Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Сердюка Артура Олеговича*
(ПІБ)

академічної групи *122-19-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка веб-додатку для інтернет-магазину з продажу одягу з використанням HTML, CSS, JavaScript, Python.*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*		відмінно	
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-2* (група) *Сердюк А.О.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатку для інтернет-магазину з продажу одягу з використанням HTML, CSS, JavaScript, Python.*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц. Спірінцев В.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Сердюк А.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12,06.2023 р*

РЕФЕРАТ

Пояснювальна записка: 91 с., 35 рис., 3 дод., 20 джерел.

Об’єкт розробки: веб-додаток для інтернет магазину з продажу одягу.

Мета кваліфікаційної роботи: розробити інтернет-магазин одягу з метою поліпшення комфорту покупок для клієнтів та збільшення прибутку для власника магазину.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформу для розробки, виконано проектування і розробку веб-орієнтованої інформаційної системи, описана робота системи, алгоритм і структура його функціонування, а також виклик та завантаження додатку, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню додатку та розраховано час на його створення.

Практичне значення роботи полягає у створенні функціонального інтернет-магазину, який може ефективно використовуватись для продажу одягу. Він має зручний інтерфейс, що дозволяє користувачам швидко знайти потрібний товар.

Прогнозуючи розвиток можна відзначити, що дана сфера буде продовжувати рости у майбутньому. Кількість покупок онлайн продуктів, зокрема одягу, активно зростає щороку тим самим збільшуючи попит на продукцію і це дає поштовх іншим магазинам створювати веб-орієнтовні додатки для інтернет продажу своєї продукції.

Список ключових слів: ІНТЕРНЕТ-МАГАЗИН, ВЕБ-ДОДАТОК, БАЗА ДАНИХ, PYTHON, HTML, CSS, JAVASCRIPT.

ABSTRACT

Explanatory note: 91 pp., 35 fig., 3 extra, 20 sources.

The object of development: a web application for an online store selling clothes.

The purpose of the diploma project is to develop an online clothing store to improve the shopping experience for customers and increase profits for the store owner.

The introduction discusses the analysis and current state of the problem, specifies the purpose of the qualification work and its scope, provides a justification for the relevance of the topic, and clarifies the task statement.

The first section analyzes the subject area, determines the relevance of the task and the purpose of the development, formulates the task statement, and specifies the requirements for software implementation, technologies, and software tools.

The second section analyzes existing solutions, selects a development platform, designs and develops a web-based information system, describes the system's operation, algorithm and structure of its functioning, as well as the application call and loading, defines input and output data, and characterizes the composition of technical means parameters.

In the economic section, the labor intensity of the developed information system is determined, the cost of creating the application is calculated, and the time for its creation is estimated.

The practical significance of the work is to create a functional online store that can be effectively used to sell clothes. It has a user-friendly interface that allows users to quickly find the right product.

Forecasting the development, it can be noted that this area will continue to grow in the future. The number of online purchases of products, in particular clothing, is actively growing every year, thereby increasing the demand for products and this gives impetus to other stores to create web-based applications for online sales of their products.

Keyword list: ONLINE STORE, WEB APPLICATION, DATABASE, PYTHON, HTML, CSS, JAVASCRIPT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ - Програмне забезпечення;

ІС - Інформаційна система;

ОС - Операційна система;

БД - База даних;

СУБД - Система управління базами даних;

CSS - Cascading Style Sheets;

SQL - Structured Query Language;

HTML - Hyper Text Markup Language.

ВСТУП

Зараз, у добу інтернет-технологій, з'явилась можливість здійснювати покупки онлайн. Інтернет-магазини стали дуже популярними серед людей, які цінують свій час та зручність покупок. Однак, зростання популярності цього виду торгівлі призвело до збільшення конкуренції серед магазинів.

У зв'язку з цим, метою дипломної роботи є дослідження та аналіз інтернет-магазинів одягу з метою вдосконалення їх функціонування та підвищення ефективності роботи на прикладі власного проєкту. Галуззю застосування кваліфікаційної роботи є сфера електронної комерції, зокрема продажів одягу в інтернеті.

Актуальність теми полягає в тому, що кількість покупців, які переходять на онлайн-шопінг, з кожним роком зростає. Наприклад, в умовах поширення COVID-19 та карантинних обмежень люди почали переходити на покупку різних речей через інтернет, і одяг не став виключенням. Також зазначу, що під час карантину люди почали переносити свій бізнес до мережі, таким чином їхній бізнес продовжував функціонувати і приностити прибуток. Не можу не зазначити і війну, яка є складним фактором, що впливає на електронну комерцію. У період конфлікту звичайні канали постачання товарів та послуг можуть бути порушені або обмежені, що ставить підприємства перед викликом забезпечити продовження своєї діяльності. В таких умовах електронна комерція стає важливим інструментом для забезпечення постачання товарів та послуг, оскільки дозволяє підприємствам здійснювати торгівлю та взаємодіюти з клієнтами віртуально.

Для вирішення поставленої мети необхідно вирішити наступні задачі:

- розробити вимого до функціональності веб-додатку на основі аналізу потреб інтернет-магазину та його цільової аудиторії;
- створити прототип веб-додатку з основними функціональними можливостями та дизайном;
- розробити архітектуру веб-додатку;

– розробити функціонал, що дозволяє клієнтам здійснювати замовлення товарів;
– здійснити розрахунок трудомісткості та витрат на розробку програмного продукту.

Практичне значення кваліфікаційної роботи полягає в тому, що її результати можуть бути використані для покращення роботи інтернет-магазину одягу та полегшення процесу продажів. Розроблений веб-додаток забезпечить зручну інтерактивну платформу для користувачів, де вони зможуть швидко знайти та придбати потрібний їм одяг. Крім того, робота над цією кваліфікаційною роботою може дати мені змогу отримати нові знання та навички у галузі веб-розробки, що дозволить стати більш конкурентоспроможним на ринку праці. Таким чином, практичне значення цієї кваліфікаційної роботи є важливим для самого автора, інтернет-магазину та його клієнтів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

1.1.1 Особливості електронної комерції

Електронна комерція (e-commerce) - це практика проведення бізнес-транзакцій з купівлі-продажу товарів або послуг в Інтернеті. Вона передбачає використання веб-сайтів, онлайн-платформ і каналів соціальних мереж для полегшення транзакцій, отримання платежів і збору цінної інформації про клієнтів у цифрових форматах. Основною особливістю електронної комерції є те, що вона не обмежена географічною локацією, що дає можливість покупцям замовляти товари з будь-якої точки світу. Крім того, інтернет-магазини можуть працювати цілодобово, що також є великим плюсом для клієнтів. Іншою важливою особливістю є можливість збереження історії замовлень та налаштування автоматичних процесів відстеження замовлень, що полегшує роботу як покупців, так і продавців. Крім того, електронна комерція дозволяє рекламувати товари та послуги у соціальних мережах та на різноманітних інтернет-платформах. Однією з найбільших переваг являється можливість здійснювати електронні платежі та використовувати різноманітні способи оплати, що дуже зручно для покупців. Крім того, це дозволяє магазинам більш ефективно використовувати свій інвентар та запобігти необхідності зберігання товарів в фізичних магазинах. Також для самого магазину це зниження витрат: дозволяє продавцям знизити витрати на зберігання, персонал та оренду приміщень. Окрім цього, віртуальний магазин не потребує затрат на зовнішнє оздоблення, ремонт [1].

1.1.2 Аналіз існуючих аналогів

Під час проектування власного продукту, було проведено аналіз декількох веб-додатків для того, щоб зрозуміти які методології просування та зацікавлення клієнта вони використовують. Також отримати шаблон, яким користуються усі інтернет-магазини. Для аналізу було обрано 3 магазини: “Answear” [2], “JolyBell” [1] та “Cropp” [3]. На всіх трьох сайтах можна побачити хедер з вибором типу одягу. Також у всіх трьох сайтах присутній логотип, який знаходиться на видному місці - посередині хедеру, таким чином люди, під час заходу на сайт, будуть постійно бачити та запам’ятовувати назву та логотип бренду. Також на рис. 1.1 - 1.2 можна побачити задній фон у вигляді фотографії моделі або відео з моделлю, це виглядає привабливо та клієнт може одразу побачити людей, які носять цей одяг.

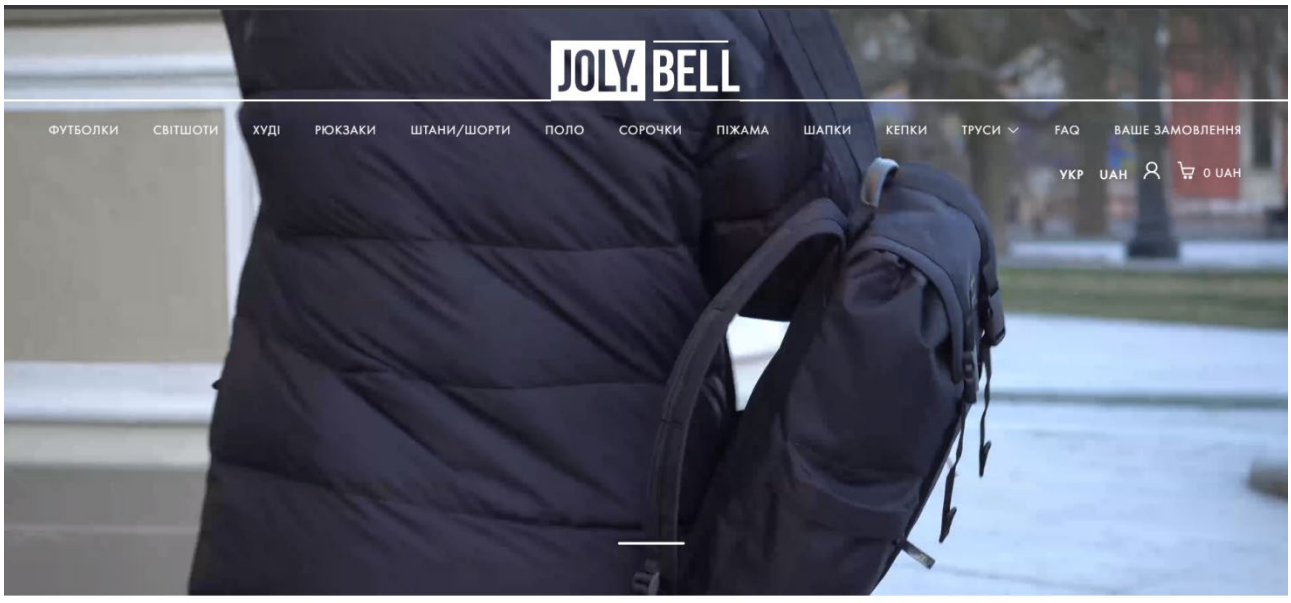

Рис. 1.1. Головна сторінка магазину “JolyBell”

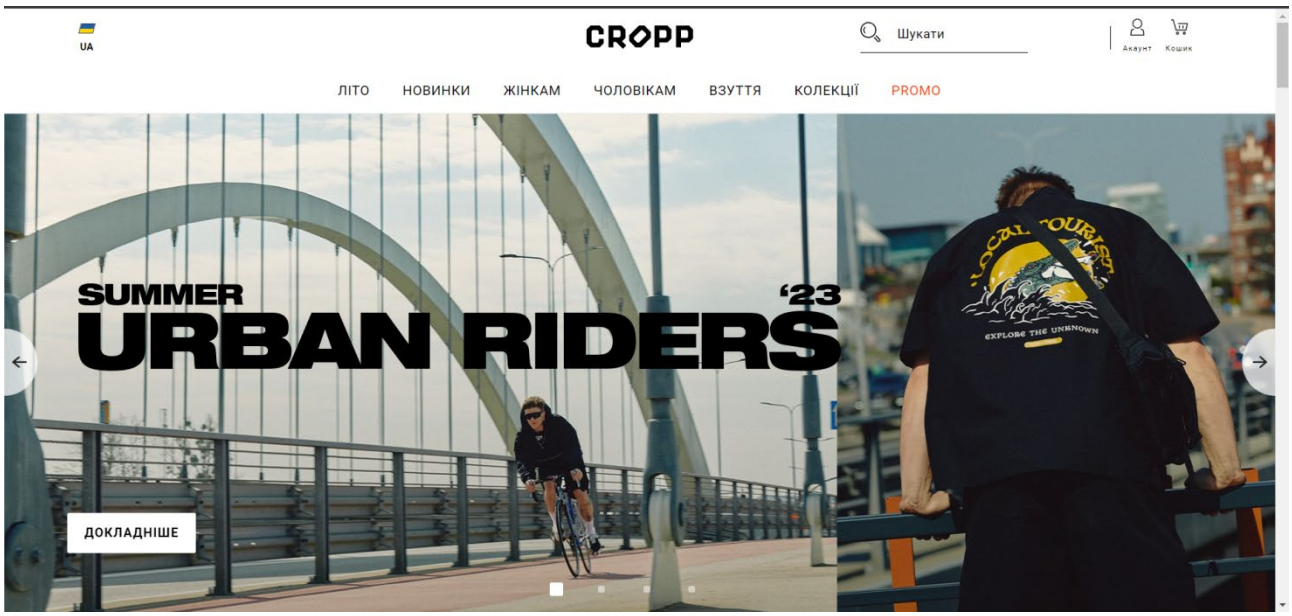

Рис. 1.2. Головна сторінка магазину “Cropp”

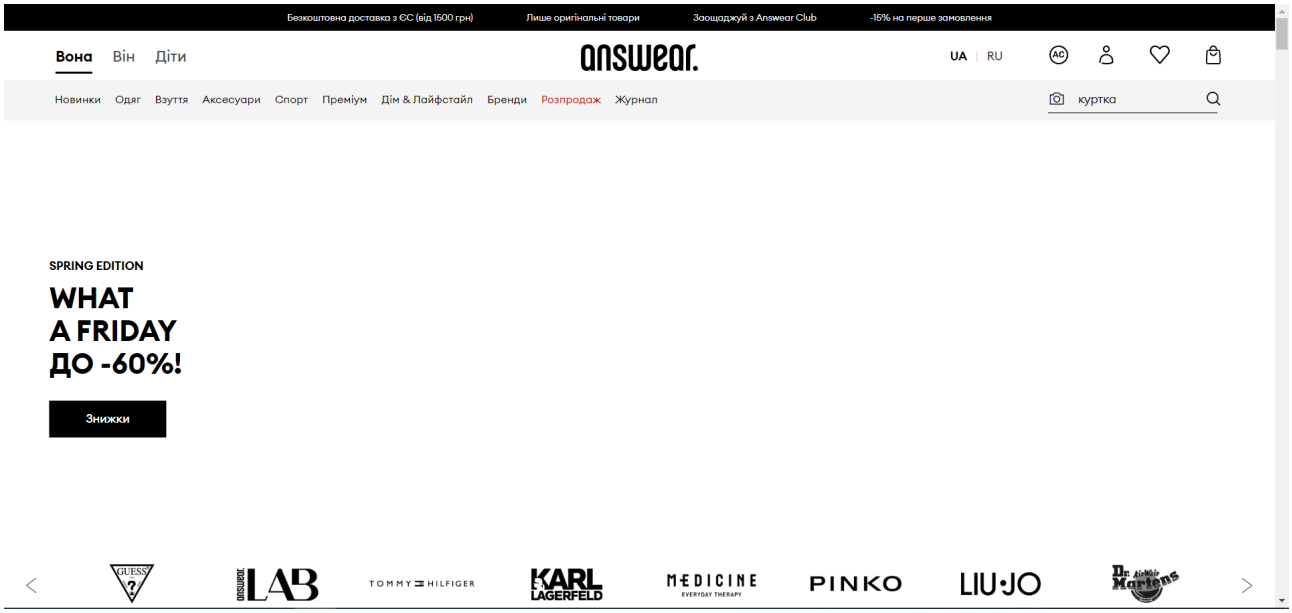

Рис. 1.3. Головна сторінка магазину “Answear”

В магазинах “Answear” та “Cropp” можна проскролити нижче та побачити приклади одягу, які є в наявності. Таким чином нові користувачі зможуть подивитись який одяг пропонує їм даний магазин. Такі приклади можна побачити на рис. 1.4 - 1.5.

