МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система онлайн-магазину з використанням фреймворку Django»

Завідувач

випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Олексієнко Г.А.**

Студента групи ІН-63 **Свістельнік А.О.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020г.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-63 спеціальності "Інформатика" денної форми навчання Свістельніка Артема Олександровича.

Тема: «Інформаційна система онлайн-магазину з використанням фреймворку Django»

Затверджена наказом по СумДУ

№_____ от _____2020 г.

Зміст пояснювальної записки: 1) інформаційний огляд , постановка завдання й формування завдань дослідження; 2) опис основних положень і моделі веб-ресурсу; 3) реалізація веб-ресурсу;

Дата видачі завдання "_____"_____2020 г.

Керівник випускної роботи _____ Олексієнко Г.А.

Завдання приняв до виконання _____ Свістельнік А.О.

РЕФЕРАТ

Записка: 54 стор., 28 рис., 1 додаток, 9 літературних джерел.

Об’єкт дослідження — веб-платформа для торгівлі через власний інтернет-магазин, маркетплейс.

Мета роботи — розробити і програмно реалізувати веб-ресурс, для продажу товару чи послуг через власний інтеренет-магазин.

Методи досліджень — системно-інформаційний аналіз, інформаційне моделювання та комп’ютеринй експеримент

Результати — розроблений веб-ресурс за допомогою HTML/CSS, Python, фреймворку Django. Веб-ресурс реалізований як торговий майданчик (маркетплейс).

Ключові слова — Веб-ресурс, Python, Django , маркетплейс , торговий майданчик.

ЗМІСТ

ВСТУП

В наш час є актуальною тема продажу товару або послуг, а особливо, можливість продажу через мережу інтернет, так як не завжди є можливість сходити в звичайний магазин і придбати необхідний товар чи замовити послугу, як під час всесвітнього карантину в 2020 році. Тому на 1 місце виходить інтернет-торгівля. Також зараз з’являється велика кількість різних за розмірами підприємств, стає питання, яким чином краще продавати свій продукт. Є декілька варінтів для цього. Найчастіше великі підприємства мають ресурси для створення як і офлайн так і онлайн магазинів. На відміну від гігантів індустрії, малим підприємствам часто бракує можливостей на створення чи просування власного інтернет-магазину, або магазину у звичному розумінні. По цій причині в наш час великою популярністю користуються торгові майданчики , які допомагають в цьому.

Саме тому дана робота присвячена розробці веб-ресурсу - торгівельного майданчика для продажу товарів і послуг . Цей веб-ресурс буде вирізнятися серед інших простотою роботи і зрозумілістю як для продавця, так і для клієнта. Оскільки для успішної покупки не потрібний особистий контакт між продавцем і покупцем – цей ресурс можна вважати безпечним і актуальним.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Загальна інформація про торгові майданчики

Торгові майданчики (Маркетплейс) - це веб-ресурс на якому продаються послуги чи товари інших компаній. Аналогом торгового майданчика є великий гіпермаркет, на території якого знаходяться різні продавці, а покупці можуть купувати товар в різних магазинах, але в цей час знаходитися в одній будівлі.

Існує 3 види організаї торгових майданчиків

- Залучаються продавці, але продаж проходить від імені свого бренда (назви торгово майданчика)
- Продаж проходить від імені кожного продавця окремо, але по правилам маркетплейсу
- Змішаний тип, де продаж відбувається як і від імені власного бренду, так і від імені продавців

Маркетплейс виступає в ролі зв’язуючої ланки між клієнтом і продавцем. Його ціль забезпечити продавця покупцями, а покупця необхідними йому товарами. Зазвичай торговий майданчик не займається відправкою товару, а лише приймає замовлення і передає його продацю. Також маркетплейс займається маркетингом і приверненням нових клієнтів, а також відповідає за роботоспособність сайту.

На рисунку 1.1 зображено типову схему роботи маркетплейсу: Клієнт робить замовлення через сайт, маркетплейс його через свою систему відправляє продавцю, після чого продацець займається доставкою товару до покупця. Оплата покупок, може здійснюватися і через сайт, в такому разі часто маркеплейс бере свою комісію і відправляє решту коштів продацю. При оплаті товару при отриманні, розрахунок відбувається за правилами маркетплейсу або за домовленістю сторін

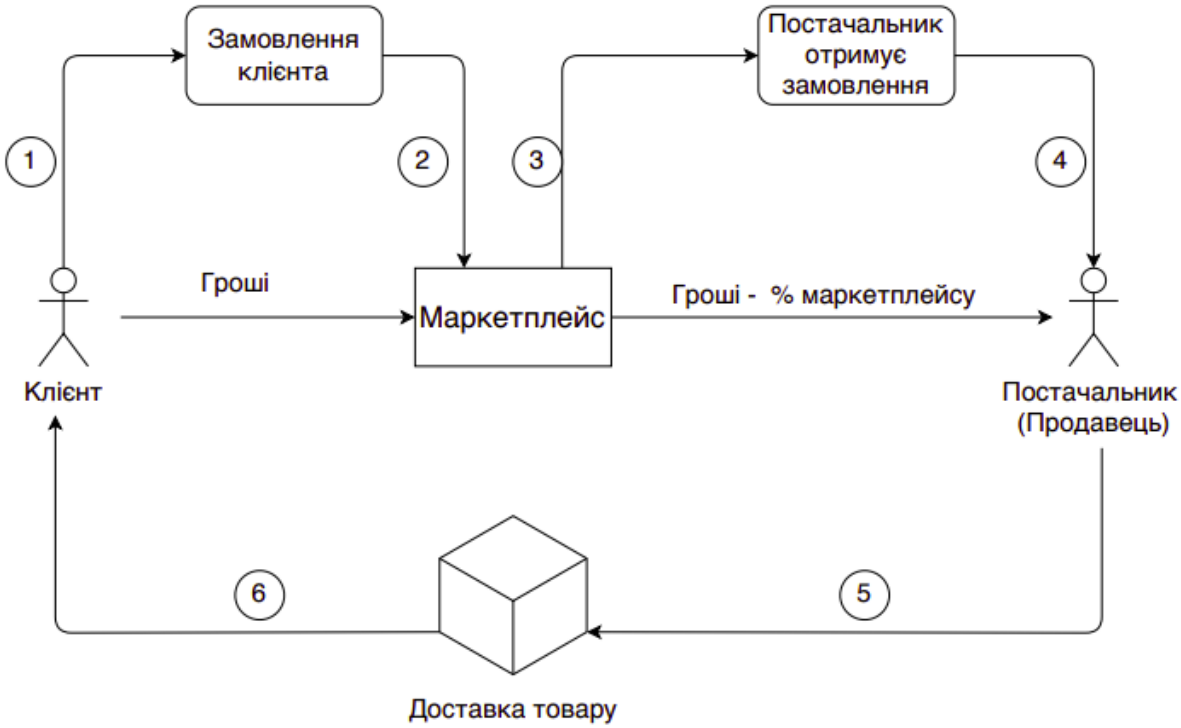

Рисунок 1.1 – Типова схема роботи маркетплейсу

1.2. Огляд існуючих торгових майданчиків

Першим прикладом маркетплейсу є всім відовий торговий майданчик Prom. Він надає можливість продавцю створити бізнес аккаунт і продавати товар на умовах цього ресурсу. Також він має можливість невеликої кастомізації власного онлайн-магазину. Однак сама платформа не продає власний товар.

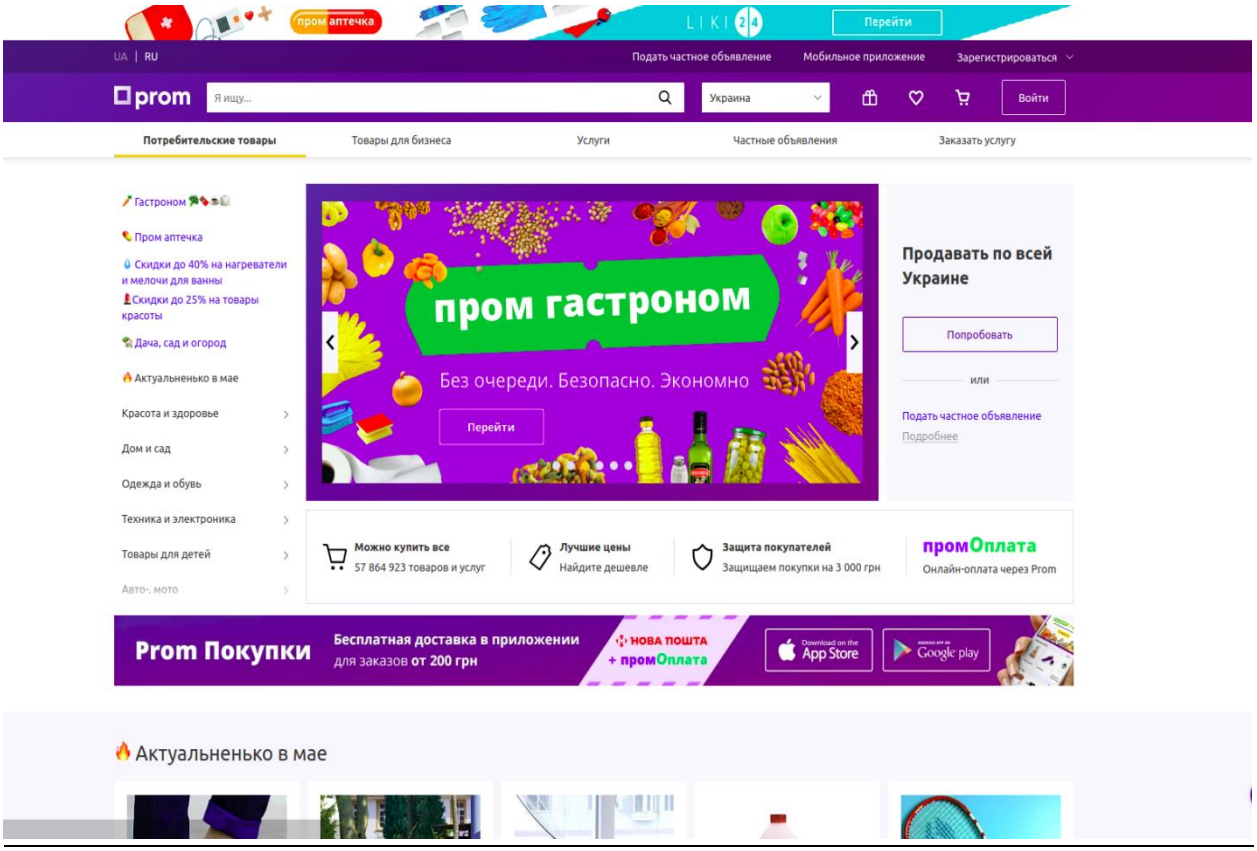

Рисунок 1.2 – Головна сторінка Prom

Другий приклад торгового майданчику є веб-сайт Rozetka. Цей ресурс розпочинав свою роботу як звичайний інтернет магазин і продавав лише власний товар. Але час йде і умови ринку змінюються і тепер на цьому сайті є можливість продажу товару і іншим продацям. З чого можна зробити висновок, що створення свого маркетплейсу є актуальним.

