МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА БЕКЕНДУ ІНТЕРНЕТ-МАГАЗИНУ СПОРТИВНОГО ХАРЧУВАННЯ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ DJANGO»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.І. Гречаний
(ініціали та прізвище)
Керівник професор кафедри програмної інженерії, професор, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри комп'ютерних наук, доцент, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Гречаному Вячеславу Ігоровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка бекенду інтернет-магазину спортивного харчування з використанням фреймворку Django

керівник роботи Чопоров Сергій Вікторович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи 03.06.2024 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка REST FULL сервісу для спотривного магазину.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	05.02.2024	
3.	Обробка методичних та теоретичних	01.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	19.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ В.І. Гречаний
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка бекенду інтернет-магазину спортивного харчування з використанням фреймворку Django»: 44 с., 24 рис., 7 джерел.

БЕКЕНД, ЕЛЕКТРОНА КОМЕРЦІЯ, РОЗРОБКА, СПОРТИВНЕ ХАРЧУВАННЯ, API, DJANGO, DJANGO REST FRAMEWORK.

Розробка бекенду частини вебдодатку електронної комерції для магазину спортивного харчування, який забезпечить зручність та ефективність для клієнтів, що шукають продукти для покращення свого здоров'я та фізичної форми. Оптимізація алгоритмів та використання передових технологій можуть значно полегшити процес замовлення товарів, забезпечуючи високий рівень задоволення клієнтів.

Головною метою цього дослідження є проєктування та дослідження особливостей програмування розробки вебдодатків у сфері електронної комерції для магазинів спортивного харчування. Під час дослідження, виявленно, що ефективні вебдодатки в цій галузі мають суттєвий вплив на підвищення продуктивності та конкурентоспроможності бізнесу. Дослідження та порівняльний аналіз різних підходів та технологій дозволили ідентифікувати ключові фактори успіху.

На основі результатів аналізу розроблений вебдодаток для магазину спортивного харчування з урахуванням сучасних підходів та технологій розробки програмного забезпечення, таких як методологія Scrum, авторизація та шифрування (JWT, HTTPS), Django, Django REST framework, Sql3Lite. Цей додаток дозволяє клієнтам зручно та швидко замовляти необхідні продукти, а власникам магазину – ефективно управляти асортиментом та замовленнями. Такий вебдодаток може значно полегшити процес електронної комерції для магазину спортивного харчування, забезпечуючи високу якість обслуговування для клієнтів та підвищуючи конкурентоспроможність бізнесу.

SUMMARY

Bachelor’s qualifying paper “Development of a Backend of an Online Sports Nutrition Store Using the Django Framework”: 44 pages, 24 figures, 7 references.

BACKEND, E-COMMERCE, DEVELOPMENT, SPORTS NUTRITION, API, DJANGO, DJANGO REST FRAMEWORK.

The backend development of an e-commerce web application for a sports nutrition store ensures convenience and efficiency for customers seeking products to improve their health and fitness. Optimizing algorithms and using advanced technologies can significantly ease the product ordering process, providing a high level of customer satisfaction.

The primary goal of this research is to design and explore the programming features of developing web applications in the e-commerce sector for sports nutrition stores. Our research has shown that effective web applications in this field significantly impact the productivity and competitiveness of the business. Research and comparative analysis of different approaches and technologies have identified key success factors.

Based on the analysis results, a web application for a sports nutrition store was developed, considering modern software development approaches and technologies such as Scrum methodology, authorization and encryption (JWT, HTTPS), Django, Django REST framework, and SQLite. This application allows customers to conveniently and quickly order the necessary products, while store owners can efficiently manage their inventory and orders. Such a web application can significantly streamline the e-commerce process for a sports nutrition store, ensuring high-quality customer service and increasing business competitiveness.

ЗМІСТ

ВСТУП

У зв’язку з стрімким розвитком технологій та їх інтеграцією у наше повсякденне життя, перехід більшості видів діяльності до інтернету став важливим та невід’ємним аспектом сучасного світу. Цей прогрес обумовлений рядом факторів, включаючи поліпшення обчислювальної потужності комп’ютерів, збільшення швидкості інтернет-з’єднань, зменшення розміру електронних пристроїв та підвищення їх енергоефективності.

Зростання числа користувачів інтернету по всьому світу свідчить про те, що доступ до мережі став масовим явищем. Люди використовують Інтернет для різних цілей, включаючи спілкування, пошук інформації, покупки товарів та послуг, розваги, освіту та багато іншого. Це зумовлює зростання попиту на якісні та надійні інтернет-послуги.

Такий розквіт технологій сприяє переходу компаній та послуг до онлайн-простору, що стає важливим кроком у розвитку бізнесу та досягненні успіху. Інтернет став найбільш доступним та широко використовуваним засобом комунікації, що суттєво впливає на розвиток електронної комерції.

Електронна комерція – це процес купівлі та продажу товарів або послуг через Інтернет, що дозволяє компаніям пропонувати нові продукти та послуги, вдосконалювати існуючі технології та способи продажу, розширювати аудиторію та збільшувати прибутки [1]. Крім того, такі сервіси надають безліч можливостей для реклами та просування товарів та послуг. Компанії можуть використовувати електронну комерцію як платформу для ефективного просування своїх пропозицій та досягання більш широкої аудиторії. Це дає можливість максимально використовувати потенціал електронної комерції з метою збільшення продажів та популярності продуктів і послуг. За допомогою електронної комерції, компанії можуть використовувати соціальні медіа для реклами своїх товарів та послуг, а також співпрацювати з іншими платформами та ресурсами для досягнення більшого кола потенційних клієнтів.

Об’єктом дослідження є процес розробки вебдодатків для електронної комерції та торгівлі. Предметом дослідження виступають сучасні підходи, методи та технології створення таких додатків. Метою роботи є аналіз особливостей розробки вебдодатку для електронної комерції в сфері торгівлі спортивним харчуванням з використанням актуальних технологій та розробка прототипу такого додатку.

Отже, розвиток електронної комерції та маркетингова орієнтація бізнес-структур робить досить актуальною темою створення віртуального майданчика торгівлі та бартерного обміну, що буде працювати як вебдодаток та надавати необхідний функціонал. Цей додаток буде корисним як для компаній, так і для звичайних користувачів, які бажають активно взаємодіяти у сфері електронної комерції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

У галузі електронної комерції проводяться значні наукові дослідження та аналізи, присвячені різним аспектам цього сегменту [1, 2]. Вітчизняні вчені та автори розглядають ключові поняття та принципи електронної комерції та Інтернет-торгівлі [1]. Вони відзначають, що електронний бізнес можна описати як проведення ділової діяльності через мережу Інтернету. Сьогодні спостерігається збільшення цього типу послуг та поступове перетворення вебсайтів у повноцінні інтернет-магазини. Це дозволяє підприємствам розширити географію свого присутності на ринку, швидше реалізувати товари та послуги та отримати додаткові можливості для маркетингу та реклами.

Електронна комерція, також відома як е-комерція, є одним із способів здійснення електронного бізнесу. Це широкий спектр інтерактивних методів, що використовуються для надання споживачам товарів та послуг. Під е-комерцією розуміють будь-які форми ділових операцій, де сторони взаємодіють за допомогою електронних технологій, а не фізичного обміну або прямого контакту.

Останні тренди та перспективи розвитку були розглянуті у дослідженні [2]. Автори підкреслюють, що внаслідок пандемії COVID-19 підходи до міжнародного бізнесу радикально змінилися. Світова економіка пройшла глобальну трансформацію, зосереджуючись на цифровізації. Пандемія прискорила використання цифрових технологій компаніями, що змусило їх змінити власні бізнес-моделі для адаптації до нових умов.

Електронна комерція стала надзвичайно важливою для багатьох міжнародних бізнес-суб’єктів і стала єдиним способом збереження або зростання конкурентних позицій на ринку. Модернізація бізнес-процесів за допомогою новітніх інтернет-технологій не тільки створює можливості для

розширення діяльності, але також стимулює формування нових видів діяльності та суб'єктів бізнесу через створення нового типу інфраструктури.

Як показано на рисунку 1.1, до 2024 року очікується майже п'ятикратне зростання роздрібних продажів у цій галузі порівняно з 2014 роком. Це свідчить про значне підвищення популярності та прийняття електронної комерції як основного каналу продажу для багатьох бізнесів і споживачів. Тобто цифровізація торгівлі та інших бізнес-процесів призводить до збільшення уваги до електронної комерції при укладанні міжнародних двосторонніх та багатосторонніх угод. Впровадження цифрових рішень та інтернет-технологій відкриває нові можливості для привернення нових клієнтів та розширення географії присутності на ринках, що дозволяє підприємствам ефективно просувати свої товари та послуги у міжнародному масштабі.

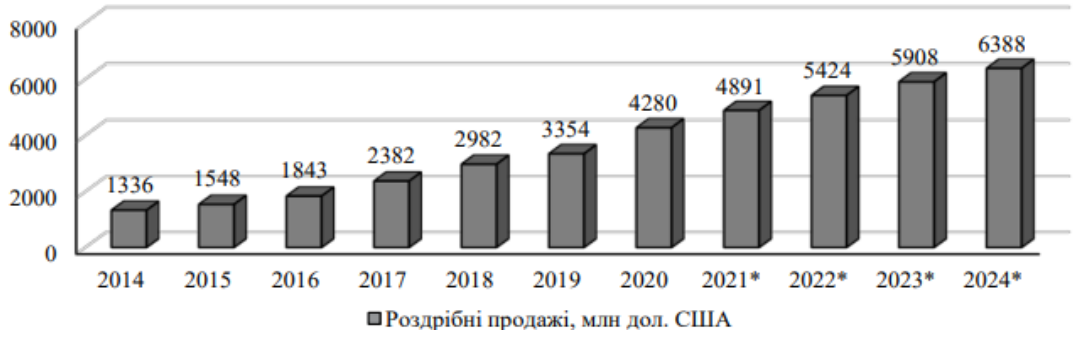

Рисунок 1.1 – Роздрібні продажі засобами електронної комерції в глобальному вимірі, млн. дол. США

Усе це свідчить про те, що електронна комерція в міжнародному бізнесі є необхідним і динамічно розвиваючимся напрямком, який впливає на стратегії компаній. Ріст роздрібних продажів в електронній комерції вказує на збільшення популярності онлайн-покупок серед споживачів і важливість присутності бізнесів в цьому сегменті. Компанії пристосовуються до змін, впроваджуючи нові технології, розвиваючи електронні платформи, модернізуючи маркетингові стратегії та забезпечуючи зручність та безпеку для своїх клієнтів.

1.2 Аналіз програмних продуктів-аналогів

Сайти спортивного харчування – це онлайн-ресурси, які спеціалізуються на продажі різних видів спортивного харчування, таких як протеїнові батончики, порошки для приготування коктейлів, вітаміни, амінокислоти та інші добавки для підтримки фізичної форми та збільшення м’язової маси.

Розглянемо декілька прикладів популярних сайтів спортивного харчування.