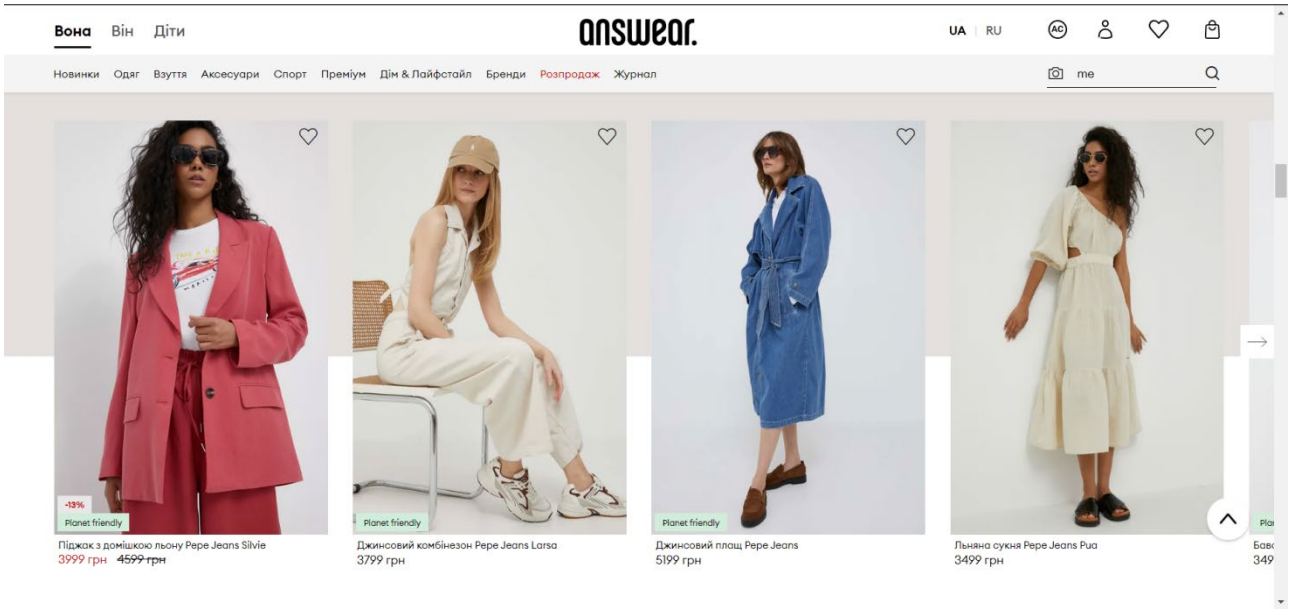

Рис. 1.4. Приклади одягу з магазину “Answear”

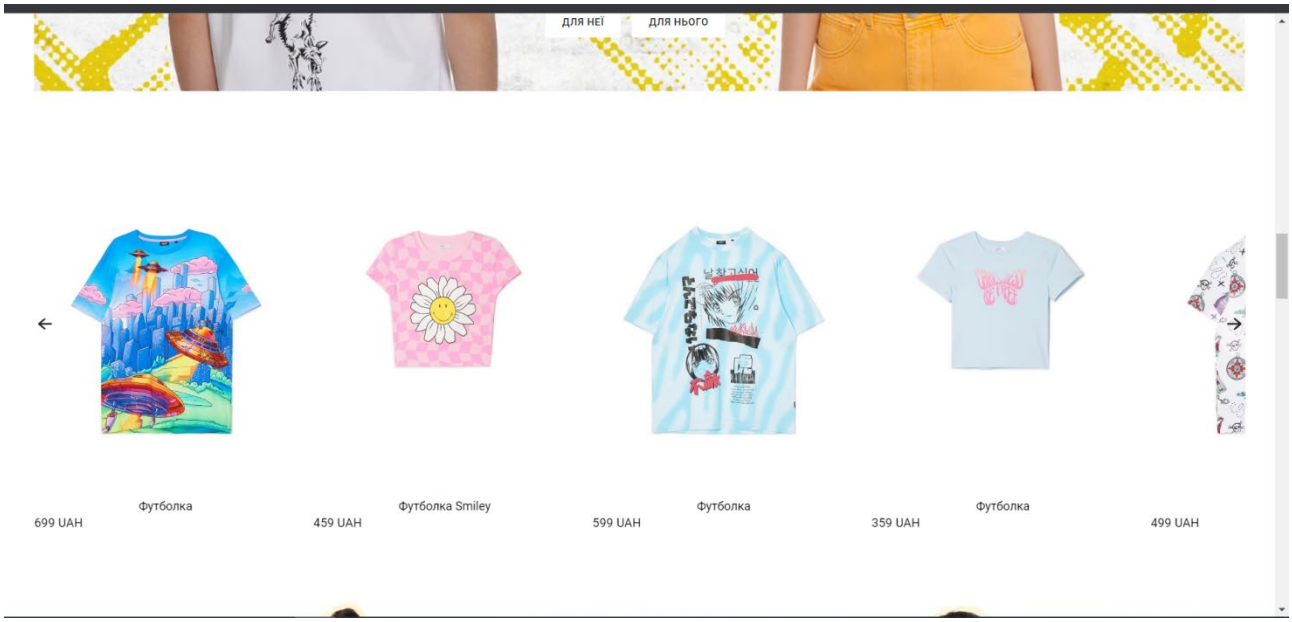

Рис. 1.5. Приклади одягу з магазину “Cropp”

На рис. 1.6 - 1.8 можна побачити приклади футерів магазинів. На них зображена додаткова інформація, яка може зацікавити користувачів інтернет-магазинів.

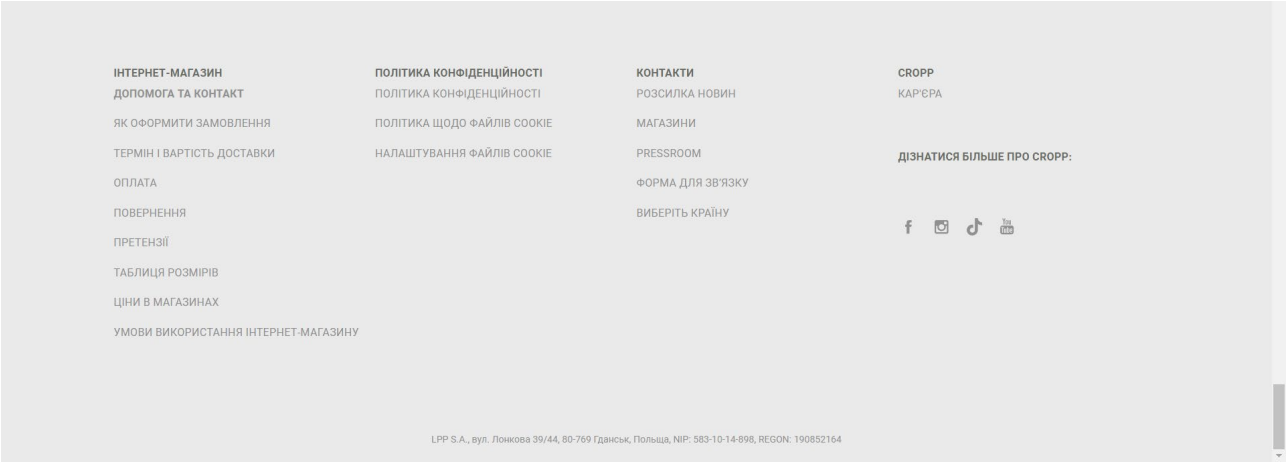

Рис. 1.6. Футер мазагину “Cropp”

Рис. 1.7. Футер магазину “JolyBell”

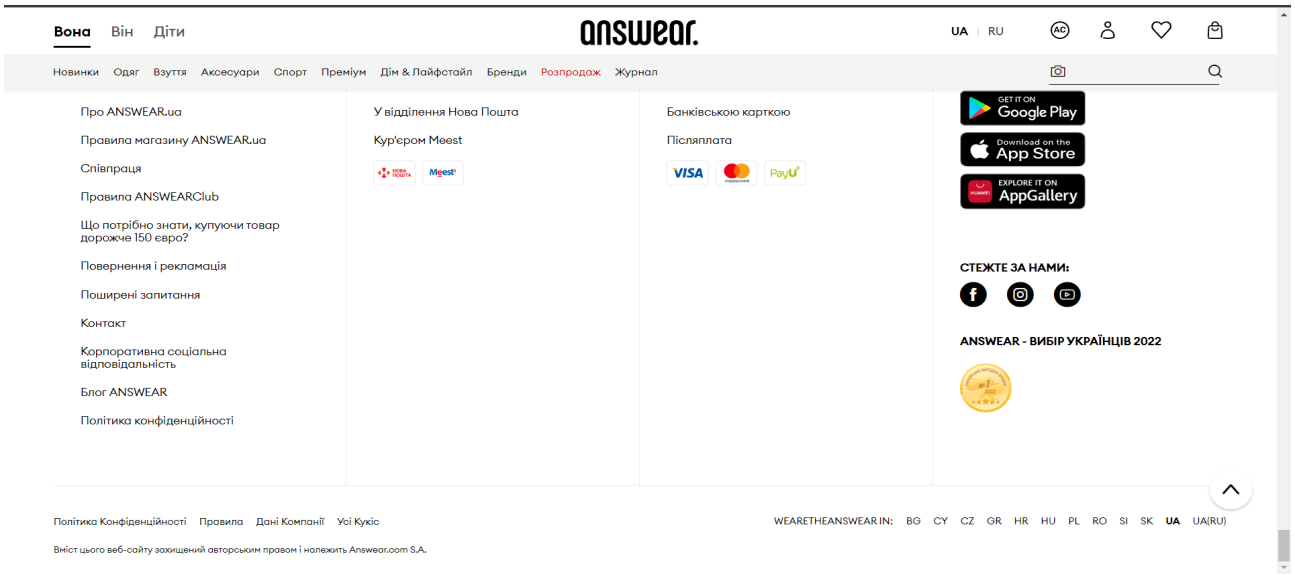

Рис. 1.8. Футер магазину “Answear”

Аналіз існуючих магазинів показав, що інтернет-магазин складається з різних компонентів, які разом формують його структуру. Основні складові та структура інтернет-магазину включають наступне:

- головна сторінка: це перша сторінка, на яку потрапляють відвідувачі, вона повинна містити ключову інформацію про магазин, привабливі акції або нові продукти, посилання на важливі категорії товарів та навігаційні елементи.
- категорії товарів: інтернет-магазин повинен мати структуровану систему категорій товарів, що допомагає клієнтам швидко знаходити потрібні продукти.
- сторінки товарів: кожен товар має свою окрему сторінку, на якій міститься його опис, фотографії, ціна, характеристики та інша інформація.
- кошик: це місце, де клієнти можуть додавати товари, які вони бажають придбати, переглядати замовлення, обирати спосіб доставки та оформлювати покупку.
- система оплати: інтернет-магазин повинен мати інтегровану систему оплати, яка дозволяє клієнтам здійснювати безпечні та зручні платежі.

1.2 Призначення розробки та галузь застосування

Як об'єкт впровадження розроблюваної веб-орієнтованої інформаційної системи для роботи інтернет-магазину одягу розглядається веб-сайт “Munera”, який буде корисний для маленьких та середніх магазинів, які прагнуть полегшити процес продажу своїх товарів та збільшити свою клієнтську базу шляхом відкриття онлайн-магазину. Додаток також буде корисним для покупців, які можуть легко знайти та купити необхідний одяг зручним для них способом.

Після проведення дослідження існуючих технічних рішень, були виявлені основні вимоги, які встановлюються для ефективної роботи інтернет-магазину:

- зручність навігації;
- зручніть оформлення замовлення;
- якість фото товарів;
- орієнтованість на клієнта.

Аналізуючи вищезазначені вимоги, було зроблено висновок про те, що обов'язково буде реалізовано в проекті:

– хедер з категоріями одягу та логотипом посередині;
– лендінг з прикладами одягу;
– сторінка категорії;
– сторінка товару;
– кошик;
– футер з додатковою інформацією.

Головна сторінка першою постає перед користувачем, тому її дизайну та функціональності необхідно приділити чимало уваги, щоб надати змогу забезпечити користувача достатньою кількістю необхідної інформації.

Відповідно приклад прототипу можна побачити на рис. 1.9.

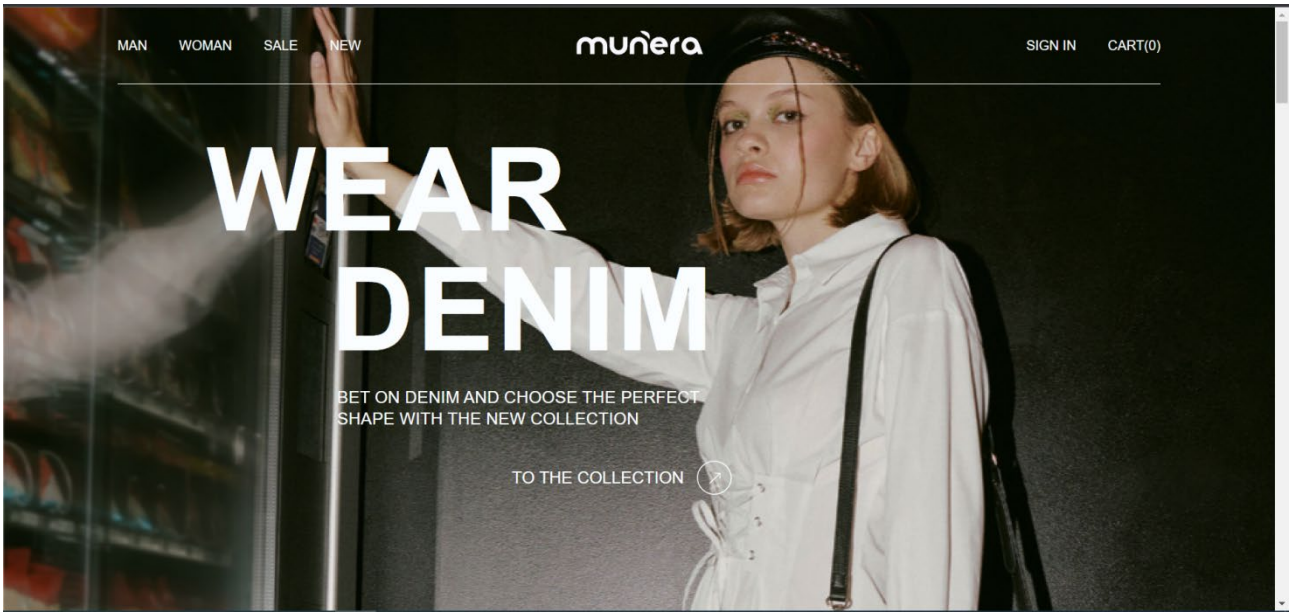

Рис. 1.9. Прототип головної сторінки лендінгу

Головна сторінка містить:

– логотип, розроблений звичайною стилізацією тексту;
– система навігації, яка знаходиться у хедері;
– надпис про нову колекцію та лінк на неї;
– вибір мови: укр. або англ.;

– кошик.

Відповідно до прототипу головної сторінки було здійснено її верстку із застосуванням технологій HTML та CSS.

Розроблена інформаційна система призначена для:

– зручності клієнтів: легко та просто можна переглянути товар та замовити;
– доступність різних видів одягу;
– об’єднання елементів прямого маркетингу та традиційної торгівлі;
– можливість розширити географію свого бізнесу до світових ринків.

1.3 Підстави для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

– навчальний план та графік навчального процесу;
– наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;
– завдання на кваліфікаційну роботу на тему «Розробка веб-додатку для інтернет-магазину з продажу одягу з використанням HTML, CSS, JavaScript, Python».

1.4. Постановка завдання

В даній кваліфікаційній роботі розглядається створення інформаційноїго веб-орієнтованого додатку для магазина одягу.

Інформаційна система створюється для надання послуг з продажу одягу. Аудиторію інформаційної системи становлять люди у віці від 15 до 40 років.

Тематикою інформаційної системи є реалізація продажу одягу через мережу “Інтернет”.

Інформаційна система повинна задовольняти наступним основним вимогам: висока швидкість завантаження сторінок; максимальна зручність

роботи із системою для користувача; оптимізація сторінок під пошукові системи; легкість сприйняття інформації; простота й повнота управління змістом.

