Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри

_____Олена ОЛЬХОВСЬКА

(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«**РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ ОДЯГУ З ВИКОРИСТАННЯМ DJANGO**»

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр

Виконавець роботи Цілуйко Ілля Ярославович

_____«__»_____202_р.

(підпис)

Науковий керівник к.ф.-м.н.,доц.,_____

_____«__»_____202_р.

(підпис)

Рецензент

ПОЛТАВА 2025

ЗАТВЕРДЖУЮ

Завідувач кафедри _____Олена ОЛЬХОВСЬКА
«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФАКАЦІЙНОЇ РОБОТИ

на тему «Розробка інтернет магазину одягу з використанням Django»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Цілуйко Ілля Ярославович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___» _____ 20__ р.
Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні відеоматеріали, технічна документація.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Вибір інструментарію
2.2. Планування табличок бази даних
2.3. Планування структури сайту
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Теоретичний опис інструментарію
3.2. Переваги та недоліки обраного інструментарію
3.3. Опис задач та алгоритмів вирішення
4. ПРАКТИЧНА ЧАСТИНА
4.1. Опис розробки сайту
4.2. Тестування та налагодження
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
Перелік графічного матеріалу: Необхідні ілюстрації.
Консультанти розділів кваліфікаційної роботи

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Черненко О. О.		
Інформаційний огляд	Черненко О. О.		
Теоретична частина	Черненко О. О.		
Практична частина	Черненко О. О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.
Здобувач вищої освіти Цілуйко Ілля Ярославович
Науковий керівник к. ф.-м. н., доц. Оксана ЧЕРНЕНКО

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на

_____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____

(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.

Погоджено
Науковий керівник _____
к. ф.-м. н., доц. Оксана ЧЕРНЕНКО
«____» _____ 202_ р.

План

кваліфікаційної роботи на тему
«Розробка інтернет магазину одягу з використанням Django»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
ступеня бакалавра
Цілуйко Ілля Ярославович
Прізвище, ім'я, по батькові

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Вибір інструментарію
2.2. Планування табличок бази даних
2.3. Планування структури сайту
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Теоретичний опис інструментарію
3.2. Переваги та недоліки обраного інструментарію
3.3. Опис задач та алгоритмів вирішення
4. ПРАКТИЧНА ЧАСТИНА
4.1. Опис розробки сайту
4.2. Тестування та налагодження
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Здобувач вищої освіти _____ Ілля ЦІЛУЙКО
«____» _____ 202_ р.

АНОТАЦІЯ

Записка: 61 сторінка, в тому числі з яких: основна частина – 61 сторінка, літературних джерел – 20 назв, рисунків – 38.

Мета роботи – розробка інтернет-магазину з використанням фреймворку Django, що включає функціонал реєстрації та автентифікації користувачів, обробки замовлень, інтеграції платіжної системи Stripe, підтримки пошуку товарів через Elasticsearch та реалізації асинхронних задач за допомогою Celery, і кешування через Redis.

Об'єкт розробки – веб-сайт для електронної комерції – вебдодаток інтернет-магазину.

Методи дослідження та інформаційне забезпечення – аналіз існуючих технологій веброзробки, порівняння фреймворків і сервісів, моделювання архітектури проєкту, алгоритмізація бізнес-логіки, тестування функціоналу. Технічна документація до Django, Redis, Celery, Stripe API, Elasticsearch, а також інтернет-ресурси, офіційна документація і приклади реалізацій.

Результати дослідження. Розроблено прототип інтернет-магазину з функціоналом: реєстрації/авторизації користувачів, відправки email-повідомлень при реєстрації та відновленні пароля, інтеграції Stripe для обробки платежів, реалізації корзини та оформлення замовлень, пошуку товарів через Elasticsearch, виконання асинхронних задач через Celery, кешування Redis.

Рекомендації щодо використання результатів дослідження. Результати можуть бути використані: як основа для створення повноцінного комерційного вебсайту, при навчанні розробці вебдодатків з використанням Django та суміжних технологій, для впровадження подібних архітектурних рішень у реальних e-commerce проєктах.

Ключові слова: Інтернет-магазин, Django, Stripe, Redis, Celery, Elasticsearch, реєстрація користувачів, обробка замовлень, веброзробка, асинхронні задачі.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ 7
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД 9
2.1. Вибір інструментарію 9
2.2. Планування табличок бази даних 11
2.3. Планування структури сайту 13
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА 22
3.1. Теоретичний опис інструментарію 22
3.2. Переваги та недоліки обраного інструментарію 31
3.3. Опис задач та алгоритмів вирішення 34
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА 38
4.1. Опис розробки сайту 38
4.2. Тестування та налагодження 55
ВИСНОВКИ 57
СПИСОК ЛІТЕРАТУРИ 58

ВСТУП

Сучасний розвиток інформаційних технологій та інтернет-комерції вимагає від розробників створення високопродуктивних, масштабованих і надійних веб-додатків. Для задоволення цих потреб використовуються різноманітні інструменти та технології, які дозволяють оптимізувати процеси реєстрації користувачів, обробки платежів, пошуку товарів та забезпечення стійкості до високих навантажень.

Мета даного проєкту полягає у розробці інтернет-магазину, який підтримує реєстрацію користувачів, обробку замовлень, платіжну систему Stripe, функціонал для управління корзиною покупок, а також реалізацію розширеного пошуку товарів за допомогою Elasticsearch. Для забезпечення асинхронних завдань, таких як відправка електронних листів при реєстрації чи відновленні паролів, використовується Redis у поєднанні з Celery.

Об'єктом розробки є система для електронної комерції. Предмет розробки - реалізація архітектури інтернет-магазину з інтеграцією сервісів Redis, Celery, Elasticsearch і Stripe.

Проєкт складається з кількох розділів, де детально розглянуто архітектуру додатка, інструменти та технології, а також алгоритми взаємодії між сервісами.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

У рамках даної дипломної роботи необхідно розробити вебдодаток для електронної комерції, який забезпечуватиме наступні функціональні можливості:

1. Реєстрація та автентифікація користувачів: Реалізувати механізм реєстрації, входу та виходу користувачів, а також відновлення пароля через електронну пошту. Для цього потрібно налаштувати систему відправки листів із підтвердженням реєстрації та відновленням доступу.
2. Інтеграція платіжної системи Stripe: Забезпечити можливість оплати товарів через систему Stripe з підтримкою різних платіжних методів.
3. Корзина покупок та управління замовленнями: Користувачі повинні мати можливість додавати товари до корзини, переглядати її вміст і оформлювати замовлення з подальшою оплатою.
4. Реалізація асинхронних задач: Використати Redis у поєднанні з Celery для виконання таких асинхронних задач, як відправка електронних листів при реєстрації, оновленні паролів та обробці замовлень.
5. Пошук товарів за допомогою Elasticsearch: Інтегрувати систему пошуку товарів, яка дозволяє виконувати швидкий і точний пошук з використанням Elasticsearch.
6. Масштабованість і оптимізація: Забезпечити можливість масштабування додатка та оптимізації його роботи при збільшенні кількості користувачів і замовлень, використовуючи Redis для кешування та оптимізації бази даних.

Завданням дипломної роботи є алгоритмізація та програмування основних компонентів вебдодатка з використанням Django та інтеграція додаткових інструментів для підвищення продуктивності та забезпечення сучасного користувацького досвіду.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Вибір інструментарію

Використовується такий вибір інструментарію для проєкту, так як це важливий крок для розуміння завдань і цілей на майбутнє для цього проєкту. Спочатку визначається мова програмування, використовується - Python, так як мій проєкт це вебсайт, а Python гарно підходить під веб-розробку. Також гарними виборами були б Java, PHP, Javascript(це трішки буде використовуватися), так як ці мови програмування також мають дуже сильні фреймворки для веб-розробки такі як - Laravel (PHP), Spring (Java).

Далі фреймворк Django через його великий інструментарій, через дуже гарні вбудовані засоби для адміністрування, ORM для баз даних. Також гарними виборами були б такі інструменти як Flask, FastAPI, якщо з Flask все зрозуміло, у нього в принципі менше компонентів для веброзробки. То з FastAPI все не так однозначно, в Django є можливість працювати з API і це також величезний плюс цього фреймворку, але з цим компонентом в проєкті праці не буду, по причині - особисто для мене проще написати візуальну частину за допомогою Django, Jinja, бо я один розробник який буде працювати з проєктом, але на більшості реальних проєктів використовують саме API для взаємодії між візуальною частиною та серверною частиною сайту [3, 4].

API - дає змогу Back-end розробнику надавати серверні данні великій кількості потрібних додатків таких як - візуальна частина сайту, мобільні додатки, взаємодія між різними додатками і т.д. Якщо почати писати це за допомогою API це затягнеться на довго через малий досвід роботи з front-end фреймворками типу React, Angular. Ну і на реальних проєктах цим займаються два різних розробника якщо це не Full-Stack developer. З Фреймворком розібралися, настав час вибору бази даних, PostgreSQL дуже надійна реляційна база даних, яка підтримує складні запити, і дає змогу

працювати з великими обсягами даних, хоча на перших етапах буде використовуватися вбудована база даних яку надає Django(ще один його плюс) [10].

Далі про кеш, при великій кількості користувачів на сайті, буде робитися багато запитів до бази даних - це час, тому потрібно попіклуватися про те, щоб деякі сторінки загружалися швидко, з цим нам допоможе нереляційна база даних Redis. Що він вміє? Redis надає швидкий доступ до даних так як зберігає інформацію в оперативній памяті, а не на диску, тоді що робити якщо оперативна память буде зайнятою через Redis? Діло в тому що Redis вміє запамятовувати інформацію як назавжди, так і на визначений час, тобто якщо ми зазначимо час зберігання інформації в Redis, він буде зберігати її стільки, скільки вказано. Як це працює - спочатку користувач заходить на сторінку, відбувається запит до Redis, якщо інформації там немає, відбувається запит до нашої БД, в ній беруться данні, і записуються в Redis, після чого, інший користувач, або той самий користувач який оновить сторінку, помітить різницю між часом завантаження сторінки, так як далі, звернення буде відбуватися до Redis, який працює швидше, після вказаного часу на видалення даних з Redis, данні будуть очищені. Тобто Redis дає змогу швидкої роботи сайту [7].

Ще одним важливим інструментом який потрібно вказати Celery, іноді час на обробку деяких даних дуже довгий, і користувачу не цікаво буде сидіти і очікувати поки сторінка загрузиться, тоді чому не виконати це завдання в фоні? Сторінка користувачу загрузиться моментально, а завдання яке потрібно виконати, виконається але згодом, таку можливість надає Celery.

Ще одним компонентом який потрапляє до опису будуть фікстури, важлива річ для простоти роботи з базою даних. Коли проєкт на стадії розробки, не завжди є можливість звернутися до БД, в такому разі нам допоможуть фікстури, що вони роблять? Зберігають дані з таблиць бази

даних в JSON форматі, після чого якщо база даних пуста, ми її швидко заповнюємо даними з цього файлу. На ранніх етапах буде використовуватися вбудована в Django БД, і заповнюватиметься даними, щоб при переході на іншу БД не заповнювати її заново, використаються створені фікстури і БД заповниться даними [11, 12].

Також є проблема того, що потрібно витрачати час на візуальну частину сайту, щоб це швидше зробити, я використаю bootstrap, гарний інструмент для швидкого створення html сторінок, який також містить css стилі [14].

2.2. Планування табличок бази даних

Важливим завданням перед початком роботи з сайтом є планування табличок баз даних. Так як потрібно розуміти задачі які потрібно реалізувати. Наприклад в Django є вбудована модель User в якій вже є поля, і можна використовувати стандартну модель, та чи це підходить під мій сайт?

В стандартній моделі немає наприклад поля image, щоб на сайті користувач міг встановлювати фото. Якщо такий функціонал потрібно додати, потрібно переоприділити цю модель, і вже потім заповнювати її даними, якщо цього не зробити, спочатку заповнити даними, а вже потім зрозуміти що потрібно добавляти картинки користувача, для переоприділення бази моделі User знадобиться видалення всієї БД, щоб додати лише одне поле, або декілька, якщо ж попіклуватися про це спочатку, і визначитися які ж поля будуть в таблиці User, далі потрібно буде тільки проводити міграції [1, 2].

Тому планування БД є важливим завданням перед початком роботи. Спочатку розберемося як працює БД в Django, ми працюємо через моделі ORM, Object-Relational Mapping це технологія, яка дозволяє розробникам працювати з базою даних, використовуючи об'єктно-орієнтовані принципи, а не SQL-запити. Тобто працювати доведеться тільки з моделями, після чого

проводити міграції(файли в яких відбувається перетворення написаної моделі, до SQL запиту з використанням Python).

Коли я буду проводити зміни в моделі, і проводитиму міграцію, зміни будуть і в БД. Такий принцип роботи ORM Django, тепер потрібно зрозуміти які є завдання, після чого перейти до схематичного представлення БД.

На сайті я хочу реалізувати реєстрацію/авторизацію, тому потрібна модель User, далі буде модель продуктів на сайті, тобто потрібно реалізувати модель Product, щоб користувач міг заповнювати свій кошик товарів продуктами на сайті, значить потрібно реалізувати модель Basket з зв'язками до користувача User, і до продукту Product, також на моєму сайті повинні бути відгуки до товарів, потрібно реалізувати модель Review, зі зв'язками до моделей User, Product, було б не погано ще проводити сортування продуктів, тобто додати модель ProductCategory, з зв'язком до моделі Product [4].

На перших етапах потрібних для реалізації задуманого використовується бд, далі моделі будуть додаватися і оновлюватися за рахунок розробки, тобто вже на проєкті буде видно що потрібно додати, або оновити. На реальних проєктах цей етап доводять до кінця, тобто мають детальний план бази даних, але в подальшому уже на реалізації, можуть з'являтися нові ідеї, які будуть вноситися, ну і на проєкті також вже при виході, реалізація може оновлюватися. Перейдемо до опису моделей бази даних, безпосередньо полів бази даних.