MyProtein – це один з найбільших онлайн-магазинів спортивного харчування у Великобританії та Європі. Сайт пропонує широкий вибір високоякісних протеїнових сумішей, амінокислот, вітамінів, спеціалізованих напоїв та інших додатків для активних людей. MyProtein відомий своїми власними торговими марками та часто пропонує вигідні акційні пропозиції.

Інтерфейс головної сторінки сайту наведено на відповідному рисунку (див. рис. 1.2).

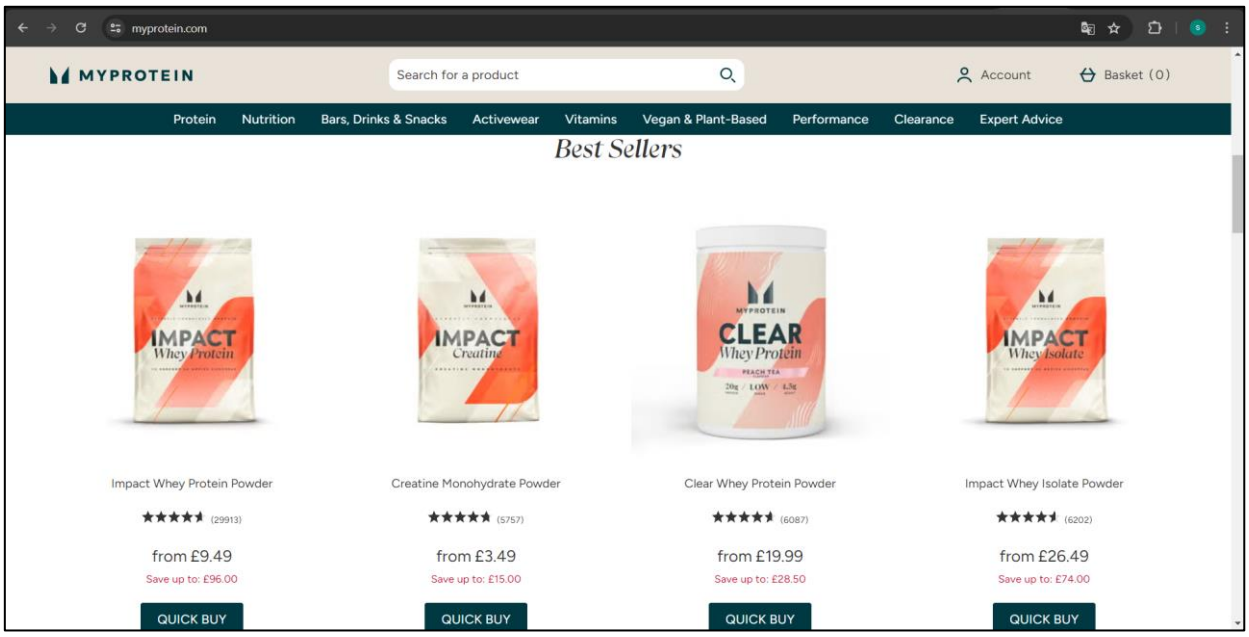

Рисунок 1.2 – Інтерфейс головної сторінки сайту MyProtein

Головні переваги:

– широкий асортимент товарів;
– висока якість продукції;

– доступні ціни;

– зручна система замовлень;

– безкоштовна доставка при певних умовах.

BioTech USA – американська компанія, яка виробляє та продає широкий асортимент спортивного харчування, включаючи протеїни, гейнери, жиророзчинні вітаміни, ферменти та добавки для схуднення. Вони пропонують продукцію власного виробництва за доступними цінами.

Інтерфейс головної сторінки платформи наведено на відповідному рисунку (див. рис. 1.3).

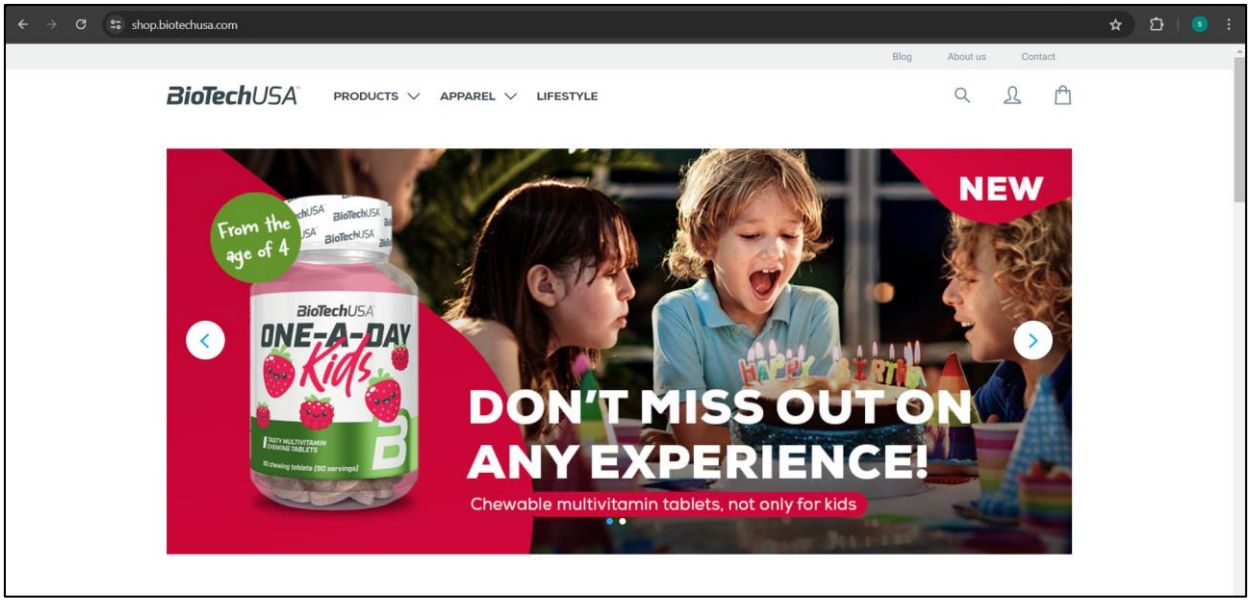

Рисунок 1.3 – Головна сторінка сервісу BioTech USA

Головні переваги:

– широкий вибір власної продукції;

– доступні ціни;

– інформативний сайт з детальним описом товарів;

– можливість придбати товари оптом.

Scitec Nutrition – угорська компанія, яка виробляє високоякісні добавки для спортсменів та любителів активного способу життя. Їхній асортимент включає протеїни, гейнери, вітаміни, амінокислоти та багато іншого.

Інтерфейс головної сторінки платформи наведено на відповідному рисунку (див. рис. 1.4).

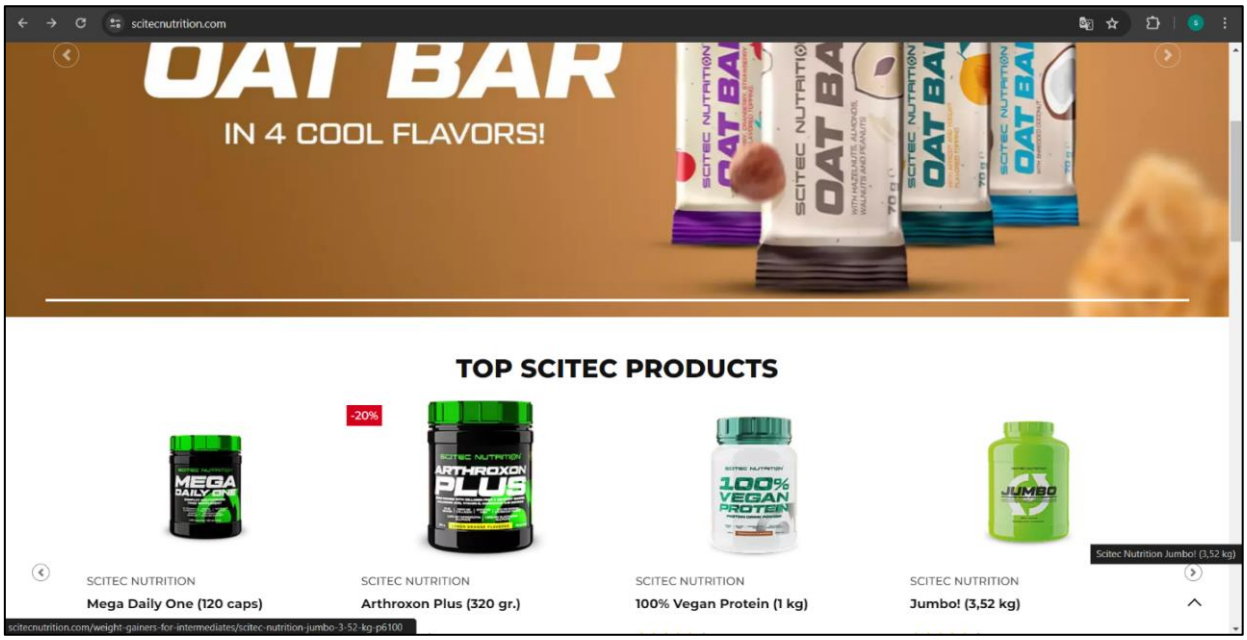

Рисунок 1.4 – Головна сторінка сервісу Scitec Nutrition

Головні переваги:

– висока якість продукції;
– широкий асортимент додатків;
– зручна онлайн-крамниця;
– безкоштовна консультація фахівців.

Доречі, багато великих спортивних брендів, таких як Optimum Nutrition, Dymatize та інші, також мають власні інтернет-магазини, де можна придбати їхню продукцію.

До потенційних недоліків таких сайтів можна віднести високі ціни на деякі види товарів, вартість доставки (хоча часто пропонуються безкоштовні варіанти) та, як і у випадку з усіма онлайн-покупками, дещо складнішу процедуру повернення/обміну товару.

1.3 Постановка завдання

Мета кваліфікаційної роботи – створити бекенд для вебдодатку інтернет-магазину спортивного харчування з використанням фреймворку Django та Django REST Framework.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати предметну область електронної комерції та специфіку продажу спортивного харчування, визначити необхідні функціональні вимоги до майбутнього додатку;
- спроєктувати модель даних для зберігання інформації про товари, категорії, замовлення, користувачів тощо; визначити необхідні сутності, їх атрибути та зв’язки між ними;
- використовуючи Django ORM (Object-Relational Mapping), створити моделі даних та відповідні міграції для генерації схеми бази даних;
- розробити бекенд застосунку з використанням Django, налаштувати необхідні маршрути (urls), вью-функції та шаблони для обробки HTTP-запитів;
- інтегрувати Django REST Framework для створення RESTful API;
- використовуючи можливості Django REST Framework, описати серіалізатори для перетворення моделей даних у формат JSON або інший необхідний формат для передачі даних через API;
- реалізувати необхідну функціональність;
- забезпечити безпеку додатку, налаштувавши належні дозволи та обмеження доступу до різних частин API та функціоналу;
- задокументувати створене API, описавши доступні ендпоінти.