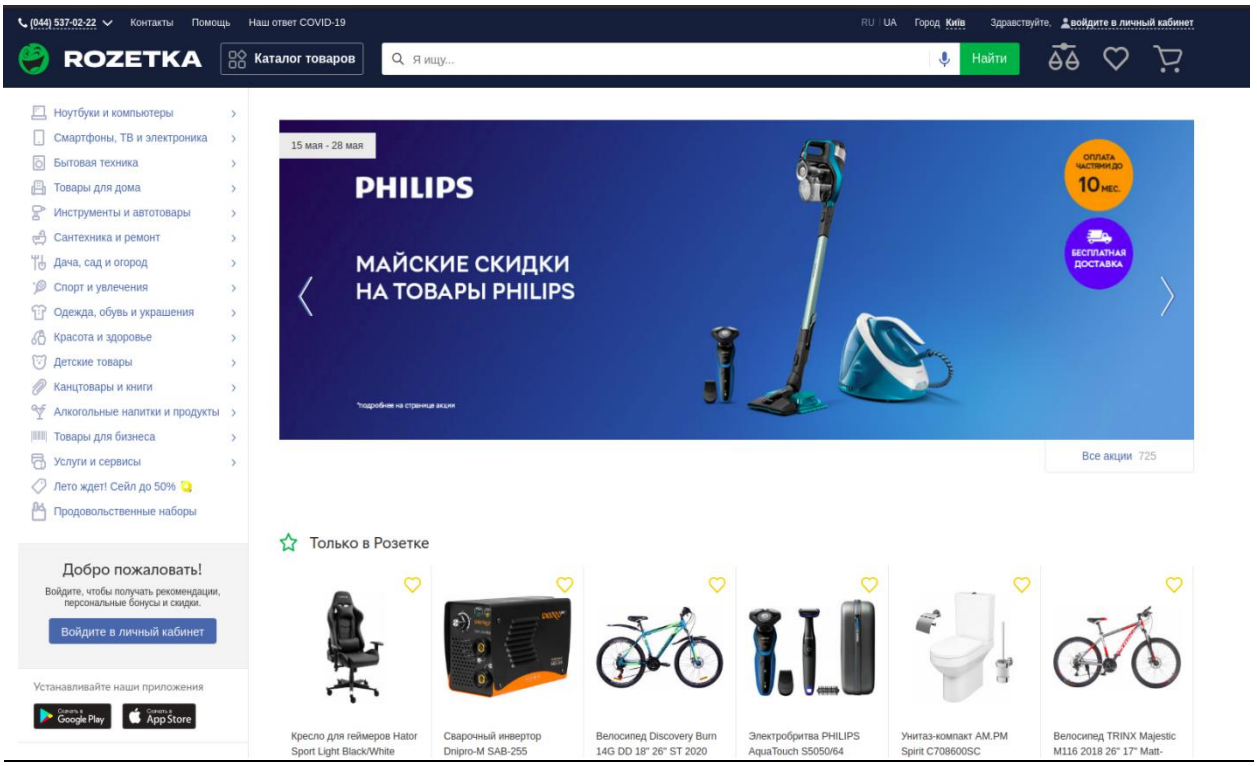

Рисунок 1.3 – Головна сторінка сайту Rozetka

Наступний аналог - це сайт olx. Цей ресурс має можливість створити як бізнес аккаунт так і аккаунт продавця для одного чи декількох товарів. Які в цьому є проблеми. Через те що створити аккаунт просто і продавці ніяк не перевіряються - цей ресурс славиться великою кількістю шахраїв і недобросовісних продавців , що є великим недоліком цього ресурсу

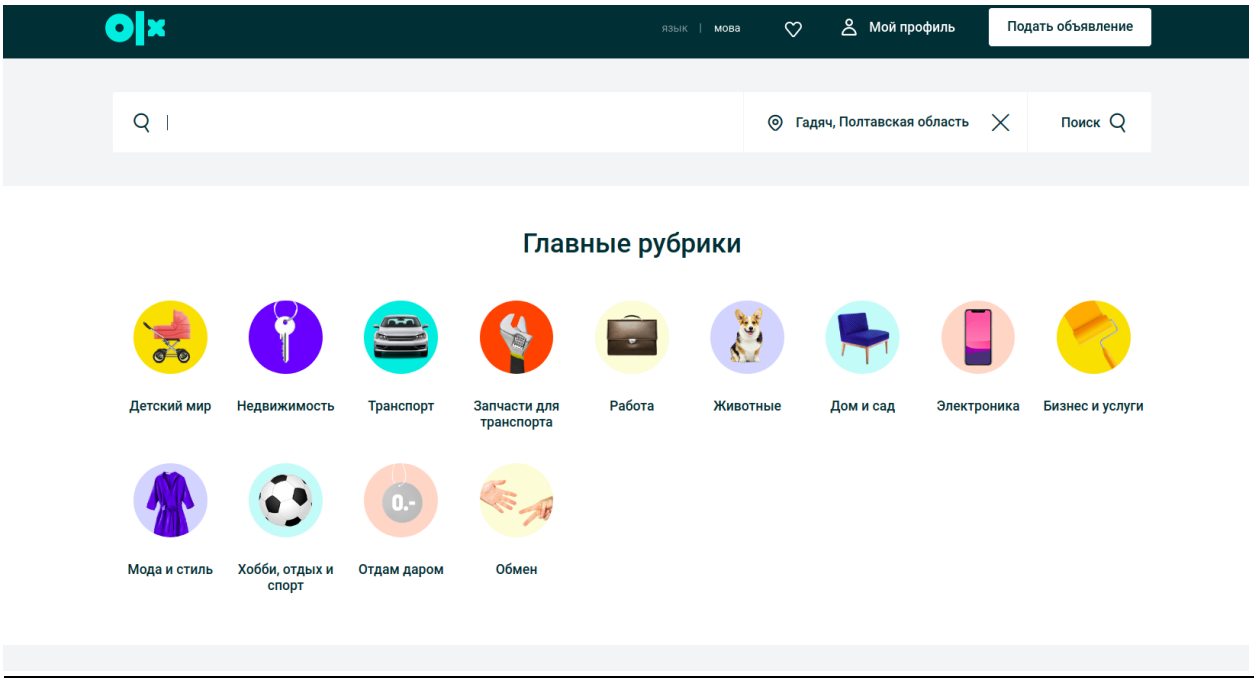

Рисунок 1.4 – Головна сторінка сайту OLX.

1.3. Постановка задачі

Метою роботи є розробка веб-ресурсу (маркетплейс) який допоможе людям, які доклавши мінімум зусиль зможуть продавати свої товари чи послуги через цей сервіс. Така організація торгівлі є взаємовигідною, оскільки сервіс так само зацікавлений в приваблені клієнтів для продавця, як і сам продавець цього товару, оскільки найчастіше прибуток сервіса це відсоток з продажів кожного товару.

Головною ідеєю сайту є максимальне спрощення процедури продажу і покупки товару як для продавців так і для клієнтів. А також можливості вести простий облік замовлень для продавця. Саме тому цей веб-ресурс буде користуватися популярністю

На сайті, для продавця будуть присутні такі розділи, як безпосередньо ”мої товари”- для адміністрування роботи з товарами, також можливість переглядати і обробляти замовлення .

Основні вимоги до веб-ресурсу:

- Для користувача

  1)Можливість пошуку товару за назвою, чи описом, включаючи можливість помилки при введенні;

  2)Перегляд товарів за категоріями, магазином чи пошуком;

  3)Можливість формування корзини покупок, включати товари від різних магазинів;

  4)Великий асортимент пропозицій;

- Для продавця

  1)Легкий і зручний спосіб створення власного магазину;

  2)Зручне адміністрування товарів;

  3)Систему оброблення замовлень;

- Для всіх

  1)Стабільність роботи сайту;

  2)Зручність використання сайту;

2. ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ ЗАВДАННЯ

2.1. Вибір мови програмування

В 2020 році для створення сайту можна використати велику кількість мов програмування. Для написання backend частини сервісу підходять такі популярні мови як: PHP, Python, Java, JavaScript та інші.

Сильні сторони цих мов програумвання :

PHP:

- Орієтацію на web-розробку . Php створювася як мова для створеннння веб-сайтів.
- Кросплатформеність.
- Низький поріг входу.

Java:

- Надійність. Java має статичну типізацію.
- Швидкодія.

JavaScript – найчастіше використовуються у frontend розробці, але за допомогою програмної платформи Node.js, можна використовувати для розробки backend частини. Це зручно тому, що одна можна на одній мові програмування створити весь проект.

Але , для створення торгівельного майданчика був обраний Python.

Чому саме Python? Python - це сучасна, універсальна, високорівнева мова програмування. До переваг Python можна віднести його ефективність і добре сконструйований легко читаємий код. У python простий для читання і написання синтаксис, що легко дозволяє вичити його навіть новачку у програмуванні.

Python був написаний на мові програмування C (сі), що дозволяє створювати доповнення і бібліотеки до нього на самому C. Такий спосіб доповнення використовується у випадках коли необхідна критична швидкодія. Python підтримує багато стилів розробки, до яких входять популярні: ООП (об'єктно-орієнтоване програмування) і функціональне програмування.

Для чого використовують python? Python підходить для розробки будь-яких проектів. Також він підтримується майже усіма платформами. З операційнми системами на базі лінукс python буде встановлений по замовчуванням . Найчастіше python використовують у сферах роботи з Big Data, нейромережами, штучним інтелектом, машинним навчанням і у web-розрозбці . Для веб-розробки часто використовуються такі фреймворки як Flask , Pylons і найпопулярніший серед всіх - Django. Також python часто використовуються для розробки різноманітних парсерів, які збирають різноманітну інформацію з мережі інтернет.

2.2. Вибір фреймворку

Існує велика кількість веб-фреймворків для Python. Їх можна поділити на 2 категорії : Full-stack фреймворки і мікрофреймворки.

Full-stack фреймворки гарні в тому що мають вбудовану по замовчуванням велику кількість функціоналу, часто систему ORM (Object-Relational-Mapping) для спрощення роботи з базами даних і функціоналу. Найпопулярніший фреймворк цієї категорії це Django.

Серед плюсів мікрофреймворків можна відзначити, що вони досить компактні , і нагадують конструктор, де ви можете на ваш смак налаштувати майже все. Вони включають лише найнеобхідніший для роботи функціонал. Найпопулярнішим мікрофреймворком для python є Flask.

Отже вибір був між фремоврками Flask і Django.

Для створення веб-ресурсу був використаний фреймворк Django. Які причини такого вибору ? На відміну від Flask , Django має вбудовані форми, які легко інтегруються з системою ORM і ідмін-панеллю . А також вбудовану ORM, систему аутентифікації і ролей користувачій і панель ідміністрування.

Що таке Django?

Django - це безкоштовний веб-фреймворк с відкритим вихідним кодом, написанний на Python. Веб-фреймворк це набір компонентів, який полегшує розробку будь-яких продуктів web-сайтів і сервісів , мобільних або декстопник додатків.

Коли ви створюєте веб-сайт, вам завжди потрібен набір компонентів таких як: спосіб обробки аутентифікаціїї користувача (реєстрація, авторизація, вихід), панель керування для вашого сайту , форми, можливість завантаження файлів і т.д.

На щастя для нас, інші люди, це помітили, що веб-розробнкик зустрічаються з схожими проблемами при створенні нового сайту, тому вони об’єднались і створили фреймворки. Django - один з них.

Ось 3 головні тези якими розробники характеризують Django:

- Ridiculously fast, що означає дуже швидку роботу
- Reassuringly secure, що позначає його надійність і безпеку
- Exceedingly scalable, що позначає його гнучкіть і легкість масшатубування

Django притримується концепції розробки MVT, що означає Model, View, Template. Ця концепція біль відома як MVC: Model, View, Controller.

Ось короткий опис того, як M, V і T розділені в середовищі Django:

- M - Model (Модель) - відповідає за бізнес-логіку, і роботу з даними. Також model дозволяє створювати, читати, редагувати і видаляти об’єкти в базі даних.
- V - View (Представлення) - є зв’язуючою ланку між Model і Template. View відповідає за обробку HTTP-запитів, а також надає доступ до даних з БД для templates.
- T - Templates (Шаблони) - це файлы з кодом HTML, за домогою яких і відображаються дані.

2.3. Середовище розробки

Інтегроване середовище розробки (IDE) (англ. Integrated Development Environment) - програмний комплекс, який призначений для ефективної розробки різної складності систем. IDE складається з:

- Редактора текста і ресурсів. Текстовий редактор призначений для зручного і швидкого створення і редагування тексту (коду). Часто містить автодоповнення і підказки
- Компілятора. Компілятор - це спеціалізоване пз (програмне забезпечення) яке перетворює програмний текст, який написаний на одній з мов програмування в набір машинних команд, які будуть зрозумілі цій машині.
- Відладчика. Відладчик - це спеціалізована програма вбудована в IDE для пошука помилок в коді програми, чи інших системних помилок.
- Засобів управління проектом. Цей пункт зображує аналог вбудованого файловго менеджера
- Стандартних заготовок, спрощуючик розробку стандартних задач

Як IDE для створення торгового майданчик було обрано PyCharm IDE. Це IDE від компанії Jetbrains, яке підходить для роботи на мові програмування

Python, а також для використання популярних фреймворків таких як Django. PyCharm містить в собі всі пункти вказані вище які допомагають у розробці будь-яких програм чи сервісів.

2.4. Вибір бази даних

База даних – це спеціально розроблене сховище для різних типів даних.

Серед сучасних баз даних можна виділити популярні зараз реляційні SQLite, MySQL, PostgreSQL.

За замовчування у обраному для розробки веб-сервісу фреймовркі Django використовуються за замовчунням SQlite як база даних. Це файлова база даних. Це зручно коли необхідно її перемістити чи замінити. SQlite добре підходить для розробки і тестування простих додатків розрахованих на одного користувача. Але вона не підходить для додатків, якими можуть одночасно користуватися декілька користувачів , як у цьому випадку, коли створюється торговий майданчик. А також не підтримує пошук за триграмами, який запланований у даному веб-ресурсі.

MySQL – це найбільш розповсюджена серверна база даних. MySQL має дуже великий функціонал. До плюсів цієї БД можна віднести простоту в роботі, адже її легко встановити і вона має додатки у вигляді GUI. А також швидкість, адже спрощення деяких стандартів дозволило значно пришвидшити продуктивність. З мінусів цієї БД можна відзначити заложене розробниками обмежнення функціоналу, а також проблеми з надійністю через деякі способи обробки даних.