Для моделі User - залишаються всі її стандартні поля, і добавляється тільки поле Image, це робиться для того щоб мати можливість додавати фото юзера на сайті, і також щоб потім була можливість оновлювати модель User без видалення БД:

- Модель Product - для неї потрібні поля з іменем продукту, описом, ціною, кількістю товарів, категорією(зв'язок з моделлю ProductCategory), та картинкою товару.
- Модель Basket - поле користувача(зв'язок з User), поле продукту(зв'язок з Product), кількість товарів, і створення.

- Модель Review - поле автора(User), поле продукту(Product), поле тексту відгуку, поле створення відгуку.
- Модель ProductCategory - поле імені категорії, поле опису категорії.

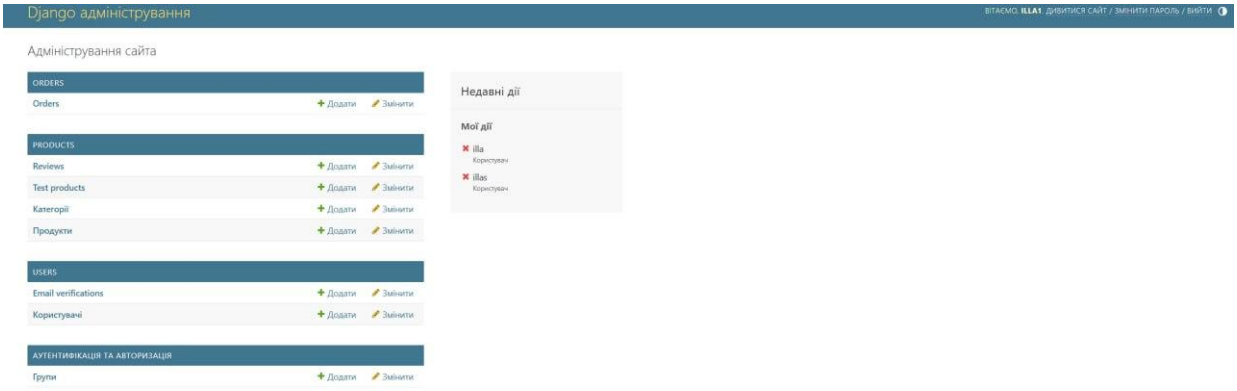


Рисунок 2.2.1 - Адмін панель з табличками

2.3. Планування структури сайту

Теоретичний опис сайту:

На сайті планується розробити головну сторінку, каталог з товарами, сторінку детального опису товару, систему реєстрації користувача на сайті, систему по відновленню паролю поштою, систему реєстрації через підтвердження електронної пошти, сторінку користувача, в якій буде видно його кошик товарів, та інформацію про цього користувача, також планую додати відгуки до товарів на сайт, підключення тестової оплати товарів на сайті, і реалізувати пошук по товарам на сайті, також зроблю розбивку товарів по категоріям, для зручності користувачів.

1. Головна сторінка.

Головна сторінка буде точкою входу для користувача, тому важливо оптимізувати її для пошукових систем.

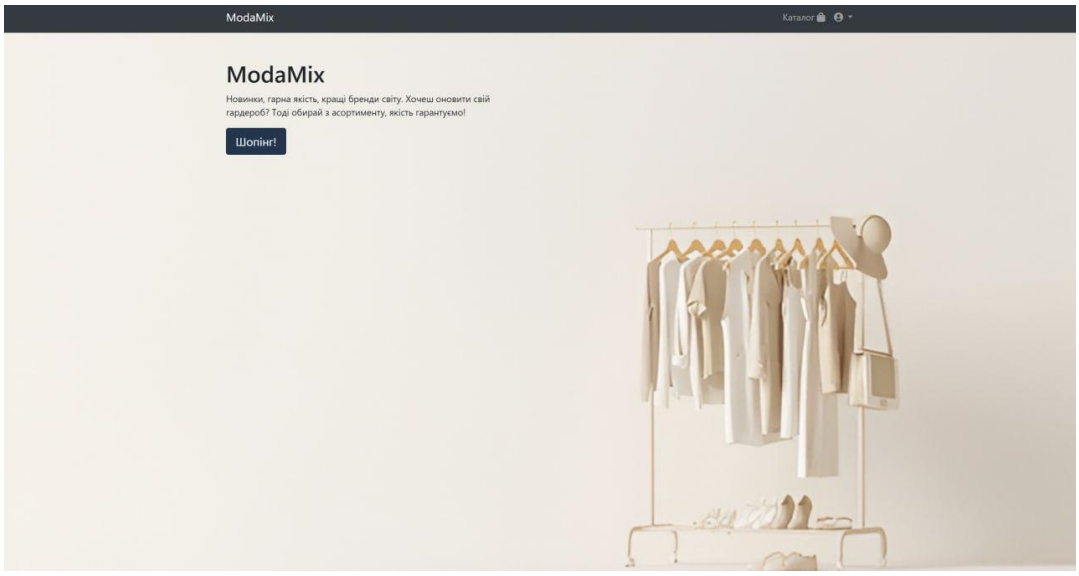


Рисунок 2.3.1 - Головна сторінка сайту

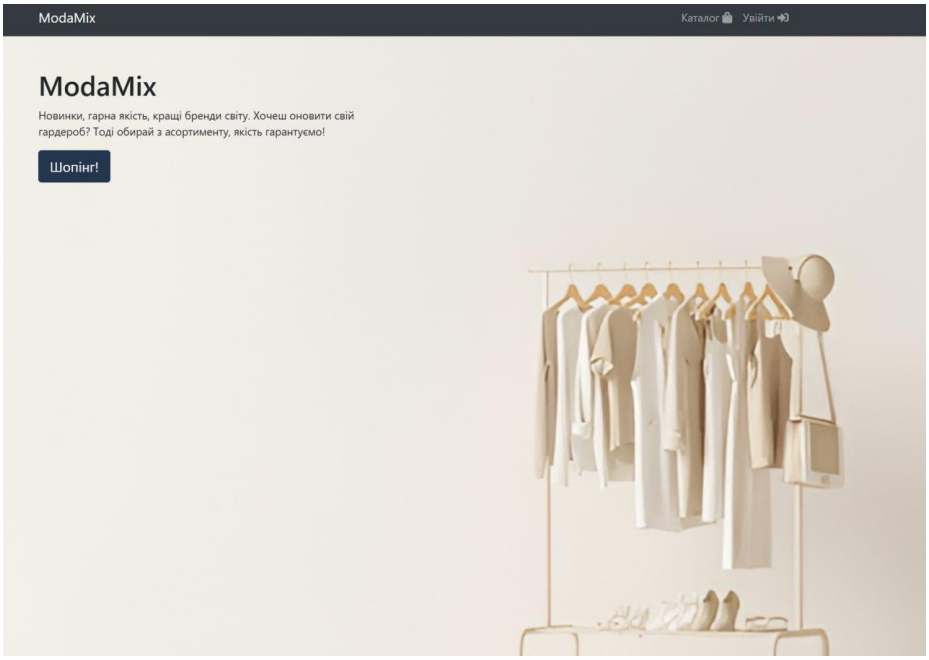


Рисунок 2.3.2 - Головна сторінка не авторизованого користувача

2. Каталог товарів.

Планується відобразити основні категорії товарів та популярні продукти, використовуючи Django views та шаблони. Для покращення швидкості завантаження сторінки можна реалізувати кешування даних, використовуючи Redis [3, 7].

Каталог відображатиме товари, згруповані за категоріями. Використовуючи Django models, можна створити модель для товарів, яка міститиме поля для назви, опису, ціни, зображення та категорії. Фільтрація товарів за категоріями та пошук товарів за ключовими словами буде реалізована за допомогою ListView і спеціальних фільтрів [4].

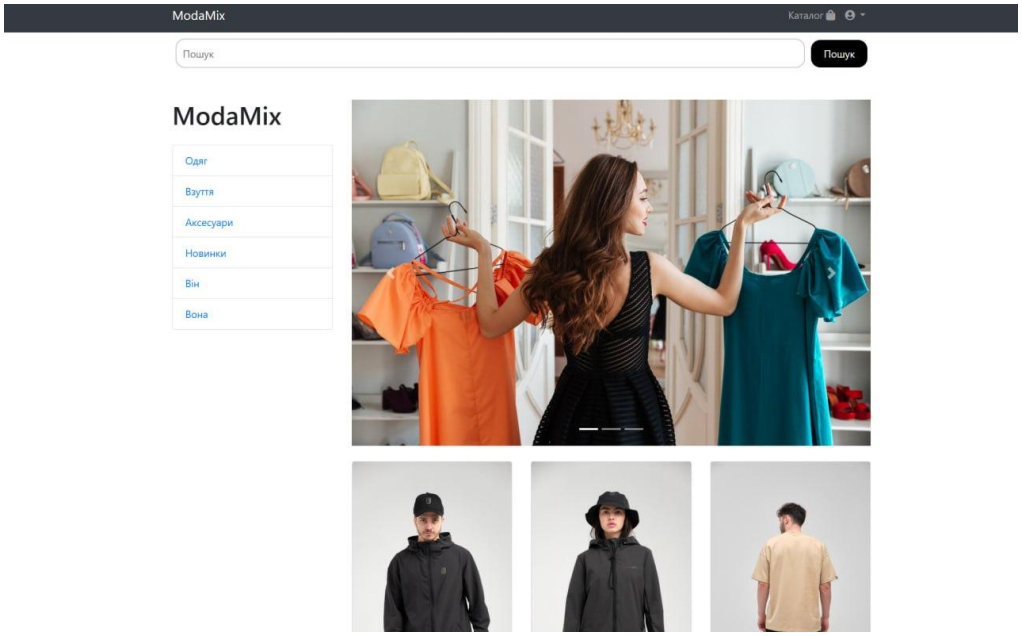


Рисунок 2.3.3 - Каталог товарів, головна сторінка

Технічна реалізація:

Створення моделі продукту з полями (назва, ціна, категорія).

Реалізація фільтрації товарів за категоріями та ціною.

Використання класових представлень (ListView) для відображення каталогу.

3. Сторінка детального опису товару.

Сторінка детального опису товару буде відображати всі характеристики продукту. Реалізація передбачає використання Django шаблонів та класу DetailView, що дозволить відображати дані конкретного товару.

Технічна реалізація:

Використання DetailView для відображення окремих товарів.

Додавання відгуків для товарів.

Використання ForeignKey для зв’язку з продуктами, користувачами та відгуками.

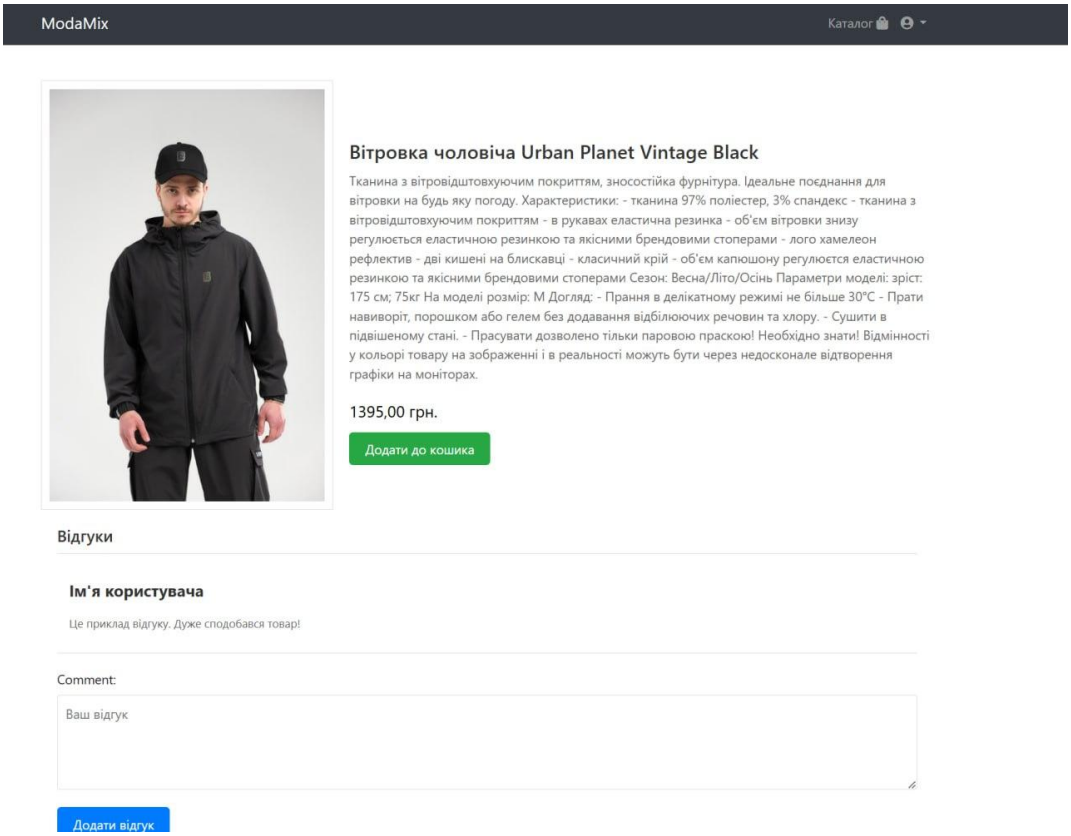


Рисунок 2.3.4 - Детальний опис товару

4. Система реєстрації користувача.