Використання Django дозволить швидко розробити надійний та масштабований бекенд для інтернет-магазину спортивного харчування, в той час як Django REST Framework забезпечить зручний спосіб створення API для взаємодії з фронтенд-частиною додатку.

1.4 Аналіз сучасних технологій розробки серверної частини

Розробка серверної частини для інтернет-магазину спортивного харчування є важливою задачею, яка вимагає ретельного аналізу існуючих технологій та фреймворків. Для цього проєкту, враховуючи особливості Python та його фреймворків, можна розглянути наступні варіанти.

Django – це високорівневий фреймворк Python для розробки вебдодатків, який забезпечує швидку та безпечну розробку. Він має ряд переваг.

Швидка розробка: Django дозволяє розробникам створювати вебдодатки швидше завдяки своїй потужній функціональності та вбудованим компонентам, таким як система автентифікації, адміністративна панель та маршрутизація URL.

Безпека: Django має вбудовані механізми безпеки, які захищають від багатьох типових вразливостей вебдодатків, таких як SQL-ін’єкції, міжсайтовий скриптинг (XSS) та підробка міжсайтових запитів (CSRF).

Масштабованість: Django розроблений з урахуванням масштабованості, що дозволяє легко розширювати додатки для обробки великих обсягів трафіку та даних.

Велика екосистема: Django має велику та активну спільноту розробників, що означає широку доступність бібліотек, плагінів та ресурсів для розробки.

Недоліки Django можуть включати більш високі вимоги до ресурсів у порівнянні з легшими фреймворками, а також деяку складність при налаштуванні та розгортанні додатків.

Flask – це мінімалістичний фреймворк Python, який відомий своєю гнучкістю та легкістю. Наведемо переваги.

Простота: Flask має простий та зрозумілий синтаксис, що робить його ідеальним вибором для невеликих проєктів або прототипування.

Гнучкість: Flask дозволяє розробникам вибирати лише ті компоненти,

які їм потрібні, уникаючи зайвих залежностей та складності.

Розширюваність: Flask підтримує велику кількість розширень, які можна легко інтегрувати для додавання нової функціональності.

Швидкодія: завдяки своїй легкості, Flask може забезпечувати високу продуктивність та швидкість обробки запитів.

Однак Flask може мати деякі недоліки, такі як відсутність вбудованих функцій для більш складних завдань (наприклад, система автентифікації або адміністративна панель) та менша доступність ресурсів та бібліотек у порівнянні з більш популярними фреймворками.

FastAPI – це сучасний, високопродуктивний фреймворк Python для створення API з використанням асинхронної обробки запитів. Розглянемо переваги.

Висока продуктивність: FastAPI використовує асинхронні операції для максимальної ефективності та швидкодії.

Автоматична генерація документації: FastAPI автоматично генерує інтерактивну документацію API на основі визначень моделей даних та маршрутів.

Проста валідація даних: FastAPI забезпечує вбудовану валідацію вхідних даних на основі визначень моделей даних.

Підтримка асинхронних операцій: FastAPI повністю підтримує асинхронні операції, що робить його ідеальним вибором для створення високопродуктивних та масштабованих API.

Недоліки FastAPI можуть включати відносно невелику спільноту та екосистему у порівнянні з більш популярними фреймворками, а також фокус на створенні API, що може зробити його менш підходящим для розробки повноцінних вебдодатків.

Вибір фреймворку залежить від конкретних вимог проєкту, наявних ресурсів та досвіду команди розробників. Для інтернет-магазину спортивного харчування, де важливими є швидкодія, безпека та масштабованість, Django або FastAPI можуть бути гарними варіантами. Проте, якщо потрібна швидка

розробка невеликого додатку, Flask також може розглядатися як варіант. Таким чином, для розробки серверної частини інтернет-магазину спортивного харчування найкращим вибором буде використання фреймворку Django на мові Python.

1.5 Опис інструментарію

Серверна частина інтернет магазину спортивних товарів на базі Python backend була реалізована за допомогою фреймворку Django.

Django є високорівневим вебфреймворком, спеціально розробленим для швидкого і зручного створення складних вебдодатків. Цей фреймворк має багато переваг, які роблять його відмінним вибором для розробки системи управління замовленнями в магазині техніки.

Однією з ключових особливостей Django є його модульність. Фреймворк дозволяє розділити функціональність на набір окремих додатків, що спрощує розробку та розширення системи. Для цієї системи можна створити окремі додатки для керування замовленнями, товарними позиціями, клієнтами тощо.

Додатково, Django має вбудований REST framework, який надає потужний інструментарій для створення вебсервісів API. Django REST framework (DRF) дозволяє визначати серіалізатори, які перетворюють дані у формати, зручні для передачі через мережу, і надає можливості для автентифікації, авторизації, обробки запитів, створення моделей представлення та багато іншого. DRF є потужним і гнучким інструментом для розробки API в Django, який в свою чергу був використаний в проєкті (див. рис. 1.5).

Нарешті, Django ORM забезпечує зручний спосіб взаємодії з базою даних (див. рис. 1.6). Він дозволяє використовувати Python-об’єкти для виконання різних операцій з базою даних, замість прямого писання SQL-запитів. Django ORM автоматично генерує необхідні SQL-запити на основі

визначених моделей даних та забезпечує зручний інтерфейс для фільтрації, сортування, об’єднання даних та багато іншого.

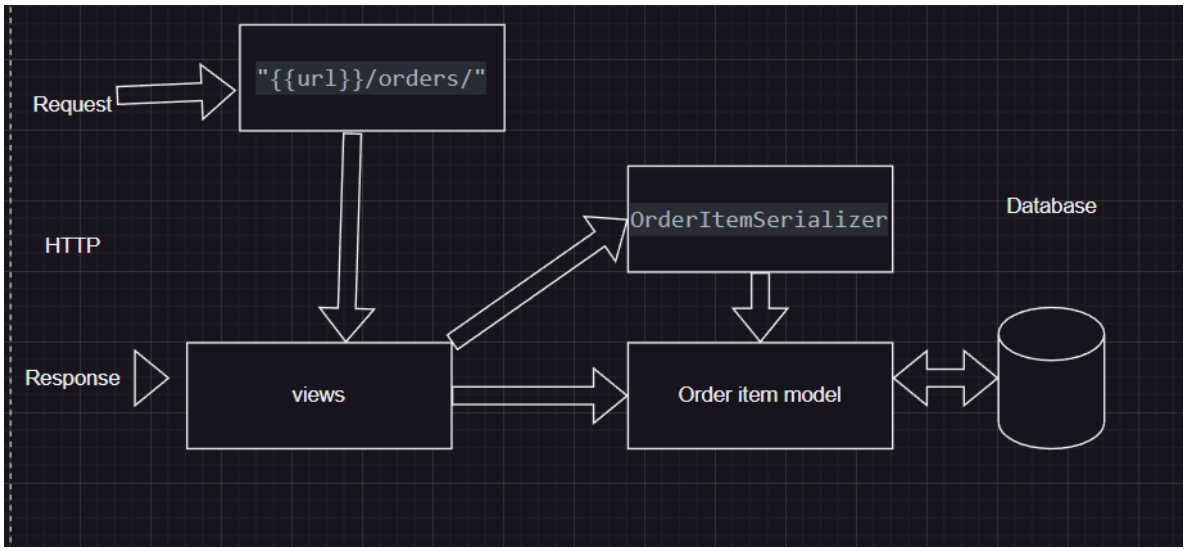

Рисунок 1.5 – Принцип роботи Django застосунку

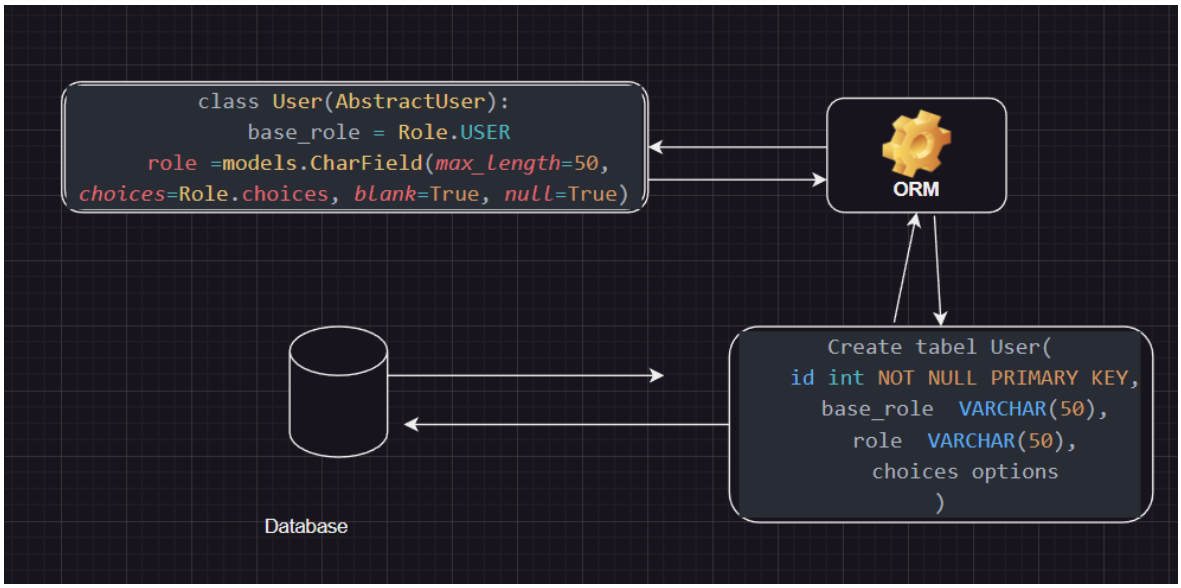

Рисунок 1.6 – Таблиця приклад роботи Django ORM

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Проєктування системи вебдодатку

Під час етапу проєктування було виконано структурно-функціональне моделювання, яке дозволило детально прослідкувати виконання всіх етапів проєктування вебзастосунку для спортивного магазину з використанням Django ORM. Цей аналітичний підхід також дозволив проаналізувати можливі напрямки удосконалення роботи системи з метою покращення її ефективності та задоволення потреб користувачів.

Для виділення основних можливостей системи було розроблено діаграми варіантів використання, які відображають взаємодію акторів (користувачів та адміністраторів) з системою та описують різні сценарії взаємодії. Діаграми варіантів використання для користувачів та адміністраторів наведені на рисунках 2.1 та 2.2 відповідно.

