розроблено автором

Панель адміністратора є зручним інструментом для керування каталогом велосипедів, компонентів та конфігурацій, доступним як для адміністратора, так і для контент-менеджера (рис. 3.12 та рис. 3.13). Вона забезпечує повний контроль над інформацією про велосипеди, дозволяючи: обирати бренд і серію моделі, задавати технічні характеристики (такі як рама, вилка, колеса, трансмісія, гальма, кермо та інші), вказувати вагу та ціну, додавати опис особливостей та використаних технологій. Крім того, підтримується завантаження основного зображення велосипеда та галереї додаткових фотографій.

Функціонал панелі також включає можливість збереження, редагування або видалення запису про велосипед, що значно спрощує процес оновлення даних і формування актуального асортименту товарів в інтернет-магазині.

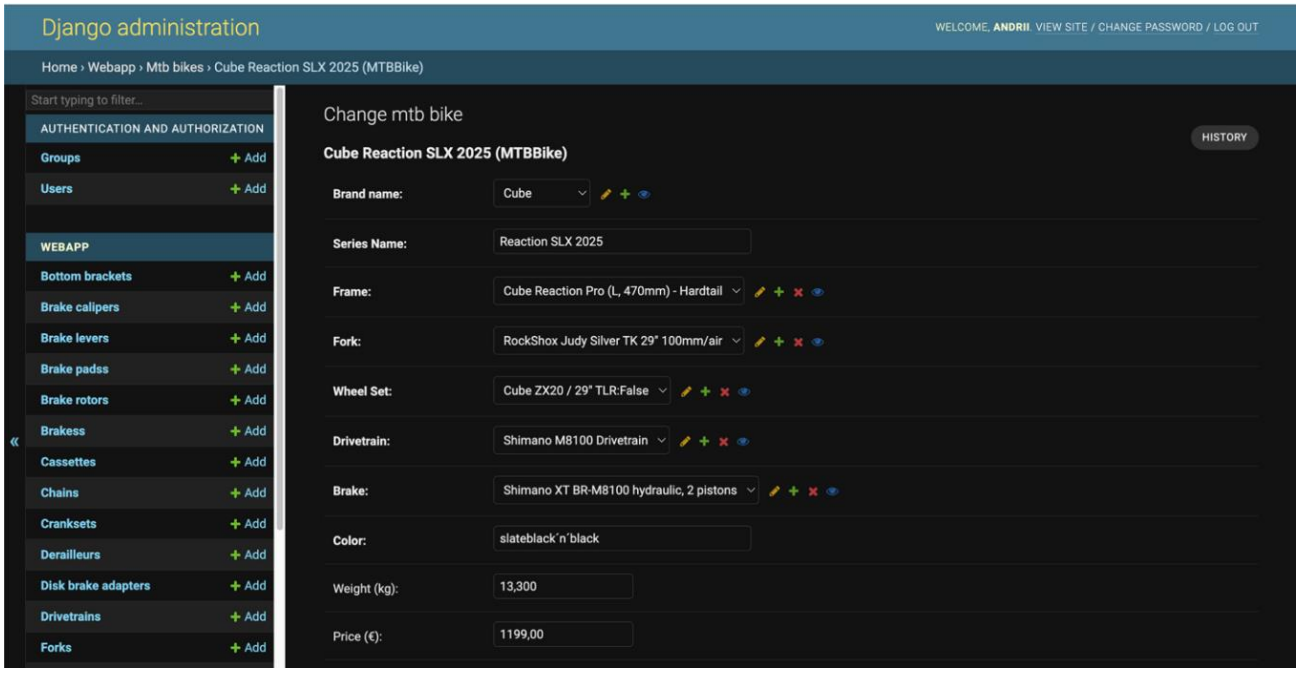


Рисунок 3.12 – Сторінка редагування велосипеду панелі адміністратора

Джерело: розроблено автором

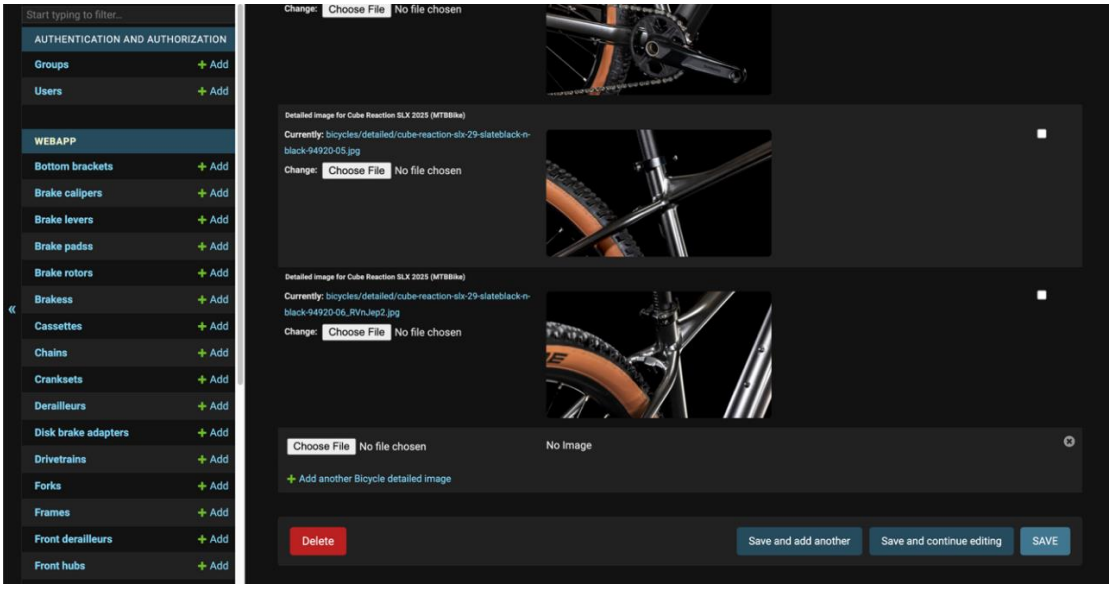


Рисунок 3.13 – Сторінка редагування велосипеду панелі адміністратора 2

Джерело: розроблено автором

Висновки до розділу 3

У третьому розділі було детально розглянуто процес реалізації веб-платформи інтернет-магазину з продажу велосипедів із використанням багатокритеріальних фільтрів, рекомендаційної системи та онлайн-конфігуратора індивідуального велосипеду. Була обґрунтована доцільність вибору технологічного стеку, який включає Python, Django, Django REST Framework для серверної частини, React.js для клієнтської частини, а також MySQL як основну систему управління базами даних із використанням phpMyAdmin та середовища XAMPP для локальної розробки.