Створення й розробка інформаційної системи повинно включати наступні етапи:

- аналіз цільової аудиторії та визначення її вподобання і потреби;
- розробка концепції дизайну сайту, що відображає імідж бренду магазину одягу;
- розробка контент-стратегії для веб-сайту, включно з описами продуктів, публікаціями в блозі та іншою відповідною інформацією;
- розробка платформи електронної комерції, яка включає функціонал кошика для покупок;
- встановлення та конфігурування додатку на веб-сервері, запуск та тестування [5];

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставленої в роботі мети в інформаційній системі, що розробляється, повинні бути реалізовані (рис. 1.10) [6]:

- зручний інтерфейс: веб-сайт повинен мати інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам легко орієнтуватися і швидко знаходити продукти, які вони шукають;
- каталог товарів: на сайті повинен бути каталог товарів, який відображає всі доступні товари, а також їхні ціни, описи та зображення;
- кошик для покупок: на сайті має бути кошик, який дозволяє користувачам додавати товари до кошика і переходити до оформлення замовлення;

– функція пошуку: веб-сайт повинен мати функцію пошуку, яка дозволяє користувачам шукати конкретні продукти за ключовими словами, категоріями або іншими фільтрами;
– безпека: веб-сайт повинен мати надійні засоби безпеки, які захищають дані користувачів і запобігають несанкціонованому доступу.

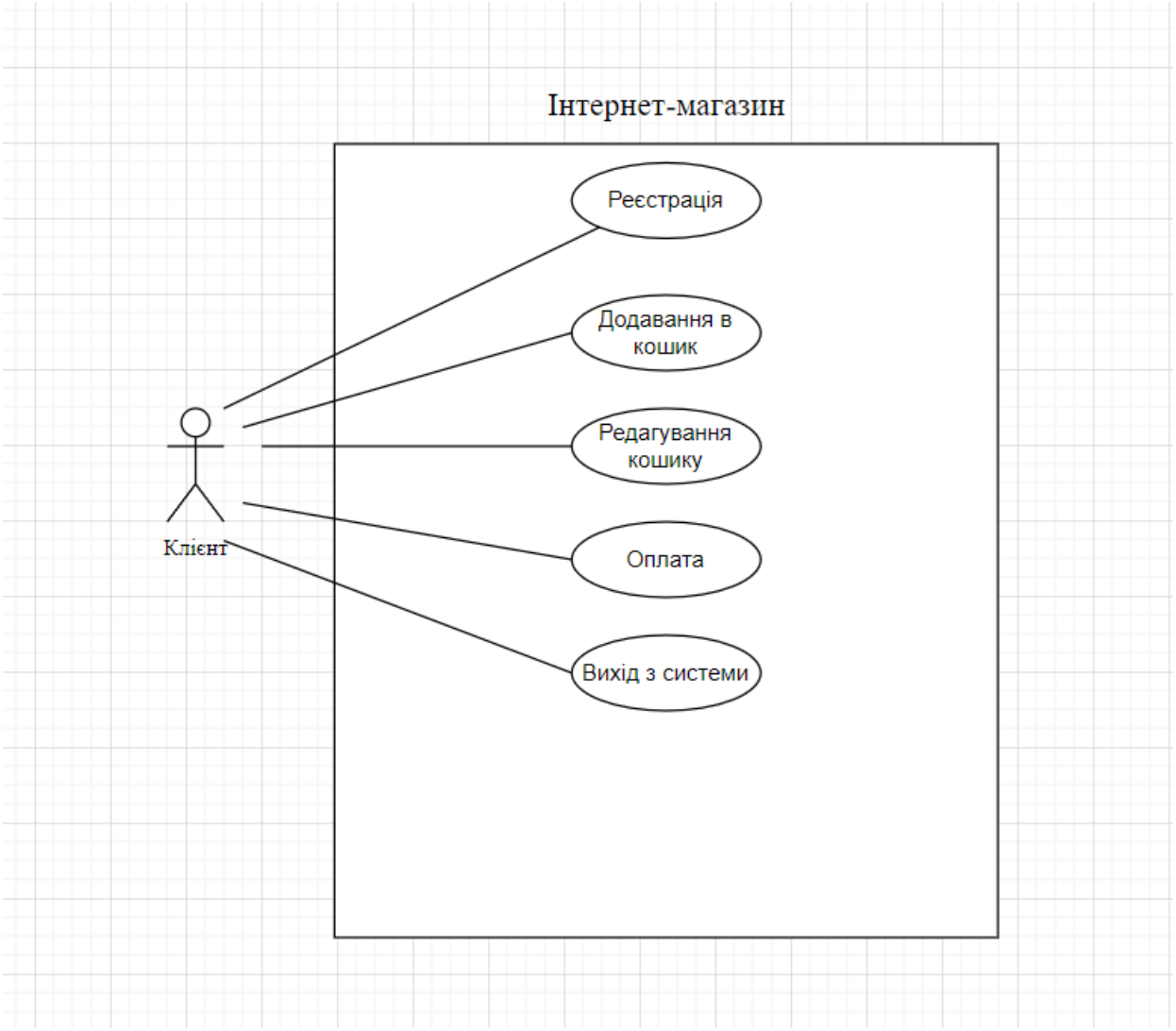

Рис. 1.10. Use Case Diagram

1.5.2. Вимоги до інформаційної безпеки

Під інформаційною безпекою розуміють стан захищеності систем обробки і зберігання даних, при якому забезпечено конфіденційність, доступність і цілісність інформації, використання й розвиток в інтересах громадян або

комплекс заходів, спрямованих на забезпечення захищеності інформації особи, суспільства і держави від несанкційованого доступу, використання, оприлюднення, руйнування, внесення змін, ознайомлення, перевірки запису чи знищення. Таким чином в даній кваліфікаційній роботі задля забезпечення інформаційної безпеки будуть використані наступні [7]:

- аутентифікація та авторизація: впровадження безпечних механізмів автентифікації та авторизації, щоб запобігти несанкціонованому доступу до веб-сайту та бази даних;
- шифрування: використання технологій шифрування для захисту даних під час передачі та зберігання;
- безпечне зберігання: конфіденційні дані повинні зберігатися надійно, з відповідним контролем доступу та шифруванням;
- валідація даних: перевірка всіх даних, що вводяться на сайт, щоб запобігти ін'єкційним атакам та іншим уразливостям безпеки;
- обробка помилок та запис історії: впровадження обробки помилок та ведення іторії для виявлення та реагування на інциденти безпеки.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для ефективного функціонування веб-орієнтованої інформаційної системи, повинні виконуватися певні вимоги до технічних засобів [6].

Для клієнтської частини:

- сучасний веб-браузер: Google Chrome, Mozilla Firefox, Microsoft Edge та Apple Safari;
- операційна система: повинна працювати будь-яка сучасна операційна система, така як Windows, macOS, Linux;
- дисплей: рекомендується монітор або екран з роздільною здатністю не менше 1024x768 пікселів;
- підключення до інтернету: стабільне та надійне інтернет-з'єднання необхідне для доступу до контенту та ресурсів веб-сайту;

– оперативна пам’ять: для безперебійної роботи в Інтернеті, особливо для веб-сайтів з мультимедійним контентом, рекомендується щонайменше 4ГБ оперативної пам'яті;
– процесор: для кращої продуктивності рекомендується сучасний процесор з тактовою частотою не менше 2 ГГц.

Для серверної частини:

– процесор: щонайменше двоядерний процесор з тактовою частотою 3 ГГц або вище;
– оперативна пам’ять: щонайменше 8 ГБ оперативної пам'яті;
– жорсткий диск: щонайменше 100 ГБ вільного місця на жорсткому диску;
– операційна система: серверна операційна система, наприклад, Windows Server або дистрибутив Linux, наприклад, Ubuntu Server;
– веб-сервер: програмне забезпечення, таке як Apache, Nginx або IIS для розміщення веб-сайту;
– сервер бази даних: якщо веб-сайту потрібна база даних, знадобиться таке програмне забезпечення, як MySQL або PostgreSQL;
– підключення до мережі: швидке і стабільне інтернет-з'єднання необхідне для того, щоб веб-сайт був доступний для користувачів.

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для коректної роботи програми необхідно, щоб програмне забезпечення на комп'ютері, на якому буде функціонувати веб-орієнтована підсистема, відповідало наступним вимогам:

- операційна система повинна бути Unix, Linux або Microsoft Windows XP/7/8/10;
- потрібен браузер, наприклад, Microsoft Internet Explorer, Mozilla Firefox, Opera або Google Chrome;

Веб-орієнтована інформаційна система має бути реалізовано на мові програмування Python, HTML5 і CSS для верстання веб-сторінок, бібліотеки Django для написання клієнтських сценаріїв, бібліотеки Angular для написання клієнтської частини та СУБД PostgreSQL.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Функціональне призначення веб-сайту для магазину одягу полягає в тому, щоб надати клієнтам онлайн-платформу для перегляду, вибору та придбання предметів одягу. Веб-сайт повинен дозволяти покупцям шукати конкретні товари, фільтрувати результати за категоріями. Він також повинен містити детальні описи товарів та високоякісні зображення, щоб допомогти покупцям прийняти обґрунтоване рішення про покупку.

Функціональне призначення системи інтернет-магазину включає:

- реєстрація та обліковий запис користувача: дозволяє користувачам створювати облікові записи, зберігати інформацію про адресу, історію замовлень і переглядати статус замовлень;
- категорії та підкатегорії товарів: допомагає організувати товари у логічні групи для легкого навігації та пошуку;
- картки товарів з детальним описом, зображеннями та цінами: надають повну інформацію про товар, його характеристики, варіанти кольорів або розмірів, фотографії та ціни;
- корзина покупок: Дозволяє клієнтам додавати товари до корзини, переглядати вміст, редагувати кількість товарів та переходити до оформлення замовлення;
- обробка платежів і системи оплати: забезпечує безпечну оплату товарів за допомогою різних платіжних методів, таких як кредитні картки, електронні гроші або платіжні шлюзи;
- безпечність та захист персональних даних клієнтів: забезпечує захист конфіденційної інформації клієнтів, такої як особисті дані та інформація про платежі.

Для покращення клієнтського досвіду веб-сайт може також включати такі функції, як персоналізовані рекомендації, списки побажань та рекламні пропозиції [8].

2.2. Опис використаних технологій та мов програмування

Оновлені вимоги передбачають використання фронтенд-технологій HTML, CSS та Angular для розробки клієнтської частини веб-додатку. Бізнес-логіка додатку буде реалізована з використанням Django - фреймворку на мові програмування Python. Для зберігання та управління даними використовуватиметься система керування базами даних PostgreSQL.

Для розробки веб-додатку було використано наступні технології та мови програмування:

- Python: мова програмування високого рівня, що використовується для серверного програмування та веб-розробки, була використана для розробки внутрішньої логіки сайту;
- JavaScript: мова сценаріїв, що використовується для програмування на стороні клієнта для додавання інтерактивності та функціональності веб-сайту, використовувалася для реалізації динамічних функцій на веб-сайті;
- TypeScript: мова програмування, яка розширює JavaScript, додаючи до нього статичну типізацію. Вона розроблена для покращення розробки складних програм, забезпечуючи переваги типізації, контролю помилок та підтримки сучасних функцій JavaScript.
- HTML: мова розмітки, що використовується для створення структури та змісту веб-сторінок;
- CSS: мова таблиць стилів, що використовується для стилізації зовнішнього вигляду веб-сторінок;

– PostgreSQL: реляційна система управління базами даних, що використовується для зберігання даних, пов'язаних з продуктами, клієнтами, замовленнями тощо.

Для полегшення процесу розробки було використано декілька бібліотек та фреймворків для вищезгаданих мов програмування:

– Django - це високорівневий веб-фреймворк на Python, який сприяє швидкій розробці та чистому дизайну веб-додатків;
– Angular - це потужний фреймворк для розробки веб-додатків, який дозволяє створювати складні, багатофункціональні інтерфейси користувача з багатими функціональними можливостями;
– Django REST framework: потужний і гнучкий інструментарій для створення веб-аплікаторів на мові Python.

Якщо є проект з обмеженим часом на його виконання, використання Django може бути розумним вибором для його реалізації. Крім того, Django має можливість налаштування конфігурації користувача, що дозволяє значно прискорити процес розробки. Раніше, коли створення програмного коду займало багато часу, ця можливість дозволила значно спростити процес розробки, знизивши вартість розробки практично на 80%.

Переваги Django :

– він був створений з метою допомогти розробникам створювати додатки якомога швидше. Це охоплює весь процес, від ідеї до випуску проєкту, де значно економиться час та ресурси на кожному етапі. Тому це ідеальний вибір для розробників, в яких питання дедлайнів на першому місці;
– забезпечує повну функціональність, яка включає десятки додаткових функцій, таких як аутентифікація користувача, карти сайту, управління вмістом, RSS і багато іншого. Ці функції допомагають значно полегшити кожен етап процесу веб-розробки;
– дає високий рівень безпеки для вашого проекту, допомагаючи уникнути помилок, які можуть порушити безпеку вашого додатку. Він має

ефективні механізми логінів та паролів, а також систему користувальницької аутентифікації, що дозволяє забезпечити безпеку вашого проекту на високому рівні;

- це ідеальне рішення для проектів з великим трафіком, оскільки фреймворк надійно підтримує високу масштабованість. Тому багато веб-сайтів з великим обсягом трафіку використовують його для задоволення вимог щодо обробки великого потоку користувачів;
- є універсальним рішенням, яке може бути застосовано в різних сферах, таких як управління контентом, наукові обчислювальні платформи або навіть великі організації.

Незважаючи на багато переваг, у фреймворку Django також є деякі недоліки. Ось деякі з них:

- може бути складним для вивчення, особливо для початківців. Хоча існує багато ресурсів для навчання, вивчення фреймворку може зайняти час і зусилля;
- зазвичай працює дуже добре для типових веб-сайтів, але може бути важко налаштувати його для більш складних або нестандартних проектів;
- вимагає відносно великої кількості ресурсів, щоб працювати швидко та ефективно, зокрема вимоги до обсягу оперативної пам'яті [9].

Angular - це фреймворк JavaScript, створений для розробки складних та потужних веб-сайтів, веб-додатків для мобільних платформ (iOS та Android) та систем. Він був розроблений Google та має відкритий вихідний код. Є одним з найбільш популярних фреймворків, який надає універсальне інтерфейсне середовище розробки з шаблоном для проєктування MVC. Він побудований на мові програмування TypeScript, що дозволяє створювати продуктивні додатки з меншою кількістю помилок.

Переваги:

- використовує новий стандарт веб-компонентів, що відрізняється від закритої системи модуляризації в AngularJS. Це означає, що з Angular

можна безпосередньо використовувати будь-який компонент, написаний як веб-компонент, не потребуючи коду верстки;

- TypeScript надає значні комерційні переваги завдяки своєму багатому інструментарію. Ця мова дозволяє використовувати сучасні інструменти автозаповнення, навігації та рефакторінгу, що стають незамінними при роботі з великими проектами;
- не проводить глибокого порівняльного аналізу об'єктів, тобто якщо в масиві даних додати новий елемент, то зміна не буде автоматично відображена в View, якщо властивості об'єкта не пов'язані безпосередньо з View;
- Angular CLI є потужним інструментом, який дозволяє створювати нові проекти за допомогою вбудованого шаблонування та автоматизовувати багато завдань, таких як генерація компонентів та модулів, встановлення залежностей та більш ефективне управління проектами. Це дозволяє розробникам швидко створювати високоякісні додатки, що відповідають сучасним вимогам ринку.

Недоліки:

- важко оволодіти: якщо немає досвіду роботи з TypeScript, то буде потрібно витратити певний час на його вивчення;
- regular DOM: прямо працює з DOM, що може знизити швидкість та ефективність в деяких випадках;
- складнощі при серверної шаблонізації: у Angular фізично зберігається розділення між кодом JavaScript, який відповідає за логіку додатку, та HTML-кодом, який відповідає за відображення додатку на стороні клієнта [10].

PostgreSQL - це потужна та надійна об'єктно-реляційна база даних з відкритим вихідним кодом, яка стала популярною серед розробників та підприємств завдяки своїм перевагам.

Одна з найвагоміших переваг - це його розширені можливості. Він має багато функцій, які дозволяють розширювати та змінювати функціональність

бази даних за допомогою власних доповнень, що робить його більш гнучким та адаптивним до різних потреб користувачів.

Також PostgreSQL має потужну систему забезпечення безпеки та захисту даних. Вона включає в себе такі функції, як захист паролів, шифрування даних, мережевий доступ, контроль доступу до бази даних, аудит тощо.

Ця база є високопродуктивною, яка може оптимально працювати з великими обсягами даних. Він має підтримку оптимізації запитів, в тому числі підтримує індекси, матеріалізовані види, агрегацію та інші функції, що дозволяють розширити можливості для ефективної роботи з даними. Крім того, PostgreSQL підтримує реплікацію та кластеризацію, що дозволяє підвищити надійність та масштабованість бази даних.