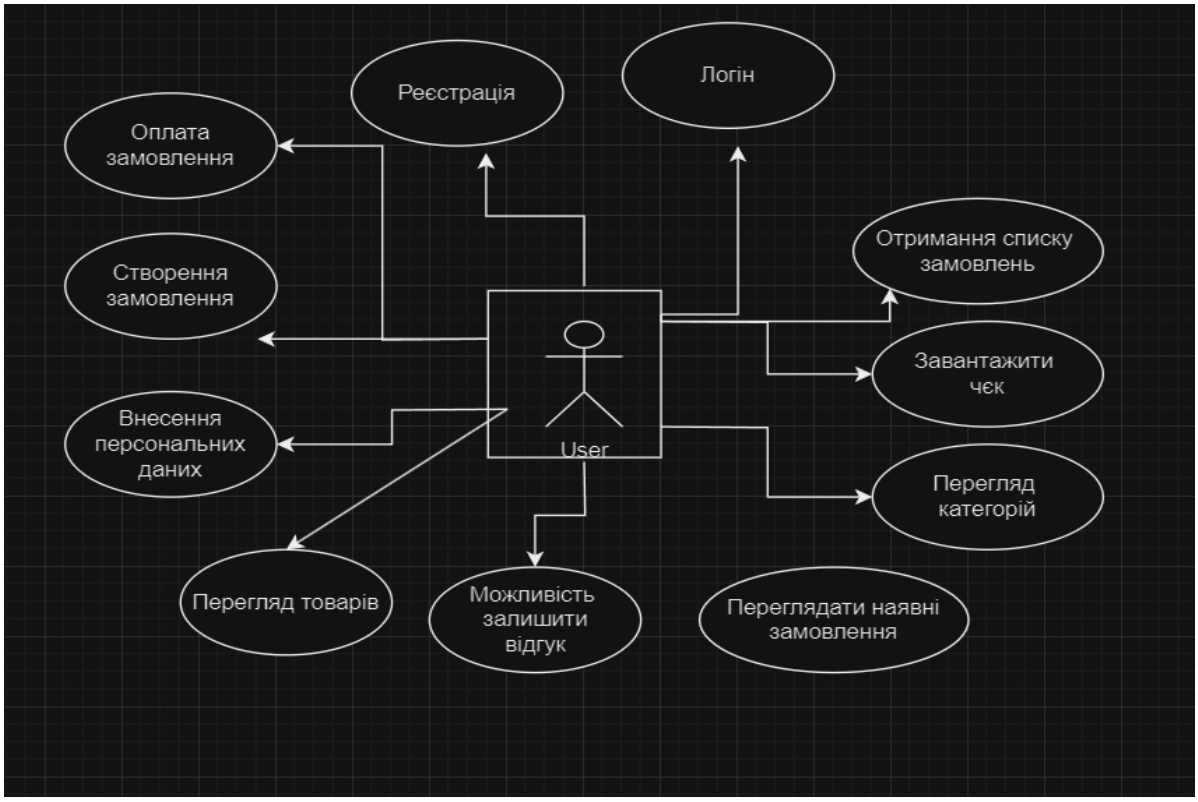

Рисунок 2.1 – Діаграма варіантів використання для користувача

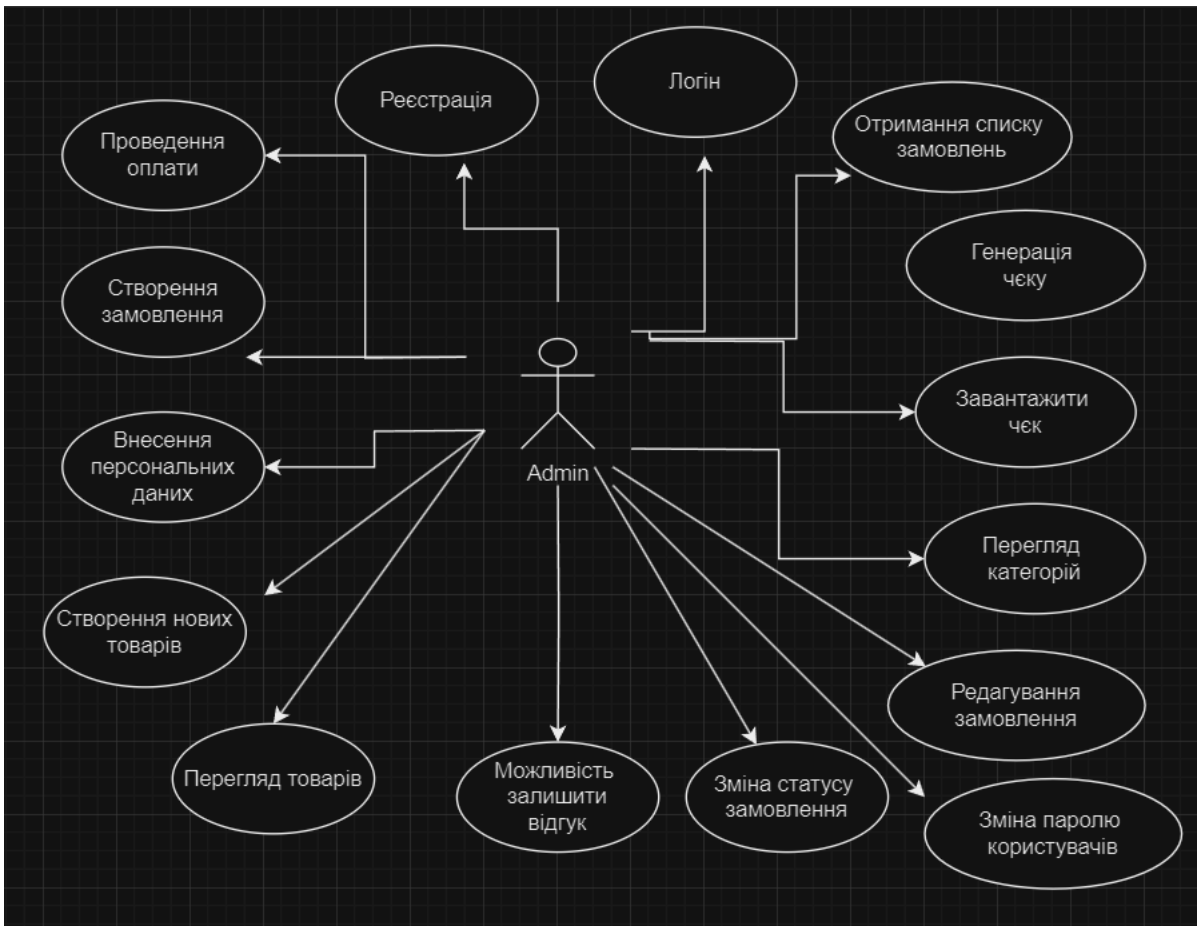

Рисунок 2.2 – Діаграма варіантів використання для адміністратора

Діаграма з рисунку 2.1 ілюструє прецеденти для користувача вебдодатку, ймовірно, електронної комерції або магазину спортивного харчування.

– оплата замовлення: це прецедент, що дозволяє користувачеві здійснити оплату для завершення процесу замовлення товарів;
– створення замовлення: прецедент, який передбачає, що користувач формує нове замовлення, додаючи необхідні товари;
– реєстрація: дозволяє новому користувачеві зареєструватися та створити обліковий запис у системі;
– логін: прецедент для авторизації зареєстрованих користувачів шляхом введення облікових даних;
– отримання списку замовлень: користувач може переглядати свій список активних або виконаних замовлень;
– внесення персональних даних: прецедент для введення та оновлення

особистої інформації користувача, такої як адреса, контактні дані тощо;

- перегляд товарів: дозволяє користувачеві переглядати каталог доступних товарів та їх опис;
- перегляд категорій товарів: користувач може вибирати та переглядати товари за певними категоріями;
- перегляд товарів за навчанням: можливість отримувати рекомендації товарів на основі минулих покупок або переглядів;
- завантажити як: прецедент, що дозволяє користувачеві завантажити свій список покупок для подальшого використання;
- можливість залишити відгук: користувачі можуть залишати відгуки чи рецензії на придбані товари.

Таким чином, діаграма охоплює основні прецеденти використання для користувачів вебдодатку електронної комерції, включаючи процеси замовлення, перегляд товарів, управління обліковим записом та інші пов’язані дії.

Діаграма з рисунку 2.2 ілюструє прецеденти використання для адміністратора (Admin) вебдодатку, ймовірно, електронної комерції або магазину спортивного харчування. Ось опис цих прецедентів:

- проведення оплати: адміністратор може здійснювати операції з оплатою замовлень від користувачів;
- створення замовлення: адміністратор має можливість створювати нові замовлення від імені користувачів або для внутрішніх цілей;
- внесення персональних даних: адміністратор може вносити та редагувати персональні дані користувачів або іншу інформацію в системі;
- створення нових товарів: цей прецедент дозволяє адміністратору додавати нові товари в каталог магазину;
- перегляд товарів: адміністратор може переглядати список усіх наявних товарів у каталозі;

- перегляд категорій: можливість переглядати та керувати категоріями товарів у каталозі;
- реєстрація: адміністратор має права на реєстрацію нових користувачів у системі;
- логін: адміністратор може авторизуватися в системі для отримання адміністративних прав доступу;
- отримання списку замовлень: адміністратор може переглядати список усіх активних або виконаних замовлень у системі;
- генерація чеку: прецедент, що дозволяє адміністратору генерувати чеки або квитанції для замовлень;
- завантажити чек: адміністратор може завантажувати згенеровані чеки у різних форматах;
- можливість залишити відгук: адміністратор має права на додавання відгуків або рецензій на товари від імені магазину;
- зміна статусу замовлення: адміністратор може змінювати поточний статус замовлень, наприклад, на «Виконано», «Відправлено» тощо;
- зміна набору користувачів: цей прецедент дозволяє адміністратору керувати доступом та правами користувачів у системі;
- редагування замовлення: адміністратор має можливість редагувати деталі існуючих замовлень у системі.

Таким чином, діаграма охоплює широкий спектр адміністративних функцій та прецедентів використання, необхідних для управління вебдодатком електронної комерції.

База даних є невід’ємною та критично важливою складовою такої системи. Вона виконує роль центрального сховища для збереження, організації та керування великим обсягом даних, що використовуються в системі.

Для створення бази даних було використано ORM Django. Завдяки моделі ORM (Object-Relational Mapping) ми можемо розглядати дані бази даних як об’єкти, що взаємодіють один з одним. ORM-система виконує мапінг

(відображення) між об’єктно-орієнтованою моделлю даних, яка використовується в програмі, і реляційною моделлю бази даних.

Завдяки використанню Django ORM всі моделі даних були описані у вигляді коду, використовуючи мову Python. Після цього Django автоматично згенерував необхідний SQL-код для створення таблиць, виконання запитів та здійснення зв’язків між таблицями [7].

Діаграма бази даних, зображена на рисунку 2.3, надає візуальне відображення структури всіх таблиць і полів, які використовуються для збереження інформації.