Реалізація бази даних була здійснена за допомогою Django ORM, що забезпечило логічну цілісність даних і спростило керування ними. Було створено окремі модулі для роботи з каталогом велосипедів, онлайн-конфігуратором, системою замовлень та механізмами аутентифікації й авторизації користувачів за допомогою JWT-токенів. Також продемонстровано інтеграцію візуальної та серверної частини через REST

API, що забезпечило зручну і швидку взаємодію між клієнтською та серверною частинами системи.

Було проведено функціональне тестування ключових модулів: інтерфейсу, реєстрації та авторизації користувачів, роботи конфігуратора, фільтрації товарів, збереження конфігурацій та оформлення замовлень, що підтвердило працездатність системи та її відповідність поставленим вимогам.

Для організації роботи над кодом і контролю версій використовувалася система Git у поєднанні з GitHub, що дозволило зберігати історію змін, координувати командну розробку, проводити рев'ю коду та створювати окремі гілки. Завдяки цьому процес розробки був структурованим, контрольованим і прозорим.

У результаті розробки створено стабільну, масштабовану та адаптивну платформу, яка відповідає сучасним стандартам веб-розробки та забезпечує високий рівень зручності й ефективності для кінцевого користувача.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено повноцінну веб-платформу інтернет-магазину з продажу велосипедів, особливістю якої є інтеграція багатокритеріальних фільтрів, рекомендаційної системи та онлайн-конфігуратора індивідуального велосипеду під замовлення.

У першому розділі проведено аналіз предметної області, визначено основні проблеми, що виникають при підборі та замовленні велосипедів через онлайн-сервіси. Було обґрунтовано актуальність розробки інструменту, який дозволяє не лише обирати готові моделі, а й створювати індивідуальні конфігурації велосипедів відповідно до фізіологічних параметрів користувача та особистих уподобань.

Другий розділ було присвячено етапу проєктування системи. Було розроблено структуру майбутньої платформи із застосуванням сучасних архітектурних підходів. Обрано клієнт-серверну архітектуру з використанням RESTful API, що забезпечує масштабованість, гнучкість та простоту інтеграції. Побудовано діаграми варіантів використання, діяльності та послідовності, а також змодельовано структуру бази даних із визначенням основних сутностей і зв’язків між ними.

У третьому розділі реалізовано основні функціональні модулі системи. Серверну частину розроблено за допомогою Django та Django REST Framework, клієнтську – з використанням React.js. Базу даних створено на основі MySQL з адмініструванням через phpMyAdmin у середовищі XAMPP. Реалізовано функціонал авторизації користувачів за допомогою JWT, динамічний онлайн-конфігуратор, каталог велосипедів із багатокритеріальним фільтром, модуль обробки замовлень і збереження конфігурацій. Проведено тестування основних процесів, яке підтвердило працездатність системи.

Окрему увагу під час розробки було приділено організації командної роботи та контролю версій – для цього використовувалася система керування версіями Git у поєднанні з платформою GitHub. Це дозволило ефективно керувати кодовою базою, фіксувати зміни, проводити код-рев’ю та

забезпечити безпечне злиття функціональних гілок у загальну структуру проєкту.

Таким чином, поставлені задачі були виконані в повному обсязі: розроблено платформу для замовлення велосипедів із можливістю створення індивідуального велосипеду під замовлення. Реалізована система відповідає актуальним вимогам до веб-розробки, забезпечує високий рівень взаємодії з користувачем і може бути основою для подальшого розвитку та розширення функціональності у майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сторінка компанії Fanatikbike. – URL: https://www.fanatikbike.com/ (дата звернення 7.03.2025).

2. Що таке API: навіщо використовується програмістами та базові основи роботи з ним - URL: https://cloud.itstep.org/blog/what-is-an-api-why-is-it-used-by-programmers-and-the-basics-of-working-with-it#1 (Дата звернення 11.03.2025).

3. Стаття у вікіпедії «Django» – URL: https://uk.wikipedia.org/wiki/Django

4. Архітектура вебзастосунків 2024: Ультимативний гайд для розробників — URL: https://robotdreams.cc/uk/blog/567-arhitektura-vebzastosunkiv (Дата звернення 11.03.2025).

5. Веброзробка для початківців: що потрібно знати? - URL: https://landinglist.com.ua/vebrozrobka-dlya-pochatkivcziv-shho-potribno-znaty/ (Дата звернення 11.03.2025).

6. Стаття у вікіпедії «MySQL» – URL: https://uk.wikipedia.org/wiki/MySQL (Дата звернення 11.03.2025).

7. Was ist eine REST API? - URL: https://www.uptrends.de/was-ist/rest-api (Дата звернення 11.03.2025).

8. 3. Стаття у вікіпедії «Knowledge-based configuration» - URL: https://en.wikipedia.org/wiki/Knowledge-based_configuration (Дата звернення 18.03.2025).

9. Інтернет-магазин «Велопланета2» – URL: https://veloplaneta.ua/ua/gornye (Дата звернення 11.03.2025).

10. Microservices vs. monolithic architecture – URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (Дата звернення 11.03.2025).

11. MVC Architecture: Simplifying Web Application Development – URL: https://www.ramotion.com/blog/mvc-architecture-in-web-application/ (Дата звернення 11.03.2025).