PostgreSQL це найбільш продвинута база даних, яка орієнтована на відповідність стандартам і масштабованість. Ця база створена на технології Posgres і чудово справляється з обробкою декількох запитів одночасно , що добре підходить для такого веб-сервісу як торговий майданчик.

Саме тому було обрано PostgreSQL як основну базу даних для розробки веб-ресурсу.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Проектування інформаційної системи веб-ресурсу

Перед початком проектування інформаційної системи веб-ресурсу були виконані наступні пункти:

- Підключені необхідні бібліотеки;
- Підключена база даних;

Деякі головні пакети, які були використані і інформація про них:

- Django (версія 3.0.5) - веб-фреймворк на якому вирішено розробляти веб-ресурс;
- django-bootstrap4 - пакет, який дозволяє завдяки шаблонним тегам django використовути bootstrap. Наприклад {% bootstrap_form form %}. Шаблонним тегом у цьому прикладі являється тег bootstrap_form заключений у {%%}. Він додає до форми form у шаблоні необхідні bootstrap стилі.
- django-mptt - спеціальний пакет для роботи з базою даних. Він дозволяє будувати таблицю в виді дерева, коли один об’єкт може бути нащадком іншого.
- django-extensions, graphviz - необхідні для автоматичної побудовати ERD діаграми з існуючої бази даних.
- python-slugify - використовується для генерації поля slug, який необхідний для створення унікального і зрозумілого url.
- psycopg2 - адаптер PostgreSQL для python

На рисунку 3.1 відображені всі поля таблиці Shop. Ця таблиця містить в собі дані про власника (користувача), та інші дані. Поле owner - є зв’язаним методом “один до одного” з таблицею User, яка надається стандартною бібліотекою django. Решта полів моделі:

- Name - типу CharField позначає назву магазину
- Slug - типу SlugField - генерується автоматично для унікально url
- Phone - типу CharField, позначає номер телефону
- Address - типу CharField, позначає адресу
- Logo - типу ImageField, містить посилання на зображення (логотип магазину)
- Created - типу DateTimeField (автоматично додає дані про дату і час створення магазину)

```python
from django.db import models

# Create your models here.
from django.utils import timezone
from django.contrib.auth.models import User
from slugify import slugify
from mptt.models import MPTTModel, TreeForeignKey

class Shop(models.Model):
    owner=models.OneToOneField(User,related_name='shop',on_delete=models.CASCADE,verbose_name='Владелец')
    name=models.CharField(max_length=100,verbose_name='Название')
    slug=models.SlugField(max_length=250,unique_for_date='created')
    phone=models.CharField(max_length=30,verbose_name='Номер телефона')
    address=models.CharField(max_length=30,verbose_name='Адресс')
    logo=models.ImageField(upload_to='shop_logo/',blank=False,verbose_name='Логотип')
    created=models.DateTimeField(default=timezone.now)

    def __str__(self):
        return self.name
    def save(self,  *args, **kwargs):
        self.slug = slugify(self.name)
        return super(Shop, self).save(*args, **kwargs)
```

Рисунок 3.1 – підключені бібліотеки та таблиця Shop

Таблиця Категорія зображена на рисунку 3.2 . Ця таблиця містить в собі дані про всі категорії. Для її створення у вигляді дерева, коли одна категорія може

бути нащадком іншої було використане підключення пакету MPTTModel. Модель має такі поля:

- Name - типу CharField позначає назву магазину
- Slug - типу SlugField - генерується автоматично для унікально url
- Parent - типу TreeForeignKey, наслідується від самого себе, містить інофрмацію про батьківський об’єкт цього ж класу.

```
class Category(MPTTModel):
    name = models.CharField(max_length=64, unique=True)
    parent = TreeForeignKey('self', null=True,db_index=True,
                            blank=True, related_name='children',
                            on_delete=models.CASCADE)
    slug=models.SlugField(max_length=250,db_index=True)

    class MPTTMeta:
        order_insertion_by = ['name']

    def __str__(self):
        return self.name

    def save(self,  *args, **kwargs):
        self.slug = slugify(self.name)
        return super(Category, self).save(*args, **kwargs)
```

Рисунок 3.2 – таблиця Category

На рисунку 3.3 зображено 2 таблиці Product і ProductImages. Модель Product (продукт) містить в собі такі поля:

- Shop - типу ForeignKey, зв’язується як первинний ключ з таблицею Shop, тобто кожен товар належить певному магазину.
- Name - типу CharField позначає назву магазину

- Category - типу ManyToManyField, містить інформацію про категорії до яких належить даний товар. Тобто у товара може бути багато категорій, а до кожної категорії може належати багато товарів.
- Slug - типу SlugField - генерується автоматично для унікально url
- Short_description - поле типу CharField, позначає короткий опис товару
- Description - типу TextField, поле призначане для зберігання повного опису товару (великої кількості тексту) і на сторінкі html відображається як TextArea
- Image - типу ImageField, містить посилання на зображення (головне зображення товару)
- Price - типу PositiveIntegerField, позначає ціну товару і може містити лише цілі числа більше нуля
- Created - типу DateTimeField (автоматично додає дані про дату і час створення магазину)
- Avaibility - полу типу BooleanField, містить інформацію про наявність товару лише True (є) або False (відсутній товар)

Модель ProductImages (галерея зображень) містить в собі такі поля:

- Product - типу ForeignKey, зв’язується як первинний ключ з таблицею Product, тобто кожне зображення належить певному товару.
- Product_image - типу ImageField, містить посилання на зображення (одне зображення для галереї)

Для збереження зображень до галереї був використаний власний метод save_images, який приймає назву файлу, і підставлені дані певного товара, і зберегіє зображення в форматі “gallery_images/id магазину/id товару/назва файлу”

```python
class Product(models.Model):
    shop = models.ForeignKey(Shop, on_delete=models.CASCADE, verbose_name='Магазин')
    product_name = models.CharField(max_length=30, verbose_name='Название товара')
    category = models.ManyToManyField(Category, verbose_name='Категория', blank=True, default=None, null=True)
    slug = models.SlugField(max_length=250, unique_for_date='created', blank=True)
    short_description = models.CharField(max_length=100, verbose_name='Короткое описание')
    description = models.TextField(max_length=1000,verbose_name='Полное описание')
    image = models.ImageField(upload_to='product_images/', blank=False, verbose_name='Изображение')
    price = models.PositiveIntegerField(default=0,verbose_name='Цена грн.')
    created = models.DateTimeField(default=timezone.now)
    availability=models.BooleanField(default=True,verbose_name='Наличие')


    def __str__(self):
        return self.product_name

    def save(self, *args, **kwargs):
        self.slug = slugify(self.product_name)
        return super(Product, self).save(*args, **kwargs)


def save_images(instance,filename):
    shop_id=instance.product.shop.id
    id=instance.product.id
    return 'gallery_images/{}/{}/{}'.format(shop_id,id,filename)


class ProductImages(models.Model):
    product=models.ForeignKey(Product, on_delete=models.CASCADE,default=None)
    product_image = models.ImageField(upload_to=save_images, blank=True, verbose_name='Галерея изображений')
```

Рисунок 3.3 – Таблиці Product (товари) та ProductImages (галерея зображень)

На рисунку 3.4 зображені моделі Order (замовлення) і модель OrderItem (елемент замовлення/товар)

Модель Order містить такі поля:

- First_name - поле типу CharField, містить ім’я замовника
- Last_name - поле типу CharField, містить прізвище замовника
- Phone - типу CharField, позначає номер телефону
- Email - типу EmailField, містить email замовника
- City - поле типу CharField, містить назву Міста для доставки
- Address - поле типу CharField, містить точну адресу для доставки

- Created - типу DateTimeField автоматично додає дані про дату і час створення замовлення
- Updated - типу DateTimeField, автоматично додає дані про дату і час зміни в замолвенні (зміни статусу)
- Status - поле типу CharField, позначає статус замовлення

Також модель Order має метод get_total_cost, який повертає повну суму замовлення

Модель OrderItem містить інформацію про товари які були замовлен і має такі поля:

- Order - типу ForeignKey, зв’язується як первинний ключ з таблицею Order, тобто кожний елемент змовлення належить певному замовленню.
- Product - типу ForeignKey, зв’язується як первинний ключ з таблицею Product, тобто кожний елемент замовлення є певним товаром.
- Price - типу DecemalField, позначає ціну товару і може містити лише числа, з двома знаками після після коми.
- Quantity - типу PositiveIntegerField, позначає кількість товару і може містити лише цілі числа більше нуля, по замовчуння кількість дорівнює 1.

Також модель OrderItem має метод get_cost, який повертає повну суму товара помножену на його кількість.

```python
class Order(models.Model):
    first_name=models.CharField(max_length=50,verbose_name='Имя')
    last_name=models.CharField(max_length=50,verbose_name='Фамилия')
    phone=models.CharField(default=0,max_length=30,verbose_name='Номер телефона',blank=False)
    email=models.EmailField()
    city = models.CharField(max_length=100,verbose_name='Город')
    address=models.CharField(max_length=250,verbose_name='Адрес')
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status=models.BooleanField(default=False,verbose_name='Статус')

    class Meta:
        ordering = ('-created',)

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

    def get_total_cost(self):
        return sum(item.get_cost() for item in self.items.all())

class OrderItem(models.Model):
    order=models.ForeignKey(Order,on_delete=models.CASCADE,related_name='items',verbose_name='Заказ')
    product=models.ForeignKey(Product,on_delete=models.CASCADE,related_name='order_items',verbose_name='Товар')
    price = models.DecimalField(max_digits=10, decimal_places=2,verbose_name='Цена')
    quantity = models.PositiveIntegerField(default=1,verbose_name='Количество')
    def __str__(self):
        return '{}'.format(self.id)

    def get_cost(self):
        return self.price * self.quantity
```

Рисунок 3.4 – Таблиці Order (замовлення) і OrderItem (товар із замовлення)

Для генерації ERD діаграми були використані вказані вище пакети.

Рисунок 3.5 можна умовно розділити на 2 частини по вертикалі. З лівого боку - стандартні моделі django, які були підключені і використані у проеті. В правій частині знаходяться створені автором проетку моделі веб-ресурсу.

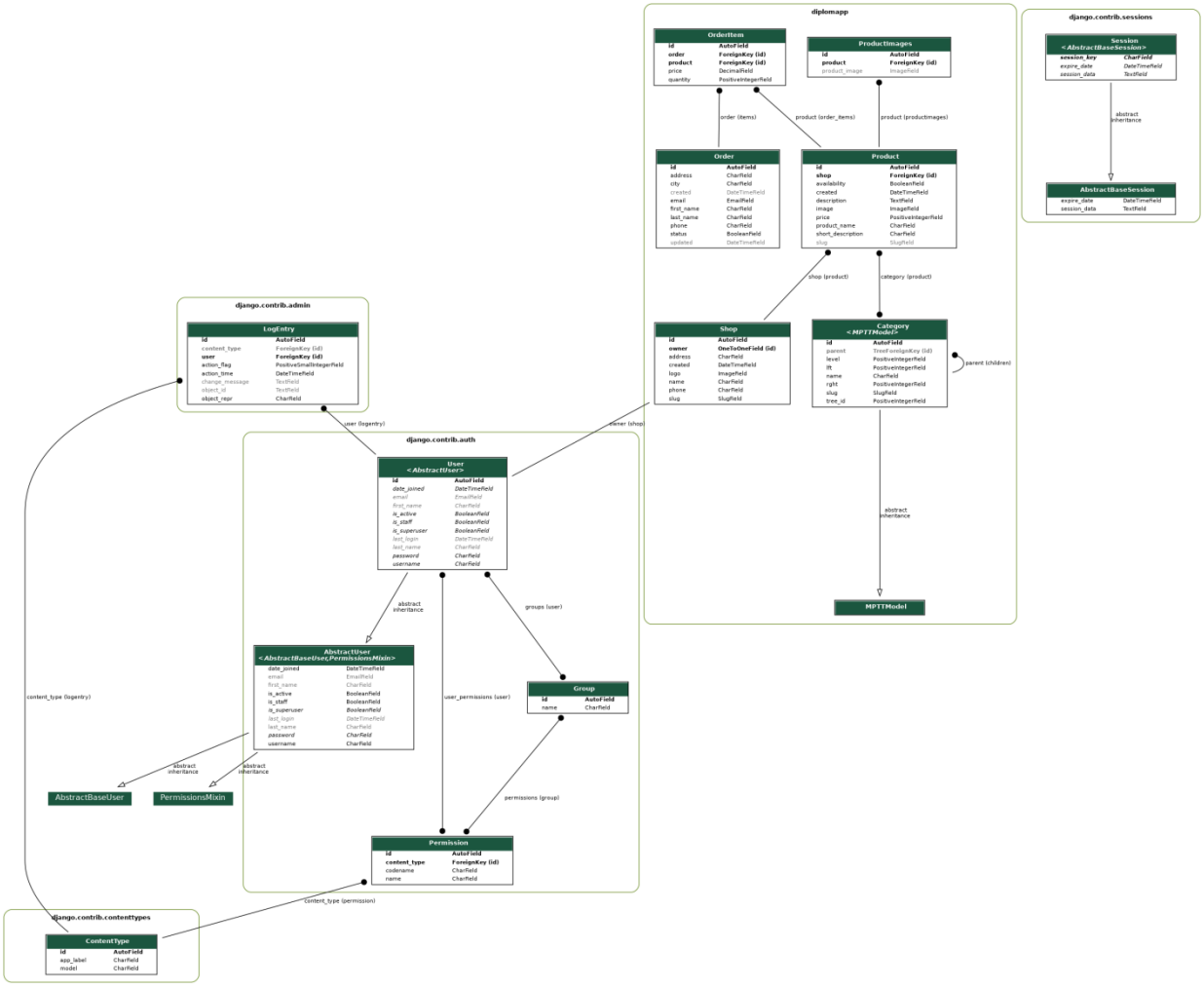

Рисунок 3.5 – Згенерована ERD діаграма

3.2. Опис основного функціоналу торгового майданчика для клієнта

Кожен користувач має можливість без реєстрації переглядати товари по категоріям, по магазинам, заповнювати корзину покупок та робити замовлення.