Для реєстрації користувачів буде використано форму на основі Django forms. Реєстрація включатиме перевірку унікальності email та пароль, а також відправлення листа для підтвердження реєстрації. Це можна реалізувати за допомогою бібліотеки django-allauth або власного рішення з використанням Django signals. В своїй роботі використовувався django-allauth через його простоту.

Технічна реалізація:

Використання UserCreationForm для створення користувачів.

Реалізація системи активації користувачів через email.

Перевірка наявності користувача за email та унікальність.

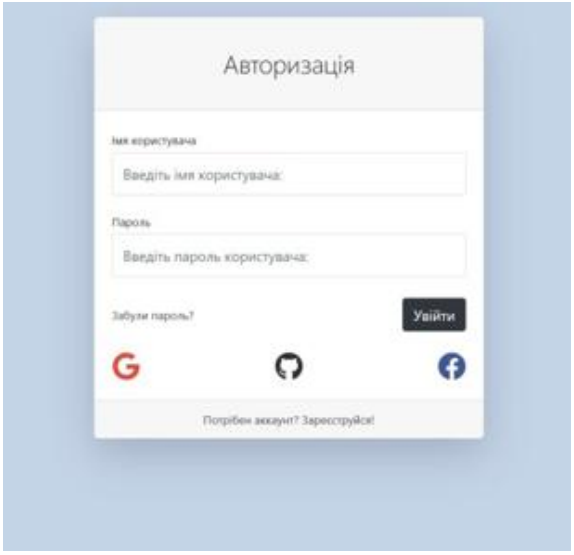


Рисунок 2.3.5 - Вхід в аккаунт користувача

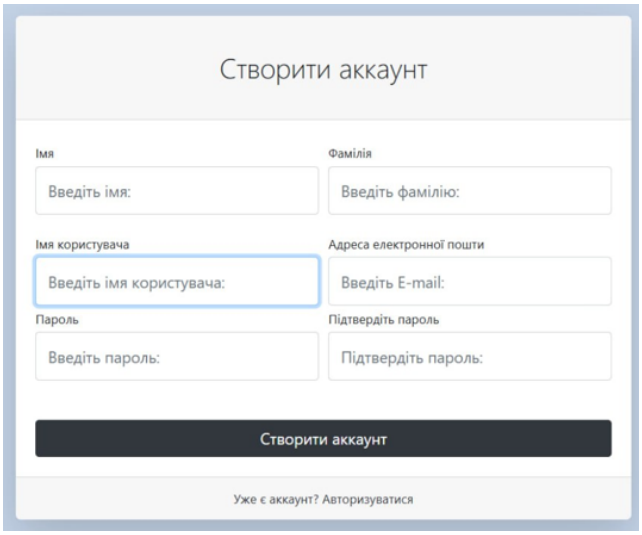


Рисунок 2.3.6 - Реєстрація користувача

5. Відновлення пароля поштою.

Для відновлення пароля планується використовувати стандартну Django функціональність для роботи з поштою та формами. Після введення email, користувачеві буде надсилатись лист з інструкціями щодо зміни пароля.

Технічна реалізація:

Використання вбудованого функціоналу Django для відновлення пароля.

Налаштування email-сервера для відправки листів.

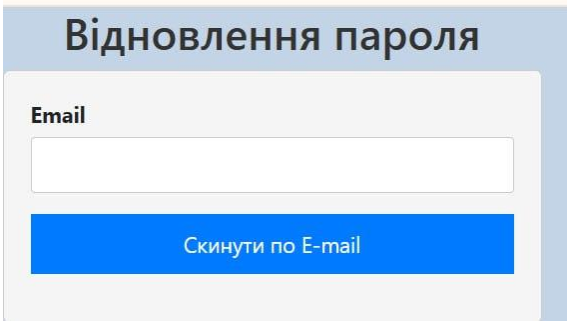


Рисунок 2.3.7 - Форма відновлення паролю

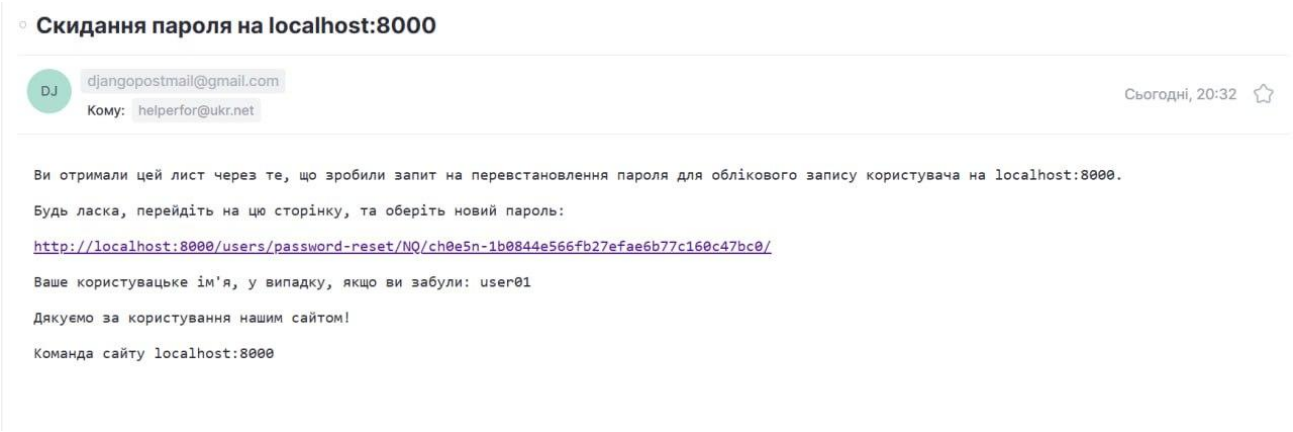


Рисунок 2.3.8 - Повідомлення на пошті від сайту, про скидання паролю

6. Сторінка користувача.

На сторінці користувача буде відображатися його особиста інформація, історія покупок та поточний кошик.

Технічна реалізація:

Створення моделі профілю користувача.

Додавання можливості редагування даних профілю.

Відображення кошика користувача та замовлень.

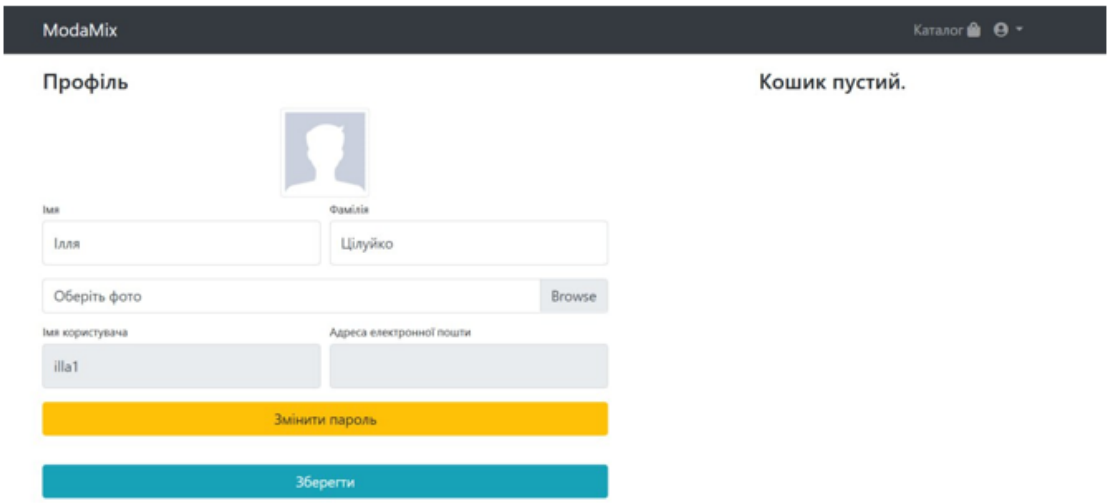


Рисунок 2.3.9 - Профіль користувача

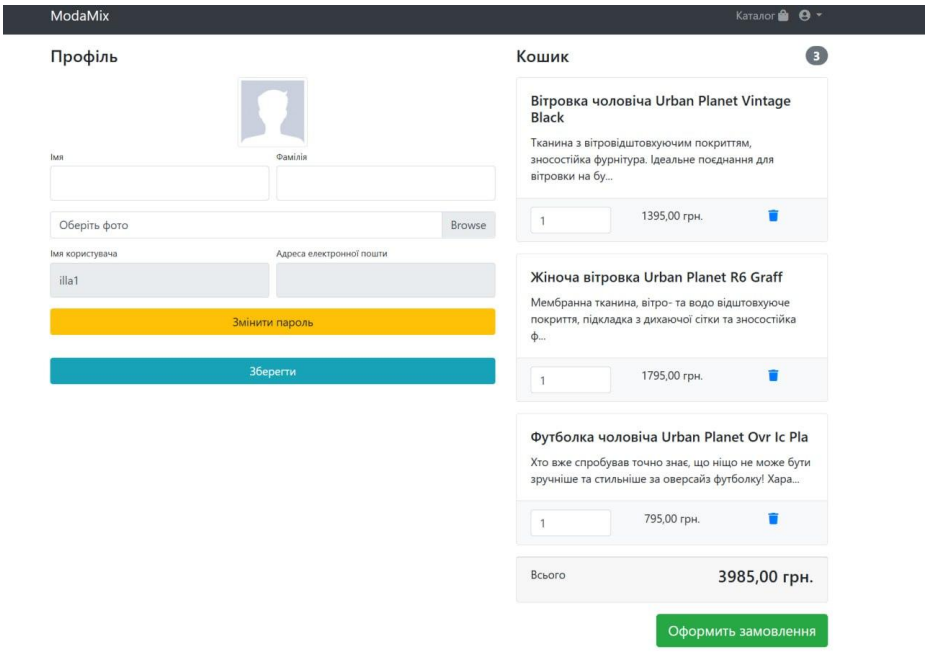


Рисунок 2.3.10 - Профіль користувача з кошиком

7. Відгуки до товарів.

Кожен товар на сайті може мати відгуки від користувачів. Для цього буде створено окрему модель відгуків, пов’язану з моделлю товару. Користувачі зможуть залишати текстові коментарі. Система відгуків до товарів вказана на рис. 2.3.4.

Технічна реалізація:

Створення моделі для відгуків з полями для тексту.

Відображення відгуків на сторінці товару.

Обмеження можливості залишати відгуки тільки для зареєстрованих користувачів.

8. Тестова оплата товарів.

Для реалізації тестової оплати планується інтеграція платіжної системи Stripe. Платіжна система дозволить користувачам здійснювати покупки, використовуючи тестовий режим для перевірки функціональності.

Технічна реалізація:

Інтеграція Stripe API для обробки платежів.

Реалізація вебхуків для обробки успішних та невдалих оплат.

Налаштування тестового режиму для перевірки платіжної функціональності.

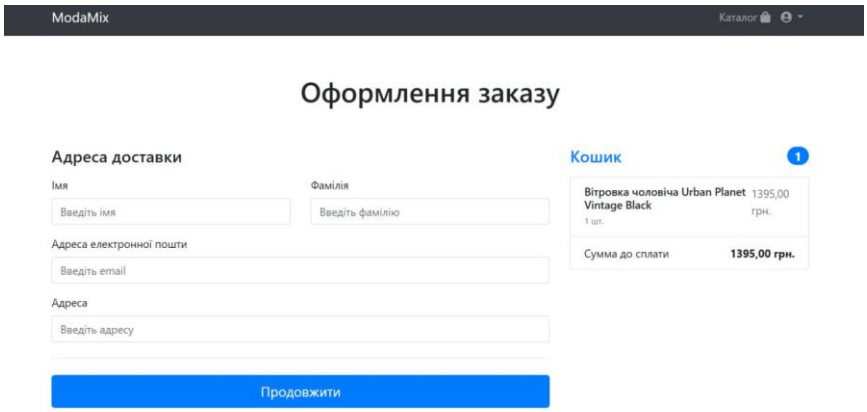


Рисунок 2.3.11 - Сторінка оформлення замовлення

9. Пошук за товарами.

На сайті буде реалізовано функцію пошуку товарів за ключовими словами. Це буде зроблено через створення форми пошуку та фільтрацію результатів на основі введених даних.

Технічна реалізація:

Реалізація форми пошуку на основі Django forms.

Використання функції search() у моделі товару для фільтрації результатів.

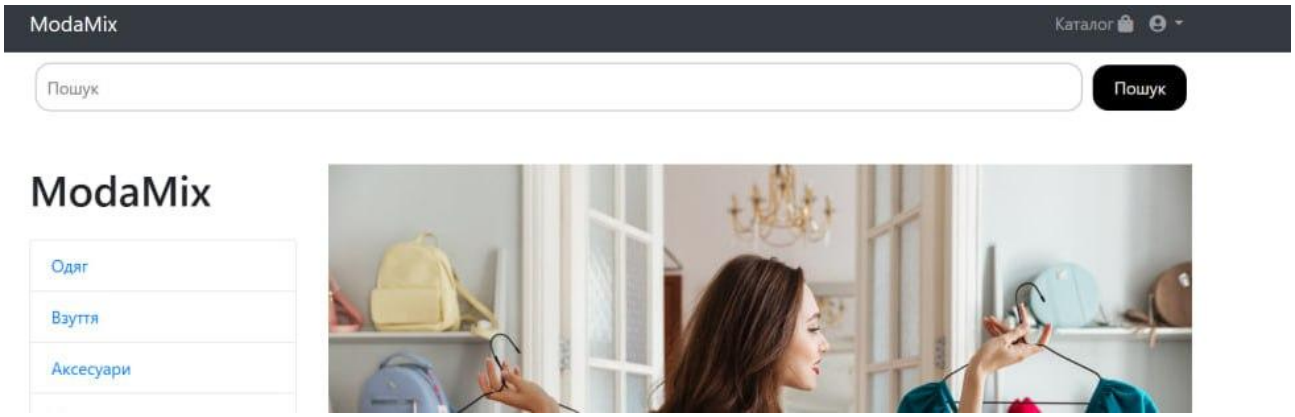


Рисунок 2.3.12 - Пошук товарів

10. Категоризація товарів.