12. What is a RESTful API? – URL: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls (Дата звернення 11.03.2025).

13. Microservices: Advantages and Disadvantages (And Whether They're Right For Your Business) – URL: https://www.shopify.com/enterprise/blog/disadvantages-microservices (Дата звернення 11.03.2025).

14. 2025's Most Competitive 3D Product Configurators for E-Commerce, Furniture & Industrial Manufacturing – URL: https://www.linkedin.com/pulse/2025s-most-competitive-3d-product-configurators-e-commerce-furniture-jduie/ (Дата звернення 11.03.2025).

15. Бочкарьов С. О., Бондаренко О. О. UML: об'єктно-орієнтований підхід до аналізу та проектування програмних систем : навч. посіб. - Харків: ХНУ імені В. Н. Каразіна, 2018. - 132 с.

16. Гриценко В. І. Електронна комерція: проектування інтернет-магазинів. – Харків: Фоліо, 2019. – 184 с.

17. Filtering – URL: https://www.django-rest-framework.org/api-guide/filtering/ (Дата звернення 14.03.2025).

18. Simple JWT – URL: https://django-rest-framework-simplejwt.readthedocs.io/en/latest/#simple-jwt (Дата звернення 14.03.2025).

19. 110% Complete JWT Authentication with Django & React-2020 – URL: https://hackernoon.com/110percent-complete-jwt-authentication-with-django-and-react-2020-iejq34ta (Дата звернення 14.03.2025).

20. Website Functionality Testing: How To Do It? A Full Guide – URL: https://luxequality.com/blog/website-functionality-testing/ (Дата звернення 14.03.2025).

21. Web application testing: Getting started with functional testing – URL: https://www.rainforestqa.com/blog/web-application-testing (Дата звернення 14.03.2025).

22. Django documentation – URL: https://docs.djangoproject.com/en/5.2/ (Дата звернення 14.03.2025).

23. Django REST framework – URL: https://www.django-rest-framework.org/ (Дата звернення 14.03.2025).

24. Django Web Framework (Python) – URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django (Дата звернення 11.03.2025).

25. Hostinger – URL: https://support.hostinger.com/en/ (Дата звернення 24.03.2025).

26. Bluehost – URL: https://www.bluehost.com/ (Дата звернення 24.03.2025).

27. SiteGround – URL: https://www.siteground.com/ (Дата звернення 24.03.2025).

28. Databases – URL: https://docs.djangoproject.com/en/5.2/ref/databases/ (Дата звернення 24.03.2025).

29. Connector/Python Django Back End – URL: https://dev.mysql.com/doc/connector-python/en/connector-python-django-backend.html (Дата звернення 25.03.2025).

30. Create an XAMPP MySQL Database in 9 Easy Steps – URL: https://hevodata.com/learn/xampp-mysql/ (Дата звернення 25.03.2025).

31. Bringing MySQL to the web – URL: https://www.phpmyadmin.net/ (Дата звернення 25.03.2025).

32. phpMyAdmin – URL: https://www.ntchosting.com/encyclopedia/databases/mysql/phpmyadmin/ (Дата звернення 25.03.2025).

33. Visual Studio Code documentation – URL: https://code.visualstudio.com/docs (Дата звернення 11.04.2025).

34. PyCharm documentation – URL: https://www.jetbrains.com/help/pycharm/getting-started.html (Дата звернення 11.04.2025).

35. Docker Container Deployment: Checklist for DevOps Leaders – URL: https://duplocloud.com/blog/docker-container-deployment/ (Дата звернення 15.04.2025).

36. Models and databases – URL: https://docs.djangoproject.com/en/5.1/topics/db/ (Дата звернення 27.03.2025).

37. How to work with Django models and databases – URL: https://www.hostinger.com/tutorials/django-models-and-databases (Дата звернення 25.03.2025).

38. Django Database Relationships with MySQL – URL: https://www.digitalocean.com/community/questions/django-database-relationships-with-mysql (Дата звернення 13.04.2025).

39. Стаття у вікіпедії «XAMPP» – URL: https://uk.wikipedia.org/wiki/XAMPP (Дата звернення 13.04.2025).

40. Django 5.2: What's New? A Look at the Latest LTS Release – URL: https://medium.com/%40kajendiran/django-5-2-whats-new-a-look-at-the-latest-lts-release-bd0563761a52 (Дата звернення 13.04.2025).

41. Dubois, P. MySQL: The Complete Reference / Paul Dubois. – New York : McGraw-Hill, 2003. – 1224 с.

42. MySQL 8.0 Reference Manual – URL: https://dev.mysql.com/doc/refman/8.0/en/ (Дата звернення 14.04.2025).

43. Django Software Foundation. Databases Backends. MySQL – URL: https://docs.djangoproject.com/en/5.2/ref/databases/#mysql-notes (Дата звернення 14.04.2025).

44. A2 Hosting – URL: https://hosting.com/ (Дата звернення 24.04.2025).

45. Django ORM Моделі Зв’язків: Ефективні Шляхи Керування Зв’язками Бази Даних – URL: https://javascript.org.ua/django-orm-modeli-zvyazkiv-efektivni-shlyahi-keruvannya-zvyazkami-bazi-danih/ (Дата звернення 24.04.2025).

46. Репозиторій на GitHub – URL: https://github.com/andriimilevskyi/Diploma_project (Дата звернення 24.04.2025).

47. What Is Content-Based Filtering? Benefits and Examples in 2025 – URL: https://www.upwork.com/resources/what-is-content-based-filtering (Дата звернення 24.04.2025).

48. Content-based filtering – URL: https://developers.google.com/machine-learning/recommendation/content-based/basics (Дата звернення 24.04.2025).

49. How do hybrid recommender systems combine different techniques? – URL: https://milvus.io/ai-quick-reference/how-do-hybrid-recommender-systems-combine-different-techniques (Дата звернення 24.04.2025).