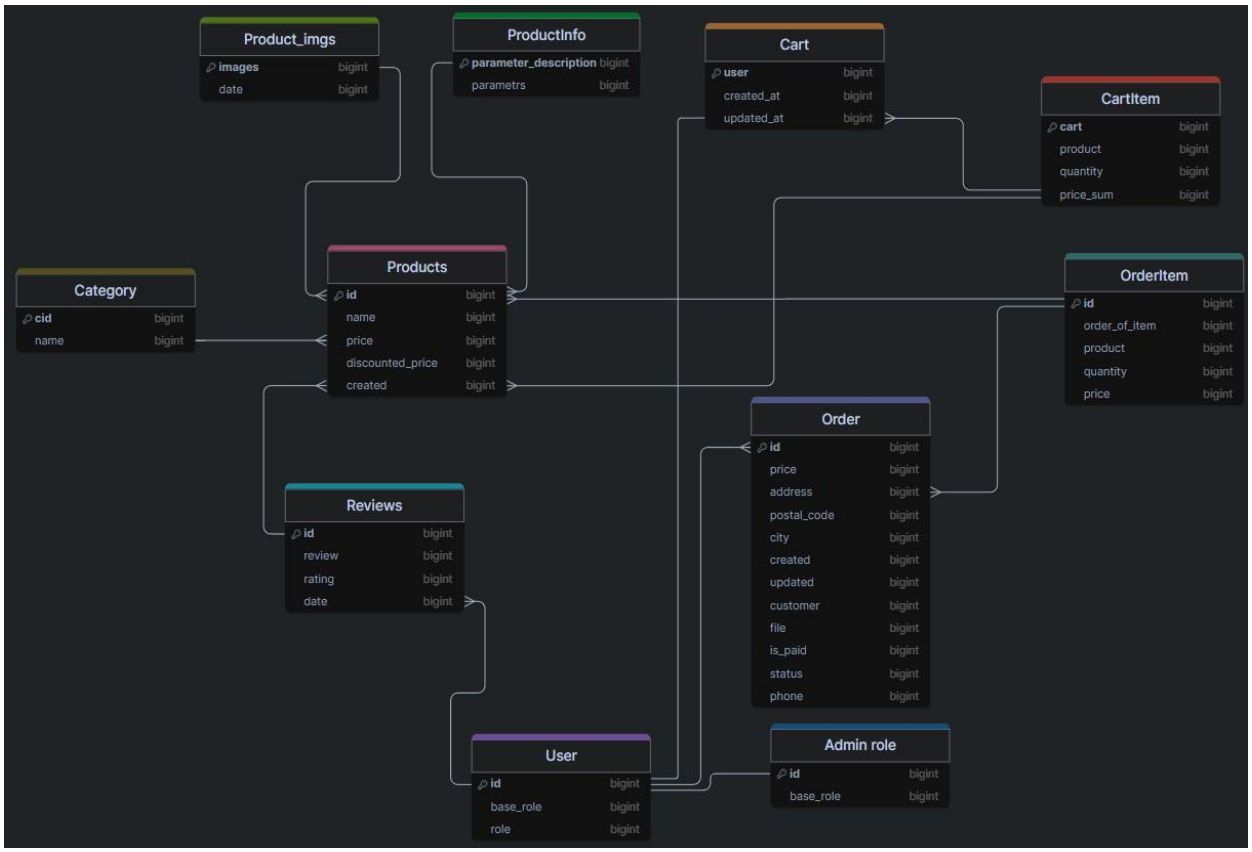

Рисунок 2.3 – Схема бази даних спортивного магазину

Кожна таблиця має унікальний первинний ключ, який ідентифікує кожен запис у таблиці. Представленні атрибути можна описати наступним чином.

Сутність **RATING**:

– це кортеж, що містить кортежі з цілими числами та відповідними їм

рядками, що представляють рейтинг від однієї до п’яти зірочок.

Сутність **Category**:

– cid: унікальний ідентифікатор категорії, представлений як ShortUUIDField з префіксом ‘cat’;
– name: назва категорії, що зберігається у текстовому полі CharField з максимальною довжиною 100 символів.

Сутність **Product**:

– category: зовнішній ключ (ForeignKey) на модель Category, що вказує на категорію, до якої належить продукт;
– name: назва продукту, що зберігається у текстовому полі CharField з максимальною довжиною 150 символів;
– image: зображення продукту, що зберігається у полі ImageField з шляхом завантаження ‘products/’;
– price: ціна продукту, що зберігається у полі DecimalField з максимальною кількістю цифр 10 та кількістю десяткових знаків 2;
– short_description: короткий опис продукту, що зберігається у текстовому полі TextField з максимальною довжиною 200 символів;
– description: більш детальний опис продукту, що зберігається у полі RichTextField;
– old_price: стара ціна продукту, що зберігається у полі DecimalField з максимальною кількістю цифр 10 та кількістю десяткових знаків 2;
– created: дата та час створення продукту, що зберігається у полі DateTimeField.

Сутність **ProductImages**:

– images: зображення продукту, що зберігається у полі ImageField з шляхом завантаження ‘product-images’;
– product: зовнішній ключ (ForeignKey) на модель Product, що пов’язує зображення з певним продуктом;
– date: дата та час завантаження зображення, що зберігається у полі DateTimeField.

Сутність **ProductInfo**:

– product: зовнішній ключ (ForeignKey) на модель Product, що пов’язує інформацію з певним продуктом;
– parametrs: параметри продукту, що зберігаються у текстовому полі CharField з максимальною довжиною 100 символів;
– parameter_description: опис параметрів продукту, що зберігається у текстовому полі CharField з максимальною довжиною 200 символів.

Сутність **ProductReview**:

– user: зовнішній ключ (ForeignKey) на модель User, що вказує на користувача, який залишив відгук;
– product: зовнішній ключ (ForeignKey) на модель Product, що вказує на продукт, для якого залишено відгук;
– review: текст відгуку, що зберігається у текстовому полі TextField;
– rating: рейтинг продукту, наданий користувачем, що зберігається у числовому полі IntegerField з вибором значень із кортежу RATING;
– date: дата та час залишення відгуку, що зберігається у полі DateTimeField.

Сутність **Cart**:

– user: зовнішній ключ (ForeignKey) на модель User, що пов’язує кошик з певним користувачем;
– created_at: дата та час створення кошика, що зберігається у полі DateTimeField;
– updated_at: дата та час останнього оновлення кошика, що зберігається у полі DateTimeField.

Сутність **CartItem**:

– cart: зовнішній ключ (ForeignKey) на модель Cart, що пов’язує елемент кошика з певним кошиком;
– product: зовнішній ключ (ForeignKey) на модель Product, що вказує на продукт, який додано до кошика;
– quantity: кількість одиниць продукту в кошику, що зберігається у полі

PositiveIntegerField;

– price_sum: загальна вартість одиниць продукту в кошику, що зберігається у полі PositiveIntegerField.

Сутність **OrderItem**:

– order_of_item: зовнішній ключ (ForeignKey) на модель Order, що пов’язує елемент замовлення з певним замовленням;
– product: зовнішній ключ (ForeignKey) на модель Product, що вказує на продукт, який був замовлений;
– quantity: кількість одиниць продукту в замовленні, що зберігається у полі PositiveIntegerField;
– price: ціна одиниці продукту в замовленні, що зберігається у полі DecimalField з максимальною кількістю цифр 10 та кількістю десяткових знаків 2.

Сутність **Order**:

– customer: зовнішній ключ (ForeignKey) на модель User, що вказує на користувача, який зробив замовлення;
– address: адреса доставки замовлення, що зберігається у текстовому полі CharField з максимальною довжиною 250 символів;
– postal_code: поштовий індекс для доставки замовлення, що зберігається у текстовому полі CharField з максимальною довжиною 20 символів;
– department_number: номер квартири або офісу для доставки замовлення, що зберігається у текстовому полі CharField з максимальною довжиною 100 символів;
– created: дата та час створення замовлення, що зберігається у полі DateTimeField;
– updated: дата та час останнього оновлення замовлення, що зберігається у полі DateTimeField;
– file: файл, пов’язаний із замовленням, що зберігається у полі FileField;
– is_paid: логічне поле BooleanField, що вказує, чи було оплачене

замовлення;

– status: статус замовлення, що зберігається у текстовому полі CharField з вибором значень із кортежу STATUS_CHOICES;
– total_price: загальна вартість замовлення, що зберігається у полі DecimalField з максимальною кількістю цифр 10 та кількістю десяткових знаків 2.

Сутність **User:**

– role: роль користувача, що зберігається у текстовому полі CharField з вибором значень із класу Role (яким є TextChoices). Допустимі значення – ‘USER’ та ‘ADMIN’;
– base_role: базова роль користувача, яка встановлюється як Role.USER за замовчуванням.

Сутність **AdminManagger:**

– це менеджер моделі (BaseUserManager), який використовується для отримання записів користувачів з роллю ‘ADMIN’.

Сутність **Admin:**

– це проксі-модель (proxy=True), що успадковується від моделі User;
– base_role для цієї моделі встановлено як User.Role.ADMIN;
– менеджером для цієї моделі є AdminManagger.

Загалом, ця структура моделей представляє систему електронної комерції. Модель Product зберігає інформацію про продукти, включаючи назву, зображення, ціну, опис тощо. Модель Category використовується для групування продуктів за категоріями. Користувачі можуть залишати відгуки про продукти за допомогою моделі ProductReview. Моделі Cart, CartItem та Order забезпечують функціональність кошика та оформлення замовлень. Модель User зберігає інформацію про користувачів, включаючи їхню роль (користувач чи адміністратор).

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРАКТИЧНОГО ЗАВДАННЯ

3.1 Програмна реалізація серверної частини

Програмна реалізація системи управління замовленнями в магазині техніки на базі Python backend з використанням фреймворку Django включає розробку моделей даних, роутингу URL, виглядів (views), шаблонів та додаткових функцій, що забезпечують функціональність додатку [5].