Товари будуть розбиті по категоріях для зручності користувачів. Це передбачає створення моделі категорій та їх ієрархічне відображення на сайті. Відвідувачі зможуть переглядати товари, фільтруючи їх за відповідними категоріями.

Технічна реалізація:

Створення моделі категорій з ієрархією.

Відображення категорій у вигляді меню.

Фільтрація товарів за категоріями.

Кожен з цих кроків допоможе розробити гарний вебсайт який охоплює конкретну функціональність, де використовується Django як основа для backend-логіки. Функціональність сайту буде базуватися на використанні класичних Django механізмів, таких як views, models, forms, а також додаткових бібліотек для інтеграції з платіжними системами та email-сервісами [3].

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Теоретичний опис інструментарію

Django

Django – високорівневий фреймворк на Python, створений для швидкого створення складних веб-додатків. Дотримуючись принципів DRY (Don't Repeat Yourself) і Convention over Configuration, він зменшує повторення коду та спрощує налаштування, забезпечуючи чистий синтаксис і високу ефективність розробки [3].

Фреймворк використовує архітектуру MTV (Model-Template-View), яка розділяє логіку, дані та інтерфейс, полегшуючи організацію та підтримку коду. Завдяки вбудованому ORM (Object-Relational Mapping) розробники працюють із базами даних через Python-класи, уникаючи прямого використання SQL. Це робить Django сумісним із різними базами даних, такими як PostgreSQL, MySQL і SQLite [4].

Django має потужні інструменти безпеки, які захищають від XSS, CSRF і SQL-ін'єкцій, а також автоматично генерує адміністративну панель для керування даними через веб-інтерфейс. Гнучка система маршрутизації спрощує організацію URL, а шаблони забезпечують динамічне створення HTML із чітким відокремленням логіки та подання [16].

Підтримка кешування та інтеграція з асинхронними задачами (наприклад, Celery і Redis) роблять Django придатним як для невеликих проектів, так і для масштабних систем із високим навантаженням. Його структура дозволяє швидко створювати прототипи та стабільні додатки завдяки стандартним налаштуванням і можливості їх легко адаптувати [7].

Jinja

Jinja – Система шаблонів для веб-додатків

Jinja – це інструмент для розділення бізнес-логіки й подання у веб-додатках. Вона дозволяє створювати динамічні HTML-документи, інтегруючи серверні дані у шаблони. Хоча Jinja широко використовується у різних фреймворках, у Django застосовується власна система шаблонів із подібним синтаксисом.

Jinja має простий синтаксис із використанням подвійних фігурних дужок {{ }} для змінних і {% %} для логічних конструкцій, таких як цикли чи умови. Серед її ключових можливостей – підтримка умовних операторів, циклів для відображення даних зі списків, а також вбудовані фільтри для форматування, наприклад, |lower для зниження регістру або |date для форматування дати.

Система підтримує наслідування шаблонів, що дозволяє створювати базові шаблони для спільних елементів дизайну та розширювати їх у дочірніх. Макроси спрощують роботу з повторюваними блоками коду, роблячи шаблони більш структурованими.

Jinja отримує дані через контекст, переданий із сервера, що дозволяє відображати змінні, списки або об'єкти Python у шаблонах, мінімізуючи дублювання коду й забезпечуючи гнучкість у розробці.

Redis

Redis – Інструмент для кешування та управління даними

Redis (Remote Dictionary Server) – це швидке open-source сховище даних у пам’яті, оптимізоване для кешування, обробки сесій і черг повідомлень. Завдяки роботі в оперативній пам’яті Redis забезпечує доступ до даних за мілісекунди, що робить його ідеальним для веб-додатків [7].

Підтримуючи структури даних, як-от рядки, списки, геші, множини та геоіндекси, Redis використовується для кешування, TTL-управління даними, публікацій/підписок і черг повідомлень.

Хоча він працює в пам’яті, Redis дозволяє зберігати дані на диск за допомогою знімків або журналу (AOF), забезпечуючи стійкість у разі збою. Висока швидкість (до мільйонів запитів/секунду) і підтримка масштабування (шардінг, реплікація) роблять його надійним для завдань із великим навантаженням.

Інтеграція з Lua-скриптами і підтримка асинхронності підвищують ефективність роботи з великими обсягами даних, особливо у зв’язці з інструментами на кшталт Celery.

Celery

Celery – Система управління асинхронними завданнями

Celery – це інструмент для асинхронного виконання завдань і управління чергами у веб-додатках. Він дозволяє виконувати тривалі або ресурсомісткі завдання у фоновому режимі, не блокуючи основний потік додатка. Наприклад, Celery може надсилати листи з підтвердженням під час реєстрації користувача, не затримуючи відповіді від сервера [12].

Цей таск-менеджер підтримує розподілену архітектуру, що дозволяє виконувати завдання на декількох вузлах (worker), забезпечуючи високу масштабованість. Завдання групуються в черги з можливістю задавати пріоритети, що гарантує ефективний розподіл ресурсів.

Celery інтегрується з брокерами повідомлень, такими як Redis і RabbitMQ, які передають завдання між додатком і робочими вузлами. Крім того, підтримується повторне виконання невдалих завдань, обмеження часу їх виконання (тайм-аут) і розклад для періодичних завдань, подібний до cron.

Для моніторингу і управління завданнями Celery пропонує інструменти, зокрема Flower, що надає веб-інтерфейс для перегляду статусу завдань у реальному часі.

Архітектура Celery складається з трьох основних компонентів:

- Брокер повідомлень (Redis, RabbitMQ) для передачі завдань.
- Робочі вузли (workers), які виконують завдання.
- Результат (backends) для збереження результатів, наприклад, у Redis чи базах даних.

Stripe

Stripe – Платіжна система для інтеграції з веб-додатками

Stripe – це платіжна платформа, яка дозволяє інтегрувати онлайн-платежі через різні канали, такі як кредитні картки, електронні гаманці (Apple Pay, Google Pay), банківські перекази та інші методи. Вона підтримує гнучкість для бізнесу та можливість обслуговувати клієнтів по всьому світу, зокрема завдяки добре задокументованому API та SDK для різних мов програмування, зокрема Python та Node.js.

Платформа також пропонує функціонал для управління підписками та періодичними платежами, що корисно для бізнес-моделей на основі регулярних оплат, таких як SaaS. Stripe забезпечує безпеку за допомогою стандартів PCI DSS Level 1 і 3D Secure, а також дозволяє легко обробляти повернення коштів та часткові відшкодування.

Завдяки розширеній аналітиці та звітності, Stripe дозволяє відстежувати всі транзакції та отримувати детальну інформацію про доходи, типи платежів та географію користувачів. Платформа підтримує більше 135 валют, що дозволяє працювати на міжнародному ринку, а також пропонує Stripe Connect для інтеграції платежів у маркетплейсах та платформах. Для мобільних додатків Stripe надає API для інтеграції мобільних платежів.

Fixtures

Django Fixtures – Інструмент для завантаження та експорту даних

Fixtures у Django – це спосіб серіалізації та десеріалізації даних з бази даних у файли, що дозволяє зберігати дані у форматах JSON, XML або YAML. Вони використовуються для завантаження або експорту наборів даних, які можуть бути імпортовані в базу даних під час розробки чи тестування. [11]

Основними цілями використання fixtures є завантаження початкових даних, тестування та резервування даних. Fixtures дозволяють завантажувати базові дані для додатків або забезпечувати стабільність тестових сценаріїв, завантажуючи фіксовані набори даних перед запуском тестів. Вони також корисні для передачі даних між різними середовищами, наприклад, для експорту з локальної бази даних і завантаження на сервер.

Django підтримує формати JSON, XML і YAML для збереження даних у fixtures. JSON є найпоширенішим форматом через свою простоту і підтримку в багатьох веб-додатках, XML менш популярний, а YAML потребує додаткової бібліотеки для роботи.

Основні переваги fixtures: автоматизація завантаження даних, стабільність тестування завдяки однаковим даним на кожному запуску тестів та мобільність при передачі даних між середовищами.

Models

Django Models – Інструмент для роботи з базою даних

Django Models є основою для роботи з базами даних у Django, де модель представляє таблицю в базі даних, а її екземпляри — записи в таблиці. Використовуючи ORM (Object-Relational Mapping), Django дозволяє взаємодіяти з базою даних через об'єкти Python без написання SQL-запитів вручну.

Моделі визначають структуру таблиці, автоматично генеруючи SQL-запити для її створення. Django ORM дозволяє створювати, читати, оновлювати і видаляти записи за допомогою Python, спрощуючи код і зменшуючи потребу в SQL. Моделі автоматично відображають поля на типи даних SQL, наприклад, CharField стає VARCHAR, а IntegerField — INTEGER. Міграції дозволяють автоматично змінювати структуру бази даних при оновленні моделей [4].

Django надає різноманітні типи полів для моделей: CharField для коротких рядків, TextField для довгих текстів, IntegerField для цілих чисел, DecimalField для десяткових чисел, BooleanField для булевих значень, DateTimeField для дати і часу, ForeignKey для зв'язків один до багатьох та ManyToManyField для зв'язків багато до багатьох.

Views

Django Views – Логіка представлення даних у веб-додатку

Django Views обробляють запити та повертають відповіді у веб-додатках, включаючи бізнес-логіку для визначення того, що відображати користувачеві. Подання отримує HTTP-запит, обробляє його і повертає відповідь. Основні функції Views включають обробку запитів, підготовку даних для шаблонів і обробку форм (отримання, перевірка та збереження даних).

Django підтримує два типи подань: функціональні (FBV) та класові (CBV). FBV є простими функціями для обробки запитів, зручними для простих випадків. CBV організовують код у вигляді класів, що дозволяє легко повторно використовувати логіку, що особливо корисно для великих проектів.

Основні CRUD-операції в Views включають:

1. Читання (ListView, DetailView) – для відображення списків чи окремих об'єктів.
2. Створення (CreateView) – для додавання нових об'єктів через форми.

3. Оновлення (UpdateView) – для редагування існуючих об'єктів.
4. Видалення (DeleteView) – для видалення об'єктів.

Urls

Django URLs – Маршрутизація запитів у веб-додатку

Django URLs – це система маршрутизації, що зіставляє веб-запити з відповідними поданнями у вашому веб-додатку. Кожна сторінка має певний URL, і коли користувач звертається до нього, запит обробляється відповідним поданням.

Основні функції Django URLs включають маршрутизацію запитів на основі URL-адрес, організацію чіткої структури URL для зручності розробників та користувачів, а також підтримку іменованих URL-адрес, що полегшує створення посилань і зміни структури без втрати гнучкості.

Щоб налаштувати URL-шляхи у Django, використовують файл urls.py, де визначають шаблон URL та відповідне подання. У великих проектах можна використовувати функцію include() для розбиття налаштувань URL по додатках.

Іменовані URL-адреси дозволяють звертатися до них за іменем, що робить код зрозумілішим і спрощує зміни в структурі.

django-allauth

Django Allauth – Система автентифікації та реєстрації

Django Allauth – це потужний пакет для Django, що забезпечує зручні та гнучкі можливості для обробки автентифікації, реєстрації та управління профілями користувачів. Пакет надає готові рішення для таких функцій, як реєстрація, вхід/вихід, підтвердження електронної пошти, відновлення пароля та соціальна автентифікація (вход через соціальні мережі).

Основні функції Django Allauth

Реєстрація користувачів: Django Allauth дозволяє легко додавати функціонал реєстрації користувачів на ваш сайт. Він включає в себе форму для реєстрації, обробку валідації та збереження нових користувачів у базі даних.

Підтвердження електронної пошти: Пакет підтримує підтвердження електронної пошти, що дозволяє вам перевіряти адресу електронної пошти користувача перед наданням повного доступу до функцій сайту. Це важливий етап для покращення безпеки та зменшення кількості фальшивих облікових записів.

Відновлення пароля: Django Allauth включає механізм для скидання пароля через електронну пошту, що дозволяє користувачам відновлювати доступ до своїх облікових записів у разі втрати пароля.

Соціальна автентифікація: Пакет підтримує входи через соціальні мережі (такі як Google, Facebook, Twitter тощо), що дозволяє користувачам реєструватися та входити на сайт за допомогою своїх облікових записів у цих мережах.

Гнучкість налаштувань: Django Allauth має безліч параметрів конфігурації, які дозволяють кастомізувати процеси реєстрації, автентифікації, а також управління профілем користувача.

Bootstrap

Bootstrap – CSS-фреймворк для швидкої розробки веб-додатків

Bootstrap – це популярний CSS-фреймворк, розроблений для спрощення процесу створення адаптивних та інтерактивних веб-сайтів. Він був створений Twitter і швидко здобув популярність завдяки своїй простоті, гнучкості та великій кількості готових компонентів [15].

Основні характеристики Bootstrap

Адаптивний дизайн: Bootstrap підтримує адаптивний (responsive) дизайн, що дозволяє веб-сайтам автоматично підлаштовуватися під різні розміри екранів (десктопи, планшети, мобільні пристрої). Це досягається за рахунок використання гнучких сіток, медіа-запитів CSS і відносних одиниць виміру.

Гнучка сітка: Bootstrap має потужну сіткову систему, що складається з 12 колонок, які дозволяють легко створювати складні макети. Компоненти сітки можна комбінувати, щоб створити бажану структуру сторінки. Вона також підтримує різні формати для різних пристроїв (мобільний, планшет, десктоп).

Готові компоненти: Bootstrap надає великий набір готових до використання компонентів, таких як кнопки, форми, таблиці, модальні вікна, навігаційні панелі, каруселі тощо. Ці компоненти мають стилі, які забезпечують єдність дизайну та зручність у використанні.