50. The Cold Start Problem in Recommender Systems: Strategies for Effective Initialization – URL: https://www.linkedin.com/pulse/cold-start-problem-recommender-systems-strategies-iain-brown-ph-d--4lsce/ (Дата звернення 24.04.2025).

ДОДАТОК А
ФРАГМЕНТИ ЛІСТИНГУ

```python
# Базовий клас для загальних характеристик велосипедних компонентів
22 usages  andriimilevskyi
class BikeComponent(models.Model):
    brand = models.ForeignKey("Brand", on_delete=models.CASCADE, verbose_name="Brand name", db_index=True)
    series = models.CharField(max_length=100, verbose_name="Series Name", db_index=True)
    application = models.ForeignKey("Application", on_delete=models.CASCADE, verbose_name="Application", db_index=True)
    material = models.ForeignKey("Material", on_delete=models.PROTECT, verbose_name="Material", db_index=True)
    features = models.TextField(verbose_name="Features", blank=True, null=True)
    technology = models.TextField(verbose_name="Technology", blank=True, null=True)
    color = models.CharField(max_length=50, verbose_name="Color", db_index=True)
    weight = models.DecimalField(max_digits=5, decimal_places=3, verbose_name="Weight (kg)", null=True, blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2, verbose_name="Price (€)", null=True, blank=True)

    andriimilevskyi
    class Meta:
        abstract = True
```

Рисунок А.1 – Базовий клас для загальних характеристик велосипедних компонентів

Джерело: розроблено автором

```python
3 usages  andriimilevskyi *
class Bicycle(models.Model):
    brand = models.ForeignKey(Brand, on_delete=models.CASCADE, verbose_name="Brand name")
    series = models.CharField(max_length=100, verbose_name="Series Name")
    frame = models.ForeignKey(Frame, on_delete=models.PROTECT, verbose_name="Frame")
    fork = models.ForeignKey(Fork, on_delete=models.PROTECT, verbose_name="Fork")
    wheelset = models.ForeignKey(WheelSet, on_delete=models.PROTECT, verbose_name="Wheel Set")
    drivetrain = models.ForeignKey(Drivetrain, on_delete=models.PROTECT, verbose_name="Drivetrain")
    brake = models.ForeignKey(Brakes, on_delete=models.PROTECT, verbose_name="Brake")
    # add rotors
    color = models.CharField(max_length=50, verbose_name="Color")
    weight = models.DecimalField(max_digits=5, decimal_places=3, verbose_name="Weight (kg)", null=True, blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2, verbose_name="Price (€)", null=True, blank=True)
    features = models.TextField(verbose_name="Features", blank=True, null=True)
    technology = models.TextField(verbose_name="Technology", blank=True, null=True)
    preview_image = models.ImageField(upload_to='bicycles/', verbose_name="Bicycle Image", null=True, blank=True)

    andriimilevskyi *
    class Meta:
        abstract = True

    andriimilevskyi
    def __str__(self):
        return f"{self.brand} {self.series} ({self.__class__.__name__})"
```

Рисунок А.2 – Базовий клас велосипеду з каталогу

Джерело: розроблено автором

```python
1 usage  andriimilevskyi
class BicycleConfigurator(models.Model):
    frame = models.ForeignKey(Frame, on_delete=models.PROTECT, verbose_name="Frame")
    fork = models.ForeignKey(Fork, on_delete=models.PROTECT, verbose_name="Fork")
    # Make wheelset or F/R wheel
    wheelset = models.ForeignKey(WheelSet, on_delete=models.PROTECT, verbose_name="Wheel Set")
    # Drivetrain
    crankset = models.ForeignKey(Crankset, on_delete=models.PROTECT, verbose_name="Crankset")
    bottom_bracket = models.ForeignKey(BottomBracket, on_delete=models.PROTECT, verbose_name="Bottom Bracket")
    cassette = models.ForeignKey(Cassette, on_delete=models.PROTECT, verbose_name="Cassette")
    chain = models.ForeignKey(Chain, on_delete=models.PROTECT, verbose_name="Chain")
    derailleur = models.ForeignKey(Derailleur, on_delete=models.PROTECT, verbose_name="Rear Derailleur")
    front_derailleur = models.ForeignKey(FrontDerailleur, on_delete=models.PROTECT, verbose_name="Front Derailleur",
                                         null=True, blank=True)
    # Brake or custom
    brake = models.ForeignKey(Brakes, on_delete=models.PROTECT, verbose_name="Brake")
    # Custom brakes

    front_rotor = models.ForeignKey(BrakeRotor, on_delete=models.PROTECT, verbose_name="Front Rotor",
                                    related_name='front_rotor_configurations', null=True,
                                    blank=True)
    rear_rotor = models.ForeignKey(BrakeRotor, on_delete=models.PROTECT, verbose_name="Rear Rotor",
                                   related_name='rear_rotor_configurations', null=True,
                                   blank=True)
    rear_shifter = models.ForeignKey(Shifter, on_delete=models.PROTECT, related_name='front_shifter_configurations',
                                     verbose_name="Rear Shifter")
    front_shifter = models.ForeignKey(Shifter, on_delete=models.PROTECT, related_name='rear_shifter_configurations',
                                      verbose_name="Front Shifter", null=True,
                                      blank=True)
    weight = models.DecimalField(max_digits=5, decimal_places=3, verbose_name="Weight (kg)", null=True, blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2, verbose_name="Price (€)", null=True, blank=True)
    preview_image = models.ImageField(upload_to='bicycles/', verbose_name="Bicycle Image", null=True, blank=True)
```

Рисунок А.3 – Базовий клас індивідуального велосипеду

Джерело: розроблено автором

```python
andriimilevskyi
def save(self, *args, **kwargs):
    # Calculate total weight and price before saving
    components = [
        self.frame, self.fork, self.wheelset,
        self.crankset, self.bottom_bracket, self.cassette,
        self.chain, self.derailleur, self.brake, self.rear_shifter
    ]

    # front derailleur, front and rear rotors, front shifter are optional
    optional_components = [self.front_derailleur, self.front_rotor, self.rear_rotor, self.front_shifter]

    total_weight = 0
    total_price = 0

    for component in components + optional_components:
        if component:  # if it's not None
            total_weight += component.weight or 0
            total_price += component.price or 0

    self.weight = total_weight
    self.price = total_price

    super().save(*args, **kwargs)
```