В загальному, PostgreSQL - це міцна, гнучка та потужна база даних з відкритим вихідним кодом, яка дозволяє розробникам створювати надійні та ефективні рішення для зберігання та роботи з даними [11].

Архітектурний шаблон MVC (рис. 2.1):

– VC (View-Controller) - це набір архітектурних ідей та принципів, що використовуються для побудови складних інформаційних систем з користувацьким інтерфейсом;
– MVC (Model-View-Controller) - це абревіатура, яка означає певну архітектурну модель для розробки програмного забезпечення;

У MVC модель програми розбивається на три основні компоненти: модель (Model), яка відповідає за обробку даних та бізнес-логіку, вигляд (View), який відображає дані користувачу та забезпечує його взаємодію з системою, та контролер (Controller), який керує потоком даних та взаємодіє з моделлю та виглядом.

Головна ідея MVC полягає в тому, щоб розділити логіку програми на ці компоненти, щоб забезпечити розподіл відповідальностей та полегшити розробку, тестування та підтримку коду. Вигляд не взаємодіє безпосередньо з моделлю, а використовує контролер для доступу до даних. Це забезпечує

відокремлення моделі даних від представлення даних та дозволяє вносити зміни в окремі компоненти, не впливаючи на решту системи.

Модель в програмуванні в основному містить чисті дані, без включення логіки, яка визначає, як ці дані будуть представлені користувачеві. Вона фактично є простим об'єктом або класом, який використовується для передачі даних усередині додатку, наприклад, між серверною частиною, інтерфейсом користувача та базою даних.

Представлення в програмуванні, наприклад, у веб-розробці, відповідає за відображення даних моделі користувачеві. Воно має доступ до даних, які знаходяться в моделі, але не знає, що саме ці дані означають або як користувач може з ними взаємодіяти. Представлення просто відображає ці дані на екрані або в іншому форматі, наприклад, в HTML-сторінках. Воно відповідає за візуальне представлення інформації з моделі без прив'язки до логіки або маніпуляцій над цими даними.

Контролер в програмуванні, особливо у веб-розробці, знаходиться між представленням і моделлю. Його основна функція - виконання фактичної бізнес-логіки. Контролер прослуховує події, які викликаються представленням або іншими зовнішніми джерелами, і реагує на ці події відповідним чином. Зазвичай ця реакція включає виклик методу в моделі. Оскільки представлення і модель пов'язані за допомогою механізму повідомлень, результат цієї дії автоматично відображається у представленні.

Таким чином, контролер виступає як посередник між представленням і моделлю, виконуючи логіку обробки подій, обмін даними з моделлю та оновлення представлення відповідно до змін в моделі. Він дозволяє взаємодіяти з моделлю, виконувати розрахунки, обробляти запити користувача та забезпечувати правильну взаємодію між компонентами додатку [12].

Застосування MVC дозволяє розробникам ефективно працювати з окремими компонентами системи, полегшує розуміння коду та сприяє його повторному використанню. Він покращує організацію програмного забезпечення та сприяє підтримці принципу єдиної відповідальності.

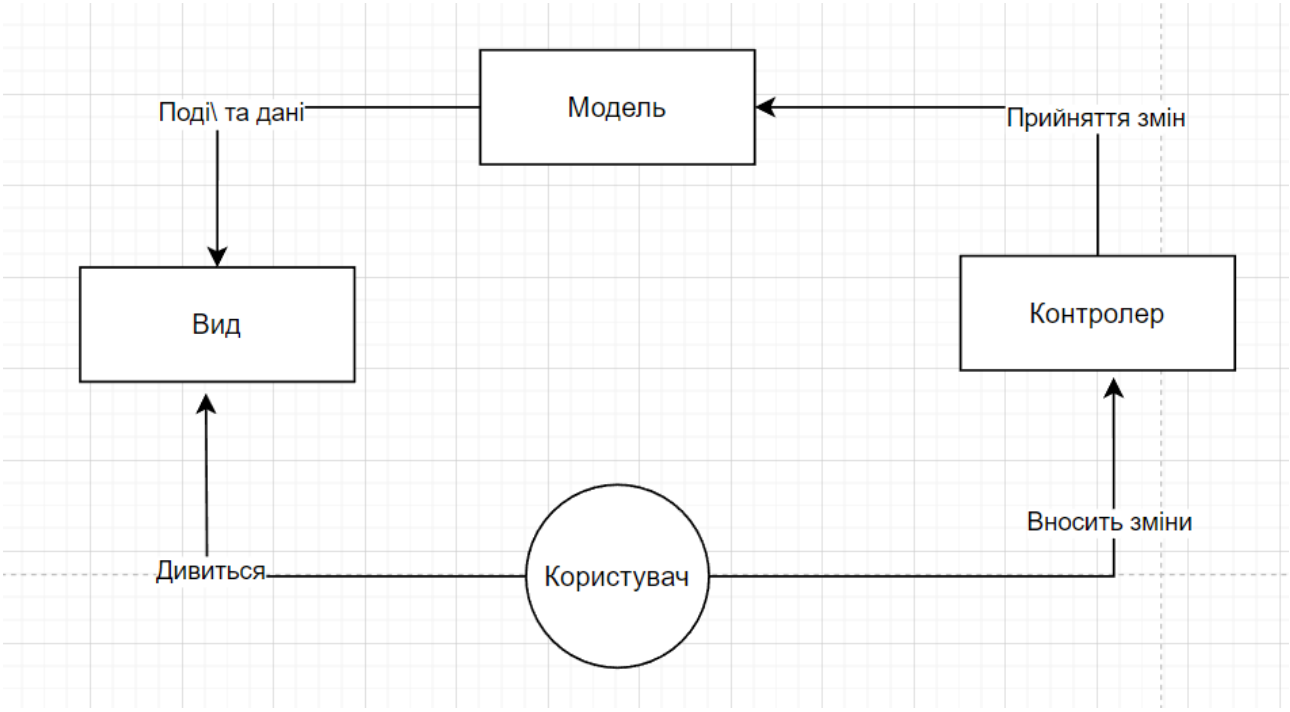

Рис. 2.1. Робота шаблону MVC

2.3. Опис структури системи та алгоритмів її функціювання

Логічна структура веб-сайту магазину одягу включатиме кілька компонентів, таких як інтерфейс, серверна частина та база даних. Фронтенд-компонент відповідатиме за представлення користувацького інтерфейсу клієнту та управління взаємодією з користувачем, наприклад, пошук та навігацію. Внутрішній компонент відповідатиме за бізнес-логіку та операції на стороні сервера, такі як аутентифікація, обробка замовлень та управління запасами. Компонент бази даних зберігатиме всю інформацію, пов'язану з продуктами, клієнтами, замовленнями та транзакціями.

Взаємозв'язок між цими компонентами передбачав би зв'язок інтерфейсу з бекендом через виклики API, який виконував би необхідну логіку та маніпуляції з даними, перш ніж надсилати відповідь назад до інтерфейсу. Бек-енд також взаємодіяв би з компонентом бази даних для отримання або зберігання даних за потреби.

Загалом, логічна структура веб-сайту магазину одягу має включати кілька компонентів, які працюють разом у скоординованій манері, щоб забезпечити безперебійний процес покупок для клієнтів. На рис. 2.2 можна побачити схему архітектури веб-додатку.

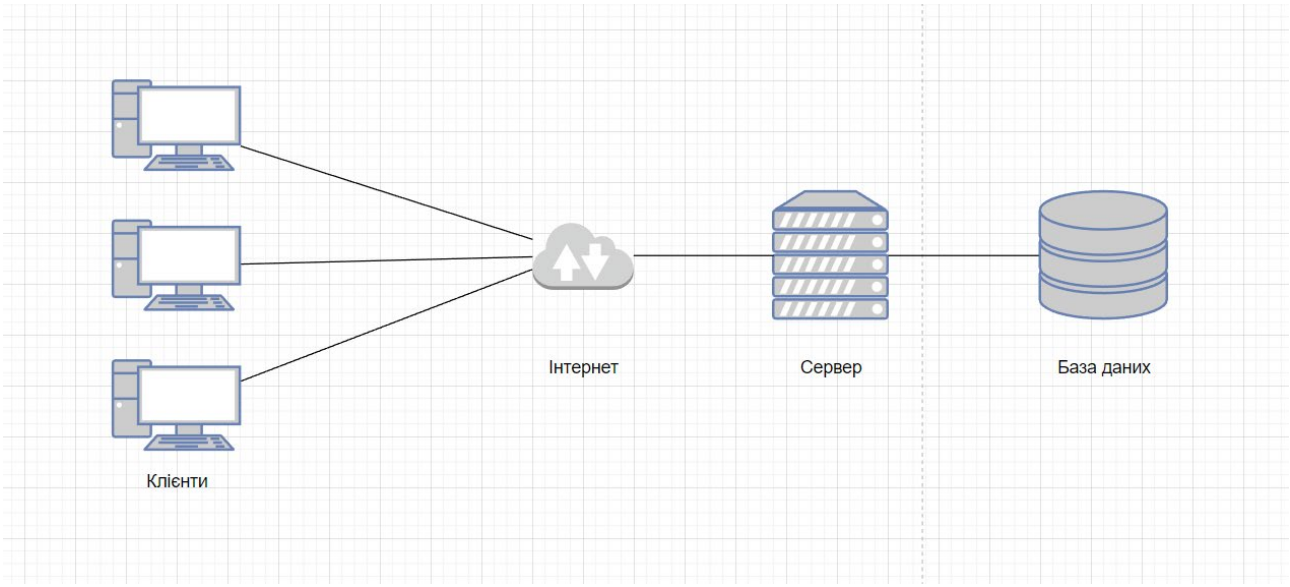

Рис. 2.2. Архітектура “клієнт-сервер”

Для розробки сайту для магазину одягу слід обрати відповідний алгоритм, який забезпечить ефективне та оптимальне функціонування програми. Розробка веб-додатку буде складатися з наступних етапів:

- розробка інтерфейсу користувача: веб-додаток повинен бути розроблений з простим і зручним інтерфейсом, який дозволяє клієнтам легко переміщатися по різних розділах сайту і шукати товари;
- проектування бази даних: ця бд повинна містити таку інформацію, як назви продуктів, описи, зображення, ціни та рівень запасів;
- формування функціоналу: повинен бути реалізований кошик для покупок, щоб клієнти могли додавати туди товари і переходити до оформлення замовлення.

Для обґрунтування обраних етапів були враховані наступні фактори:

- користувацький досвід: забезпеченння безперебійного та інтуїтивно зрозумілого користувацького досвіду, з простим і зручним інтерфейсом, який дозволяє клієнтам легко орієнтуватися на веб-сайті та знаходити те, що вони шукають;
- масштабованість: обробка великих обсягів трафіку та даних про товари без шкоди для продуктивності веб-сайту;

– безпека: заходи щодо забезпечення безпеки інформації про клієнтів;
– ефективність: мінімальний час завантаження і спрощений процес оформлення замовлення, щоб зменшити кількість покинутих кошиків.

Реалізація цих етапів надасть плавну та надійну роботу для користувачів, забезпечуючи в той же час можливість масштабування, високий рівень безпеки та ефективність веб-сайту. [13].

Структура бази даних для веб-сайту магазину одягу, включатиме таблиці (рис. 2.3):

– таблиця "Клієнти" - у цій таблиці зберігатиметься інформація про клієнтів магазину, включно з їхніми іменами, адресами, електронними адресами та номерами телефонів;
– таблиця товарів - у цій таблиці буде зберігатися інформація про товари, що продаються в магазині, включаючи їхні назви, описи, ціни та зображення;
– таблиця замовлень - у цій таблиці буде зберігатися інформація про замовлення, зроблені клієнтами, включаючи дату розміщення замовлення, клієнта, який зробив замовлення, і загальну вартість замовлення;
– таблиця позицій замовлення - У цій таблиці буде зберігатися інформація про окремі позиції в кожному замовленні, включаючи назву продукту, замовлену кількість і ціну за одиницю;
– таблиця категорій - у цій таблиці буде зберігатися інформація про різні категорії товарів, доступні в магазині, такі як "Чоловічий одяг", "Жіночий одяг" і "Аксесуари";
– таблиця "Кошик" - У цій таблиці буде зберігатися інформація про товари, додані до кошика покупця, включаючи ідентифікатор товару, кількість та ідентифікатор покупця.

Також дана база даних включає в себе таблиці, які були створені за допомогою фреймворку Django.

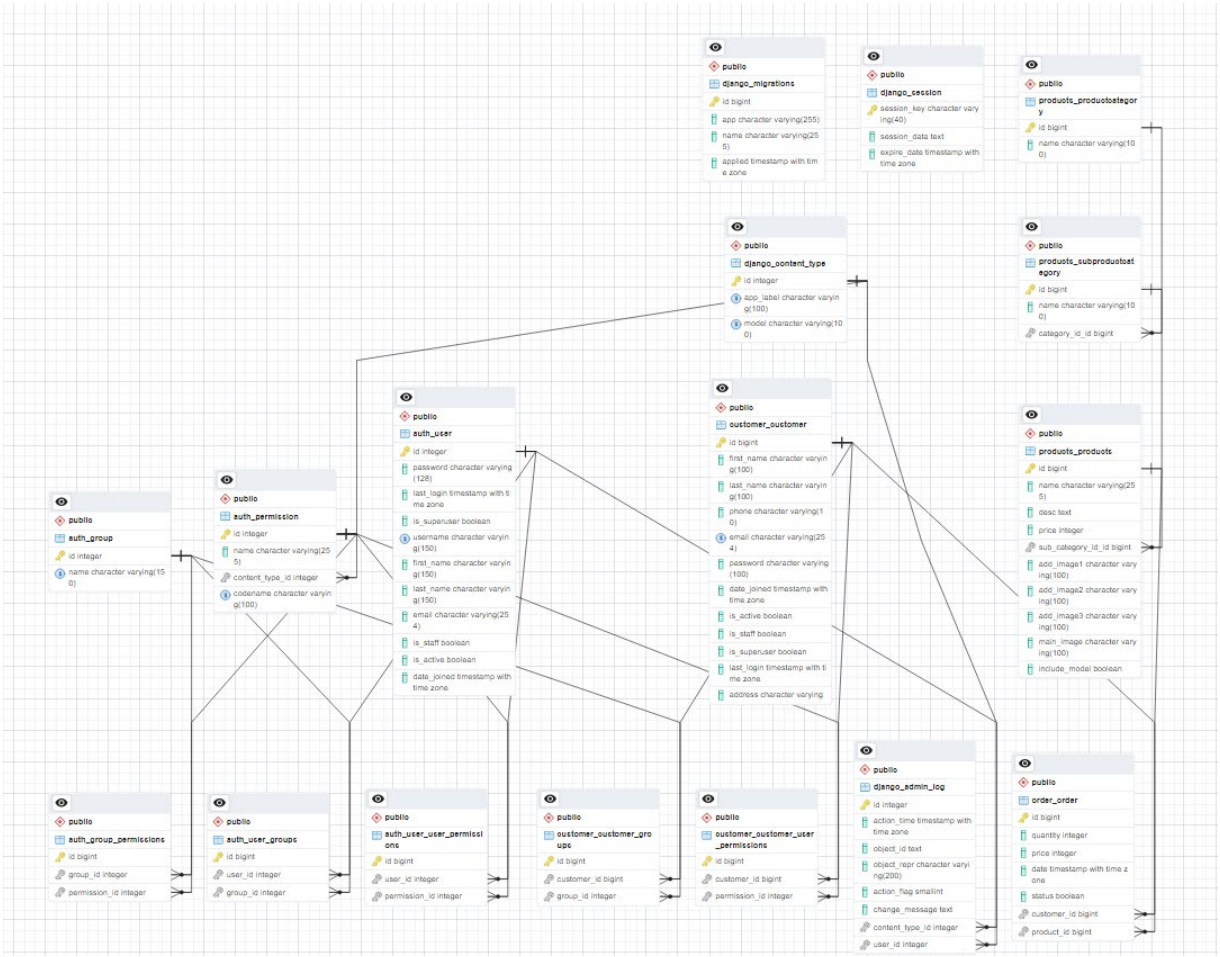

Рис. 2.3. Схема БД інтернет-магазину

В контексті веб-додатку для інтернет-магазину діаграма на рис. 2.4 відображає послідовність дій, які користувач виконує під час процесу реєстрації на сайті. Нижче наведено опис кожної активності на діаграмі:

– початок: процес реєстрації починається, коли користувач натискає кнопку "Create account" або аналогічний елемент на сторінці;
– введення даних: користувач вводить свої особисті дані, такі як ім'я, електронну пошту та пароль, відповідно до вказаних полів на формі реєстрації;
– перевірка валідності даних: система перевіряє введені дані на валідність. Це може включати перевірку правильності формату електронної пошти, унікальності користувача тощо;

– створення облікового запису: якщо введені дані валідні, система створює обліковий запис для нового користувача з використанням введених даних;
– збереження даних: створені дані про користувача зберігаються в базі даних для подальшого використання в системі;
– кінець: процес реєстрації завершується.