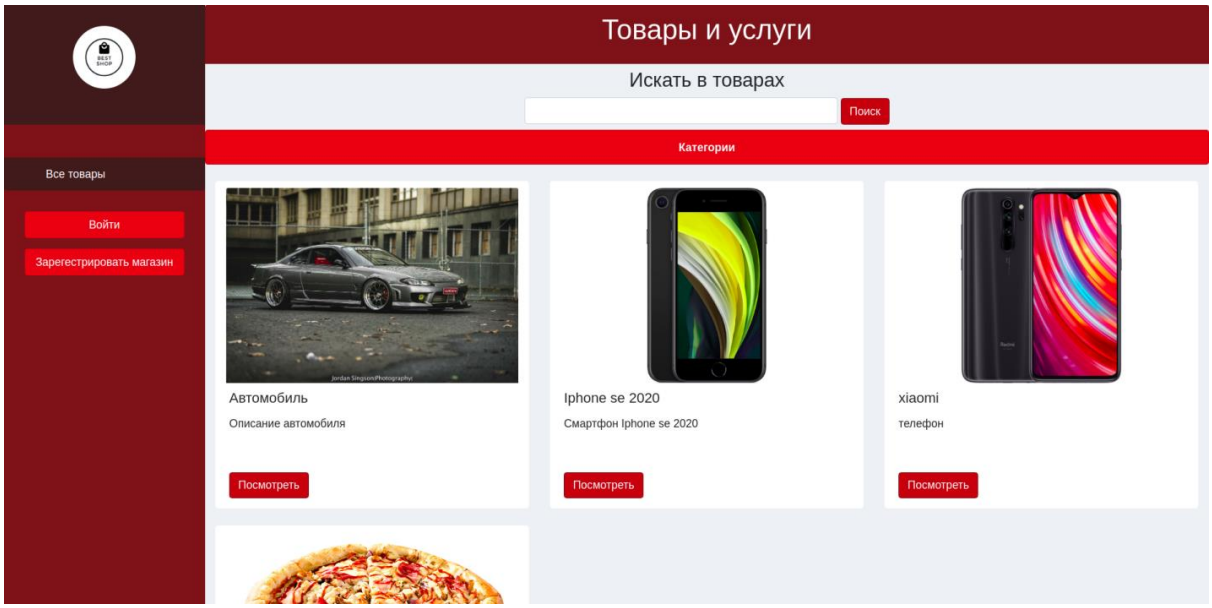

Рисунок 3.6 – Головна сторінка сайту

Клієнт має можливість здійснювати пошук по заданим словам. Система шукає в назві та короткому описі схожі слова і якщо певний налаштований % збігу буде знайдено - то система поверне результат з цим товаром. Як наведено на рисунку 3.7, в слові смартфон допущено помилку, але в результаті користувачу був відображений найбільш ймовірний результат його пошуку. Схожий приклад допущено також на рисунку 3.8 де в слові “Пицца” була пропушена одна літера “ц”.

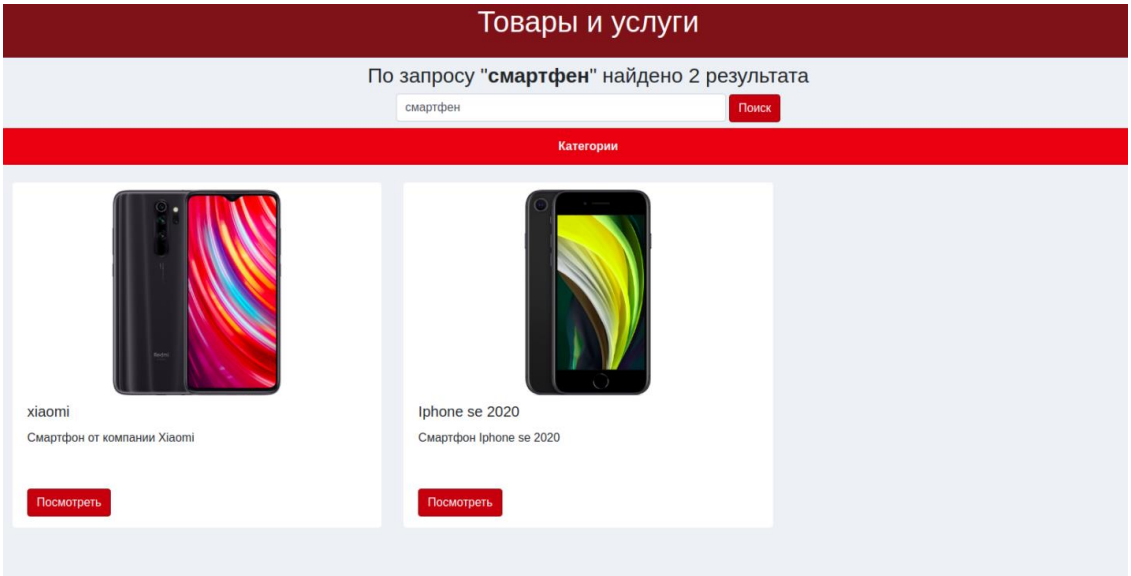

Рисунок 3.7 – Пошук по товарам

Також клієнт має можливість здійснювати пошук по категорії. Для цього потрібно навести на відповідну кнопку, і обрати з випадаючого списку потрібну категорію, чи підкатегорію. Обравши певну категорію у результаті будуть відображені товари, як вибраної категорії, так і всіх дочірніх (ті категорії, які наслідуються від обраної)

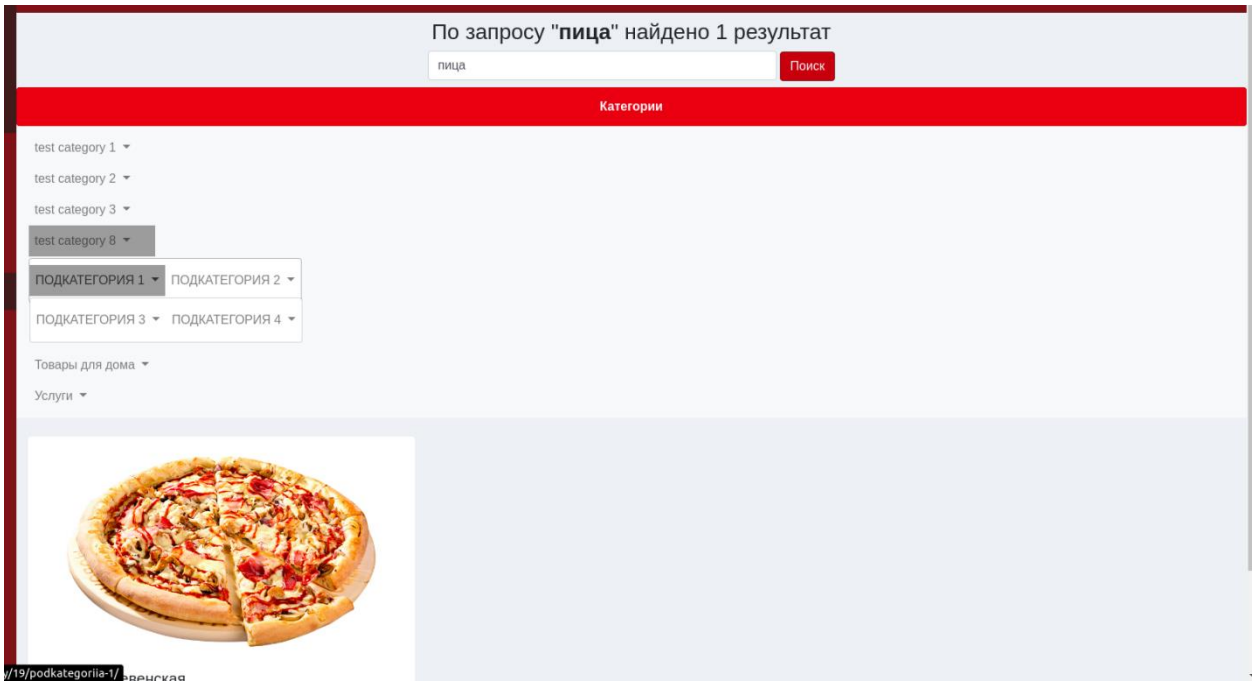

Рисунок 3.8 – Пошук по категорії

Відповідно користувач може проводити пошук за назвою товару, в певній категорії, що покращує точність необхідного для нього результату пошуку.

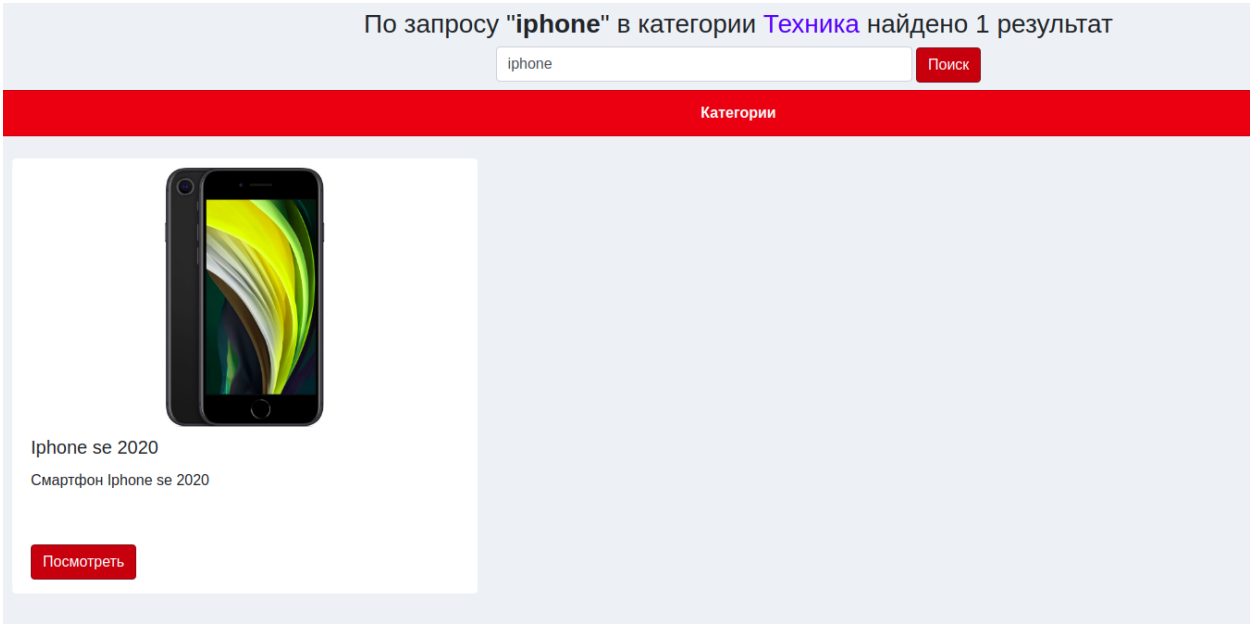

Рисунок 3.9 – Пошук по фразі в категорії

Такі ж дії користувач може проводити в перегляді товарів певного магазину. На картці останнього товару можна відмітити повідомлння про його відсутність, по цій причині він знаходиться у кінці списку товарів .