JavaScript-плагіни: Bootstrap містить також набір JavaScript-плагінів, які дозволяють додавати інтерактивність до веб-сайту. Це може бути обробка форм, спливаючі вікна, каруселі, вкладки, аккордеони та інші елементи інтерфейсу.

Кастомізація: Bootstrap дозволяє легко кастомізувати стилі та компоненти. Можна використовувати SCSS для зміни стандартних змінних Bootstrap, а також можна включити лише ті компоненти, які потрібні, для зменшення розміру файлу.

Кросбраузерна сумісність: Bootstrap підтримує більшість сучасних веб-браузерів, що забезпечує однаковий вигляд і поведінку на різних платформах.

Docker

Docker – Платформа для контейнеризації

Docker – це платформа для створення, розгортання та запуску додатків в контейнерах, що гарантує консистентність роботи програм у різних середовищах та спрощує процеси розгортання.

Основні характеристики Docker включають контейнеризацію, яка дозволяє упаковувати додатки з усіма залежностями в контейнери для забезпечення однакової роботи на різних середовищах, швидке розгортання завдяки легковісним контейнерам, ізоляцію середовищ для уникнення конфліктів версій, масштабованість для обробки більшого навантаження, управління версіями контейнерів та велику спільноту, яка підтримує різні інструменти для Docker.

Основні компоненти Docker включають Docker Engine для управління контейнерами, Docker Hub для обміну образами, Docker Images як шаблони для контейнерів, Docker Containers як запущені екземпляри образів, та Dockerfile для автоматизації створення образів.

3.2. Переваги та недоліки обраного інструментарію

Django

Переваги:

- Швидкість розробки: Django забезпечує швидке створення додатків завдяки вбудованим інструментам, як ORM, авторизація та адміністративна панель.

- Структурованість: Чітка архітектура дозволяє легко масштабувати проєкти.
- Безпека: Включає механізми захисту від SQL-ін'єкцій, CSRF, XSS тощо.
- Спільнота: Велика спільнота та хороша документація полегшують навчання та вирішення проблем.

Недоліки:

- Монолітність: Створений як монолітний фреймворк, що ускладнює модульність і мікросервісну архітектуру.
- Надмірність: Для малих проєктів Django може бути занадто складним і важким через надлишок інструментів [3].

Redis

Переваги:

- Швидкість: Redis працює в оперативній пам'яті, що дозволяє обробляти дані дуже швидко, особливо для задач кешування та зберігання тимчасових даних.
- Гнучкість: Підтримує різні структури даних, такі як хеші, списки, множини тощо.
- Масштабованість: Redis легко масштабується та може працювати як у кластері, так і в режимі реплікації.

Недоліки:

- Оперативна пам'ять: Redis зберігає дані в оперативній пам'яті, що може призводити до проблем із обсягом пам'яті при роботі з великими обсягами даних.
- Відсутність складної обробки даних: Redis призначений для простих операцій зберігання й витягування даних, тому для складних запитів або обробки потрібні додаткові інструменти [7].

Celery

Переваги:

- Асинхронність: Celery дозволяє виконувати задачі асинхронно, що знижує навантаження на сервер і підвищує швидкість відповіді для користувачів.
- Масштабованість: Легко масштабується при збільшенні кількості задач завдяки підтримці брокерів повідомлень, таких як Redis або RabbitMQ.
- Планування задач: Підтримка періодичних задач дозволяє використовувати Celery як планувальник (альтернатива cron).

Недоліки:

- Налаштування: Celery потребує додаткових налаштувань та конфігурації, особливо для інтеграції з брокером повідомлень та балансування навантаження.
- Ускладнення моніторингу: Складніше відслідковувати та дебажити асинхронні задачі, особливо у випадку помилок чи зависань [12].

Bootstrap

Переваги:

- Простота використання: Готові компоненти та шаблони полегшують розробку фронтенду.
- Адаптивність: Вбудована підтримка адаптивної верстки для різних пристроїв.
- Підтримка спільноти: Велика кількість готових рішень та активна спільнота.

Недоліки:

- Однотипність: Сайти на Bootstrap часто виглядають подібно через стандартні шаблони.
- Великий обсяг коду: Додається зайвий код, що може вплинути на продуктивність [15].

3.3. Опис задач та алгоритмів вирішення

Задача 1: Створення сайту з товарами та відгуками

Опис задачі: Користувачі повинні мати можливість переглядати товари та залишати відгуки. Кожен товар може мати кілька відгуків.

Алгоритм вирішення:

1. Моделювання даних:
- Модель для товару (Product) з полями: назва, опис, ціна, зображення.
- Модель відгуків (Review) з полями: текст, рейтинг, та зовнішніми ключами до товару та користувача.
2. Форма для відгуків: Створення форми для вводу відгуків з перевіркою валідності.
3. Відображення товарів та відгуків: На сторінці товару відображаються дані товару та відгуки, з можливістю додавання нових.
4. Реалізація логіки: Створення подань (views) для обробки запитів та збереження нових відгуків.

Задача 2: Система реєстрації та відновлення паролів

Опис задачі: Користувачі повинні мати можливість реєструватися на сайті, змінювати пароль і отримувати листи для скидання пароля.

Алгоритм вирішення:

1. Моделі користувачів: Використання стандартної або кастомізованої моделі користувача Django.

2. Форма реєстрації: Створення форми реєстрації з валідацією даних.
3. Зміна пароля: Форми для зміни пароля з перевіркою старого та нового.
4. Скидання пароля через email: Використання стандартного механізму Django для скидання пароля через email з унікальним токеном.
5. Налаштування SMTP: Конфігурація SMTP-сервера для відправки email.

Задача 3: Система оплати через Stripe.

Опис задачі: Користувачі повинні мати можливість оплачувати товари через Stripe.

Алгоритм вирішення:

1. Інтеграція Stripe: Використання Stripe API для Python.
2. Створення замовлення: Збір даних про товари в кошику.
3. Платіжна форма: Створення форми для введення платіжних даних через Stripe Checkout.
4. Обробка транзакції: Використання Stripe API для перевірки та обробки платежу, створення запису про оплату.
5. Безпека: Використання HTTPS для захисту конфіденційних даних.

Задача 4: Профіль користувача з кошиком.

Опис задачі: Кожен користувач повинен мати персональний профіль, де зберігатиметься інформація про покупки та кошик.

Алгоритм вирішення:

1. Модель кошика: Кошик зберігає товари, додані користувачем, до підтвердження замовлення. Створюється окрема модель для кошика, яка пов'язана з користувачем і товарами.

2. Інтерфейс користувача: На сторінці профілю користувач може переглядати кошик, додавати або видаляти товари, а також завершити покупку.
3. Збереження кошика: Кошик зберігається в базі даних або у сеансах (sessions), що дозволяє зберігати дані навіть після виходу з системи.

Задача 5: Реалізація категорій через Redis

Опис задачі: Категорії повинні містити товари. Цей список можна реалізувати через Redis для підвищення продуктивності.

Алгоритм вирішення:

1. Зберігання в Redis: Товари зберігаються в Redis, що дозволяє швидко витягувати дані без необхідності кожного разу запитувати базу даних.
2. Оновлення кешу: При додаванні нових товарів або змінах у популярності, відповідні дані оновлюються в Redis.
3. Отримання новинок: При запиті на сторінку новинок дані отримуються безпосередньо з Redis, що прискорює відображення списку.

Задача 6: Пошук по сайту з використанням Elasticsearch

Опис задачі: На сайті повинна бути функція пошуку товарів з використанням Elasticsearch.

Алгоритм вирішення:

1. Інтеграція з Elasticsearch: Використання бібліотеки Elasticsearch для Python.
2. Індексація товарів: Дані про товари індексуються в Elasticsearch для пошуку за різними критеріями.
3. Пошуковий інтерфейс: Форма пошуку, що відправляє запити до Elasticsearch та відображає результати.

4. Оновлення індексу: Оновлення індексу при зміні або додаванні товарів для актуальних результатів пошуку.

Задача 7: Фікстури для тестування.

Опис задачі: Для тестування необхідно створити фікстури — дані для автоматичного наповнення бази даних.

Алгоритм вирішення:

1. Формат фікстур: Фікстури створюються у форматах JSON або YAML з тестовими даними для моделей.
2. Автоматичне завантаження: Фікстури завантажуються під час тестів через команду Django loaddata.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис розробки сайту

Спочатку потрібно створити Django проєкт, в якому налаштувати основу для роботи, після чого вже йти далі. Для початку створюємо директорію в якій буде наш проєкт, після чого в любій IDE (в моєму випадку PyCharm) створюємо python project в якому прописуємо команду pip install Django, якщо воно не приймає pip потрібно в налаштуваннях обрати інтерпритер (рис. 4.1.1) (він знаходиться в директорії в яку завантажувався python).

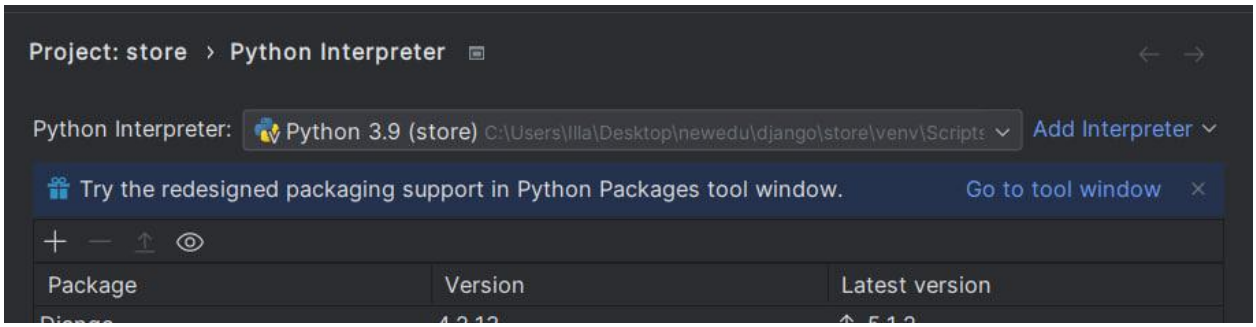


Рисунок 4.1.1 - Інтерпритатор

Зберігаємо налаштування, і встановлюємо Django

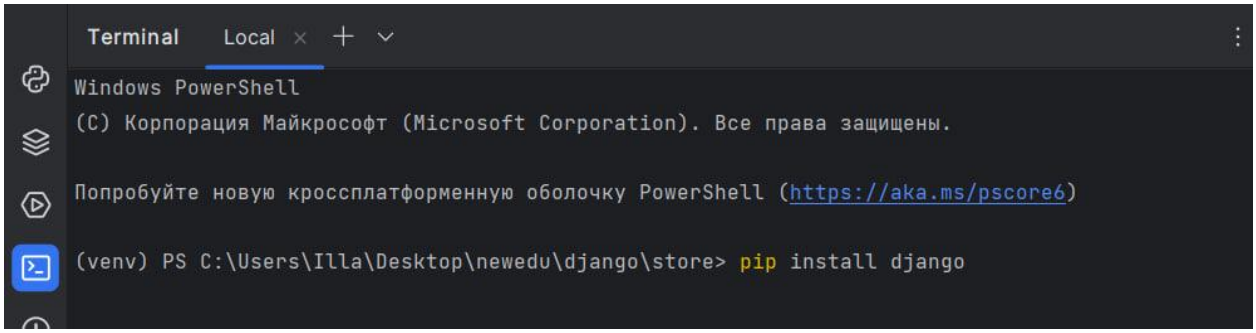


Рисунок 4.1.2 - Термінал (команда встановлення Django)

У даній роботі використовується в терміналі IDE PyCharm, але можна і у звичайному терміналі (рис. 4.1.2) завантажити джанго у свій проект, для

цього потрібно буде перейти в консолі в те віртуальне оточення того проєкту, куди потрібно завантажити фреймворк.

Після вдалого завантаження потрібно створити сам проєкт джанго, це робиться командою django-admin startproject modamix-store. Після чого в директорії проєкту з'являться стандартні файли для роботи з джанго проєктом (рис.4.1.3).

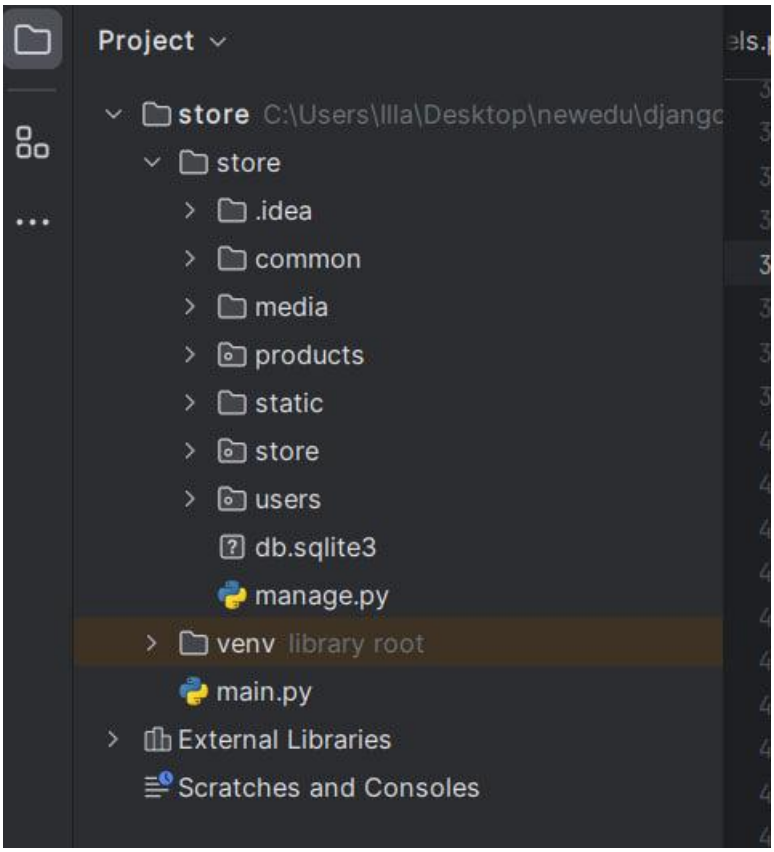


Рисунок 4.1.3 - Структура проекту

В даному прикладі видно і наступний крок який потрібно зробити, створити додаток, який буде відповідати за конкретну функцію на сайті, це команда python manage.py startapp users таким чином ми створюємо додаток в якому будемо описувати, контролери, шаблони, шляхи, форми, моделі, які відносяться якраз до додатка users. Також, щоб все працювало, в налаштуваннях проєкту які знаходяться тут (рис 4.1.4).