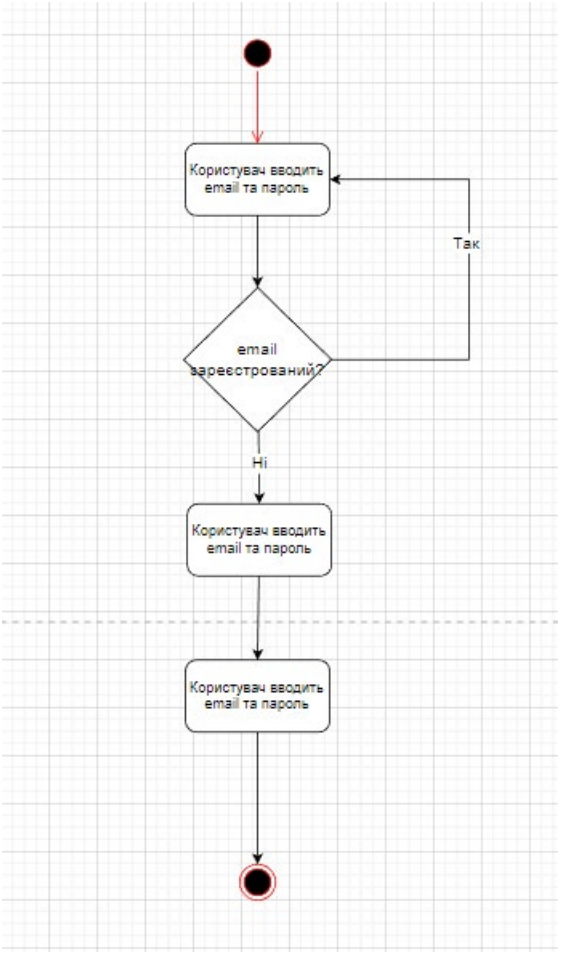

Рис. 2.4. Діаграма активності: реєстрація

В контексті веб-додатку для інтернет-магазину діаграма на рис. 2.5 відображає послідовність дій, які користувач виконує під час процесу авторизації на сайті. Нижче наведено опис кожної активності на діаграмі:

– початок: процес авторизації починається, коли користувач натискає кнопку "Sign in" або аналогічний елемент на сторінці;

– введення облікових даних: користувач вводить свої облікові дані, такі як електронна пошта та пароль, відповідно до вказаних полів на формі авторизації;
– перевірка валідності даних: система перевіряє введені дані на валідність. Це може включати перевірку правильності облікових даних та відповідність паролю до облікового запису;
– перевірка ідентифікації: якщо введені дані валідні, система перевіряє ідентифікацію користувача, перевіряючи, чи існує обліковий запис з введеними обліковими даними;
– авторизація: якщо ідентифікація пройшла успішно, система авторизує користувача, даючи токен авторизації та надаючи йому доступ до захищених ресурсів;
– кінець: процес авторизації завершується.

Рис. 2.5. Діаграма активності: авторизація

В контексті веб-додатку для інтернет-магазину діаграма на рисунку 2.6 відображає послідовність дій, які користувач виконує під час процесу покупки на сайті. Нижче наведено опис кожної активності на діаграмі:

- початок: процес покупки продукту починається, коли користувач вибирає продукт;
- додавання в кошик: обраний продукт додається до кошика користувача, який відображається на сторінці. Користувач може додати більше продуктів або перейти до оформлення замовлення;
- оформлення замовлення: користувач натискає кнопку "Cart" або аналогічний елемент для переходу до процесу оформлення замовлення;
- підтвердження замовлення: після перевірки наявності товару користувач підтверджує своє замовлення;
- завершення: процес покупки завершується і користувач може продовжити шопінг або вийти з системи.

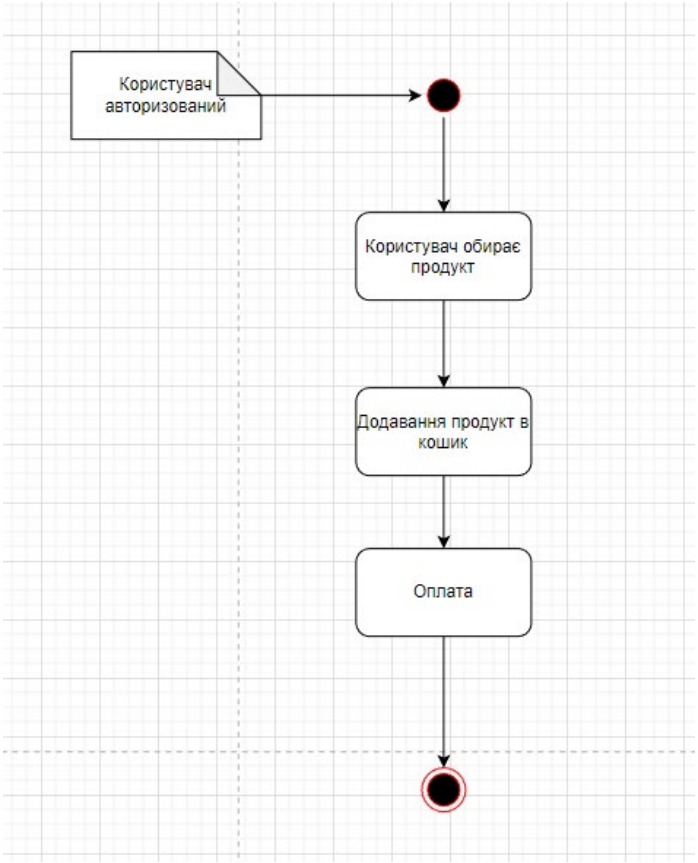

Рис. 2.6. Діаграма активності: покупка продукту

2.4 Обґрунтування та організація вхідних та вихідних даних програми

В рамках розробки веб-сайту для магазину одягу потрібно розглянути характер, організацію та попередню підготовку вхідних даних.

По-перше, вхідні дані будуть складатися з інформації про товари, які продає магазин одягу, включаючи такі деталі, як назви товарів, описи, ціни та зображення. Ці дані можуть бути зібрані з різних джерел, таких як постачальники, каталоги товарів, або введені вручну персоналом магазину.

По-друге, потрібно організувати вхідні дані у структурований спосіб, щоб система могла легко ними керувати та отримувати їх. Цього можна досягти, розробивши схему бази даних, яка включає таблиці для товарів, категорій, замовлень, клієнтів та інших відповідних сутностей. База даних може бути реалізована за допомогою реляційної системи управління базами даних, такої як PostgreSQL.

Загалом, характер, організація та попередня підготовка вхідних даних є вирішальними кроками у розробці веб-сайту для магазину одягу. Переконавшись, що вхідні дані точні, послідовні і добре організовані, можна створити надійну систему, яка відповідає потребам магазину і його клієнтів.

Щоб обґрунтувати та організувати вхідні та вихідні дані програми для веб-сайту магазину одягу, потрібно врахувати вимоги користувача та характер даних.

Вхідні дані:

- реєстраційна інформація користувача (ім'я, електронна пошта, номер телефону, адреса тощо);
- інформація про товар (назва, ціна, розмір, колір, опис тощо);
- інформація про запаси (кількість, місцезнаходження тощо);
- інформація про замовлення (інформація про клієнта, інформація про товар, інформація про оплату тощо).

Вхідні дані повинні збиратися за допомогою зручних для користувача інтерфейсів, таких як форми, меню, що випадають. Дані мають бути перевірені, щоб гарантувати їхню точність та узгодженість.

Вихідні дані:

- списки товарів (назва, ціна, зображення, опис тощо);
- підтвердження замовлень (номер замовлення, загальна ціна, дата доставки тощо);
- інформація про обліковий запис користувача (історія замовлень, особиста інформація тощо).

Вихідні дані повинні бути представлені в чіткій та організованій формі, з використанням відповідного форматування та стилю.

Для того, щоб організувати вхідні та вихідні дані програми, ми можемо використовувати систему управління базами даних для зберігання та отримання даних. База даних повинна бути масштабованою та ефективною, з відповідною індексацією та нормалізацією для зменшення надмірності та підвищення продуктивності.

Загалом, вхідні та вихідні дані програми повинні бути організовані таким чином, щоб вони були зручними та ефективними, щоб забезпечити позитивний користувацький досвід та покращити функціональність веб-сайту.

2.5 Опис розробленої системи

2.5.1 Використані технічні засоби

При розробці веб-додатку був використаний персональний комп’ютер з наступними характеристиками:

- процесор AMD Ryzen 5 4600 3.00 GHz;
- відеокарта Nvidia GeForce GTX 1650 Super;
- оперативна пам’ять 16 Гб;
- жорсткий диск SSD M2 на 500 Гб;

2.5.2 Використані програмні засоби

Для розробки веб-додатку використовувались наступні програмні засоби:

- Visual Studio Code - для верстки;
- JetBrains WebStorm - для написання фронтенд частини;
- JetBrains Pycharm - для написання бекенд частини;
- Postman - для тестування серверної частини;
- pgAdmin - для роботи з базою даних;
- Figma - для дизайну продукту.

Редактор коду Visual Studio Code (рис. 2.7).

Популярний редактор коду, який широко використовується для верстки веб-сторінок і веб-додатків. Ось деякі особливості VS Code, які роблять його зручним інструментом для верстки:

- HTML та CSS редактори: має вбудовані редактори з підсвічуванням синтаксису. Вони надають зручні функції, такі як автодоповнення тегів та властивостей, перевірка на помилки та підказки, що спрощують процес верстки;
- живий перегляд (Live Server): підтримує розширення, яке дозволяє запускати локальний веб-сервер для миттєвого перегляду змін у реальному часі. Ви можете бачити, як зміни в коді відображаються в реальному браузері без необхідності оновлювати сторінку вручну;
- плагіни для верстки: має широкий вибір плагінів для верстки, які надають розширені функції та інструменти. Наприклад, плагіни для автодоповнення коду, форматування, роботи з препроцесорами CSS, такими як Sass або Less, або плагіни для роботи з фреймворками, такими як Bootstrap або Tailwind CSS;
- інтеграція з Git: має вбудовану підтримку Git, що дозволяє легко керувати версіями вашого коду. Можна бачити зміни, виконувати коміти, злиття гілок та багато іншого, безпосередньо з редактора;

– відладка в браузері: VS Code підтримує можливість налагодження JavaScript коду в браузері, яке дозволяє знайти й виправити помилки в режимі реального часу [14].

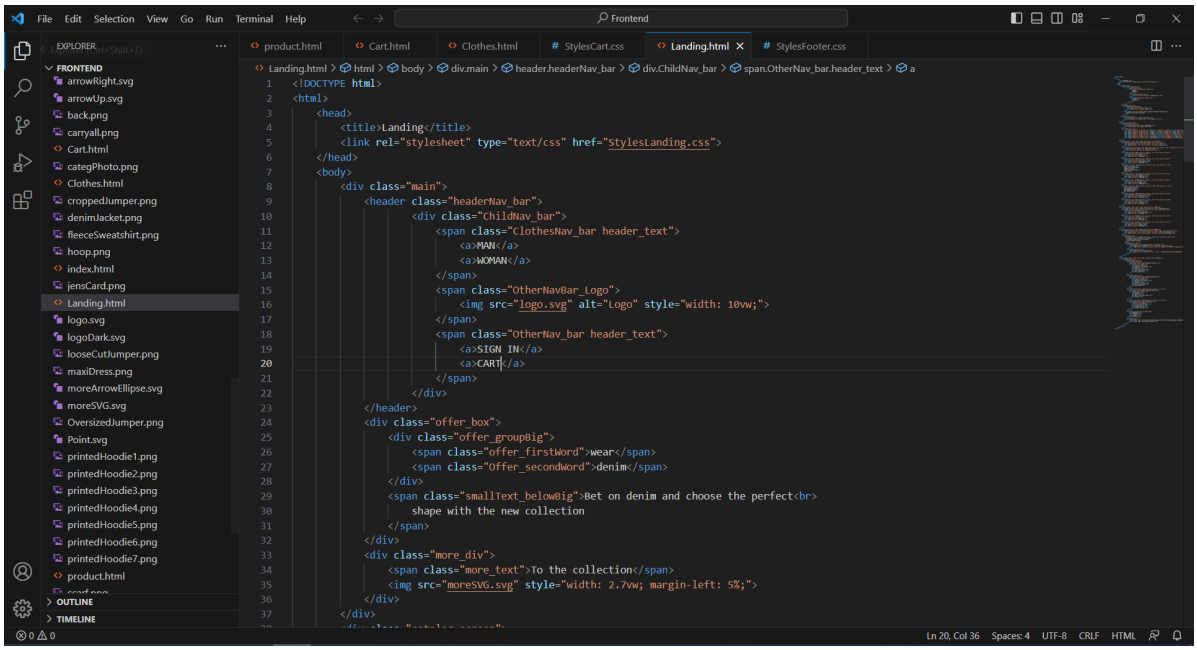

Рис. 2.7. Середовище Visual Studio code

JetBrains WebStorm (рис. 2.8).

Це інтегроване середовище розробки (IDE), спеціально призначене для розробки веб-додатків. Воно розроблене компанією JetBrains і володіє великим набором функцій, що допомагають розробникам ефективно працювати над проектами. Ось деякі особливості WebStorm:

– повна підтримка JavaScript, HTML та CSS: надає потужні редактори для JavaScript, HTML та CSS з функціями підсвічування синтаксису, автодоповненням, перевіркою помилок та багатьма іншими інструментами для полегшення верстки та розробки;
– перевірка помилок та відлагодження: IDE надає можливість виявляти та виправляти помилки в коді JavaScript, а також відлагоджати програми в реальному часі, встановлювати точки зупинки, спостерігати за значеннями змінних та аналізувати стек викликів;

– рефакторинг коду: він підтримує різні операції рефакторингу, такі як перейменування змінних, витягування функцій, оптимізація імпортів та багато іншого. Це допомагає зберігати код організованим та покращує його читабельність та підтримку;
– управління залежностями та пакетами: має інтегровану підтримку для пакетних менеджерів, таких як npm та Yarn, що спрощує встановлення, оновлення та керування залежностями проекту;
– інтеграція з системами контролю версій: підтримує популярні системи контролю версій, такі як Git, SVN та Mercurial, дозволяючи вам виконувати коміти, переглядати різницю змін, злиття гілок та багато ін.;
– інтелектуальне автодоповнення та навігація: надає потужні можливості автодоповнення коду, які допомагають розробникам швидше писати код і зменшити кількість помилок. Крім того, ви можете швидко переходити до визначення функцій, методів або змінних за допомогою функції "Go to Definition" або відкривати всі використання символу у проекті [15].