Нижче наведено фрагмент коду, що ілюструє реалізацію сутностей в базі даних (рис. 3.1).

```python
class OrderItem(models.Model):
    STATUS_CHOICES = (
        ("Undecided", "UNDECIDED"),
        ("Paid", "PAID"),
        ("Completed", "COMPLETED"),
    )

    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    price = models.DecimalField(max_digits=10, decimal_places=2, blank=True, null=True)
    address = models.CharField(max_length=250, default="")
    postal_code = models.CharField(max_length=20, default="")
    city = models.CharField(max_length=100, default="")
    created = models.DateTimeField(
        auto_now_add=False, blank=True, null=True, default=timezone.now
    )
    updated = models.DateTimeField(
        auto_now_add=False, blank=True, null=True, default=timezone.now
    )
    customer = models.ForeignKey(User, null=True, blank=True, on_delete=models.CASCADE)
    file = models.FileField(default="", null=True, blank=True)
    is_paid = models.BooleanField(default=False, null=True, blank=True)

    status = models.CharField(
        max_length=10, choices=STATUS_CHOICES, default="Undecided"
    )



    def __str__(self):
        return "{}".format(self.id)

    def save(self, *args, **kwargs):
        if self.product.discounted_price:
            self.price = self.product.discounted_price
        else:
            self.price = self.product.price
        return super().save(*args, **kwargs)

    def paid(self):
        self.is_paid = True
        self.save()

    def mark_as_paid(self):
        self.status = "Paid"
        self.save()

```

Рисунок 3.1 – Моделі даних (models.py)

Наведений код (рис. 3.2) відповідає за налаштування маршрутів URL у Django додатку з використанням модуля django.urls. В ньому також імпортується модуль views, який містить відповідні функції-вигляди (views) для обробки різних URL шляхів. Для автентифікації та авторизації користувачів використовуються класи TokenObtainPairView та TokenRefreshView із модуля rest_framework_simplejwt.views [4].

```python
from django.urls import path
from . import views

from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)


urlpatterns = [
    path("routes/", views.RoutesAPIView.as_view(), name="routes"),
    path("products/", views.ProductsAPIView.as_view(), name="products"),
    path("products/<int:pk>/", views.ProductAPIView.as_view(), name="product"),
    path(
        "products/<int:pk>/order/create/",
        views.CreateOrderAPIView.as_view(),
        name="create_order",
    ),
    path("orders/", views.OrdersAPIView.as_view(), name="orders"),
    path("orders/<int:pk>/", views.OrderAPIView.as_view(), name="order"),
    path(
        "orders/<int:pk>/payment/",
        views.OrderPaymentAPIView.as_view(),
        name="order_payment",
    ),
    path(
        "orders/<int:pk>/gen-bill/",
        views.OrderGenBillAPIView.as_view(),
        name="order_gen_bill",
    ),
    path(
        "orders/<int:pk>/change-status/",
        views.ChangeOrderStatusAPIView.as_view(),
        name="change_order_status",
    ),
    path("users/token/", TokenObtainPairView.as_view(), name="token_obtain_pair"),
    path("users/token/refresh/", TokenRefreshView.as_view(), name="token_refresh"),
]
```

Рисунок 3.2 – Роутинг URL (urls.py)

Основна частина коду – список urlpatterns, який містить шляхи (URL patterns) та пов’язані з ними вигляди (див. рис. 3.3).

```python
class ProductsAPIView(APIView):
    @extend_schema(
        description="""
                Get all products.
                Returns a list of all products available in the system.
            """,
        responses={200: ProductSerializer(many=True)},
        tags=["Products"],
    )
    def get(self, request):
        """
        Get all products.
        """
        products = Product.objects.all()
        serializer = ProductSerializer(products, many=True)
        return Response(serializer.data)


class ProductAPIView(APIView):
    @extend_schema(
        description="""
            Get a product by ID.

            Returns the product details for the specified ID.

            Parameters:
                - `pk` (int): The ID of the product.

            Example response:
            {
                "id": 1,
                "name": "Product 1",
                "price": 10.99,
                ...
            }
        """,
        responses={200: ProductSerializer()},
        tags=["Products"],
    )
    def get(self, request, pk):
        """
        Get a product by ID.
        Returns the product details for the specified ID.
        """
        product = get_object_or_404(Product, id=pk)
        serializer = ProductSerializer(product)
        return Response(serializer.data)
```

Рисунок 3.3 – Вигляди (views.py)

Наведений код (рис. 3.4) містить декілька класів-виглядів (views) для обробки різних URL шляхів в Django додатку.

Ці фрагменти коду демонструють створення моделей даних для продуктів та замовлень, роутинг URL для перегляду списку продуктів та замовлень, вигляди для API запитів на деталі та продукти, а також шаблон для відображення списку продуктів.

```
<section>
    {% if products %}
    <h1>It's a list of available products. Hope you'll find what you're looking for.</h1>
    <div class="product-list">
        {% for product in products %}
        <div class="product-item">
            <h2>{{ product.name }}</h2>
            <ul>
                <li>
                    {% if product.discounted_price %}
                    <span class="original-price">Price: {{ product.price }}</span>
                    <span class="discounted-price">{{ product.discounted_price }}</span>
                    {% else %}
                    <span class="price">Price: {{ product.price }}</span>
                    {% endif %}
                </li>
                <li>Added: {{ product.created }}</li>
            </ul>
            {% if request.user.role == 'USER' %}
            <a href="{% url 'user-order' pk=product.id %}" class="buy-button">Buy</a>
            {% endif %}
        </div>
        {% endfor %}

    </div>
    {% else %}
        <h1>No products yet...</h1>
    {% endif %}
</section>
```

Рисунок 3.4 – Шаблони (product_list.html)

Таким чином, програмна реалізація системи управління замовленнями в магазині техніки на базі Python backend з використанням фреймворку Django включає в себе створення моделей даних, налаштування роутингу URL, реалізацію виглядів та шаблонів. Всі ці компоненти співпрацюють, щоб забезпечити функціональність системи управління замовленнями та надати користувачам зручний інтерфейс для взаємодії з додатком.

3.2 Результати роботи та тестування системи

Додаток реалізований у вигляді REST API з використанням Django та Django REST Framework. Має весь функціонал, який необхідний для системи управління замовленнями в магазині техніки. Таким чином користувачі були розділенні на ролі, для кожної сторінка з замовленнями буде мати свій вигляд,

також запити на сервер через API будуть повертати різний вміст, виходячи з того, до якої ролі належить користувач [6].

Файлова структура відображає типову організацію проєкту Django, де кожен додаток або компонент розміщується у своїй власній директорії, а спільні файли, такі як файл налаштувань, знаходяться в кореневій директорії проєкту (рис. 3.5).

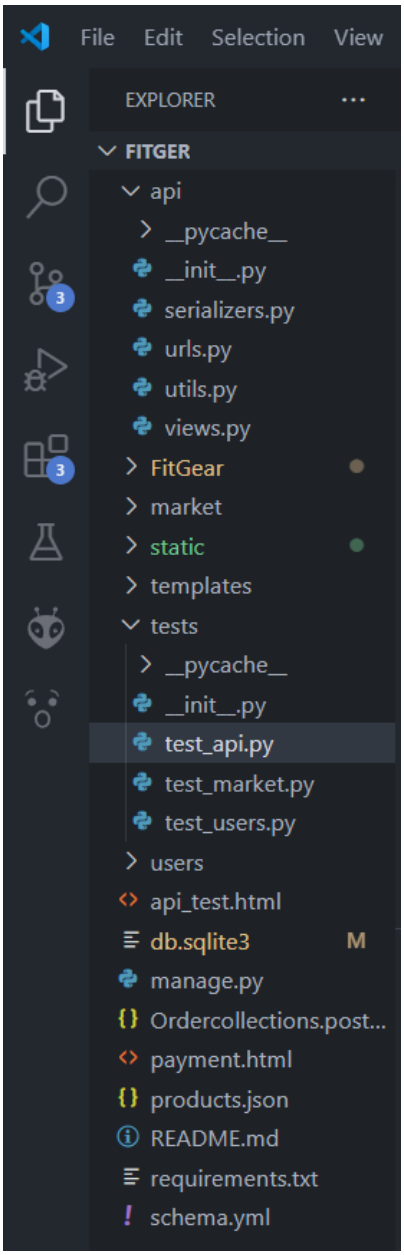

Рисунок 3.5 – Файлова структура проєкту

Api: Ця директорія, код для роботи з API (Application Programming Interface) вашого додатку Django.

Serializers.py: Цей файл, містить серіалізатори Django REST Framework, які використовуються для перетворення екземплярів моделей даних у формати JSON, XML або інші формати, придатні для передачі через API.

Urls.py: Цей файл визначає URL-маршрути для додатку Django. Він містить патерни URL-адрес, які відображаються на відповідні функції або класи представлень.

Views.py: У цьому файлі, як правило, знаходяться функції або класи представлень Django, які обробляють HTTP-запити та повертають HTTP-відповіді.

Db.sqlite3: Це файл із вбудованою базою даних SQLite, який використовується Django за замовчуванням для зберігання даних під час розробки. У виробничому середовищі часто використовуються більш потужні реляційні бази даних, такі як PostgreSQL або MySQL.

Manage.py: Цей файл є інтерфейсом командного рядка для взаємодії з проєктом Django. Він дозволяє виконувати різні завдання, такі як запуск сервера розробки, міграція баз даних, створення суперкористувача та інші.

Market, app, templates та інші директорії: Ці директорії, ймовірно, є різними додатками або компонентами вашого проєкту Django. Вони можуть містити власні моделі, представлення, шаблони та інші файли, пов’язані з їх функціональністю.

Requirements.txt: Цей файл містить список залежностей Python, необхідних для запуску проєкту Django. Його можна використовувати для відтворення середовища розробки або розгортання проєкту на іншому сервері.

Products.json, payment.html та інші файли: Ці файли, ймовірно, є різними компонентами вашого проєкту, наприклад, файлами шаблонів HTML, файлами з даними JSON або іншими ресурсами.

Для авторизації користувачів, використовуються певні механізми токенів. Давайте розглянемо функціонал кожного з них.

LoginUserAPIView: цей клас відповідає за авторизацію користувачів (див. рис. 3.6). При отриманні POST запиту з ім’ям користувача та паролем,

спочатку проводиться спроба автентифікації через Django. Якщо ця автентифікація пройшла успішно, для користувача генеруються токени доступу та оновлення за допомогою бібліотеки **django-rest-framework-simplejwt**, після чого ці токени повертаються у відповіді API.

```python
class LoginUserAPIView(APIView):
    @extend_schema(
        operation_id="login_user",
        description="Authenticate a user and obtain access and refresh tokens.",
        request={
            "application/json": {
                "type": "object",
                "properties": {
                    "username": {"type": "string"},
                    "password": {"type": "string"}
                },
                "required": ["username", "password"]
            }
        },
        responses={
            200: {
                "description": "Success",
                "examples": {
                    "Success Example": {
                        "value": {
                            "refresh": "...",
                            "access": "..."
                        },
                        "name": "Success Example"
                    }
                }
            },
            401: {
                "description": "Invalid credentials",
                "examples": {
                    "Invalid Credentials Example": {
                        "value": {
                            "error": "Invalid credentials"
                        },
                        "name": "Invalid Credentials Example"
                    }
                }
            }
        },
    )
    def post(self, request):
        username = request.data.get("username", "").lower()
        password = request.data.get("password", "")
        user = authenticate(request, username=username, password=password)
        if user is not None:
            refresh = RefreshToken.for_user(user)
            return Response({'refresh': str(refresh), 'access': str(refresh.access_token)})
        else:
            return Response({"error": "Invalid credentials"}, status=status.HTTP_401_UNAUTHORIZED)
```

Рисунок 3.6 – LoginUserAPIView для логінізації користувачів

RegisterUserAPIView: цей клас відповідає за реєстрацію нових користувачів (див. рис. 3.7). При отриманні POST запиту з даними нового