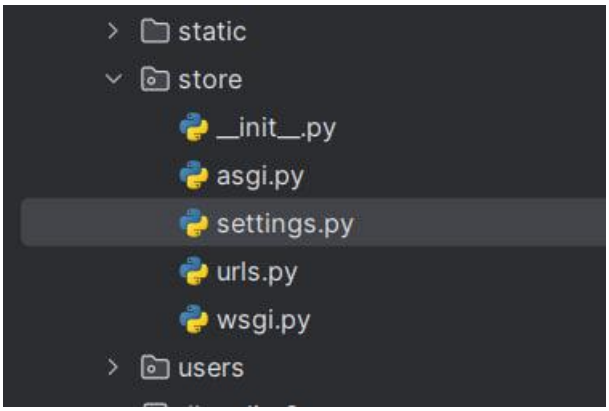


Рисунок 4.1.4 - Налаштування

Потрібно занести в список встановлених додатків назви створених для проекту додатків (рис. 4.1.5)

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites',

    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.github',

    'products',
    'users',
]
```

Рисунок 4.1.5 - Installed_apps

І далі можна працювати вже з додатками напряму, пишучи основну логіку сайту.

В urls.py прописуємо шляхи для взаємодії між додатками (рис. 4.1.6).

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', IndexView.as_view(), name='index'),
    path('products/', include( arg: 'products.urls', namespace='products')),
    path('users/', include( arg: 'users.urls', namespace='users')),
    path('accounts/', include('allauth.urls')),
]
```

Рисунок 4.1.6 - Шляхи вибору

В дочірніх додатках в urls.py прописуються шляхи конфігурації по сторінках сайту і, наприклад шлях в домені до додатка users буде виглядати як «users/profile» навіть коли шлях буде записаний як path('profile/<int:pk>/', login_required(UserProfileView.as_view()), name='profile'),

Далі по такій самій схемі створюються потрібні і вищеописані додатки до сайту, і прописуються їм шляхи як на попередніх знімках екрану проєкту. Після чого переходжу до написання front-end частини сайту. Робиться це за допомогою Bootstrap стилів, щоб вручну не прописувати велику кількість візуалу, а тільки підкорегувати під свій стиль сайту готові рішення.

Вигляд написаних сторінок для відображення користувачу в веббраузері, в структурі сайту має:

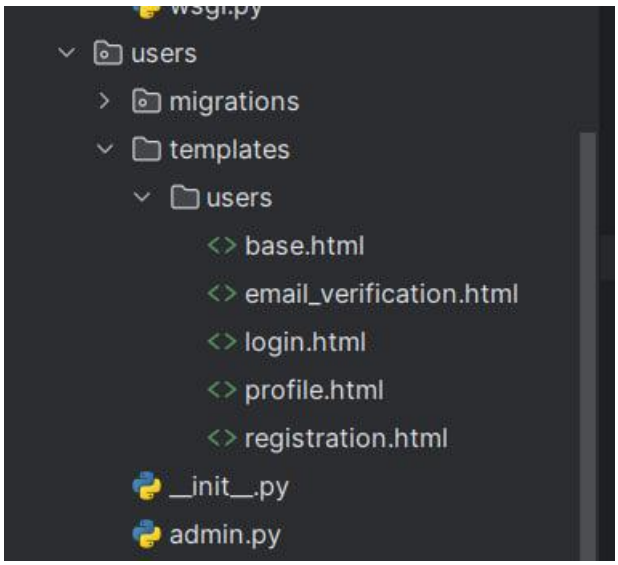


Рисунок 4.1.7 - Шаблони

В кожному додатку прописуються templates (рис. 4.1.7) в них створюється директорія з назвою додатка, і вже в ній створюються html шаблони. В них використовується Jinja описаний вище, через те що він дає змогу створити основу сторінки, і далі за допомогою наслідування просто прописувати візуальну частину інших сторінок, не прописуючи одне й те саме на них, принцип DRY.

Після цього планується створення табличок БД, створити нарешті моделі, і починати роботу з контролерами. Стандартна модель користувача була змінена (рис. 4.1.8).

```
class User(AbstractUser):
    image = models.ImageField(upload_to='users_images', null=True, blank=True)
    is_verified_email = models.BooleanField(default=False)
```

Рисунок 4.1.8 - Клас користувача

store\urls.py | users\urls.py | models.py

```
1 usage
class ProductCategory(models.Model):
    name = models.CharField(max_length=128, unique=True)
    description = models.TextField(null=True, blank=True)

    class Meta:
        verbose_name = 'Категорія'
        verbose_name_plural = 'Категорії'

    def __str__(self):
        return self.name


1 usage
class Product(models.Model):
    name = models.CharField(max_length=256)
    description = models.TextField()
    price = models.DecimalField(max_digits=6, decimal_places=2)
    quantity = models.PositiveIntegerField(default=0)
    image = models.ImageField(upload_to='products_images')
    category = models.ForeignKey(to=ProductCategory, on_delete=models.CASCADE)

    class Meta:
        verbose_name = 'Продукт'
        verbose_name_plural = 'Продукти'

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

Рисунок 4.1.9 - Моделі продуктів і категорій

За схожою схемою прописуються всі моделі з планування табличок (рис. 4.1.9), прописуються деякі методи в цих класах, для співпраці між ними, як, наприклад підрахунок суми в моделі кошика (рис. 4.1.10).

```python
class Basket(models.Model):
    user = models.ForeignKey(to=User, on_delete=models.CASCADE)
    product = models.ForeignKey(to=Product, on_delete=models.CASCADE)
    quantity = models.PositiveSmallIntegerField(default=0)
    created_timestamp = models.DateTimeField(auto_now_add=True)

    # objects = BasketQuerySet.as_manager()

    def __str__(self):
        return f'Кошик для {self.user.username} | Продукт: {self.product.name}'

    def sum(self):
        return self.product.price * self.quantity
```

Рисунок 4.1.10 - Модель кошика

Після чого потрібно застосувати міграції, щоб по написаному моделям, були створені ці самі міграції, і таблички в базі даних були зроблені. Але до цього пояснення, для того, щоб використовувати не стандартну базу даних від Джанго, потрібно в налаштуваннях увімкнути потрібну для нас базу даних, у випадку Postgresql (рис. 4.1.11), але на початку використовувалася стандартна база даних, після чого був здійснений перехід на потрібну.

```python
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql_psycopg2",
        "NAME": "store_db",
        "USER": "store_username",
        "PASSWORD": "store_password",
        "HOST": "localhost",
        "PORT": "5434",
    }
}
```

Рисунок 4.1.11 - Налаштування бази данних

В налаштуваннях прописуються параметри бази даних, і вона зв’язується з проєктом, також важливо в самому Postgresql налаштувати права доступу за допомогою команд, щоб можна було працювати з Джанго [10].

Оскільки далі не буде описано перехід на PostgreSQL, то на даному етапі здійснюється необхідна підготовка для цього переходу. Для того, щоб потім ефективно і швидко перейти на іншу БД без втрати даних, потрібно створити фікстури, після заповнення цієї БД. То який алгоритм дій? [11].

Зробивши міграції python manage.py makemigrations. Застосовуємо міграції python manage.py migrate (рис. 4.1.12).

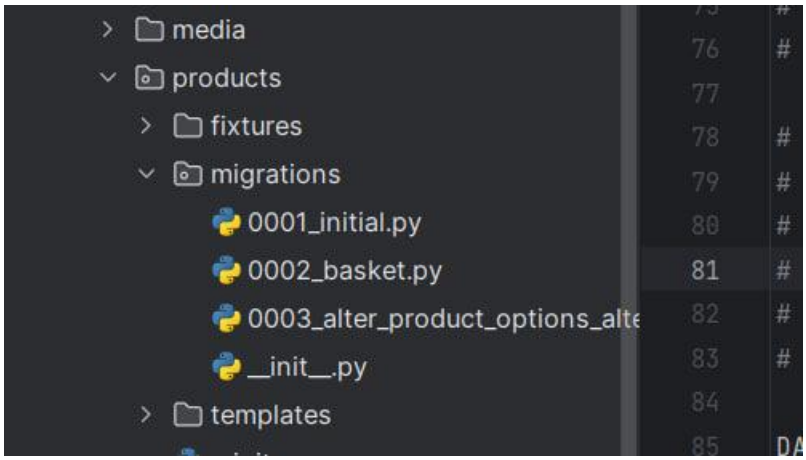


Рисунок 4.1.12 - Директорія міграцій

Міграції створюються і в них ми бачимо наступне

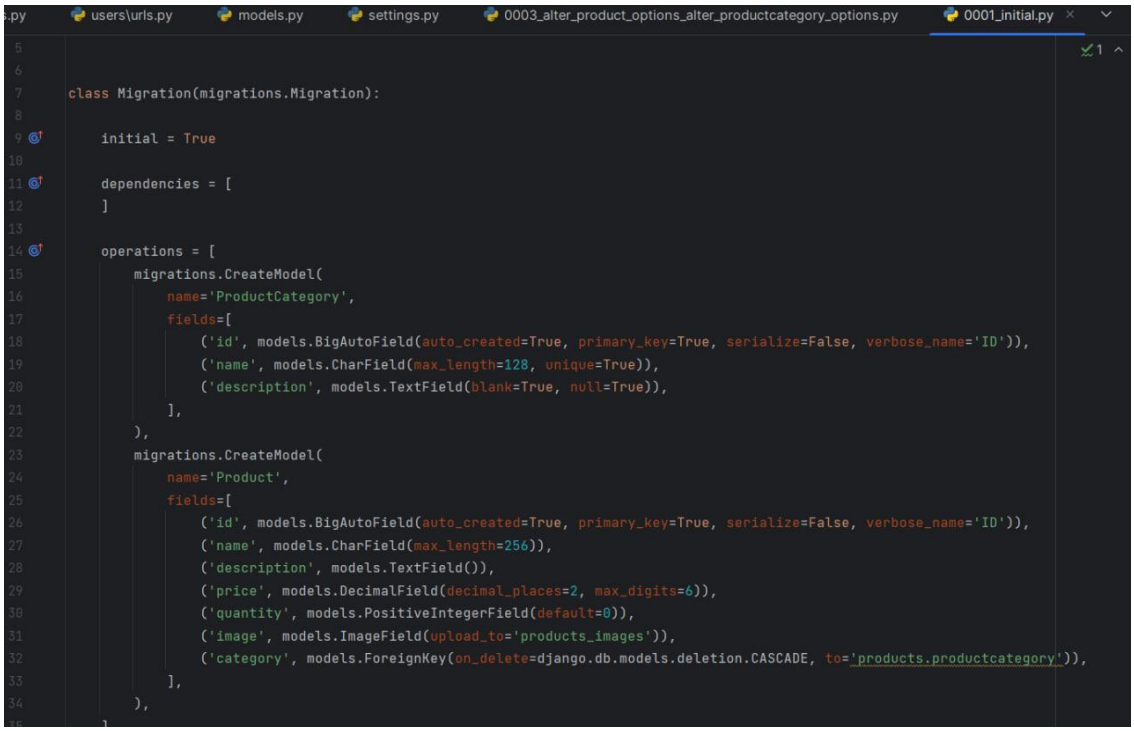


Рисунок 4.1.13 - Міграції

Тобто все спрацювало коректно і таблиці БД створені (рис. 4.1.13).

Після цього займаємося заповненням БД, її можна заповнити або вручну, або за допомогою скрипта, загалом не важливо, але якщо даних багато, то тут тільки за допомогою скрипта, можливо деякого парсера.

Після заповнення БД ми створюємо фікстури (рис. 4.1.14) щоб в майбутньому не втратити дані, з нашої БД.

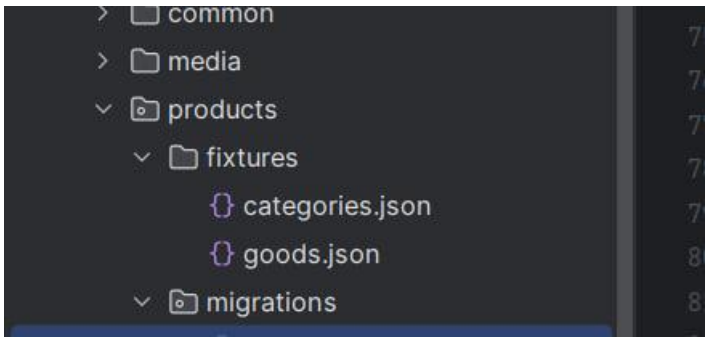


Рисунок 4.1.14 - Директорія фікстур

Створюється в JSON форматі за допомогою команди python manage.py dumpdata products.Product --output=goods.json

І дані які знаходяться в табличці продуктів записуються в JSON файл і зберігаються в ньому.

Потім щоб застосувати ці дані до нової БД python manage.py loaddata goods.json І нова БД заповниться даними.

Після цього можна створити форми які будуть взаємодіяти з моделями, і шаблонами, для коректної співпраці (рис. 4.1.15).