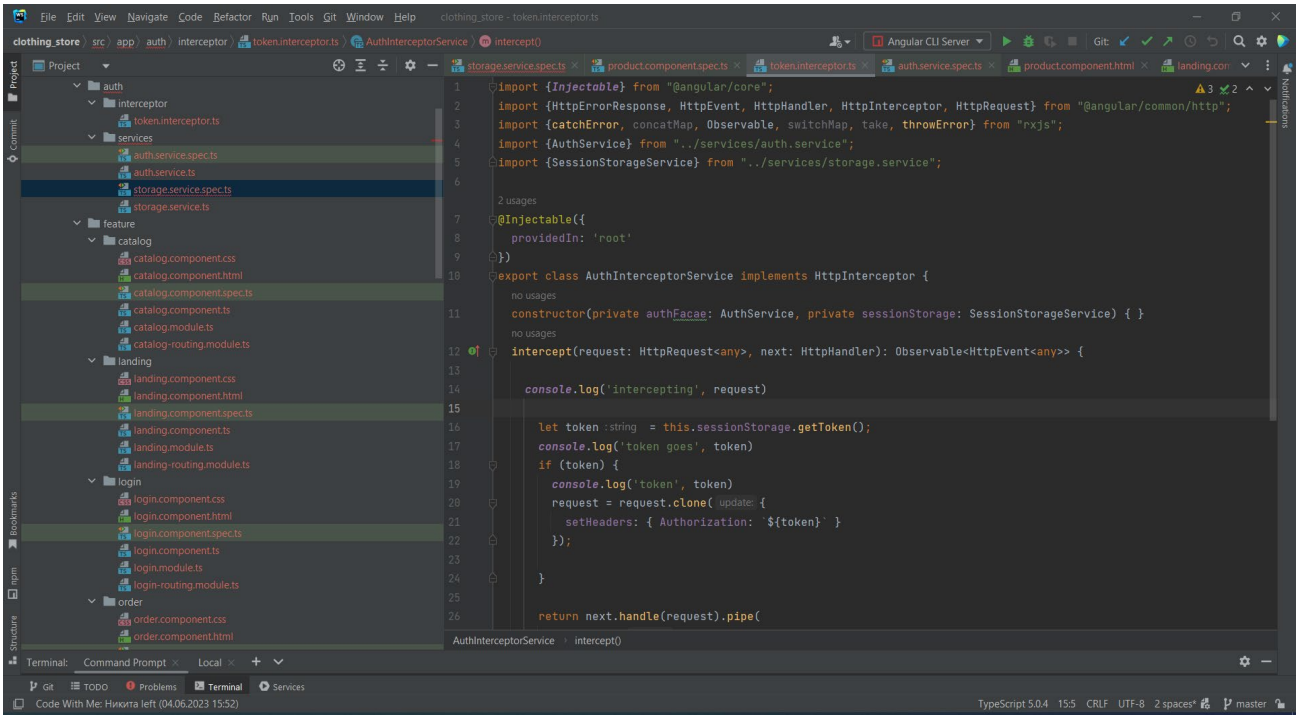

Рис. 2.8. Середовище розробки WebStorm

JetBrains Pycharm (рис. 2.9).

Це інтегроване середовище розробки (IDE), спеціально створене для розробки програм на мові Python. Воно розроблене компанією JetBrains і надає широкий набір функцій та інструментів, які полегшують розробку, налагодження та керування проектами Python. Ось деякі особливості PyCharm:

- розширена підтримка Python: надає повну підтримку Python, включаючи версії 2.x і 3.x. Він має потужний редактор коду з підсвічуванням синтаксису, автодоповненням, форматуванням, перевіркою помилок та іншими функціями, що полегшують написання коду на Python;
- умовне автодоповнення та аналіз типів: використовує вбудовану систему типізації Python для надання інтелектуального автодоповнення та аналізу типів. Він розуміє типи змінних, об'єктів, модулів та бібліотек, що дозволяє вам швидше писати код і зменшує кількість помилок;
- налагодження та профілювання: має вбудовані інструменти для відлагодження Python коду. Ви можете встановлювати точки зупинки, спостерігати за значеннями змінних, крокувати по коду, аналізувати стек викликів та використовувати інші відладочні функції. Крім того, він також надає засоби для профілювання коду, що допомагають виявити та оптимізувати швидкість виконання програми;
- управління залежностями та віртуальними середовищами: підтримує популярні менеджери пакетів, такі як pip та Conda, і надає зручний інтерфейс для керування залежностями проекту. Ви можете легко встановлювати, оновлювати та керувати пакетами залежностей вашого проекту. Крім того, PyCharm підтримує віртуальні середовища Python, що дозволяє вам ізолювати ваш проект від системного середовища та керувати версіями пакетів;
- інструменти для розробки веб-додатків: також надає підтримку для розробки веб-додатків з використанням фреймворків, таких як Django,

Flask, Pyramid тощо. Він має інтегрований сервер, що дозволяє запускати та налагоджувати ваш веб-додаток [16].

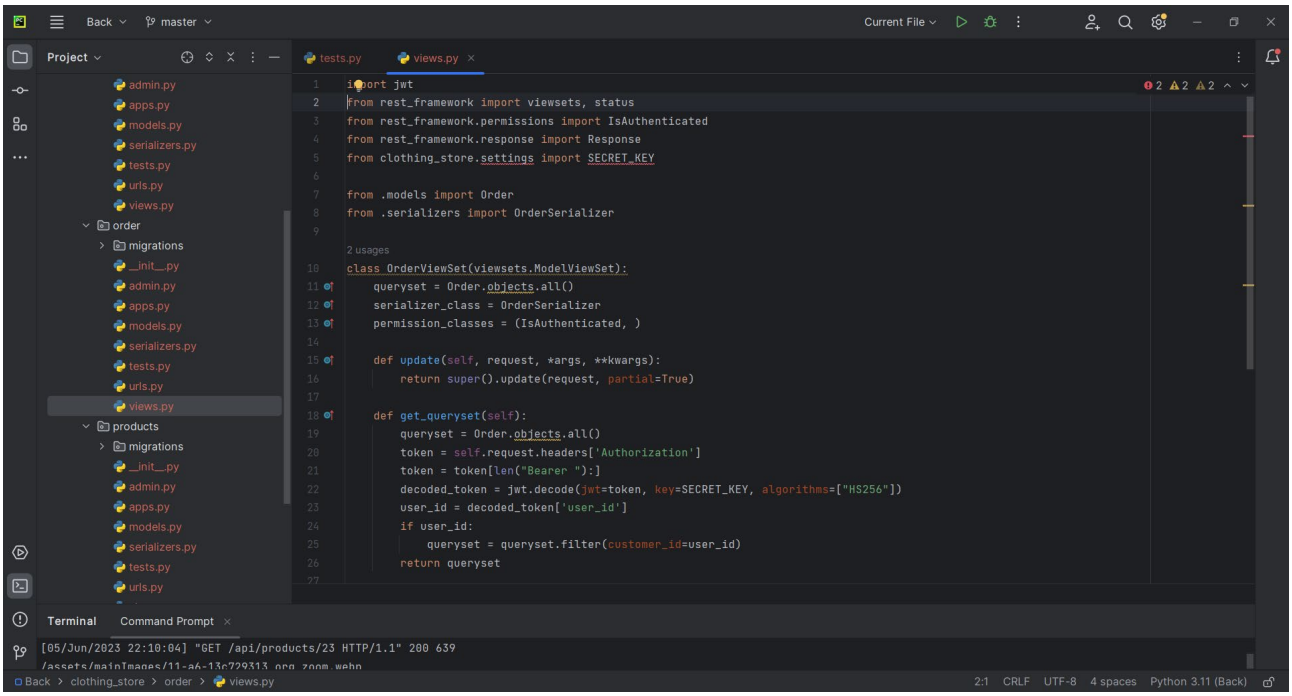

Рис. 2.9. Середовище розробки Pycharm

Postman (рис. 2.10).

Це популярний інструмент для розробки та тестування програмного забезпечення, призначений для взаємодії з API (інтерфейсом програмування додатків). Він надає потужні можливості для створення, відправлення та тестування HTTP-запитів до різних сервісів і додатків. Ось деякі особливості Postman:

– створення та надсилання HTTP-запитів: дозволяє створювати різні типи HTTP-запитів, такі як GET, POST, PUT, DELETE та інші. Ви можете налаштовувати заголовки, параметри запиту, тіло запиту (JSON, XML, форму) та інші параметри для взаємодії зі сторонніми API;
– організація запитів у колекції: дозволяє групувати запити в колекції, що спрощує організацію та керування вашими API-взаємодіями. Ви можете створювати папки, додавати описи та мітки до запитів, а також налаштовувати послідовність запитів у колекції;

– автоматизація та тестування API: дозволяє вам автоматизувати взаємодію з API за допомогою скриптів. Ви можете використовувати JavaScript для написання тестів, змінних та додаткової логіки, що допомагає автоматизувати тестування та перевірку відповідей API;
– отримання та обмін даними: дозволяє зручно отримувати та обмінюватися даними зі сторонніми сервісами. Ви можете витягувати дані з відповідей API за допомогою засобів для обробки JSON або XML, а також зберігати дані в змінних для подальшого використання у наступних запитах.

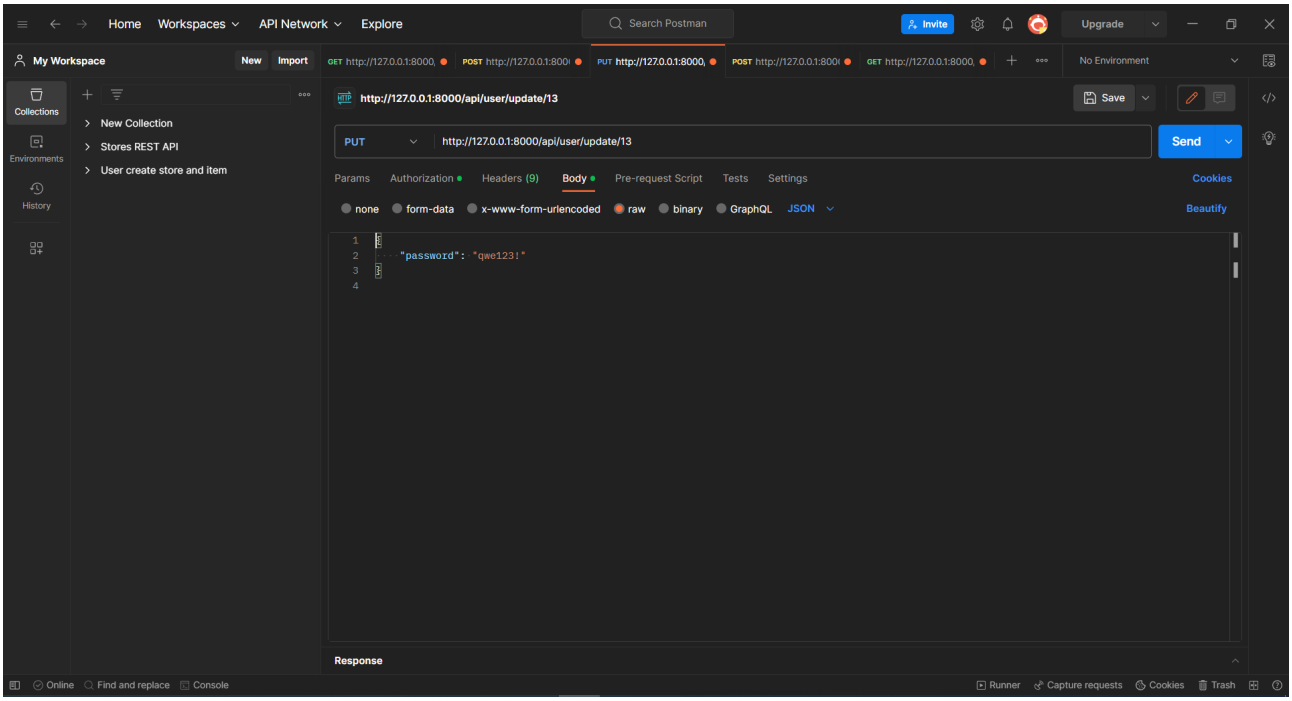

Рис. 2.10. Вікно програми Postman

pgAdmin (рис. 2.11).

Це безкоштовна графічна інтерфейсна програма для управління базами даних PostgreSQL. Вона надає зручні та потужні інструменти для адміністрування та розробки баз даних PostgreSQL. Ось деякі особливості pgAdmin:

– з'єднання з базою даних: дозволяє встановлювати з'єднання з різними серверами баз даних PostgreSQL. Можна налаштовувати параметри

підключення, такі як хост, порт, ім'я бази даних, ім'я користувача та пароль;

– графічний інтерфейс: має інтуїтивно зрозумілий графічний інтерфейс, що спрощує роботу з базами даних. Можна переглядати структуру бази даних, таблиці, схеми, індекси, переглядати та редагувати дані, виконувати SQL-запити та інші дії;
– схеми та таблиці: надає можливість створювати та керувати схемами бази даних, створювати таблиці, переглядати їх структуру, додавати, редагувати та видаляти записи у таблицях. Також є можливість керувати обмеженнями, індексами та відношеннями між таблицями;
– виконання SQL-запитів: надає зручний редактор для виконання SQL-запитів. Можна писати складні SQL-запити, виконувати їх та переглядати результати запитів у вигляді таблиць або графіків. Редактор також підтримує автодоповнення, синтаксичне виділення та інші корисні функції;
– моніторинг та логування: надає можливість моніторингу бази даних PostgreSQL. Можна переглядати активні з'єднання, виконувати запити до системних таблиць, аналізувати виконання запиті.

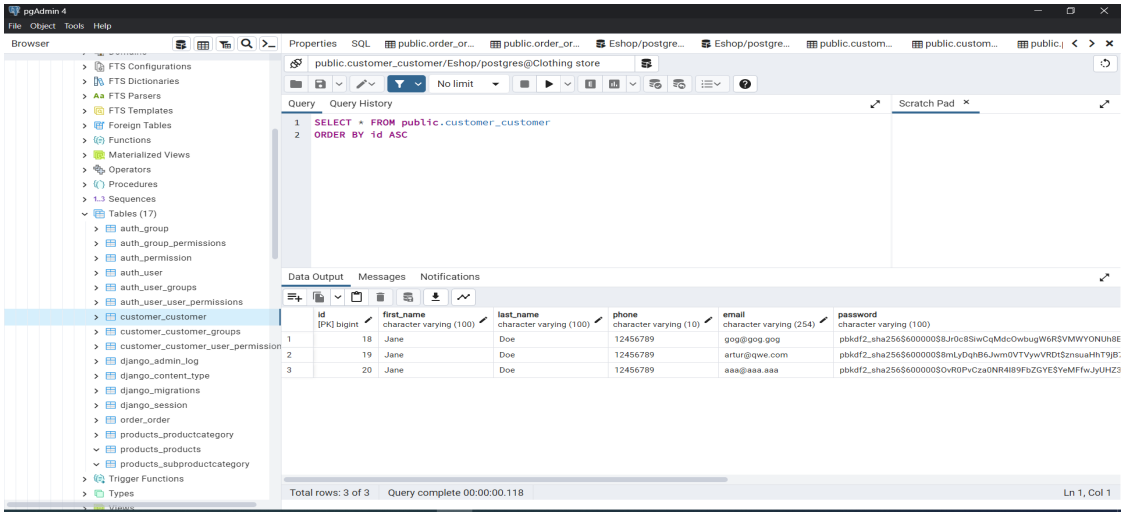

Рис. 2.11. Вікно програми pgAdmin

Figma (рис. 2.12).

Це веб-платформа та графічний редактор, призначений для розробки і дизайну інтерфейсів користувача (UI) та веб-додатків. Ось деякі особливості:

- онлайн-редагування та спільна робота: працює в браузері, що дозволяє вам редагувати та співпрацювати над проектами в реальному часі. Можна запрошувати інших користувачів, ділитися посиланнями та коментувати дизайни, що дозволяє команді працювати разом зручно та ефективно;
- векторний редактор: надає потужні можливості векторного редактора, що дозволяє створювати та редагувати графічні елементи. Можна малювати фігури, використовувати шляхи, додавати текст, застосовувати ефекти та інші векторні операції.
- компоненти та бібліотеки: дозволяє створювати компоненти, які можна повторно використовувати у вашому проекті. Ви можете створювати бібліотеки компонентів, що спрощує їх оновлення та синхронізацію у всьому проєкті. Це забезпечує консистентність дизайну та прискорює робочий процес;
- прототипування та анімація: надає можливість створювати інтерактивні прототипи вашого дизайну. Можна створювати переходи між сторінками, встановлювати взаємодію з елементами, додавати анімацію та інші взаємодійні ефекти. Це допомагає вам візуалізувати та тестувати дизайн перед реалізацією.

На рис. 2.12 можна побачити відповідно роботу з фігмою, в даному прикладі це сторінки лендінгу та каталогу.