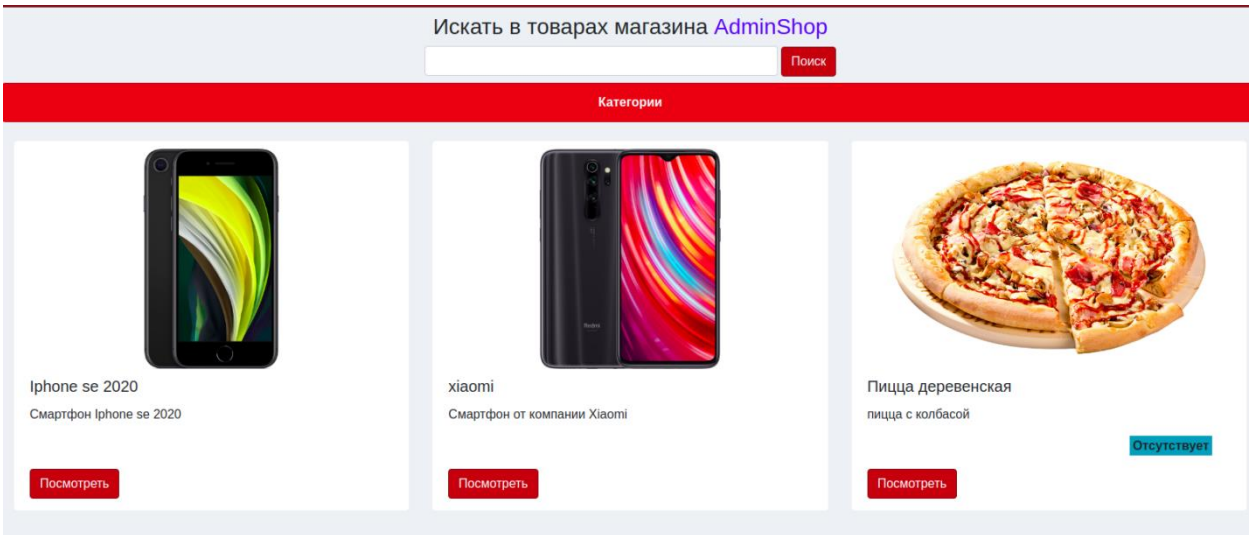

Рисунок 3.10 – Пошук товарів в певному магазині

На кожній картці товару розташована кнопка “Посмотреть”. Вона направляє на сторінку обраного товару для детального перегляду інформації про нього з можливістю перейти до всіх товарів цього продавця. Також користувач має можливість додати товар до корзини у необхідній для нього кількості.

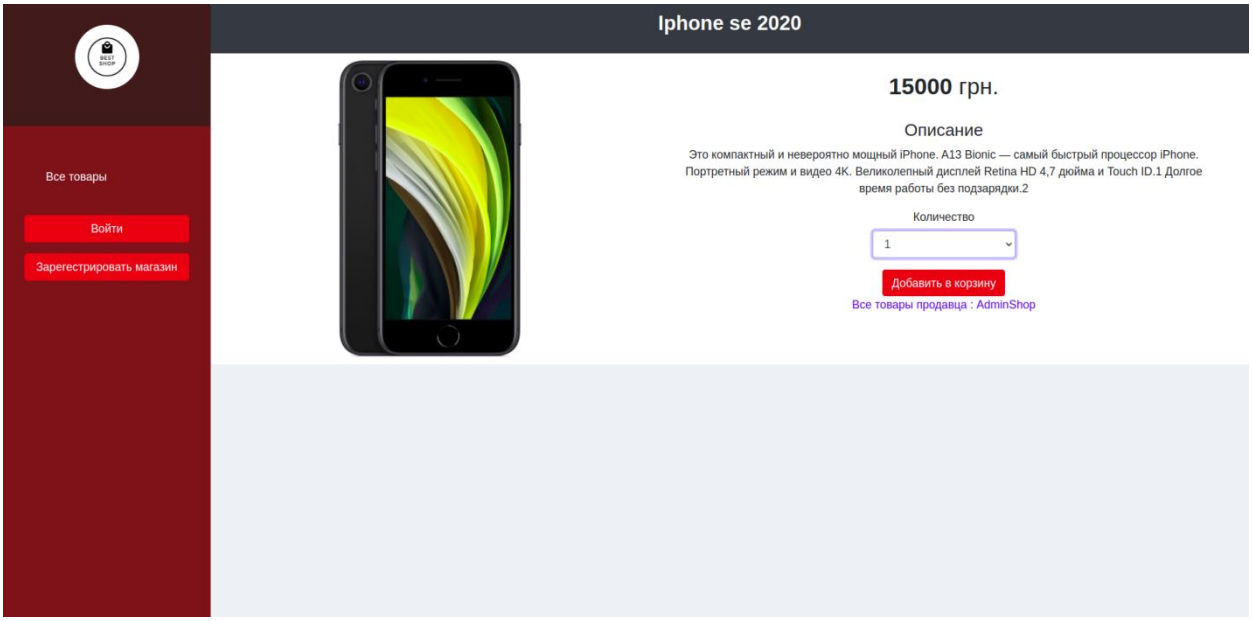

Рисунок 3.11 – Сторінка з товаром

Після додання товару у корзину на панелі зліва з’являється пункт меню з кількістю товарів у корзині і сумою замовлення а також користувач перенеправляється на сторінку з корзиною покупок. На цій сторінкі він має можливість змінити кількість обраного товару, або видалити товар з корзини. Також має можливість перейти до сторінки оформлення замовлення

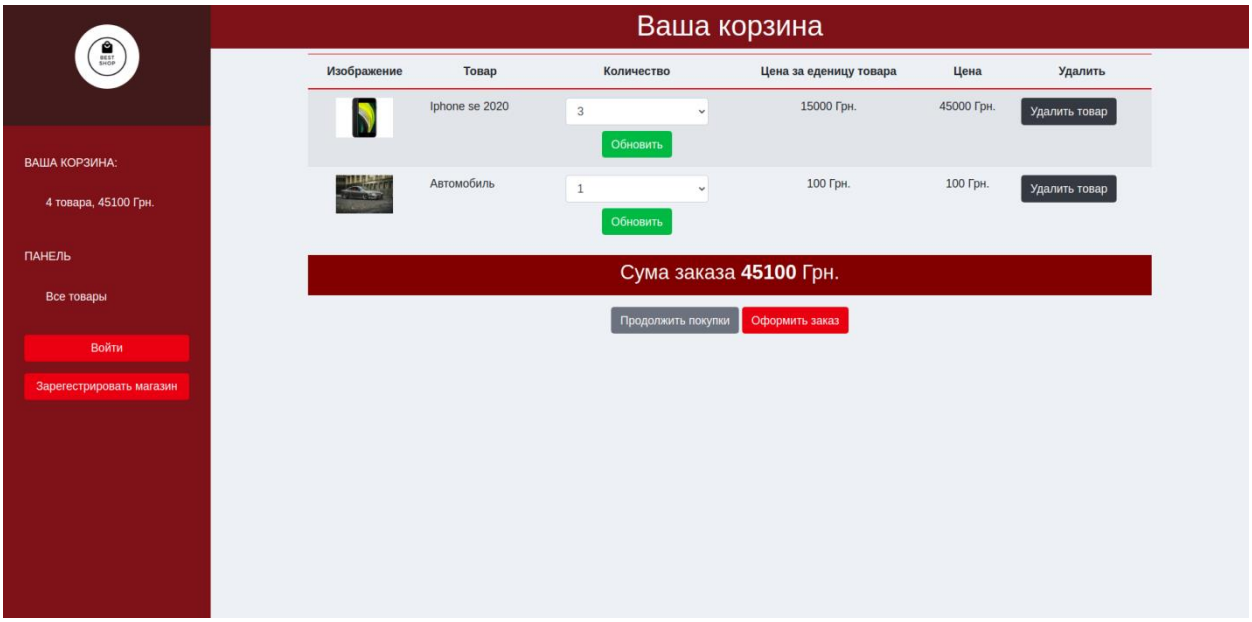

Рисунок 3.12 – Сторінка з корзиною покупок

Після натиснення кнопки “Оформить заказ” користувач перенаправляється на сторінку з формою замовлення, на якій є інформація про товари які знаходятся в корзині, також він може повернутися до корзини і змінити щось, у разі необхідності.

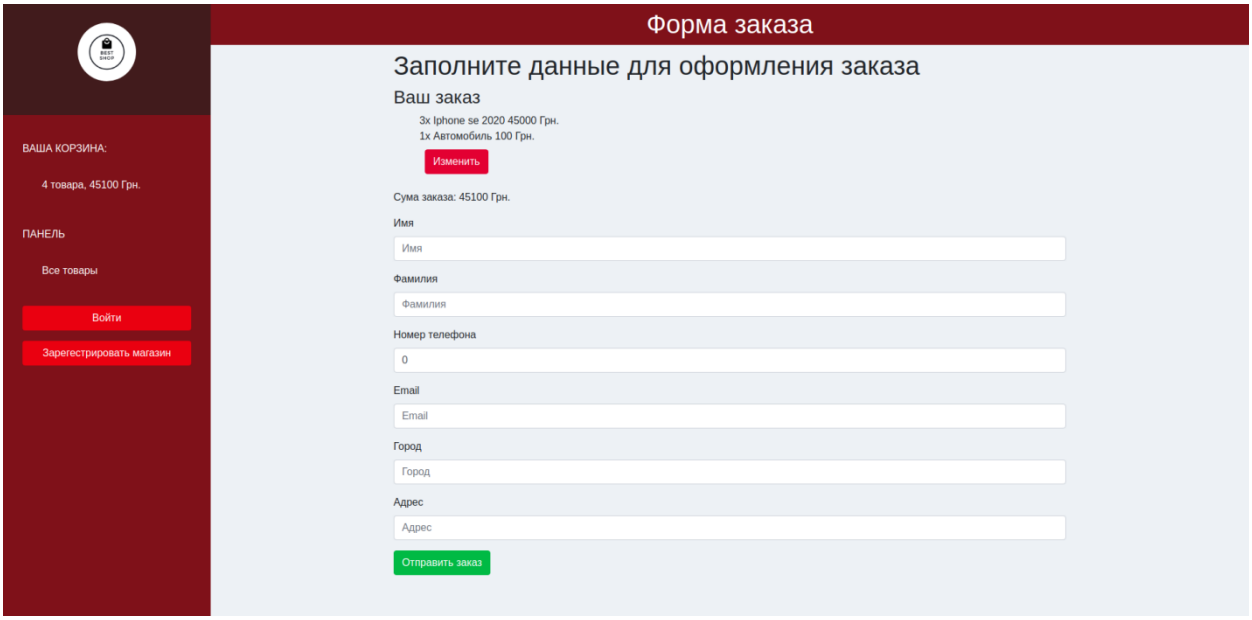

Рисунок 3.13 – Форма замовлення

Після заповнення всіх полів користувач натискає кнопку оформлення замовлення і отримує повідомлення про успішне виконання цієї операції. А

також отримує номер свого замовлення, який може бути йому необхідний для уточнення деталей з продавцем. Корзина автоматично очищається. Замовлення надсилається продавцю

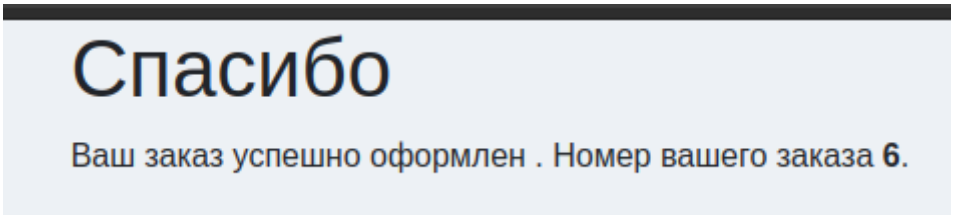

Рисунок 3.14 – Повідомлення про успішне оформлення замолвення

3.3. Опис основного функціоналу торгового майданчика для продавця

Кожен користувач має можливість зареєструвати власний магазин у системі торгового майданчика. Або, якщо це вже зроблено увійти до свого аккаунту через навігаційну панель .