Рисунок А.4 – Функція підрахунку ваги та ціни при збереженні індивідуального велосипеду

Джерело: розроблено автором

```
2 usages  andriimilevskyi
class MTBBikeSerializer(serializers.ModelSerializer):
    brand = serializers.StringRelatedField()
    frame = serializers.StringRelatedField()
    fork = serializers.StringRelatedField()
    wheelset = serializers.StringRelatedField()
    drivetrain = serializers.StringRelatedField()
    brake = serializers.StringRelatedField()
    handlebar = serializers.StringRelatedField()
    detailed_images = serializers.SerializerMethodField()

    andriimilevskyi
    class Meta:
        model = MTBBike
        fields = "__all__"

    andriimilevskyi
    def get_detailed_images(self, obj):
        """ Manually fetch all related images """
        images = BicycleDetailedImage.objects.filter(
            content_type=ContentType.objects.get_for_model(obj),
            object_id=obj.id
        )
        return BicycleDetailedImageSerializer(images, many=True, context=self.context).data
```

Рисунок А.5 – Серіалізатор гірського велосипеду з каталогу

Джерело: розроблено автором

```
2 usages  andriimilevskyi
class BicycleDetailedImageSerializer(serializers.ModelSerializer):
    image_url = serializers.SerializerMethodField()

    andriimilevskyi
    class Meta:
        model = BicycleDetailedImage
        fields = ["id", "image_url"]

    andriimilevskyi
    def get_image_url(self, obj):
        if obj.image:
            request = self.context.get("request")
            return request.build_absolute_uri(obj.image.url) if request else obj.image.url
        return None
```

Рисунок А.6 – Серіалізатор деталізованих фотографій велосипеду з каталогу

Джерело: розроблено автором

```python
andriimilevskyi
@admin.register(MTBBike)
class MTBBikeAdmin(admin.ModelAdmin):
    list_display = ("brand", "series", "frame", "fork", "wheelset", "drivetrain", "brake", "handlebar", "preview_image")
    list_select_related = ("brand", "frame", "fork", "wheelset", "drivetrain", "brake", "handlebar")
    inlines = [BicycleDetailedImageInline]

    readonly_fields = ["preview_image_admin"]

    1 usage  andriimilevskyi
    def preview_image_admin(self, obj):
        if obj.preview_image:
            return mark_safe(
                f'<img src="{obj.preview_image.url}" width="500" height="300" style="object-fit: cover; border-radius: 5px;"/>')
        return "No Image"

    preview_image_admin.short_description = "Preview Image"
```

Рисунок А.7 – Налаштування панелі адміністратора для перегляду та редагування гірського велосипеду з каталогу

Джерело: розроблено автором

```python
2 usages  andriimilevskyi
class ForkRecommendationAPIView(APIView):
    andriimilevskyi
    @swagger_auto_schema(
        operation_summary="GET рекомендовані вилки на основі обраної рами (допуск ходу ±10 мм)",
        manual_parameters=[
            openapi.Parameter('frame_id', openapi.IN_QUERY, description="ID рами", type=openapi.TYPE_INTEGER),
        ]
    )
    def get(self, request):
        try:
            frame_id = request.GET.get('frame_id')
            if not frame_id:
                return Response({"error": "Missing frame_id parameter."}, status=status.HTTP_400_BAD_REQUEST)

            frame = get_object_or_404(Frame, pk=frame_id)

            forks = Fork.objects.filter(
                wheel_size=frame.wheel_size,
                steerer_type=frame.steerer_type
                # axle_type=frame.axle_type,
                # brake_mount=frame.brake_mount
            )

            # Допуск ±10мм
            if frame.fork_travel:
                forks = forks.filter(
                    travel__gte=frame.fork_travel - 10,
                    travel__lte=frame.fork_travel + 10
                )

            serializer = ForkSerializer(forks, many=True)
            return Response(serializer.data, status=status.HTTP_200_OK)

        except Exception as e:
            return Response({"error": str(e)}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Рисунок А.8 – Реалізація API-методу для підбору вилки на основі ID рами з урахуванням допуску ходу ±10 мм

Джерело: розроблено автором

```python
2 usages  andriimilevskyi
class FrameRecommendationAPIView(APIView):
    andriimilevskyi
    @swagger_auto_schema(
        operation_summary="GET рекомендовані рами на основі зросту та висоти ноги",
        manual_parameters=[
            openapi.Parameter('discipline', openapi.IN_QUERY, description="ID дисципліни", type=openapi.TYPE_INTEGER,
                              required=True),
            openapi.Parameter('height', openapi.IN_QUERY, description="Зріст користувача (в см)",
                              type=openapi.TYPE_NUMBER, required=True),
            openapi.Parameter('inseam', openapi.IN_QUERY, description="Висота ноги (в см)", type=openapi.TYPE_NUMBER,
                              required=True),
        ]
    )
    def get(self, request):
        try:
            discipline_id = request.GET.get('discipline')
            height = request.GET.get('height')
            inseam = request.GET.get('inseam')

            if not (discipline_id and height and inseam):
                return Response({"error": "Всі параметри (discipline, height, inseam) є обов'язковими."},
                                status=status.HTTP_400_BAD_REQUEST)

            discipline_id = int(discipline_id)
            height = float(height)
            inseam = float(inseam)

            target_seatpost = inseam * 0.65

            frames = Frame.objects.filter(
                application_id=discipline_id,
                min_rider_height__lte=height,
                max_rider_height__gte=height
            ).annotate(
                seatpost_diff=Abs(F('seatpost') - target_seatpost)
            ).order_by('seatpost_diff')

            serializer = FrameSerializer(frames, many=True)
            return Response(serializer.data, status=status.HTTP_200_OK)

        except ValueError:
            return Response({"error": "Некоректні числові значення параметрів."},
                            status=status.HTTP_400_BAD_REQUEST)

        except Exception as e:
            return Response({"error": f"Внутрішня помилка сервера: {str(e)}"},
                            status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Рисунок А.9 – API контролер підбору рами на основі фізіологічних параметрів користувача