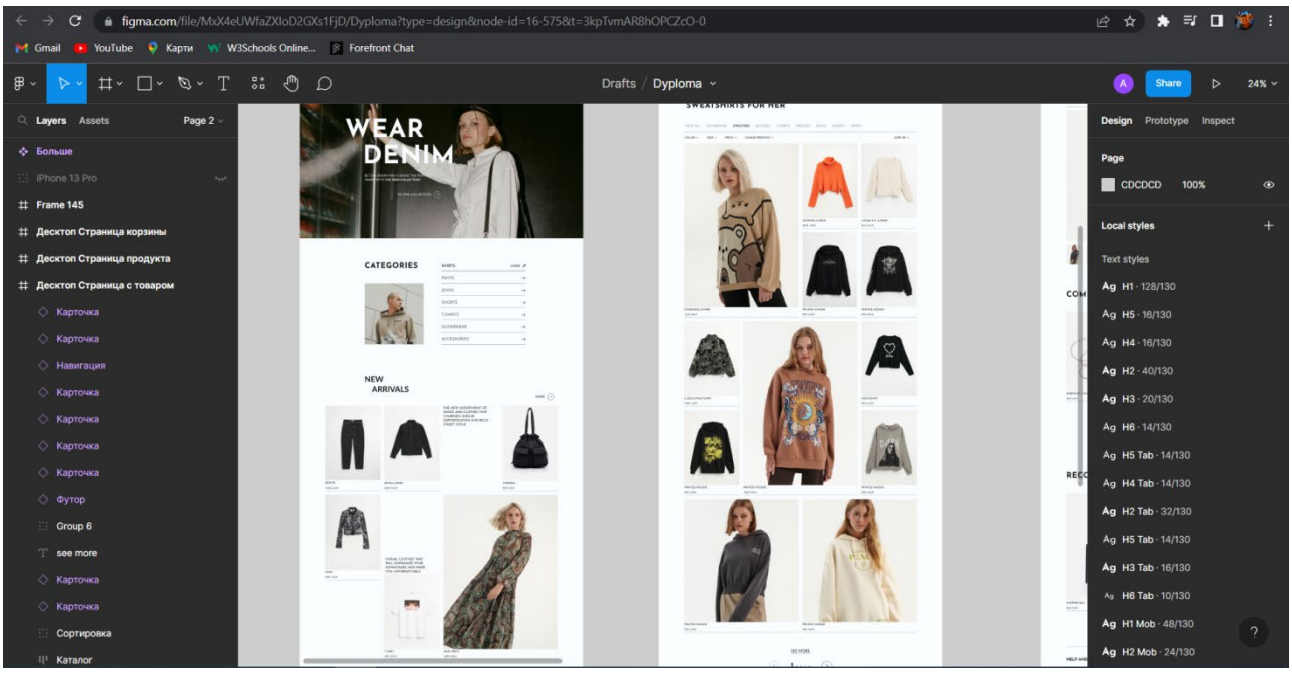

Рис. 2.12. Вікно програми Figma

2.5.3 Виклик та завантаження програми

Для запуску серверної частини потрібно з папки проєкту бекенду набрати команду py manage.py runserver, для клієнтської частини - з папки проєкту фронтенду запустити команду ng serve, таким чином обдві частини програми будуть підняті й з нею можна взаємодіяти через інтерфейс. Також в майбутньому можна буде використати якусь інфраструктуру або пптаформу для того, щоб там розвернути даний веб-додаток, наприклад: AWS, GCP або Microsoft Azure. Також цей додаток можна розширити для мобільних платформ: Android та IOS. API забезпечує спосіб взаємодії між додатком і сервером, незалежно від платформи, на якій працює програма. Для тестування серверної частини при розробці програми для інших пптаформ можна використовувати Postman.

2.5.4 Опис інтерфейсу користувача

При завантаженні веб-сайту, відвідувач автоматично перенаправляється на головну сторінку (рис. 2.13 - 2.17), де знаходиться списком доступних товарів та навігаційним меню сайту.

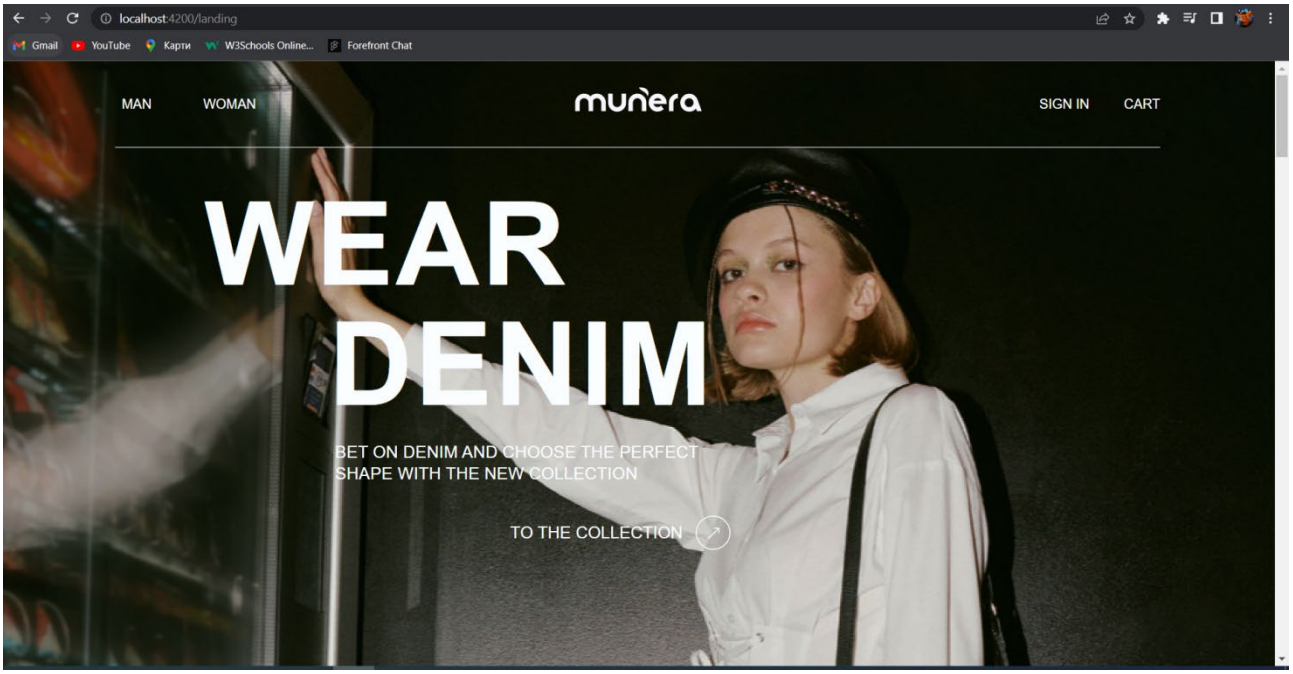

Рис. 2.13. Перша частина лендінгу

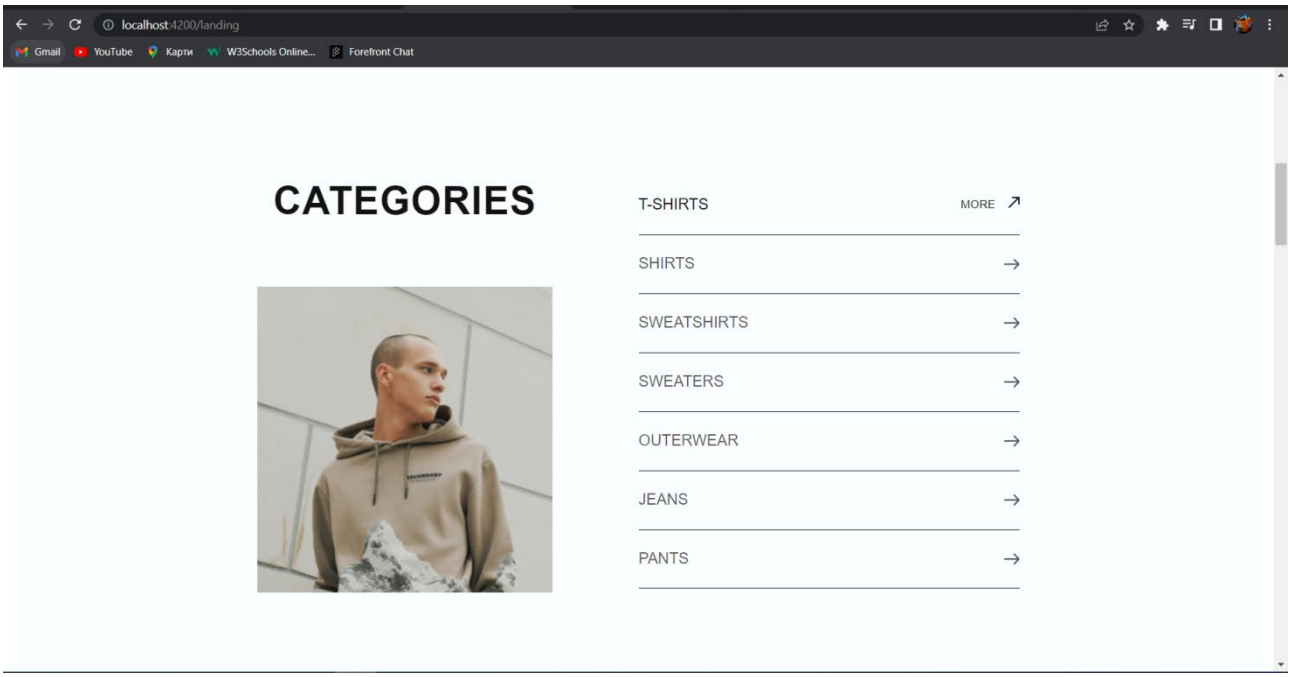

Рис. 2.14. Друга частина лендінгу

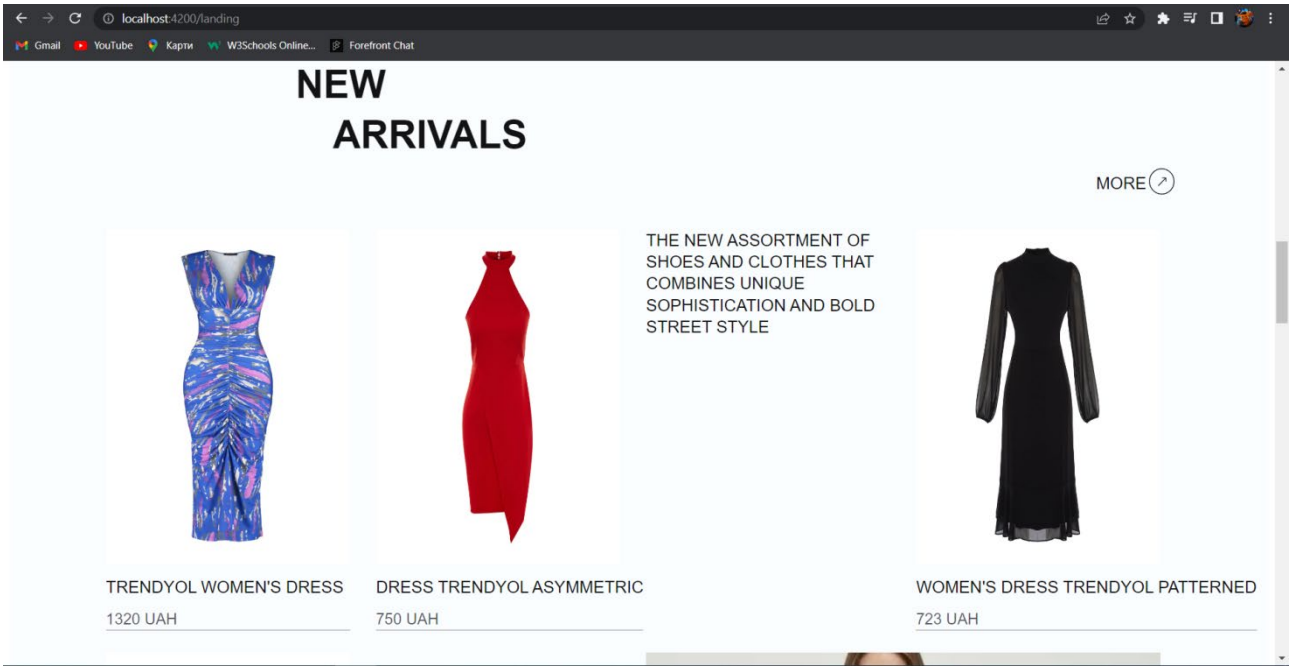

Рис. 2.15. Третя частина лендінгу

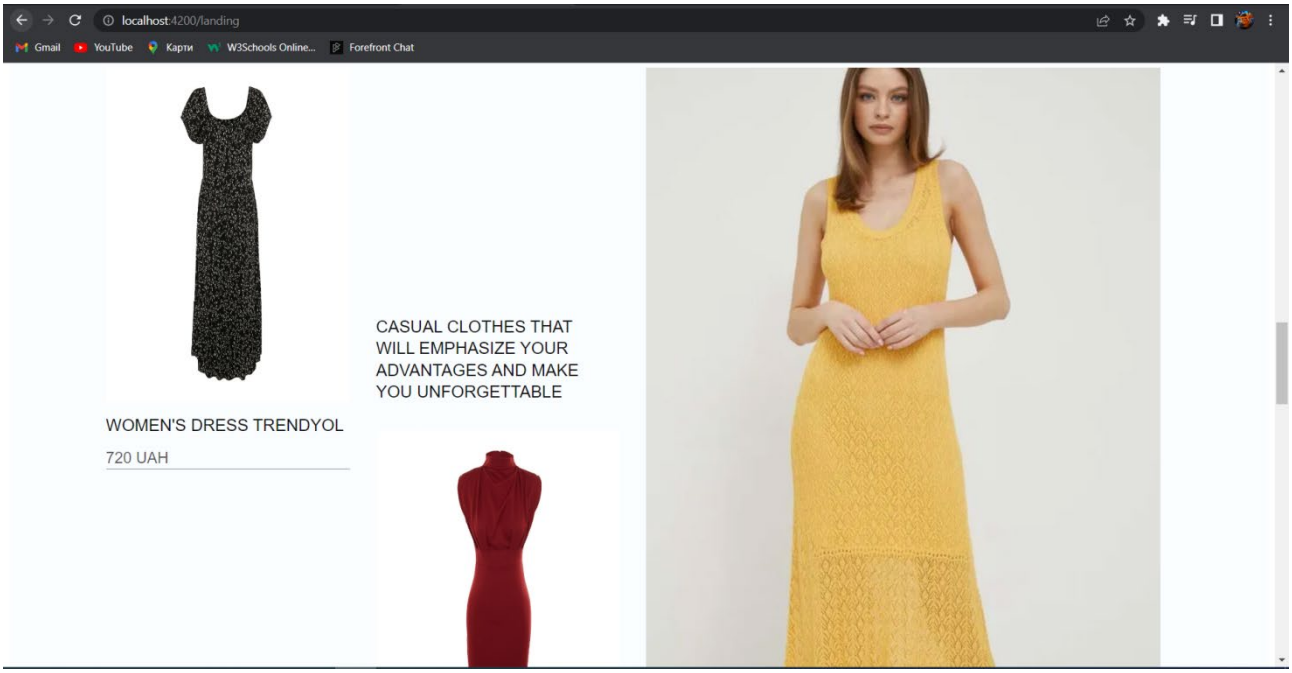

Рис. 2.16. Четверта частина лендінгу

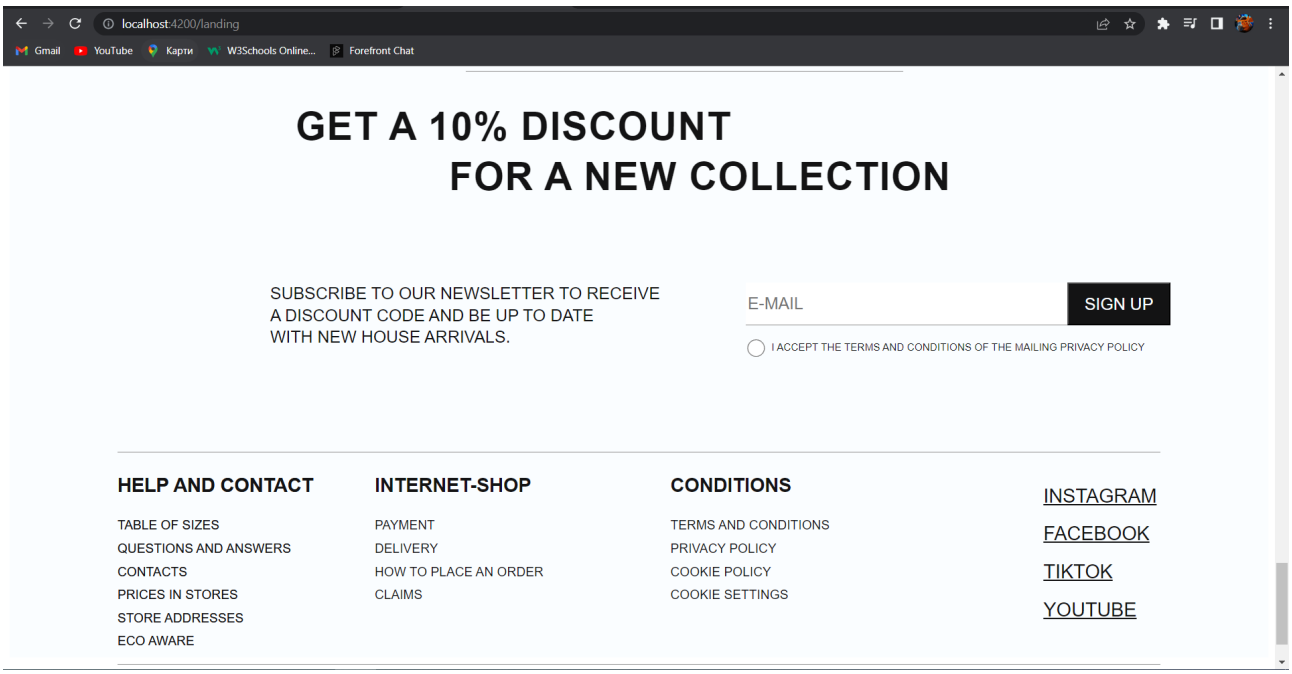

Рис. 2.17. П’ята частина лендінгу

На рис. 2.18 - 2.20 зображено сторінку з відповідними продуктами, які належать до відповідної категорії, в даному випадку - це blouses. Також можна побачити, що сторінка категорії не просто включає в себе картки з одягом, які розташовані по-порядку, але є великі картки, де користувач зможе одразу побачити людину з даною моделлю одягу. Поряд розташовані картки меншого розміру, на яких зображений одяг, але вже без людини.