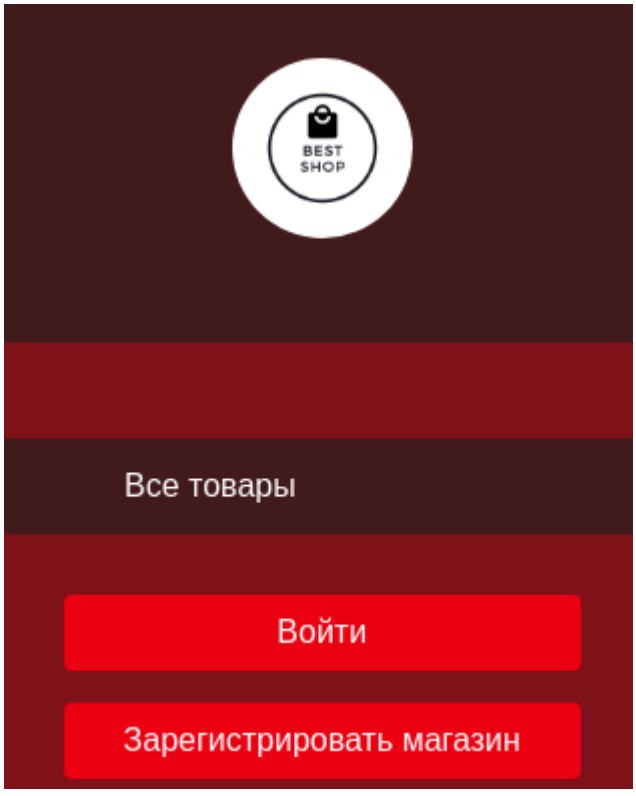

Рисунок 3.15 – Повідомлення про успішне оформлення замолвення

Для того щоб зареєструвати свій магазин потрібно заповнити форму реєстрації.

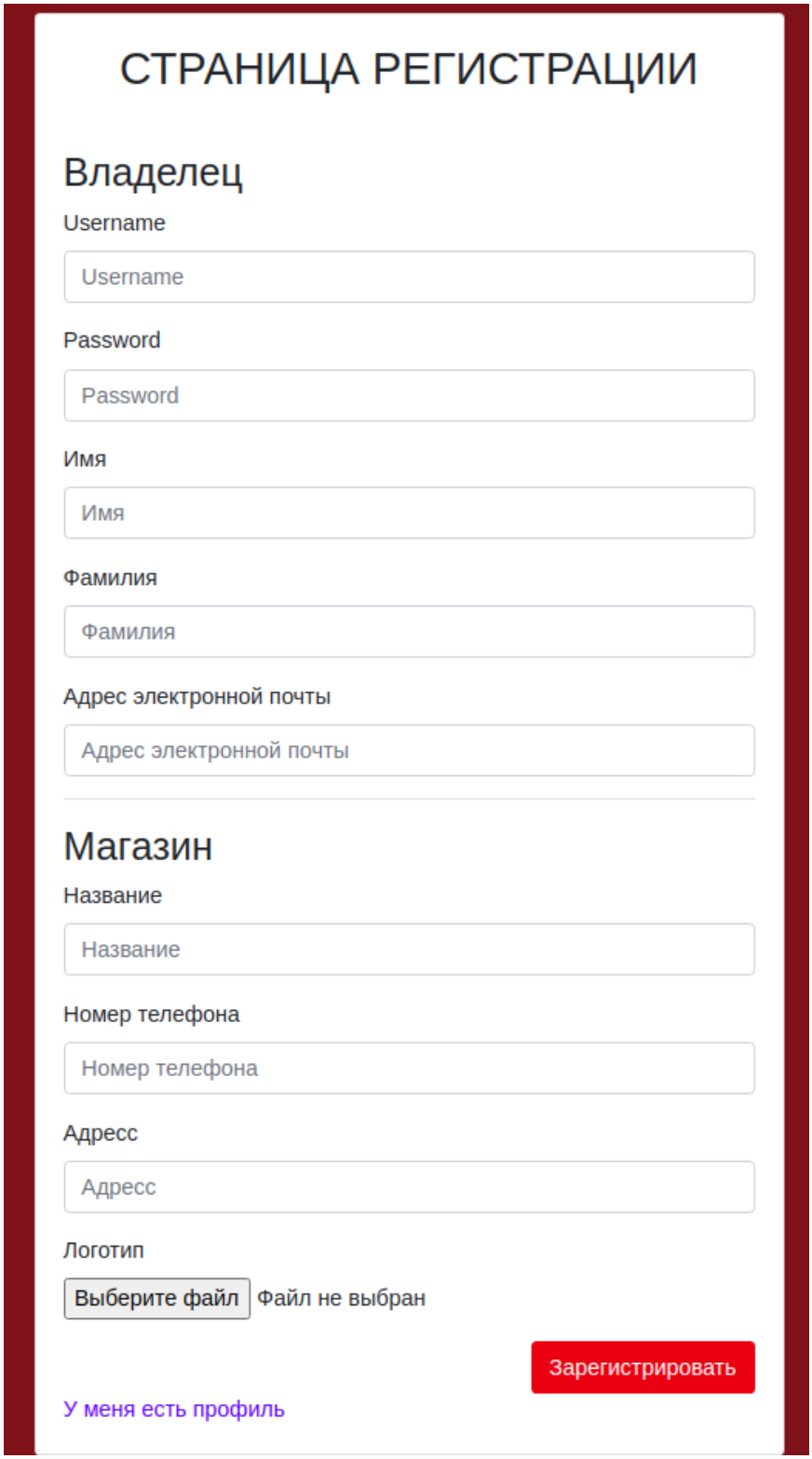

Рисунок 3.16 – Форма реєстрації магазину

Після заповнення всих данних проводитсья перевірка їх на валідність. Якщо дані валідні тоді продавець перенаправляється на сторінку магазину.

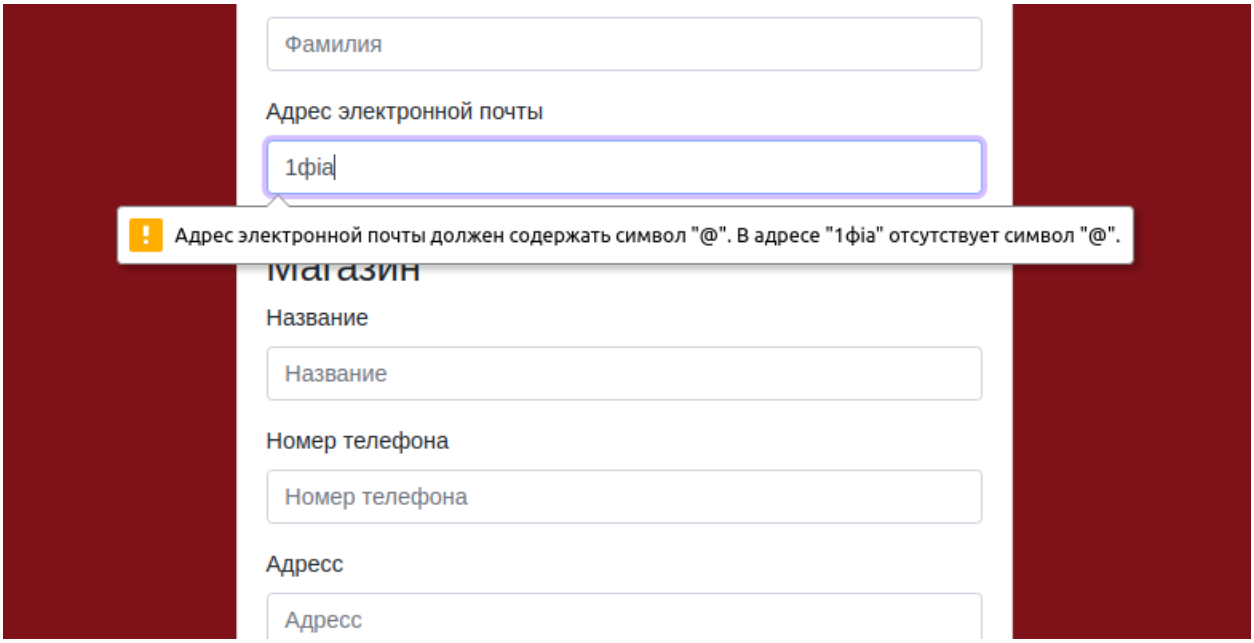

Рисунок 3.17 – Перевірка валідності даних

У авторизованого користувача на навігаційній панелі з’являються логотип магазину, ім’я і прізвище власника , назва магазину а також додаткові пункти меню , які дозволяються проводити адміністрування замовлень і свої товарів. Також користувач має можливість змінити дані свого профілю і дані про магазин.

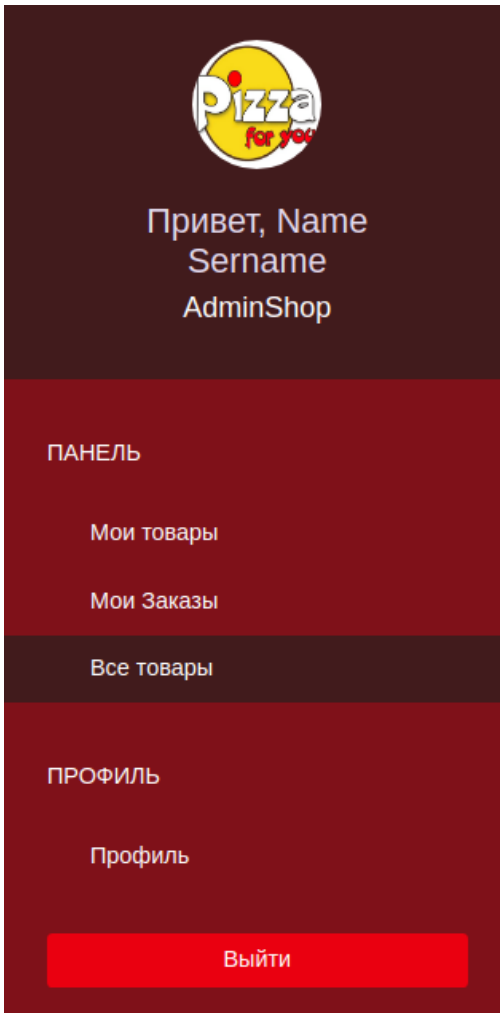

Рисунок 3.18 – Навігаційна панель авторизованого користувача

Авторизований користувач перейшовши на вкладку “Мои товары” має можливість додати, змінити чи видалити товар.

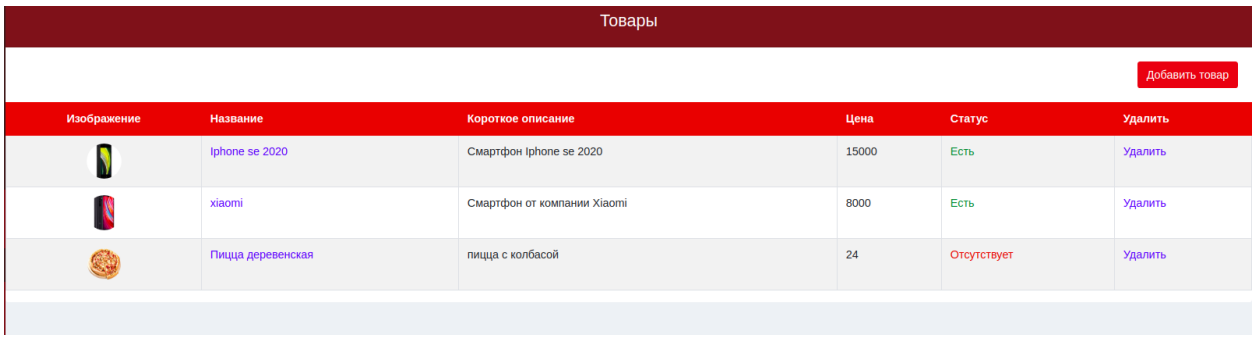

Рисунок 3.19 – Сторінка адміністрування товарів

Для додання товару продавець має перейти на відповідну сторінку, на якій знаходиться форма для заповнення данними про товар, а також продавець має змогу завантижити головне зображення для картки товару, та галерую зображень, які можна переглянути на сторінки з детальною інформацію про товар. Якщо введені дані валідні - то товар буде додано в базу даних і продавець буде перенаправлений на сторінку адміністрування товарів.