Джерело: розроблено автором

```python
2 usages  andriimilevskyi
class CranksetRecommendationAPIView(APIView):
    andriimilevskyi
    @swagger_auto_schema(
        operation_summary="Отримати сумісні шатунні системи за ID рами",
        manual_parameters=[
            openapi.Parameter('frame_id', openapi.IN_QUERY, description="ID вибраної рами", type=openapi.TYPE_INTEGER),
            # openapi.Parameter('gearing', openapi.IN_QUERY, description="Кількість передач (наприклад, 12 для 1x12)",
            #                   type=openapi.TYPE_INTEGER),
        ]
    )
    def get(self, request):
        try:
            frame_id = request.GET.get('frame_id')
            # gearing = request.GET.get('gearing')

            if not frame_id:  # or not gearing:
                return Response({"error": "Потрібні параметри 'frame_id'"},  # та 'gearing'.
                                status=status.HTTP_400_BAD_REQUEST)

            frame = Frame.objects.filter(id=frame_id).first()
            if not frame or not frame.chainline:
                return Response({"error": "Раму не знайдено або вона не має відповідний chainline."},
                                status=status.HTTP_404_NOT_FOUND)

            chainline = float(frame.chainline)
            # gearing = int(gearing)

            cranksets = Crankset.objects.filter(
                # gearing=gearing,
                chainline__gte=chainline - 1,
                chainline__lte=chainline + 1
            )

            serializer = CranksetSerializer(cranksets, many=True)
            return Response(serializer.data, status=status.HTTP_200_OK)

        except ValueError:
            return Response({"error": "Неправильні типи параметрів."}, status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response({"error": str(e)}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Рисунок А.10 – API для отримання сумісних шатунних систем за ID рами

Джерело: розроблено автором

```
2 usages  andriimilevskyi
class BottomBracketRecommendationAPIView(APIView):
    andriimilevskyi
    @swagger_auto_schema(
        operation_summary="Підбір каретки за рамою та шатуном",
        manual_parameters=[
            openapi.Parameter('frame_id', openapi.IN_QUERY, description="ID рами", type=openapi.TYPE_INTEGER),
            openapi.Parameter('crankset_id', openapi.IN_QUERY, description="ID шатунів", type=openapi.TYPE_INTEGER),
        ]
    )
    def get(self, request):
        try:
            frame_id = request.GET.get("frame_id")
            crankset_id = request.GET.get("crankset_id")

            if not frame_id or not crankset_id:
                return Response({"error": "Missing frame_id or crankset_id"}, status=status.HTTP_400_BAD_REQUEST)

            frame = Frame.objects.get(id=frame_id)
            crankset = Crankset.objects.get(id=crankset_id)

            # Підбір за типом, діаметром осі та шириною оболонки (допуск ±1 мм)
            brackets = BottomBracket.objects.filter(
                type=frame.bb_standard.bb_name,
                axle_diameter=crankset.axle_diameter,
                shell_width__gte=frame.bb_standard.shell_width - 1,
                shell_width__lte=frame.bb_standard.shell_width + 1
            )

            serializer = BottomBracketSerializer(brackets, many=True)
            return Response(serializer.data, status=status.HTTP_200_OK)

        except Frame.DoesNotExist:
            return Response({"error": "Frame not found"}, status=status.HTTP_404_NOT_FOUND)
        except Crankset.DoesNotExist:
            return Response({"error": "Crankset not found"}, status=status.HTTP_404_NOT_FOUND)
        except Exception as e:
            return Response({"error": str(e)}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Рисунок А.11 – API для підбору каретки за рамою та шатуном