```
2 usages
class UserProfileForm(UserChangeForm):
    first_name = forms.CharField(widget=forms.TextInput(attrs={'class': 'form-control py-4',}))
    last_name = forms.CharField(widget=forms.TextInput(attrs={'class': 'form-control py-4',}))
    image = forms.ImageField(widget=forms.FileInput(attrs={'class': 'custom-file-input'}), required=False)
    username = forms.CharField(widget=forms.TextInput(attrs={'class': 'form-control py-4', 'readonly': True}))
    email = forms.CharField(widget=forms.EmailInput(attrs={'class': 'form-control py-4', 'readonly': True}))

    class Meta:
        model = User
        fields = ('first_name', 'last_name', 'image', 'username', 'email')
```

Рисунок 4.1.15 - Форма профілю користувача

Тут вказуються поля форми, також модель до якої вона прив’язана, після чого застосовуємо її в класах контролерах.

Далі потрібно почати писати шляхи, і контролери (рис. 4.1.16), які і будуть важливим шляхом до запуску серверу.

```
2 usages
class UserProfileView(UpdateView):
    model = User
    form_class = UserProfileForm
    template_name = 'users/profile.html'

    def get_success_url(self):
        return reverse_lazy('users:profile', args=(self.object.id,))

    def get_context_data(self, **kwargs):
        context = super(UserProfileView, self).get_context_data()
        context['title'] = 'Store - Профіль'
        # context['baskets'] = Basket.objects.filter(user=self.request.user)
        baskets = Basket.objects.filter(user=self.request.user)
        context['total_sum'] = sum(basket.sum() for basket in baskets)
        context['total_quantity'] = sum(basket.quantity for basket in baskets)
        return context
```

Рисунок 4.1.16 - Контроллер профілю користувача

Приклад контролеру профілю користувача, в ньому ми вказуємо модель з якою працює контролер, форму, шаблон де прописаний html, метод get_success_url показує шлях на який буде перенаправлено користувача після збереження форми методу POST, та метод контексту вказує по якім ключам в Html ми можемо звертатися до даних, наприклад взяти всі дані з таблички кошика, по користувачу який зараз на сторінці і відобразити всі товари які він додав в кошик, дістанемо це по ключу baskets тоді код в html буде виглядати як:

For basket In baskets:

basket.product.name

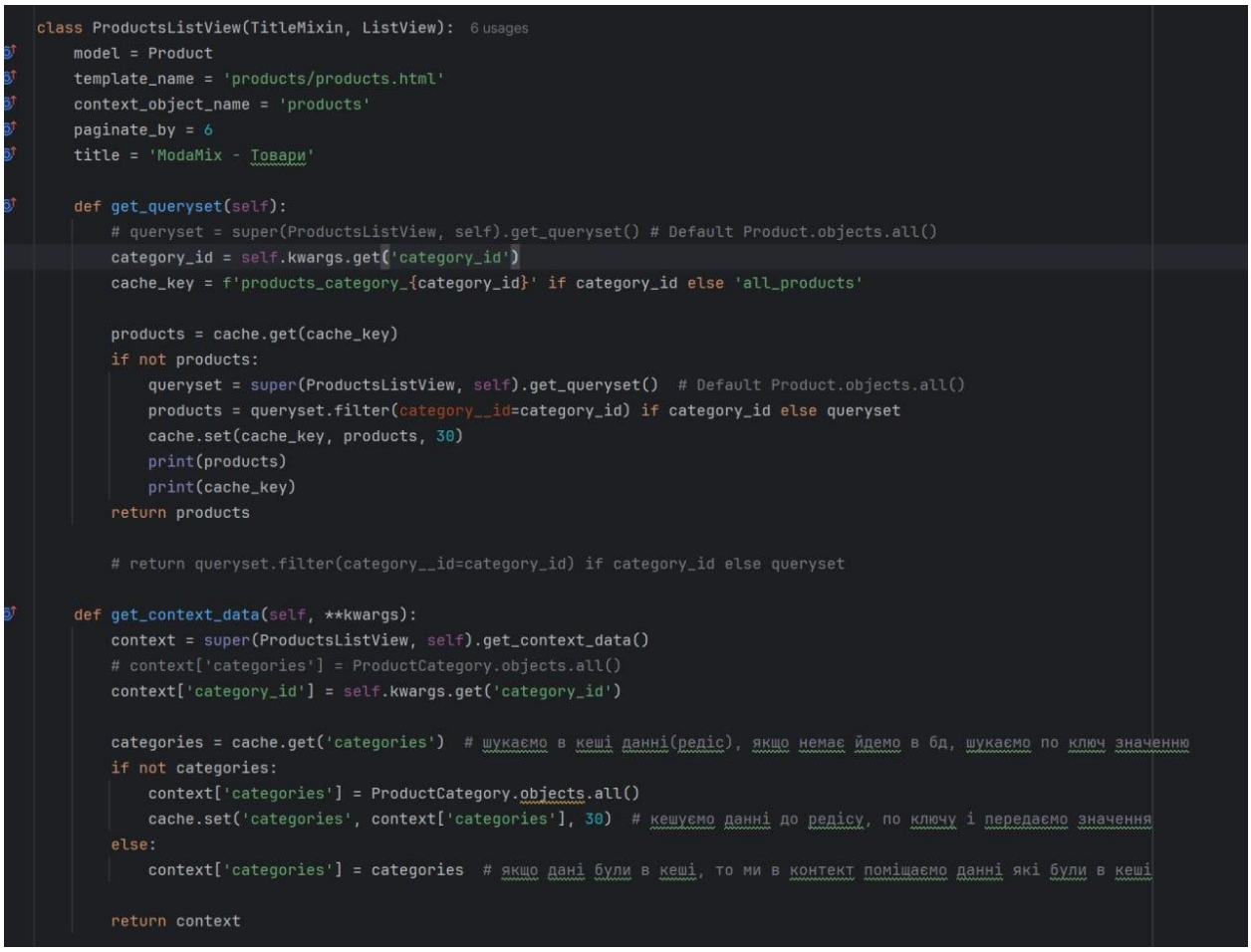
Basket.quantity

І так далі, ми будемо перебирати всі товари які є в цьому списку, так само з сумою і кількістю товарів, як вказано на знімку з екрана.

Шлях в urls.py буде виглядати так path('profile/<int:pk>/', login_required(UserProfileView.as_view()), name='profile'), звертатиметься до контролера, буде виконуватися функціонал, після чого відображатися потрібний результат користувачу.

Прописуються всі контролери, шаблони, шляхи, моделі, форми, фікстури в проєкті, створюючи при цьому і реєстрацію, і авторизацію, і каталог товарів, і категорії сортування товарів, користувачів. Прописавши весь функціонал сайту, його потрібно оптимізувати і пришвидшувати, з цим допоможе Redis і Celery [12].

Спочатку розберемося з Redis (рис. 4.1.17).



```python
class ProductsListView(TitleMixin, ListView):  6 usages
    model = Product
    template_name = 'products/products.html'
    context_object_name = 'products'
    paginate_by = 6
    title = 'ModaMix - Товари'

    def get_queryset(self):
        # queryset = super(ProductsListView, self).get_queryset() # Default Product.objects.all()
        category_id = self.kwargs.get('category_id')
        cache_key = f'products_category_{category_id}' if category_id else 'all_products'

        products = cache.get(cache_key)
        if not products:
            queryset = super(ProductsListView, self).get_queryset()  # Default Product.objects.all()
            products = queryset.filter(category__id=category_id) if category_id else queryset
            cache.set(cache_key, products, 30)
            print(products)
            print(cache_key)
        return products

        # return queryset.filter(category__id=category_id) if category_id else queryset

    def get_context_data(self, **kwargs):
        context = super(ProductsListView, self).get_context_data()
        # context['categories'] = ProductCategory.objects.all()
        context['category_id'] = self.kwargs.get('category_id')

        categories = cache.get('categories')  # шукаємо в кеші данні(редіс), якщо немає йдемо в бд, шукаємо по ключ значенню
        if not categories:
            context['categories'] = ProductCategory.objects.all()
            cache.set('categories', context['categories'], 30)  # кешуємо данні до редісу, по ключу і передаємо значення
        else:
            context['categories'] = categories  # якщо дані були в кеші, то ми в контекст поміщаємо данні які були в кеші

        return context
```

Рисунок 4.1.17 - Контроллер списку продуктів

Використовуючи ListView цей клас має свій queryset пов’язаний з моделлю, потім переозначується метод, і сортується потрібним його чином, для того, щоб працювати з кешем, тобто щоб сторінка завантажувалася швидше, бо буде звертатися до редіс. Перевіряється наявність даних в Редіс, якщо їх немає, йдемо в БД, беруться звідти потрібні дані, записуються в редіс, на деякий час їх зберігаємо в ньому, і наступний користувач, або той самий якщо оновить сторінку, далі дані будуть братися не з БД, а з Редісу, що набагато швидше. Для цього потрібно встановити Redis і налаштувати його зв’язок між проєктом і Redis, для співпраці [7].

Перейдемо до Celery, навіщо він в проєкті, оскільки при реєстрації на сайті, відправляється повідомлення поштою, далі користувач на пошті переходить за посиланням, на що в нього є два дні, якщо він не підтвердить свій акаунт, ці дані будуть видалені, тому уявимо що користувач перейшов за

посиланням, і підтвердив свій акаунт, це деякий час. При заповненні даних на реєстрацію, і при натисненні кнопки «створити акаунт», формується повідомлення на пошту, і відправляється на вказану пошту, це приблизно дві секунди, цілі дві секунди користувач буде чекати завантаження сторінки, а якщо таких користувачів буде декілька? Тоді потрібно буде ще довше чекати, в цьому випадку я використаю Celery. Для того, щоб завдання виконалося в фоні, а користувачу сторінка завантажилася швидко (рис. 4.1.18).

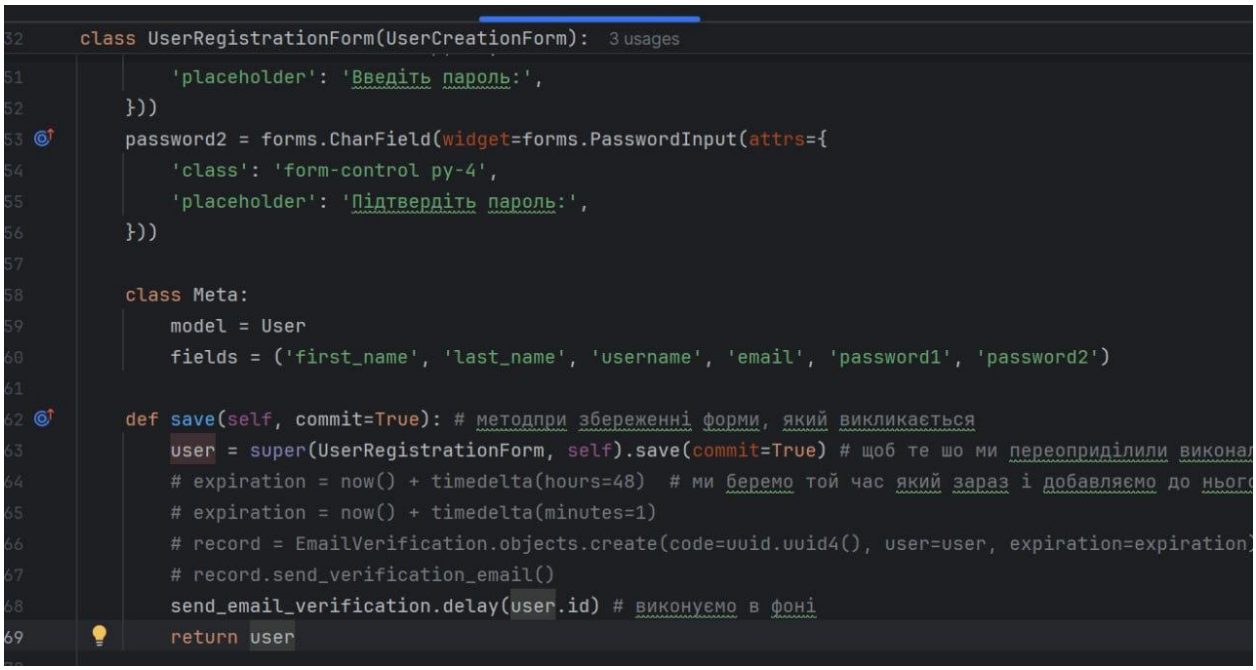


Рисунок 4.1.18 - Форма реєстрації

Тут змінено метод збереження форми, під свої потреби, щоб надсилати листа, код в коментарях, це як він виглядав до Celery [12].

Для Celery ми створюємо завдання, в які записуємо наступне (рис. 4.1.19).