користувача (ім’я, електронна пошта, пароль), спочатку дані перевіряються на валідність. Після успішної перевірки створюється новий користувач у базі даних, та для нього генеруються токени доступу та оновлення, які також повертаються у відповіді API.

```python
class RegisterUserAPIView(APIView):
    @extend_schema(
        operation_id="register_user",
        description="Register a new user and obtain access and refresh tokens.",
        request={
            "application/json": {
                "type": "object",
                "properties": {
                    "username": {"type": "string"},
                    "email": {"type": "string", "format": "email"},
                    "password": {"type": "string"},
                    "password2": {"type": "string"}
                },
                "required": ["username", "email", "password", "password2"]
            }
        },
        responses={
            201: {
                "description": "User created",
                "examples": {
                    "User Created Example": {
                        "value": {
                            "refresh": "...",
                            "access": "..."
                        },
                        "name": "User Created Example"
                    }
                }
            },
            400: {
                "description": "Bad request",
                "examples": {
                    "Bad Request Example": {
                        "value": {
                            "error": "...",
                            "details": "..."
                        },
                        "name": "Bad Request Example"
                    }
                }
            },
        },
    )
    def post(self, request):
        form = CustomUserCreationForm(request.data)
        if form.is_valid():
            user = form.save(commit=False)
            user.username = user.username.lower()
            user.save()
            refresh = RefreshToken.for_user(user)
            return Response({'refresh': str(refresh), 'access': str(refresh.access_token)}, status=status.HTTP_201_CREATED)
        else:
            return Response({"error": "An error occurred during registration.", "details": form.errors}, status=status.HTTP_400_BAD_REQUEST)
```

Рисунок 3.7 – RegisterUserAPIView для логінізації користувачів

LogoutUserAPIView: цей клас відповідає за вихід користувачів з системи (див. рис. 3.8). Для виклику цього API потрібна автентифікація користувача, яка забезпечується за допомогою дозволу **IsAuthenticated**. При отриманні POST запиту, виконується вихід користувача через Django, що

призводить до скасування його токенів доступу та оновлення. Після успішного виходу користувача повертається повідомлення про це.

```python
class LogoutUserAPIView(APIView):
    permission_classes = [IsAuthenticated]

    @extend_schema(
        operation_id="logout_user",
        description="Logout a user and invalidate access and refresh tokens.",
        responses={
            200: {
                "description": "User logged out",
                "examples": {
                    "User Logged Out Example": {
                        "value": {
                            "message": "User was logged out."
                        },
                        "name": "User Logged Out Example"
                    }
                }
            },
        },
    )
    def post(self, request):
        logout(request)
        return Response({"message": "User was logged out."}, status=status.HTTP_200_OK)
```

Рисунок 3.8 – LogoutUserAPIView для логінізації користувачів

API для отримання списку замовлень, які фільтруються на основі ролі користувача. Якщо користувач є адміністратором, він отримує всі замовлення, а якщо користувач має іншу роль (наприклад, звичайний користувач), він отримує лише замовлення, які він зробив (див. рис. 3.9).

Використання декоратора @extend_schema дозволяє автоматично генерувати документацію для цього API на основі наданих метаданих. Це полегшує розуміння функціональності API та його використання іншими розробниками або клієнтами.

Загалом, цей код демонструє типовий приклад реалізації API для роботи із замовленнями в Django REST Framework, де використовуються серіалізатори для перетворення даних в JSON, фільтрація даних на основі ролі користувача та автентифікація для забезпечення безпеки.

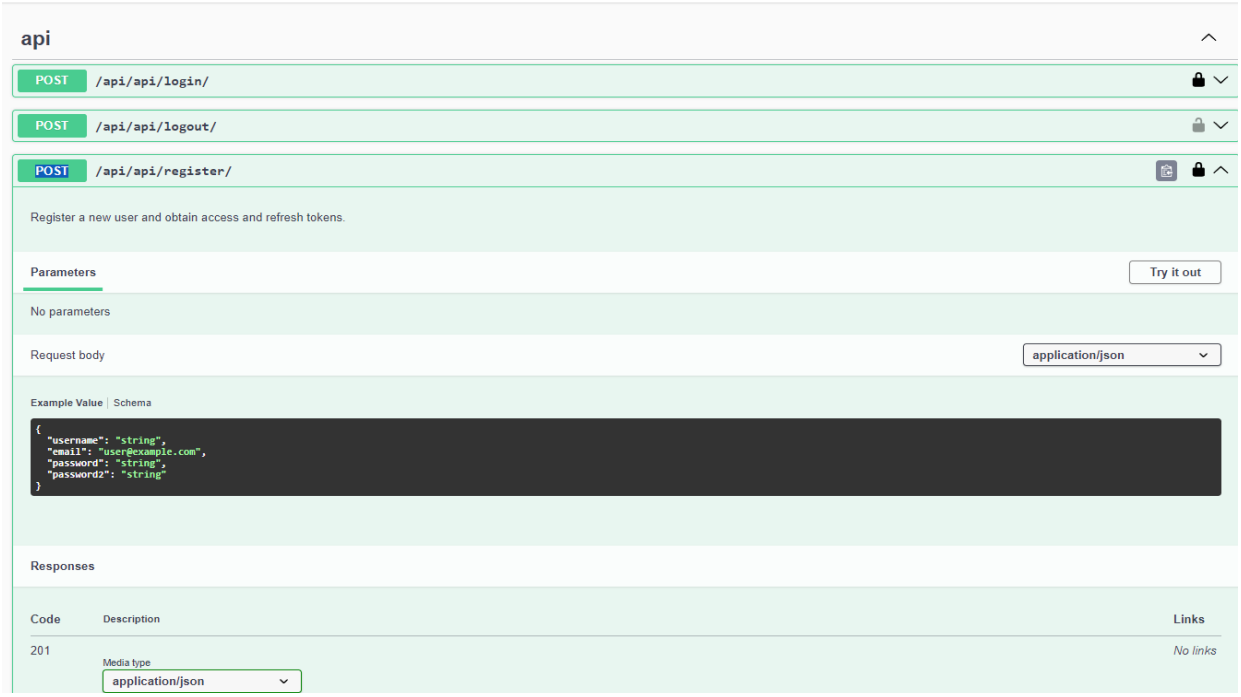

Рисунок 3.9 – Відображення документацію для API-ендпоінту, пов’язаного з реєстрацією користувача

Другим і головним компонентом додатку є його модулі, які відповідають за різні частини функціоналу, такі як, завантаження файлів, створення замовлення, виконання операцій з відповідною моделлю у базі даних і так далі. Ці модулі забезпечують реалізацію конкретних функцій та дозволяють користувачам взаємодіяти з різними частинами додатку.

Вони інкапсулюють логіку та операції, пов’язані з відповідними функціями, та надають необхідні інтерфейси для взаємодії. Особливо в системі присутні модулі, що керують доменами, тобто обробляють маршрути, спрямовані на конкретний ресурс. Такі модулі визначають, які операції можна виконати з ресурсом, які дані можна отримати або надіслати, і які дії потрібно виконати при отриманні відповідного запиту. Вони забезпечують структуровану та організовану архітектуру додатку, що дозволяє легко розширювати та модифікувати його функціонал.

Після отримання відфільтрованого списку замовлень (рис. 3.10), код створює серіалізатор OrderSerializer для перетворення екземплярів об’єктів замовлень у формат JSON (serializer = OrderSerializer(orders, many=True)).

Параметр many=True вказує на те, що серіалізатор повинен обробляти множину об’єктів замовлень.

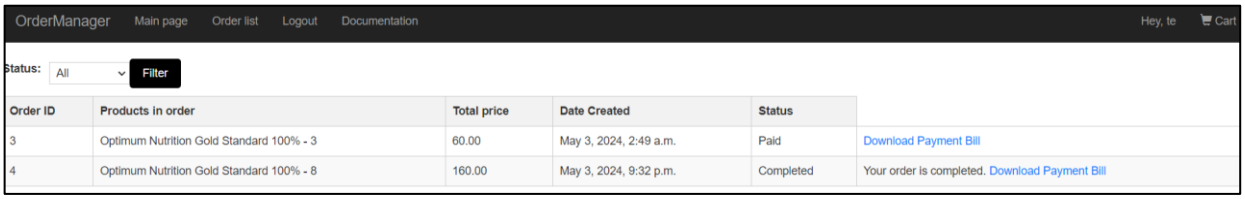

Рисунок 3.10 – Вигляд списку замовлень для звичайного клієнта

Нарешті, представлення повертає HTTP-відповідь з даними замовлень у форматі JSON, згенерованими серіалізатором (return Response(serializer.data)).

Для ролі користувача адміністратор характерною є можливість переглянути всі замовлення, їхні статуси, дату, знижку тощо (рис. 3.11). Адміністратор вказує проміжок дат, за якими необхідно вивести дані про замовлення.

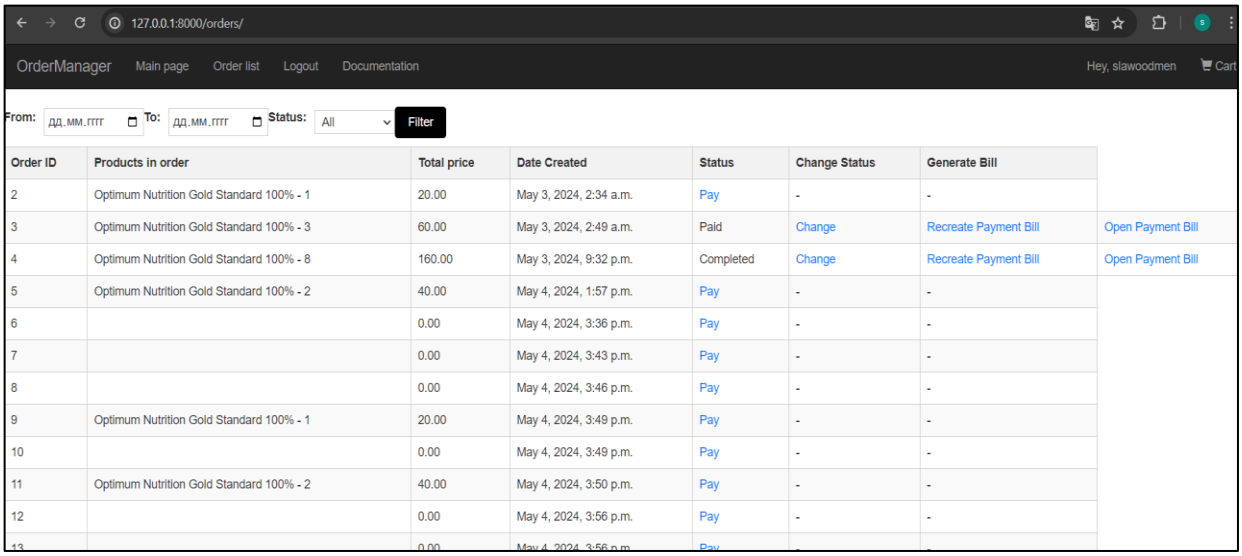

Рисунок 3.11 – Відображення списку замовлень для «Адміністратора»

Для цього запиту використовується метод GET і URL /api/products. На зображенні показана структура запиту в Postman, де вказано метод, URL і параметри запиту, які в даному випадку відсутні.

Додатково, нижче наведений приклад API-запиту для отримання деталей продукту за його ідентифікатором:

Метод: GET

URL: /api/products/{id}

У цьому запиті необхідно замінити {id} на ідентифікатор конкретного продукту, наприклад /api/products/1 для отримання деталей продукту з ідентифікатором 1.