Джерело: розроблено автором

```python
2 usages  andriimilevskyi
class CassetteByDerailleurShifterAPIView(APIView):
    andriimilevskyi
    @swagger_auto_schema(
        operation_summary="Підібрати касети за перемикачем та манеткою",
        manual_parameters=[
            openapi.Parameter('derailleur_id', openapi.IN_QUERY, description="ID перемикача",
                              type=openapi.TYPE_INTEGER),
            openapi.Parameter('shifter_id', openapi.IN_QUERY, description="ID манетки", type=openapi.TYPE_INTEGER),
        ]
    )
    def get(self, request):
        derailleur_id = request.GET.get("derailleur_id")
        shifter_id = request.GET.get("shifter_id")

        if not derailleur_id or not shifter_id:
            return Response({"error": "Необхідно передати derailleur_id і shifter_id"},
                            status=status.HTTP_400_BAD_REQUEST)

        try:
            derailleur = Derailleur.objects.get(id=derailleur_id)
            shifter = Shifter.objects.get(id=shifter_id)
        except Derailleur.DoesNotExist:
            return Response({"error": "Перемикач не знайдено"}, status=status.HTTP_404_NOT_FOUND)
        except Shifter.DoesNotExist:
            return Response({"error": "Манетка не знайдена"}, status=status.HTTP_404_NOT_FOUND)

        # Підбір касет
        if derailleur.gearing != shifter.gearing:
            return Response({"error": "Gearing перемикача і манетки не співпадає"}, status=status.HTTP_400_BAD_REQUEST)

        cassettes = Cassette.objects.filter(
            gearing=derailleur.gearing,
            smallest_gear__gte=derailleur.smallest_gear,
            biggest_gear__lte=derailleur.biggest_gear,
        )

        serializer = CassetteSerializer(cassettes, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

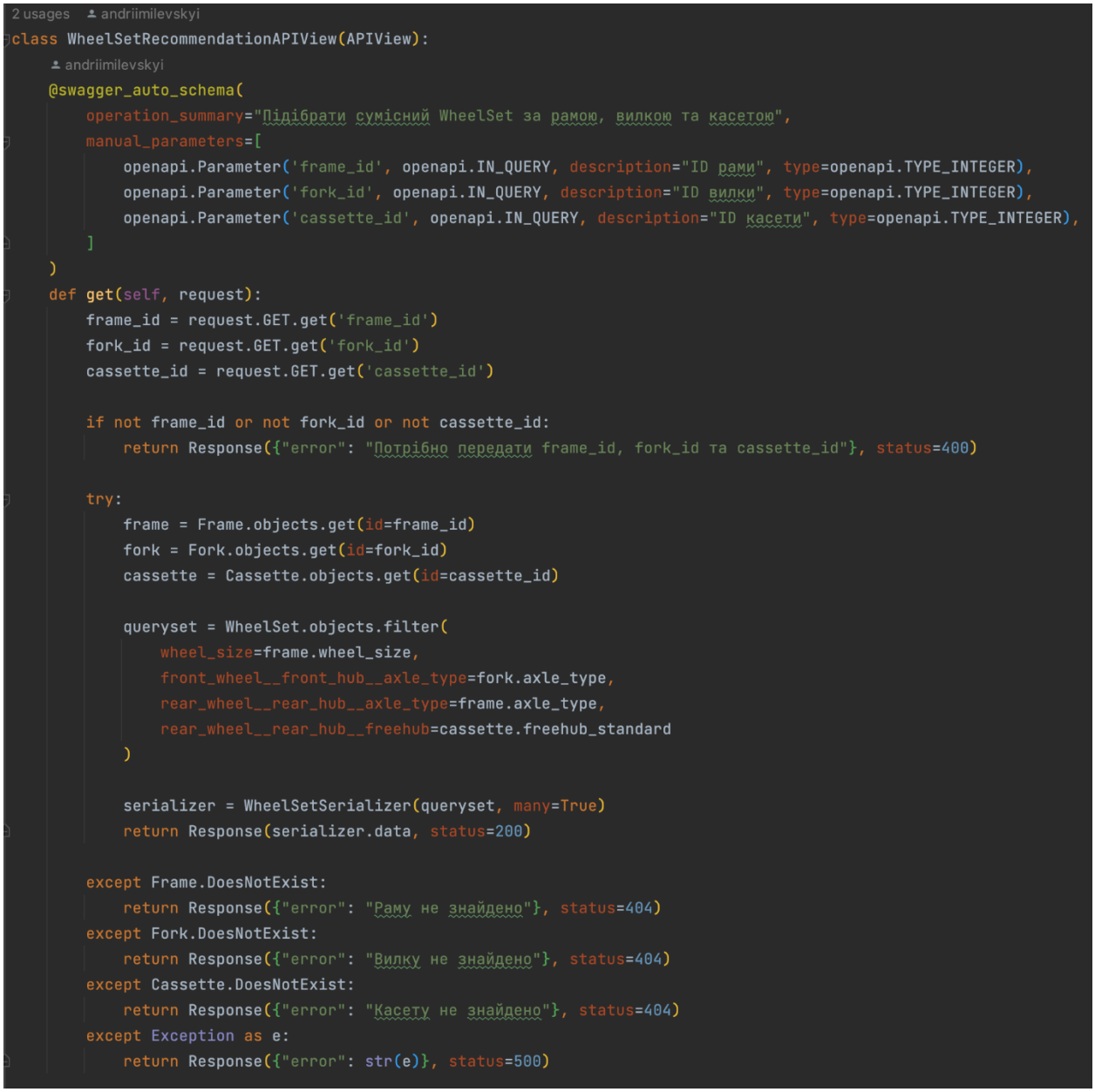
Рисунок А.12 – API для підбору касет за перемикачем та манеткою

Джерело: розроблено автором

```python
class ChainByComponentsAPIView(APIView):
    andriimilevskyi
    @swagger_auto_schema(
        operation_summary="Підібрати ланцюги за касетою, перемикачем, шатунами та швидкістю",
        manual_parameters=[
            openapi.Parameter('cassette_id', openapi.IN_QUERY, description="ID касети", type=openapi.TYPE_INTEGER,
                              required=True),
            openapi.Parameter('derailleur_id', openapi.IN_QUERY, description="ID перемикача", type=openapi.TYPE_INTEGER,
                              required=True),
            openapi.Parameter('shifter_id', openapi.IN_QUERY, description="ID манетки", type=openapi.TYPE_INTEGER,
                              required=True),
            openapi.Parameter('crankset_id', openapi.IN_QUERY, description="ID шатунів", type=openapi.TYPE_INTEGER,
                              required=True),
        ]
    )
    def get(self, request):
        cassette_id = request.GET.get("cassette_id")
        derailleur_id = request.GET.get("derailleur_id")
        shifter_id = request.GET.get("shifter_id")
        crankset_id = request.GET.get("crankset_id")

        if not all([cassette_id, derailleur_id, shifter_id, crankset_id]):
            return Response({"error": "Необхідно передати cassette_id, derailleur_id, shifter_id і crankset_id"},
                            status=status.HTTP_400_BAD_REQUEST)

        try:
            cassette = Cassette.objects.get(id=cassette_id)
            derailleur = Derailleur.objects.get(id=derailleur_id)
            shifter = Shifter.objects.get(id=shifter_id)
            crankset = Crankset.objects.get(id=crankset_id)
        except Cassette.DoesNotExist:
            return Response({"error": "Касета не знайдена"}, status=status.HTTP_404_NOT_FOUND)
        except Derailleur.DoesNotExist:
            return Response({"error": "Перемикач не знайдено"}, status=status.HTTP_404_NOT_FOUND)
        except Shifter.DoesNotExist:
            return Response({"error": "Манетка не знайдена"}, status=status.HTTP_404_NOT_FOUND)
        except Crankset.DoesNotExist:
            return Response({"error": "Шатуни не знайдені"}, status=status.HTTP_404_NOT_FOUND)