```python
from celery import shared_task
from users.models import User
from datetime import timedelta
from django.utils.timezone import now
from .models import User, EmailVerification
import uuid


@shared_task  2 usages
def send_email_verification(user_id):
    user = User.objects.get(id=user_id)
    expiration = now() + timedelta(minutes=1)
    record = EmailVerification.objects.create(code=uuid.uuid4(), user=user, expiration=expiration)
    record.send_verification_email()
```

Рисунок 4.1.19 - Список завдань для Celery

Тобто просто робиться перенесення того коду в коментарях, в завдання які будуть виконуватися в фоні, і в коді прописуємо назву методу send_email_verification і викликаємо його, тепер користувач побачить швидко відображення сторінки, і повідомлення надійде користувачу на пошту (рис. 4.1.20).

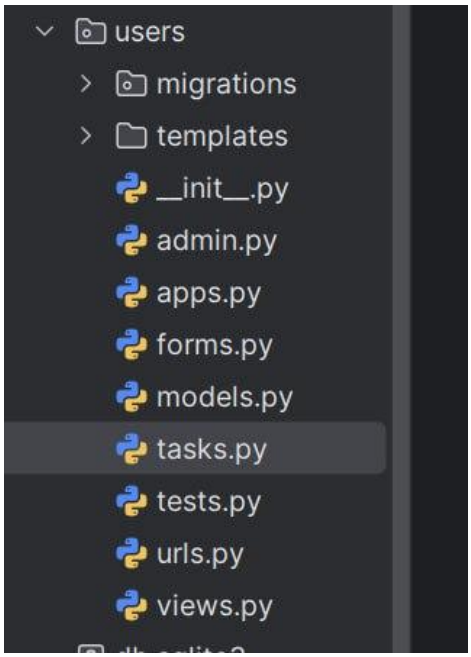


Рисунок 4.1.20 - Завдання Celery

Для налаштування Celery, його також потрібно завантажити, налаштувати, і запустити, для коректної роботи з ним [12].

Тепер перейдемо до Stripe, його також потрібно увімкнути, і налаштувати вебхук, для платежів через цей сервіс по api, налаштування саме цього сервісу потребує більше роботи, ніж все інше в проєкті, тут потрібно зареєструватися на їх сайті, створити акаунт, і увімкнути підприємство, оскільки підприємства немає, буде описаний тільки тестовий варіант оплати, Stripe надає можливість працювати з тестовими транзакціями. Після створення акаунту, потрібно взяти публічний ключ, і секретний, увімкнути в проєкт, і завантажити Stripe, коли це готово, можна підключати його в проєкт, через функціонал оплати товарів (рис. 4.1.21).

```python
class OrderCreateView(TitleMixin, CreateView):  4 usages
    template_name = 'orders/order-create.html'
    form_class = OrderForm
    success_url = reverse_lazy('orders:order_create')
    title = 'ModaMix - Оформлення заказа'

    def get_context_data(self, **kwargs):
        context = super(OrderCreateView, self).get_context_data()
        baskets = Basket.objects.filter(user=self.request.user)
        context['baskets'] = baskets
        context['total_sum'] = sum(basket.sum() for basket in baskets)
        context['total_quantity'] = sum(basket.quantity for basket in baskets)
        return context

    def post(self, request, *args, **kwargs):
        super(OrderCreateView, self).post(request, *args: *args, **kwargs)
        baskets = Basket.objects.filter(user=self.request.user)
        line_items = []
        for basket in baskets:
            item = {
                'price': basket.product.stripe_product_price_id,
                'quantity': basket.quantity,
            }
            line_items.append(item)

        checkout_session = stripe.checkout.Session.create(
            line_items=line_items,
            metadata={'order_id': self.object.id}, # передаємо для  вебхука айді товарів
            mode='payment',
            success_url='{}{}'.format( *args: settings.DOMAIN_NAME, reverse('orders:order_success')),
            cancel_url='{}{}'.format( *args: settings.DOMAIN_NAME, reverse('orders:order_canceled')),
        )
        return HttpResponseRedirect(checkout_session.url, status=HTTPStatus.SEE_OTHER)

    def form_valid(self, form):
        form.instance.initiator = self.request.user # обовязкове поле для заповнення в моделі, звязок з юзером
        return super(OrderCreateView, self).form_valid(form)
```

Рисунок 4.1.21 - Клас створення замовлення

Змінюється метод POST в створенні замовлення, де і прописується робота зі Stripe, тут передається кошик товарів користувача, підключаючи

туди кількість і ключ продукту, формується це в список і передається далі сесії, в сесії приймаються й обробляються дані, і залежно від успіху чи провалу транзакції, код працює далі.

```python
@csrf_exempt  2 usages
def stripe_webhook_view(request):
    payload = request.body # сюди прийде відповідь від страйпа
    sig_header = request.META['HTTP_STRIPE_SIGNATURE']
    event = None

    try:
        event = stripe.Webhook.construct_event(
        payload, sig_header, settings.STRIPE_WEBHOOK_SECRET
    )
    except ValueError as e:
        # Invalid payload
        return HttpResponse(status=400)
    except stripe.error.SignatureVerificationError as e:
        # Invalid signature
        return HttpResponse(status=400)

    if (
        event['type'] == 'checkout.session.completed'
        or event['type'] == 'checkout.session.async_payment_succeeded'
    ):
        fulfill_checkout(event['data']['object']['id'])

    return HttpResponse(status=200)


def fulfill_checkout(session):  1 usage
    order_id = int(session.metadata.order_id)
    order = Order.objects.get(id=order_id) # беремо обьекти корзини
    order.update_after_payment() # виконуємо функцію після успішної оплати
    print('Fullfill')
```

Рисунок 4.1.22 - Вебхук

Тут налаштовується вебхук (рис. 4.1.22), який обробляє транзакцію, і якщо вона успішна викликає функцію fulfill_checkout яка своєю чергою викликає метод над замовленням оновлення після покупки, для очищення кошика товарів користувача, і для зміни стану замовлення, видалення даних з

БД пов’язаних з кошиком, і формування JSON файлу для подальшої роботи з ним, щоб виводити користувачам інформацію про замовлення (рис. 4.1.23).

```python
class Order(models.Model):  8 usages
    CREATED = 0
    PAID = 1
    ON_WAY = 2
    DELIVERED = 3
    STATUSES = (
        (CREATED, 'Створений'),
        (PAID, 'Сплачено'),
        (ON_WAY, 'В дорозі'),
        (DELIVERED, 'Доставлено'),
    )

    first_name = models.CharField(max_length=64)
    last_name = models.CharField(max_length=64)
    email = models.EmailField(max_length=256)
    address = models.CharField(max_length=256)
    basket_history = models.JSONField(default=dict)  # тут будемо хранить словарь з інформаціє
    created = models.DateTimeField(auto_now_add=True)
    status = models.SmallIntegerField(default=CREATED, choices=STATUSES)
    initiator = models.ForeignKey(to=User, on_delete=models.CASCADE)

    def __str__(self):
        return f'Order #{self.id}. {self.first_name} {self.last_name}'

    def update_after_payment(self):  1 usage (1 dynamic)
        baskets = Basket.objects.filter(user=self.initiator)
        self.status = self.PAID
        self.basket_history = {
            'purchased_items': [basket.de_json() for basket in baskets],
            # кожен обьект матиме список з обьектів ті шо приходять з функції
            'total_sum': float(sum(basket.sum() for basket in baskets)),
        }
        baskets.delete()
        self.save()
```

Рисунок 4.1.23 - Модель замовлення

Далі створюється логіка пошуку з ElasticSearch (рис. 4.1.24), приймається запит з форми, якщо його немає, відпрацьовує стандартна логіка з Redis, якщо дані з запиту є, то надсилаються на пошук через клас документа для створення queryset (рис 4.1.25).

```python
def get_queryset(self):
    # queryset = super(ProductsListView, self).get_queryset() # Default Product.objects.all()
    category_id = self.kwargs.get('category_id')
    query = self.request.GET.get('query', None)
    cache_key = f'products_category_{category_id}' if category_id else 'all_products'

    products = cache.get(cache_key)
    if not products:
        if query:
            # Perform search using Elasticsearch
            products = ProductDocument.search().query("match", name=query).to_queryset()
        else:
            queryset = super(ProductsListView, self).get_queryset()  # Default Product.objects.all()
            products = queryset.filter(category__id=category_id) if category_id else queryset
            cache.set(cache_key, products, 30)
    return products
```

Рисунок 4.1.24 - Метод get_queryset

```python
from django_elasticsearch_dsl import Document
from django_elasticsearch_dsl.registries import registry

from .models import Product

@registry.register_document  2 usages
class ProductDocument(Document):
    class Index:
        name = 'products'
        settings = {'number_of_shards': 1, 'number_of_replicas': 0}

    class Django:
        model = Product
        fields = ['name', 'description']
```

Рисунок 4.1.25 - Документ elasticsearch

4.2. Тестування та налагодження

Для тестування сайту, використовується unittests вони перевіряють окремі частини коду, на правильність результату.

1. Основні етапи тестування:
    - Тестування моделей: Перевірка коректного збереження та обробки даних у базі.
    - Тестування подань: Перевірка, що подання правильно обробляють HTTP-запити і повертають відповідні відповіді.
    - Тестування форм: Перевірка валідації даних, що вводяться користувачем (наприклад, реєстрація або зміна пароля).

2. Тестування Redis і Celery

Для підвищення продуктивності в проєкті використовуються Redis для кешування та Celery для обробки фонових задач.

- Тестування Redis: Перевірка коректності кешування даних, наприклад, збереження нових товарів у кеш або перевірка актуальності кешованих даних.
- Тестування Celery: Використовується для перевірки роботи фонових задач, таких як надсилання електронних листів або оновлення категорії новинок. Необхідно переконатися, що завдання ставляться у чергу і виконуються вчасно.

3. Налагодження та логування

- Логування: Django підтримує вбудовану систему логування, яка дозволяє записувати події у файл. Це допомагає відстежувати проблеми під час роботи сайту.

4. Тестування продуктивності

Для забезпечення високої продуктивності необхідно протестувати навантаження на сайт, особливо для кешу та черг. Це дозволяє визначити можливі вузькі місця і покращити роботу сайту під високим навантаженням.

- Тестування часу відповіді сервера: Використовуються інструменти, такі як Apache JMeter або Locust, для симуляції одночасних запитів і вимірювання часу відповіді сервера.
- Тестування масштабованості: Перевіряється, як Redis і Celery працюють при збільшенні кількості запитів і завдань у черзі.

ВИСНОВКИ

Розробка вебмагазину на Django: У результаті виконання дипломної роботи було успішно розроблено функціональний вебмагазин з використанням фреймворку Django. Реалізовано всі основні функції, включаючи систему реєстрації користувачів, обробку платежів через Stripe, профілі користувачів з кошиком товарів, відгуки до товарів та інші ключові елементи.

Використання Redis та Celery для оптимізації: В процесі розробки було використано Redis для кешування даних та підвищення продуктивності, а також Celery для асинхронного виконання фонових задач, таких як надсилання електронних листів та оновлення категорій товарів. Це значно покращило продуктивність системи та її здатність до масштабування.

Реалізація пошуку за допомогою Elasticsearch: для полегшення навігації користувачів по сайту було впроваджено систему пошуку з використанням Elasticsearch. Це дало можливість здійснювати швидкий і точний пошук товарів по ключовому слову, що підвищує зручність користування сайтом.

Тестування та налагодження: Було проведено детальне тестування всіх компонентів системи. Застосовано модульні та інтеграційні тести для перевірки функціональності окремих модулів та їхньої взаємодії. Крім того, проведено тестування продуктивності сайту під навантаженням, що дозволило виявити й усунути потенційні вузькі місця.

Практичне значення роботи: Дипломна робота продемонструвала можливість використання сучасних технологій для створення ефективних та продуктивних вебдодатків. Вебмагазин, розроблений у ході проєкту, може бути використаний як основа для реального комерційного продукту з подальшим розширенням функціоналу.

Подальші перспективи розвитку: У подальшій роботі можна реалізувати додаткові функції, такі як система рекомендацій для користувачів, інтеграція з іншими платіжними системами, більш детальне управління замовленнями та аналітика поведінки користувачів на сайті.

Проєкт вебмагазину на Django з використанням Redis, Celery, Elasticsearch та Stripe є успішним прикладом розробки складної системи з використанням сучасних інструментів. Він демонструє важливість поєднання різних технологій для забезпечення продуктивності, масштабованості та зручності для користувачів.

СПИСОК ЛІТЕРАТУРИ

1. Urban Planet Streetwear – Режим доступу: https://urbanplanet-streetwear.com/.

2. Staff Clothes – Режим доступу: https://www.staff-clothes.com/parnyam/.

3. Django Documentation – Режим доступу: https://docs.djangoproject.com/en/5.1/.

4. Django Models – Режим доступу: https://docs.djangoproject.com/en/5.1/topics/db/models/.

5. Django Migrations – Режим доступу: https://docs.djangoproject.com/en/5.1/topics/migrations/.

6. Django Cache – Режим доступу: https://docs.djangoproject.com/en/5.1/topics/cache/.

7. Redis Documentation – Режим доступу: https://redis.io/docs/latest/.

8. Django Pagination – Режим доступу: https://docs.djangoproject.com/en/5.1/topics/pagination/.

9. Django Static Files – Режим доступу: https://docs.djangoproject.com/en/5.1/intro/tutorial06/.

10. PostgreSQL Documentation – Режим доступу: https://www.postgresql.org/docs/.

11. Django Fixtures – Режим доступу: https://docs.djangoproject.com/en/5.1/topics/db/fixtures/.

12. Celery Documentation – Режим доступу: https://docs.celeryq.dev/en/stable/.

13. Django Templates – Режим доступу: https://docs.djangoproject.com/en/5.1/intro/tutorial03/.

14. Bootstrap Introduction – Режим доступу: https://getbootstrap.com/docs/4.1/getting-started/introduction/.

15. Bootstrap Examples – Режим доступу: https://getbootstrap.com/docs/5.3/examples/.

16. Django Admin – Режим доступу: https://docs.djangoproject.com/en/5.1/intro/tutorial07/.

17. Django Installation – Режим доступу: https://docs.djangoproject.com/en/5.1/intro/install/.

18. William S. Vincent. Django for Beginners: Build Websites with Python and Django. 3rd Edition. 2022. URL: https://djangoforbeginners.com.

19. William S. Vincent. Django for APIs: Build Web APIs with Python and Django. 3rd Edition. 2022. URL: https://djangoforapis.com.

20. Harry J.W. Percival. Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript. 2nd Edition. O'Reilly Media. 2017. URL: https://www.obeythetestinggoat.com.