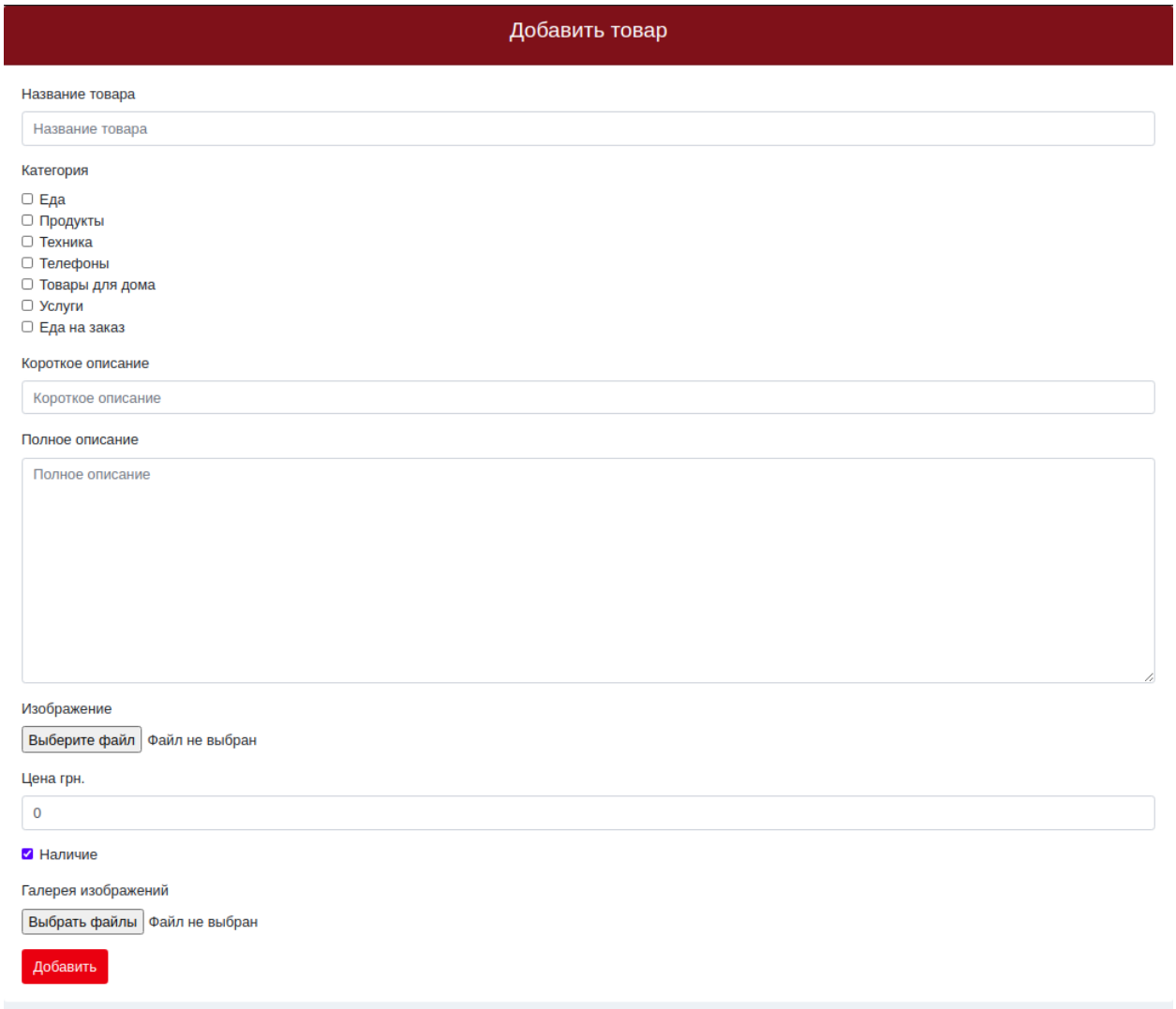

Рисунок 3.20 – Сторінка додання товару

Для переходу на сторінку редагування товару продавець має натиснути на назву товару або на його зображення. Там він має всі ті ж поля що й при доданні товару, а також видалення зображення з галереї або їх додання.

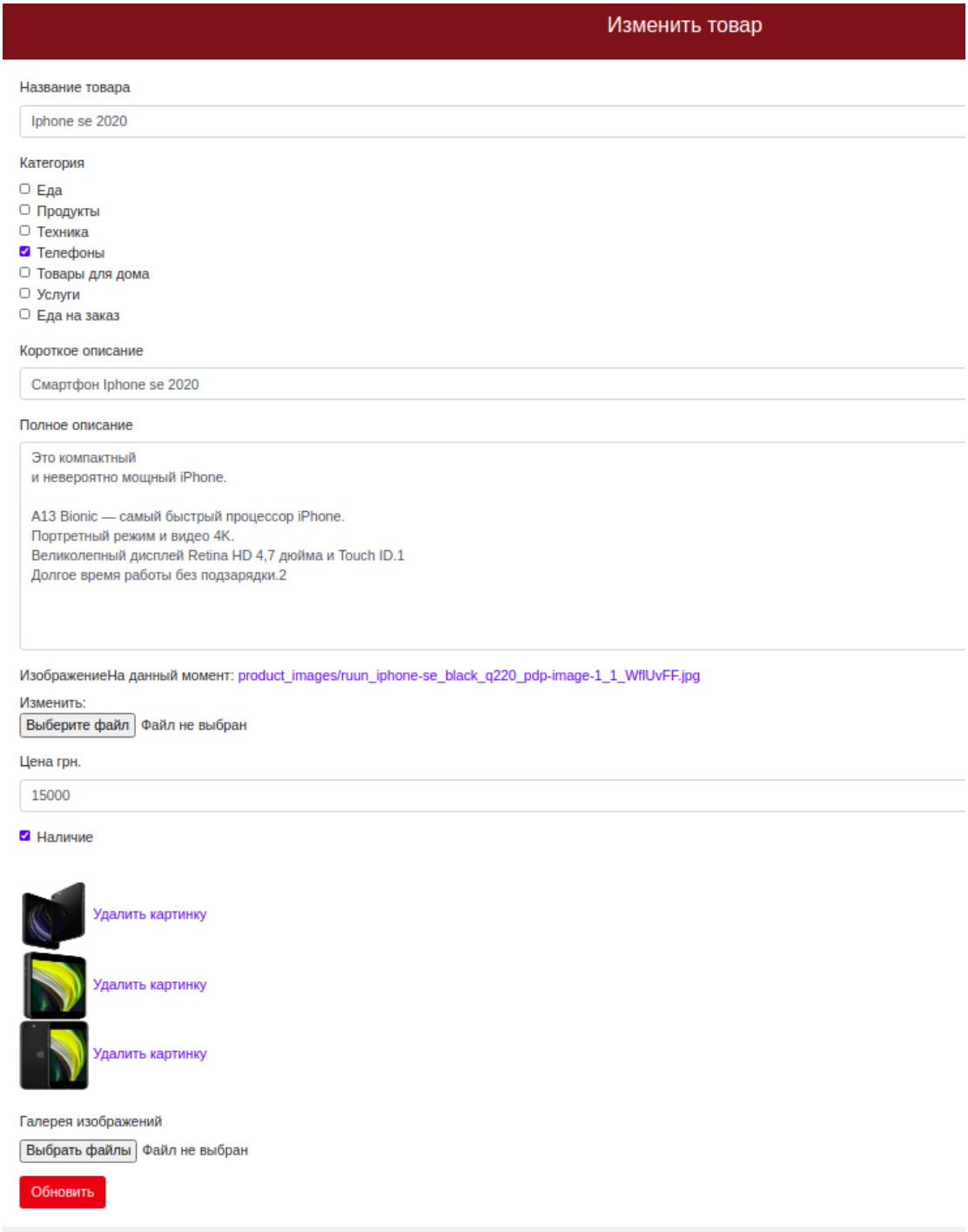

Рисунок 3.21 – Сторінка редагування товару

На сторінкі адміністрування замовлень продавець має можливість перглядати замовлення, видалати його, а також змінювати статус замовлення. Замовлення які вже були опрацьовані опускаються вниз списку.

Рисунок 3.22 – Сторінка з замовленнями

На сторінкі з замовленням у продавця є інофрмація про номер замовлення, також дані про замовника і інформація про товари , які замолвені в його магазині. Якщо клієнт замовив декілька товарів в різних магазинах в одному замовленні - то кожному продавцю окрему надійде замовлення лише з тими товарами, які належать його магазину.

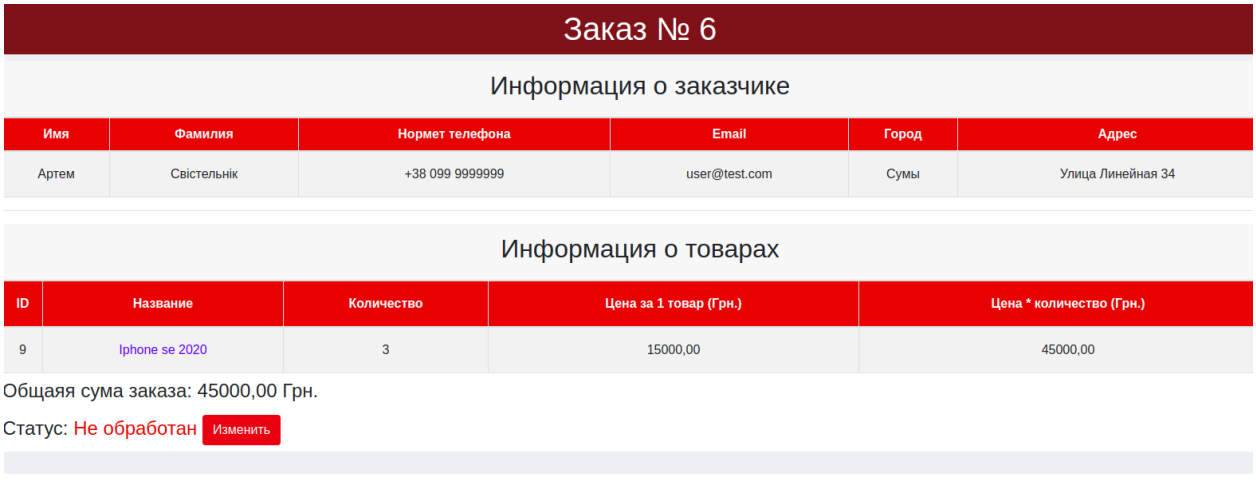

Рисунок 3.23 – Сторінка замовлення

На вкладці “Профиль” користувач може змінити деяку інформацію про магазин і особисті дані.

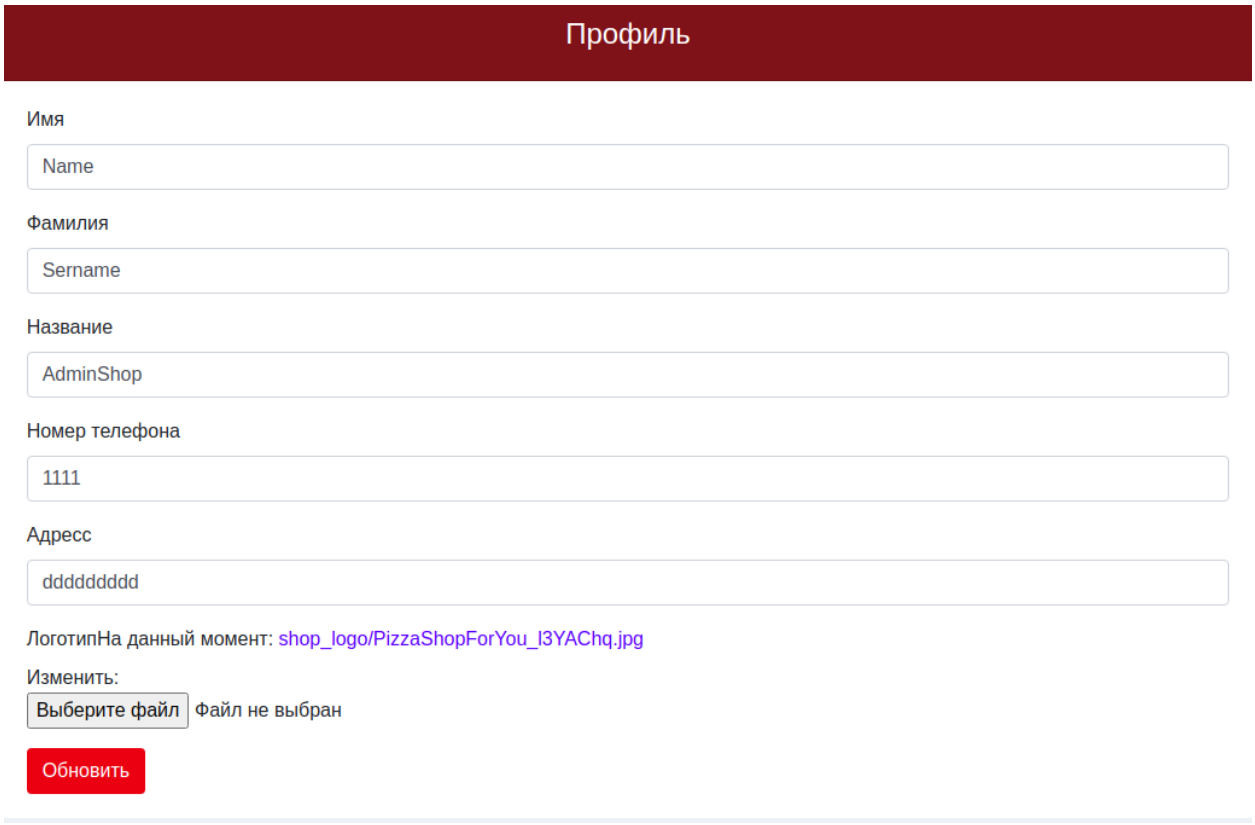

Рисунок 3.24 – Сторінка редагування профілю

ВИСНОВКИ

Проведено аналіз існуючих маркетплейсів, у результаті якого було сформовано ряд вимог до проектування власного веб-ресурсу. Також було проведено вибір сучасних технологій та методів розробки веб-платформ для торгівлі.