Ці приклади (рис. 3.12, 3.13) демонструють використання API-запитів для отримання інформації про продукти з системи замовлень [3].

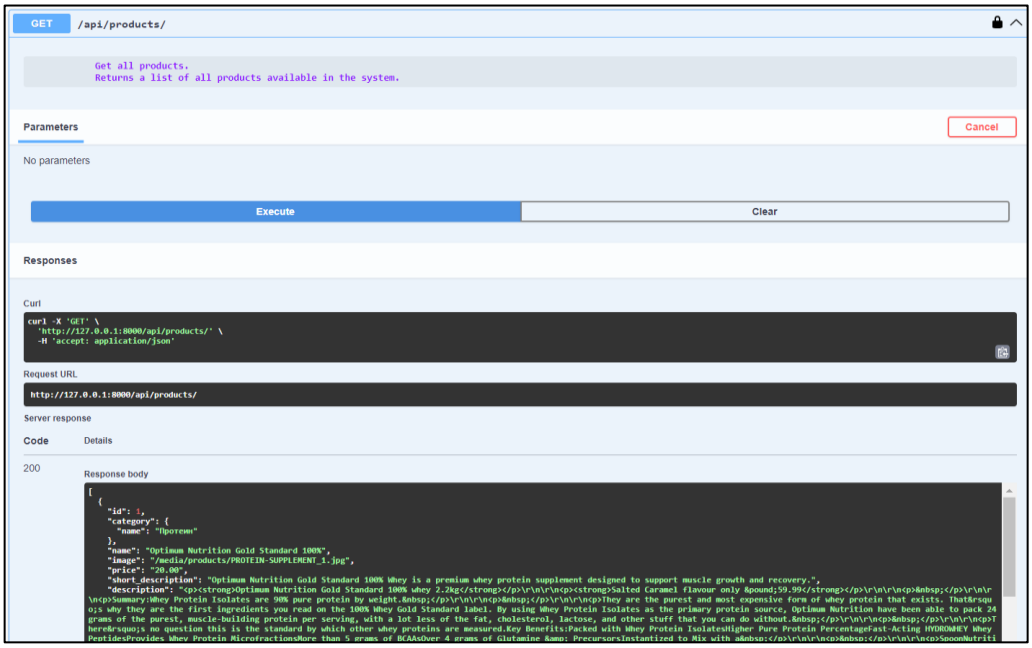

Рисунок 3.12 – Приклад API запиту для отримання всіх продуктів

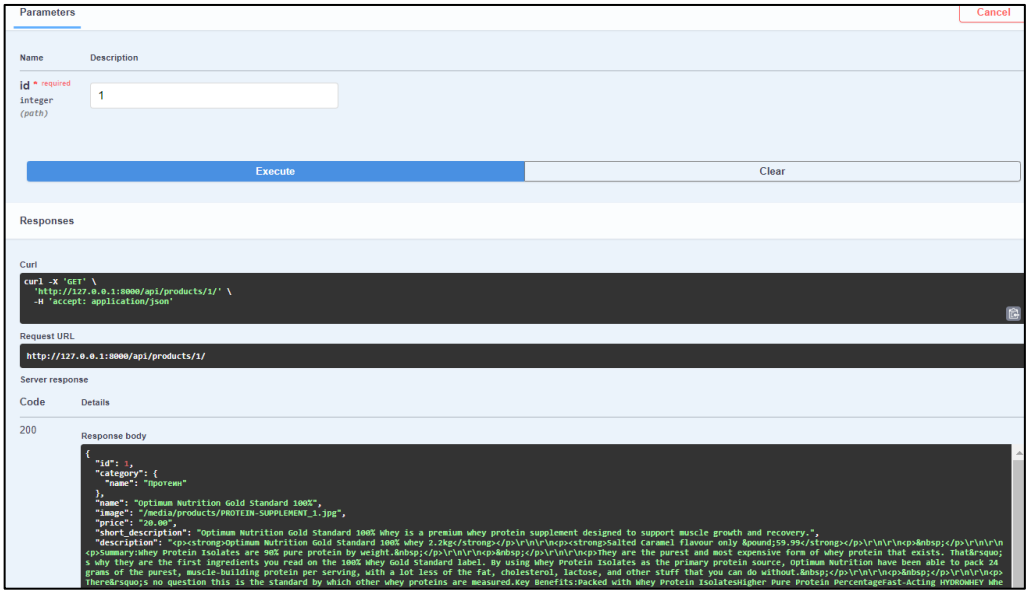

Рисунок 3.13 – Приклад API запиту для отримання даних певного продукту

Юніт-тести є важливою складовою процесу розробки програмного забезпечення. Вони дозволяють перевірити окремі модулі або функції коду на коректність роботи, тим самим забезпечуючи високу якість програмного продукту. Використання юніт-тестів допомагає виявляти та запобігати багам на ранніх стадіях розробки, що значно полегшує процес відлагодження та підтримки коду.

Наведений код (рис. 3.14) є прикладом юніт-тесту для перевірки функціональності API для роботи з замовленнями в Django REST Framework.

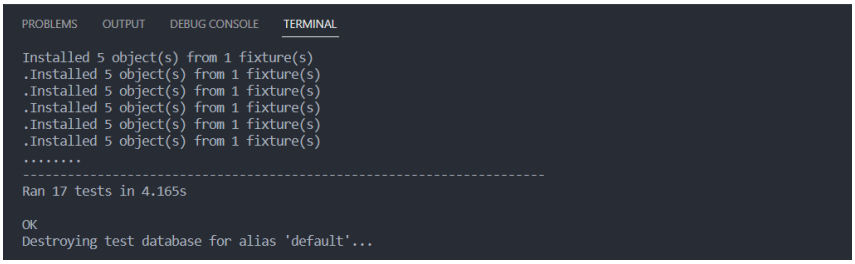

Рисунок 3.14 – Виклик реалізованних юніт етстів

Розглянемо його детальніше.

Class OrderAPIViewTest(TestCase): це клас, що успадковується від TestCase з Django. Він містить набір тестів для перевірки API для замовлень.

Def setUp(self): це метод встановлення, який виконується перед кожним тестом. Тут відбувається налаштування тестового середовища, створення тестового користувача (User.objects.create_user), автентифікація клієнта (client.force_authenticate), створення тестового продукту (Product.objects.create) та тестового елементу замовлення (OrderItem.objects.create).

Def test_get_order(self): це тест-метод, який перевіряє функціональність отримання деталей замовлення за його ідентифікатором:

– order_item = OrderItem.objects.first(): отримуємо перший елемент замовлення з бази даних;
– url = reverse("order", args=[order_item.id]): генеруємо URL-адресу для отримання деталей замовлення, використовуючи ідентифікатор

елементу замовлення;

– response = self.client.get(url): виконуємо GET-запит на згенеровану URL-адресу за допомогою тестового клієнта;
– self.assertEqual(response.status_code, status.HTTP_200_OK): перевіряємо, чи відповідь має HTTP-статус 200 OK, що означає успішне виконання запиту;
– self.assertEqual(response.data["status"], "Undecided"): перевіряємо, чи статус замовлення в отриманих даних відповіді дорівнює "Undecided", що було встановлено під час створення елементу замовлення.

Юніт-тест забезпечує автоматизовану перевірку функціональності отримання деталей замовлення через API (рис. 3.15). При внесенні змін у код, пов’язаний з цією функціональністю, тест допоможе виявити потенційні помилки або порушення очікуваної поведінки. Успішне проходження тесту гарантує, що функціональність отримання замовлення працює коректно відповідно до очікувань.

```python
class OrderAPIViewTest(TestCase):
    def setUp(self):
        self.client = APIClient()
        self.user = User.objects.create_user(
            username="testuser", password="testpassword"
        )
        self.client.force_authenticate(user=self.user)

        # Create a product object
        product = Product.objects.create(name="Product", price=9.99)

        # Create an order item with a valid product
        order = OrderItem.objects.create(
            customer=self.user, product=product, status="Undecided"
        )

    def test_get_order(self):
        order_item = OrderItem.objects.first()
        url = reverse("order", args=[order_item.id])
        response = self.client.get(url)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(response.data["status"], "Undecided")
```

Рисунок 3.15 – Код юніт-тесту

Розроблені юніт-тести в проєкті використовуються для перевірки коректності роботи окремих функцій, методів або класів програмного коду. Вони дозволяють виявляти помилки та проблеми на ранніх етапах розробки, забезпечуючи більшу надійність та стабільність системи.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведено дослідження літературних джерел за темою створення вебдодатків для спортивного інтернет-магазину та проаналізовано існуючі продукти-аналоги. Це надало глибоке розуміння теоретичних аспектів та практичних реалій розробки таких систем.

Аналіз доступних інформаційних матеріалів та результатів наукових досліджень підтвердив, що розробники та дослідники активно вивчають IT-технології, спрямовані на поліпшення процесів електронної комерції в спортивній сфері. Тому метою розробки став створення вебдодатку для спортивного інтернет-магазину з акцентом на зручну навігацію, швидке оформлення замовлень та ефективне управління товарами.

Після проведення аналізу поточного стану проблеми, було складено технічне завдання та розроблено план робіт. Перед початком реалізації було проведено проєктування архітектури серверної частини системи, бази даних та визначено варіанти використання. Розроблена система повністю відповідає всім встановленим функціональним вимогам.

Реалізована серверна частина, яка забезпечує надійну та ефективну роботу системи спортивного інтернет-магазину. Вона включає в себе розроблені модулі, сервіси та компоненти, що взаємодіють між собою з метою швидкої обробки даних та виконання запитів користувачів.

Було розроблено інтерфейс адміністратора та інтерфейси користувача, що дозволяють зручно та інтуїтивно взаємодіяти з системою, переглядати та замовляти товари, керувати особистими даними та замовленнями.

ПЕРЕЛІК ПОСИЛАНЬ

1. Краус К. М., Краус Н. М., Манжура О. В. Електронна комерція та Інтернет-торгівля : навч.-метод. посібник. Київ : «Аграр Медіа Груп», 2021. 454 с.
2. Смолій Л., Костюк В. Новітні тренди та перспективи розвитку електронної комерції в міжнародному бізнесі. *Економіка та суспільство*. 2021. Вип. 29. URL: https://doi.org/10.32782/2524-0072/2021-29-43 (дата звернення: 30.04.2024).
3. Django official site. URL: https://www.djangoproject.com (дата звернення: 05.04.2024).
4. Django ORM. URL: https://tutorial.djangogirls.org/uk/django_orm/ (дата звернення: 15.04.2024).
5. Lavin M., Solórzano J. Lightweight Django: Using REST, WebSockets, and Backbone. O'Reilly Media 2014. 246 p.
6. Shaw B., Badhwar S. Web Development with Django. 2021. 826 p.
7. Модель «сутність – зв’язок». Основні поняття моделі «сутність – зв’язок»: сутності, зв’язки, атрибути та їх класифікація. URL: https://uk.wikipedia.org/wiki/Модель_«сутність_—_зв%27язок» (дата звернення 12.04.2024).