        # Перевірка сумісності gearing
        if not (cassette.gearing == derailleur.gearing == shifter.gearing == crankset.gearing):
            return Response({"error": "Gearing касети, перемикача, манетки і шатунів не співпадає"},
                            status=status.HTTP_400_BAD_REQUEST)

        # Підбір ланцюгів за gearing (кількістю передач)
        chains = Chain.objects.filter(gearing=cassette.gearing)

        # За бажанням можна додати ще логіку фільтрації за типом замка, напрямком тощо

        serializer = ChainSerializer(chains, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

Рисунок А.13 – API для підбору ланцюгів за касетою, перемикачем, манеткою та шатуном

Джерело: розроблено автором



```
2 usages  andriimilevskyi
class WheelSetRecommendationAPIView(APIView):
    andriimilevskyi
    @swagger_auto_schema(
        operation_summary="Підібрати сумісний WheelSet за рамою, вилкою та касетою",
        manual_parameters=[
            openapi.Parameter('frame_id', openapi.IN_QUERY, description="ID рами", type=openapi.TYPE_INTEGER),
            openapi.Parameter('fork_id', openapi.IN_QUERY, description="ID вилки", type=openapi.TYPE_INTEGER),
            openapi.Parameter('cassette_id', openapi.IN_QUERY, description="ID касети", type=openapi.TYPE_INTEGER),
        ]
    )
    def get(self, request):
        frame_id = request.GET.get('frame_id')
        fork_id = request.GET.get('fork_id')
        cassette_id = request.GET.get('cassette_id')

        if not frame_id or not fork_id or not cassette_id:
            return Response({"error": "Потрібно передати frame_id, fork_id та cassette_id"}, status=400)

        try:
            frame = Frame.objects.get(id=frame_id)
            fork = Fork.objects.get(id=fork_id)
            cassette = Cassette.objects.get(id=cassette_id)

            queryset = WheelSet.objects.filter(
                wheel_size=frame.wheel_size,
                front_wheel__front_hub__axle_type=fork.axle_type,
                rear_wheel__rear_hub__axle_type=frame.axle_type,
                rear_wheel__rear_hub__freehub=cassette.freehub_standard
            )

            serializer = WheelSetSerializer(queryset, many=True)
            return Response(serializer.data, status=200)

        except Frame.DoesNotExist:
            return Response({"error": "Раму не знайдено"}, status=404)
        except Fork.DoesNotExist:
            return Response({"error": "Вилку не знайдено"}, status=404)
        except Cassette.DoesNotExist:
            return Response({"error": "Касету не знайдено"}, status=404)
        except Exception as e:
            return Response({"error": str(e)}, status=500)
```

Рисунок А.14 – API для підбору сумісного набору коліс за рамою, вилкою та касетою

Джерело: розроблено автором

```python
class MTBRecommendationAPIView(APIView):
    andriimilevskyi *
    def get(self, request, *args, **kwargs):
        bike_id = request.GET.get("bike_id")
        target_bike = MTBBike.objects.get(id=bike_id)
        bikes = MTBBike.objects.exclude(id=bike_id)
        data = []  # Створення датафрейму
        for bike in bikes:
            data.append({
                "id": bike.id,
                "application": bike.frame.application.id if bike.frame.application else 0,
                "material": bike.frame.material.id if bike.frame.material else 0,
                "brake_type": bike.brake.type if bike.brake else '',
                "weight": float(bike.weight) if bike.weight else 0.0,
                "price": float(bike.price) if bike.price else 0.0,
            })
        df = pd.DataFrame(data)
        if df.empty:
            return Response([], status=200)
        target_data = {  # Об'єкт порівняння
            "application": target_bike.frame.application.id if target_bike.frame.application else 0,
            "material": target_bike.frame.material.id if target_bike.frame.material else 0,
            "brake_type": target_bike.brake.type if target_bike.brake else '',
            "weight": float(target_bike.weight) if target_bike.weight else 0.0,
            "price": float(target_bike.price) if target_bike.price else 0.0,
        }
        df_full = df.copy()
        df_full["target_application"] = target_data["application"]
        df_full["target_material"] = target_data["material"]
        df_full["target_brake_type"] = target_data["brake_type"]
        df_full["target_weight"] = target_data["weight"]
        df_full["target_price"] = target_data["price"]
        scaler = MinMaxScaler()  # Нормалізація числових
        numeric = scaler.fit_transform(df_full[["weight", "price", "target_weight", "target_price"]])
        # Категорії
        df_full["application_sim"] = df_full["application"] == df_full["target_application"]
        df_full["material_sim"] = df_full["material"] == df_full["target_material"]
        df_full["brake_type_sim"] = df_full["brake_type"] == df_full["target_brake_type"]
        # Вектор ознак
        feature_matrix = pd.DataFrame(numeric, columns=["weight", "price", "target_weight", "target_price"])
        feature_matrix["application_sim"] = df_full["application_sim"].astype(int)
        feature_matrix["material_sim"] = df_full["material_sim"].astype(int)
        feature_matrix["brake_type_sim"] = df_full["brake_type_sim"].astype(int)
        # Розділити на запит і решту
        query_vector = \
            feature_matrix[["target_weight", "target_price", "application_sim", "material_sim", "brake_type_sim"]].iloc[
                0].values.reshape(1, -1)
        corpus = feature_matrix[["weight", "price", "application_sim", "material_sim", "brake_type_sim"]].values
        # Подібність
        similarity = cosine_similarity(query_vector, corpus)[0]
        df_full["similarity"] = similarity
        top_ids = df_full.sort_values("similarity", ascending=False)["id"].head(6)

        recommendations = MTBBike.objects.filter(id__in=top_ids)
        serializer = MTBBikeSerializer(recommendations, many=True)
        return Response(serializer.data, status=200)
```

Рисунок А.15 – Реалізація контентно-орієнтованого алгоритму рекомендацій

Джерело: розроблено автором