У результаті виконаної роботи було розроблено і реалізовано веб-ресурс для продажу товару чи послуг через власний інтеренет-магазин. У роботі були вирішені усі поставлені задачі та досягнуто мети, а саме максимальне спрощення процедури продажу і покупки товару як для продавців так і для клієнтів. А також можливість вести простий облік замовлень для продавця.

СПИСОК ЛІТЕРАТУРИ

1 Документація Django [Електронний ресурс] https://docs.djangoproject.com/en/3.0/

2 Стаття про концепцію розробки MVC https://rtfm.co.ua/django-book-model-razrabotki-mtc-model-view-controller/

3 Головний сайт фреймворку django https://www.djangoproject.com/

4 Antonio Melé Django 2 by Example

5 Відео-курс Олега Молчанова про Django 2.

6 Стаття Generating ERD for Django Applications https://wadewilliams.com/technology-software/generating-erd-for-django-applications/

7 Learning python Mark Lutz

8 Python practice book Anand Chitipothu

9 Django. Подробное руководство Адриан Головатый и Джейкоб Каплан-Мосс

ДОДАТОК

Views.py

```
from django.shortcuts import render,redirect,get_object_or_404,reverse

from django.contrib.auth.decorators import login_required

from .forms import UserForm,ShopForm, UserFormEdit,ProductForm,ImagesForm,
SearchForm,CartAddProductForm,OrderCreateForm

from django.contrib.auth.models import User

from django.contrib.auth import authenticate,login

from .models import  Product,Shop  ,Category,ProductImages,Order,OrderItem

from django.contrib.postgres.search import TrigramSimilarity

from django.views.decorators.http import require_POST

from .cart import Cart


def homeview(request):

   return redirect('diplomapp:sitename_home')


def sitename_home(request):

   return redirect('diplomapp:products')


def sitename_sign_up(request):

   user_form=UserForm()

   shop_form=ShopForm()

   if request.method=='POST':

      user_form=UserForm(request.POST)

      shop_form=ShopForm(request.POST,request.FILES)

      if user_form.is_valid() and shop_form.is_valid():
```

```
            new_user=User.objects.create_user(**user_form.cleaned_data)
            new_shop=shop_form.save(commit=False)
            new_shop.owner=new_user
            new_shop.save()
            login(request,authenticate(username=user_form.cleaned_data['username'],
                                password=user_form.cleaned_data['password']))
            return redirect('diplomapp:sitename_home')
    return render(request,'registration/sign_up.html',{'user_form':user_form,
                                        'shop_form':shop_form})


@login_required(login_url='/sitename/sign-in/')
def account(request):
    user_form=UserFormEdit(instance=request.user)
    shop_form=ShopForm(instance=request.user.shop)
    if request.method=='POST':
        user_form = UserFormEdit(request.POST,instance=request.user)
        shop_form = ShopForm(request.POST,instance=request.user.shop)
        if user_form.is_valid() and shop_form.is_valid():
            user_form.save()
            shop_form.save()
    return  render(request,'diplomapp/account.html',{'user_form':user_form,
                                        'shop_form':shop_form})


@login_required(login_url='/sitename/sign-in/')
def my_products(request):
    products=Product.objects.filter(shop=request.user.shop).order_by('-availability','-id')
    return  render(request,'diplomapp/my_products.html',{'products':products})
```

```
def products(request):
    shop = None
    category = None
    search_form = SearchForm()
    query = None
    products = []
    if 'query' in request.GET:
        search_form = SearchForm(request.GET)
        if search_form.is_valid():
            query = search_form.cleaned_data['query']
            products =
Product.objects.annotate(similarity=TrigramSimilarity('product_name',query)+TrigramSimilarity('short_description',qu
ery)
                                    ).filter(similarity__gt=0.1).order_by('-similarity')
            print('products', products)
        else:
            print(search_form.errors)
    else:
        products = Product.objects.all().order_by('-availability','-id')
    nodes = Category.objects.all()
    return render(request, 'diplomapp/products.html', {'products': products,
                                            'nodes': nodes,
                                            'category': category,
```

```
                                    'shop': shop,
                                    'search_form': search_form,
                                    'query': query,
                                    })



def product_detail(request,product,product_id):
    product_det=get_object_or_404(Product,slug=product,id=product_id)
    images=ProductImages.objects.filter(product=product_det)
    cart_product_form = CartAddProductForm()
    return render(request,'diplomapp/product_detail.html',{'product':product_det,
                                        'images':images,
                                        'cart_product_form': cart_product_form
                                            })


def shop_products(request,shop_slug,shop_id):
    search_form = SearchForm()
    query = None
    products = []
    category=None
    shop=get_object_or_404(Shop,slug=shop_slug,id=shop_id)
    if 'query' in request.GET:
        search_form = SearchForm(request.GET)

        if search_form.is_valid():
            query = search_form.cleaned_data['query']
```

```
            products=Product.objects.annotate(
                similarity=TrigramSimilarity('product_name', query) + TrigramSimilarity('short_description', query)
                ).filter(similarity__gt=0.5).order_by('similarity','-availability',)
        else:
            print(search_form.errors)
    else:
        products=Product.objects.filter(shop=shop).order_by('-availability')
    nodes = Category.objects.all()
    return render(request, 'diplomapp/products.html', {'products': products,
                                        'nodes': nodes,
                                        'category':category,
                                        'shop':shop,
                                        'search_form': search_form,
                                        'query': query,
                                        })


def test(category,n,query):
    if query:
        cats=Category.objects.filter(parent=category)
        products = Product.objects.annotate(
            similarity=TrigramSimilarity('product_name', query) + TrigramSimilarity('short_description', query)
        ).filter(similarity__gt=0.5,category=category ).order_by('-similarity')
        n += 1
        for cat in cats:
            products = products.union(products,Product.objects.annotate(
            similarity=TrigramSimilarity('product_name', query) + TrigramSimilarity('short_description', query)
        ).filter(similarity__gt=0.5,category=cat ).order_by('-similarity'))
```

```
            products=products.union(products,test(cat,n,query))
        return products
    else:
        cats = Category.objects.filter(parent=category)
        products=Product.objects.filter(category=category)
        n += 1
        for cat in cats:
            products=products.union(products,Product.objects.filter(category=cat))
            products=products.union(products,test(cat,n,query))
        return products



def product_on_category(request,category_id,category_slug):
    search_form = SearchForm()
    query = None
    products = []
    shop=None
    category=get_object_or_404(Category,id=category_id,slug=category_slug)
    if 'query' in request.GET:
        search_form = SearchForm(request.GET)
        if search_form.is_valid():
            query = search_form.cleaned_data['query']
            products = test(category, 0,query)
        else:
            print(search_form.errors)
```

```
    else:
        products=test(category,0,query)
    nodes = Category.objects.all()
    return render(request, 'diplomapp/products.html', {'products': products,
                                            'nodes': nodes,
                                            'category':category,
                                            'shop': shop,
                                            'search_form': search_form,
                                            'query': query,
                                            })
```

```
@login_required(login_url='/sitename/sign-in/')
def add_products(request):
    form = ProductForm()
    images_form=ImagesForm()
    if request.method=='POST':
        form=ProductForm(request.POST,request.FILES)
        images_form = ImagesForm(request.POST, request.FILES)
        if form.is_valid() and images_form.is_valid():
            product=form.save(commit=False)

            product.shop=request.user.shop
            product=form.save()
            product.save()
```

```
            for field in request.FILES.keys():
                if field=='image':
                    for img_1 in request.FILES.getlist(field):
                        product=Product.objects.get(id=product.id)
                        product.image=img_1
                        product.save()
                if field=='product_image':
                    for img in request.FILES.getlist(field):
                        image=ProductImages(product_image=img)
                        image.product=product
                        image.save()
            return redirect('diplomapp:products')
        else:
            form = ProductForm()
            images_form=ImagesForm()
    return  render(request,'diplomapp/add_products.html',{'form':form,
                                            'images_form':images_form})



@login_required(login_url='/sitename/sign-in/')
def edit_product(request,product_id):
    product_get=Product.objects.get(id=product_id)
    images=ProductImages.objects.filter(product=product_get)
    if request.user.shop==product_get.shop:
        form = ProductForm(instance=Product.objects.get(id=product_id))
        form_img = ImagesForm(ProductImages.objects.filter(product=product_get))
```

```
    if request.method=='POST':
        form=ProductForm(request.POST,request.FILES,instance=Product.objects.get(id=product_id))
        form_img = ImagesForm(request.POST,request.FILES,instance=Product.objects.get(id=product_id))
        if form.is_valid() and form_img.is_valid() :
            product=form.save()
            product.save()
            n = 0
            for field in request.FILES.keys():
                if field =='image':
                    for img_1 in request.FILES.getlist(field):
                        product = Product.objects.get(id=product.id)
                        product.image = img_1
                        product.save()
                if field=='product_image':
                    for img in request.FILES.getlist(field):
                        image = ProductImages(product_image=img)
                        image.product = product
                        image.save()
            return  redirect('diplomapp:my_products')
        else:
            print(form.errors)
            print(form_img)
    else:
        return redirect('diplomapp:page_404')
    return render(request, 'diplomapp/edit_product.html', {'form': form,
                                                           'images':images,
                                                          'form_img': form_img,
```

```
                                        'product':product_get
                                        })
@login_required(login_url='/sitename/sign-in/')
def delete_product(request,product_id):
    try:
        product=get_object_or_404(Product,id=product_id)
        product.delete()
        return redirect('diplomapp:my_products')
    except Product.DoesNotExist:
        print('error')
        return redirect('diplomapp:my_products')

@login_required(login_url='/sitename/sign-in/')
def delete_img(request,img_id):
    img=ProductImages.objects.get(id=img_id)
    product=img.product
    try:
        img.delete()
    except:
        pass
    return  redirect('diplomapp:edit_product' ,product_id=product.id)


def page_404(request):
    return render(request,'diplomapp/page_404.html',{})
```

```
@require_POST
def cart_add(request, product_id):
    cart = Cart(request)
    product = get_object_or_404(Product, id=product_id)
    form = CartAddProductForm()
    if request.method=='POST':
        form = CartAddProductForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            cart.add(product=product,
                     quantity=cd['quantity'],
                     update_quantity=cd['update'])
    else:
        form=CartAddProductForm()
    return redirect('diplomapp:cart_detail')

def cart_remove(request, product_id):
    cart = Cart(request)
    product = get_object_or_404(Product, id=product_id)
    cart.remove(product)
    return redirect('diplomapp:cart_detail')

def cart_detail(request):
    cart = Cart(request)
    for item in cart:
        item['update_quantity_form'] = CartAddProductForm(
            initial={'quantity': item['quantity'],
```

```
                'update': True})
    return render(request, 'diplomapp/cart/detail.html', {'cart': cart})



def order_create(request):
    cart=Cart(request)
    shop_list = []
    if request.method=='POST':
        form=OrderCreateForm(request.POST)
        if form.is_valid():
            order=form.save()
            for item in cart:
                OrderItem.objects.create(order=order,
                                         product=item['product'],
                                         price=item['price'],
                                         quantity=item['quantity'],)


            cart.clear()
            return render(request,
                          'diplomapp/order/created.html',
                          {'order': order})
    else:
        form = OrderCreateForm()
    return render(request,
                      'diplomapp/order/create.html',
                  {'cart': cart, 'form': form})
```

```
@login_required(login_url='/sitename/sign-in/')
def orders_list_page(request):
    shop=request.user.shop
    order_num=OrderItem.objects.filter(product__shop=shop).values('order').distinct()
    order_num_list=[]
    for order in order_num:
        order_num_list.append(order['order'])
    orders=Order.objects.filter(id__in=order_num_list).order_by('status','created')
    return render(request,'diplomapp/order/orders_list.html',{'orders':orders,})

@login_required(login_url='/sitename/sign-in/')
def order_remove(request,order_id):
    order = Order.objects.get(id=order_id)
    order.delete()
    return redirect('diplomapp:orders_list_page')

@login_required(login_url='/sitename/sign-in/')
def order_detail(request,order_id):
    shop=request.user.shop
    order=Order.objects.get(id=order_id)
    order_items=OrderItem.objects.filter(order__id=order_id,product__shop=shop)
    total_price=0
    for item in order_items:
        total_price+=(item.price*item.quantity)
    print(total_price)
    return  render(request,'diplomapp/order/order_detail.html',{'order':order,
```

```
                                        'order_items':order_items,
                                        'total_price':total_price})


@login_required(login_url='/sitename/sign-in/')
def update_status_order(request,order_id):
    order = Order.objects.get(id=order_id)
    print(order.status)
    if order.status==True:
        order.status=False
    else:
        order.status =True
    order.save()
    return  redirect('diplomapp:order_detail',order_id=order_id)
```