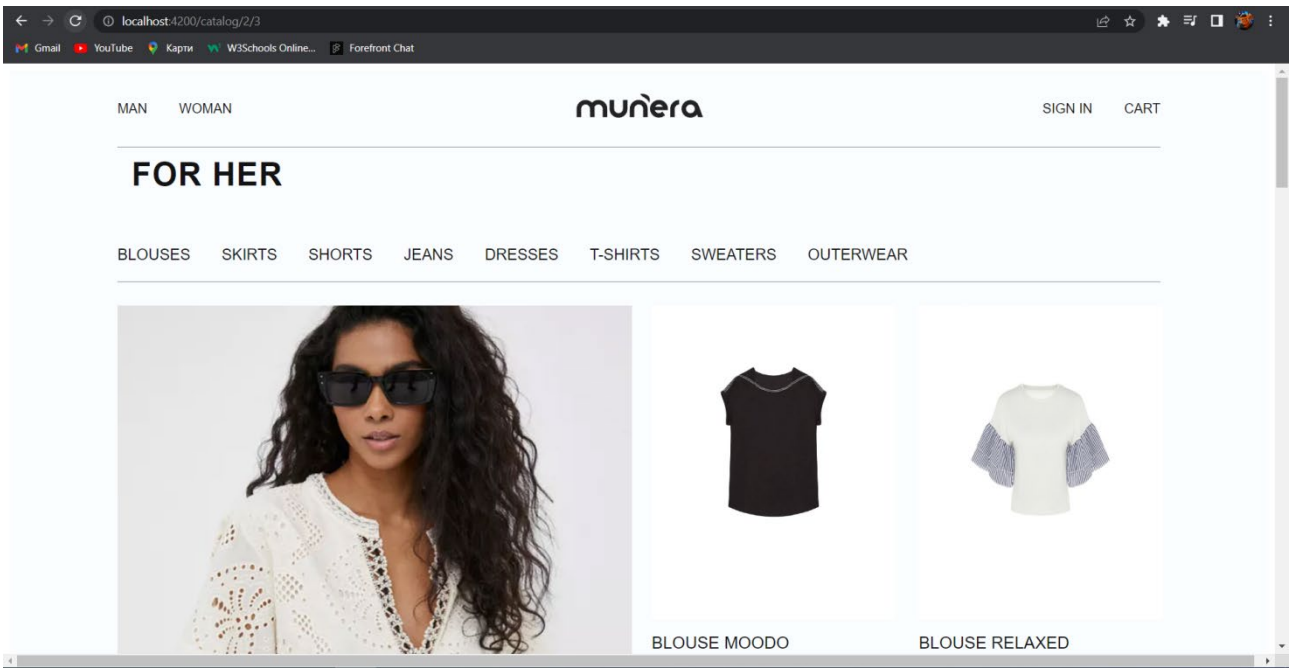

Рис. 2.18. Сторінка каталогу woman підкаталог blouses

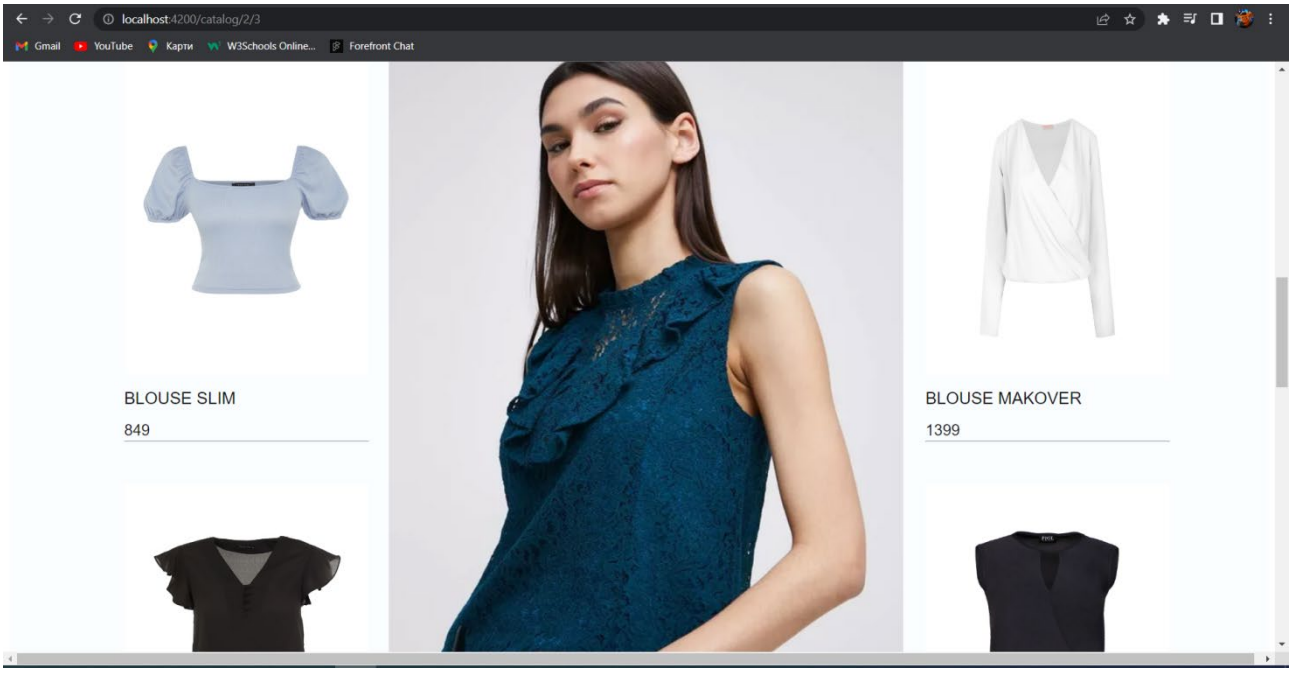

Рис. 2.19. Друга частина сторінки каталогу

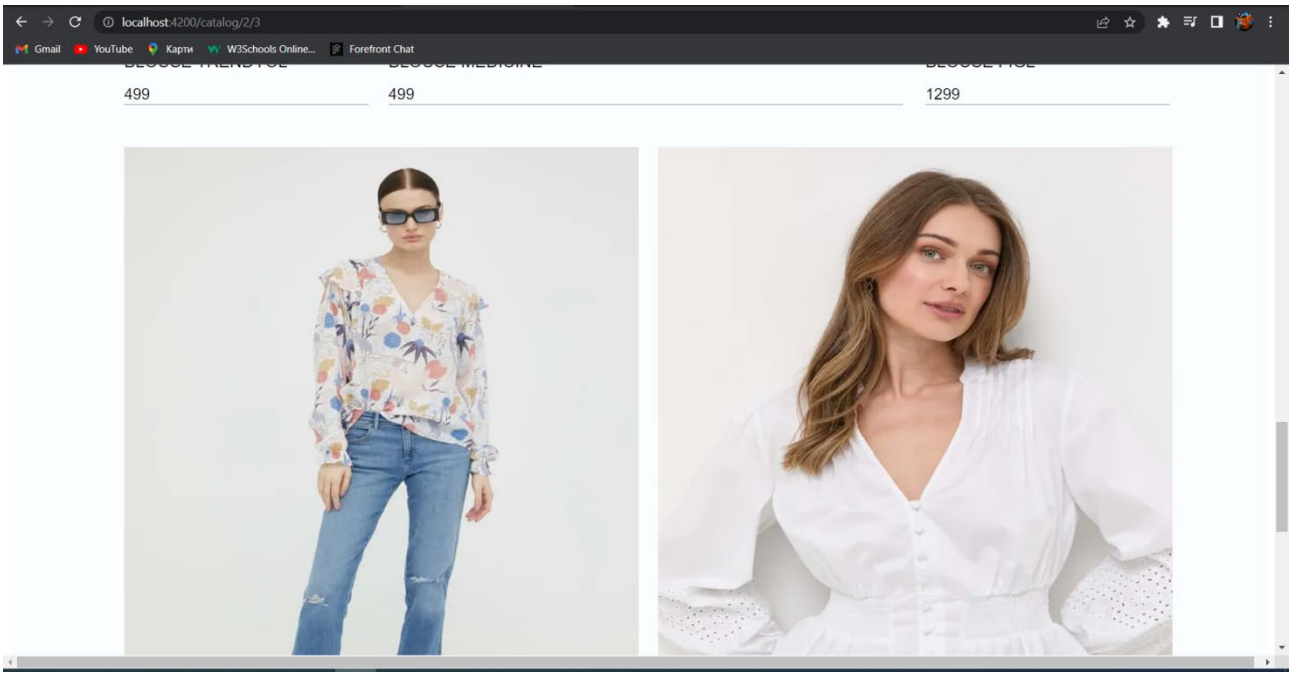

Рис. 2.20. Третя частина сторінки каталогу

Для того, щоб переглянути детальну інформацію та зображення обраного товару, необхідно клікнути на елемент, що відображає цей товар. Після цього відбудеться перехід на окрему сторінку, де буде доступна повна інформація про товар. На рисунках 2.21 - 2.22 зображено один з тих продуктів, які є в наявності.

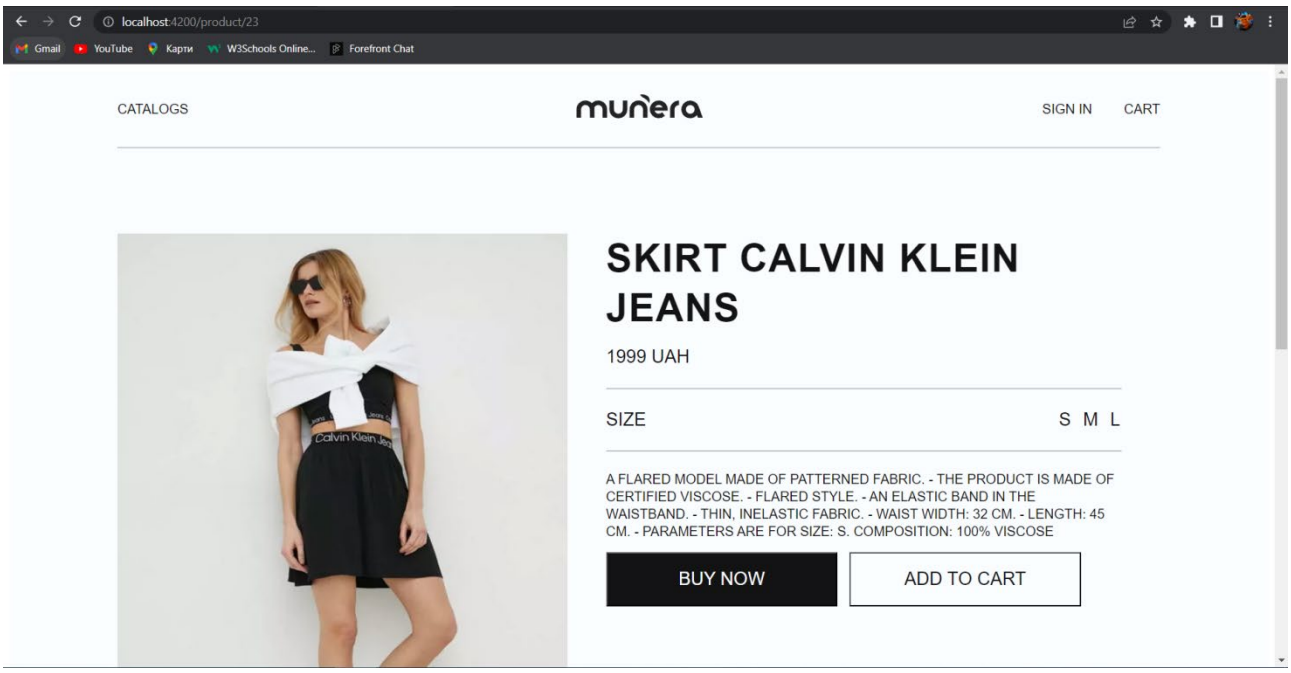

Рис. 2.21. Сторінка продукту

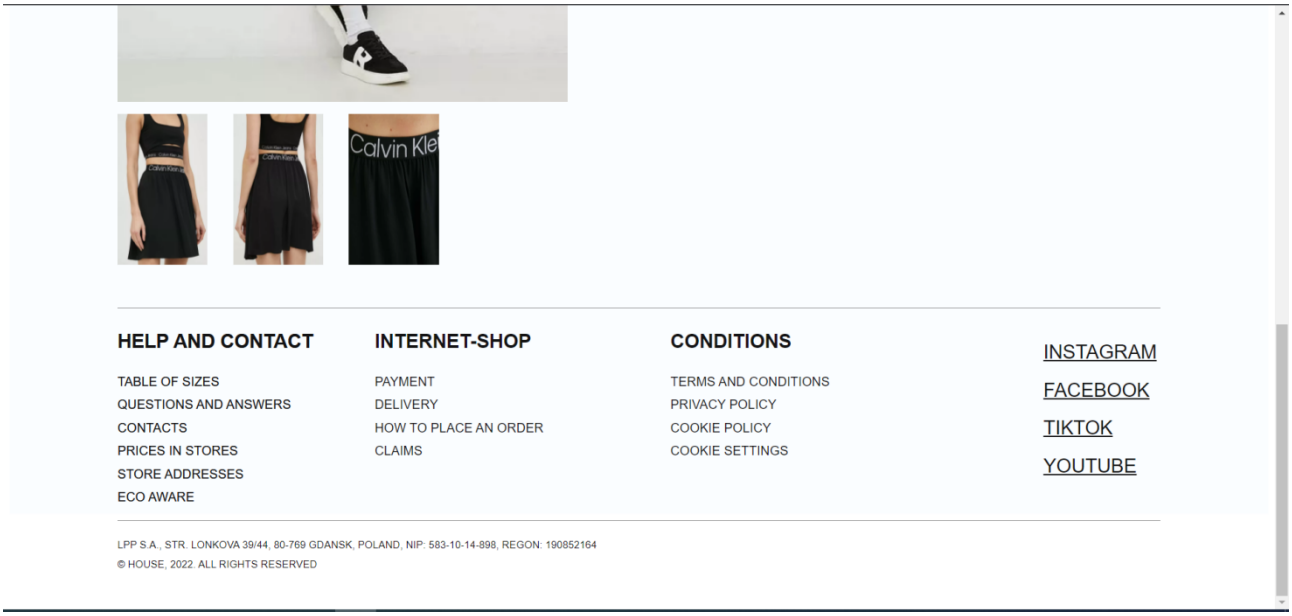

Рис. 2.22. Друга частина сторінки продукту

Відвідувач має можливість продовжити користування сайтом як гість або виконати авторизацію. Якщо відвідувач не авторизований, у правій частині хедеру буде присутній надпис sign in, яка призначений для переходу на сторінку з формою авторизації. На рис. 2.23 зображена сторінка авторизації. Якщо користувач не має акаунту в данному веб-додатку, то під кнопкою авторизації

знаходиться кнопка для створення нового акаунту. На рис. 2.24 зображена сторінка реєстрації.

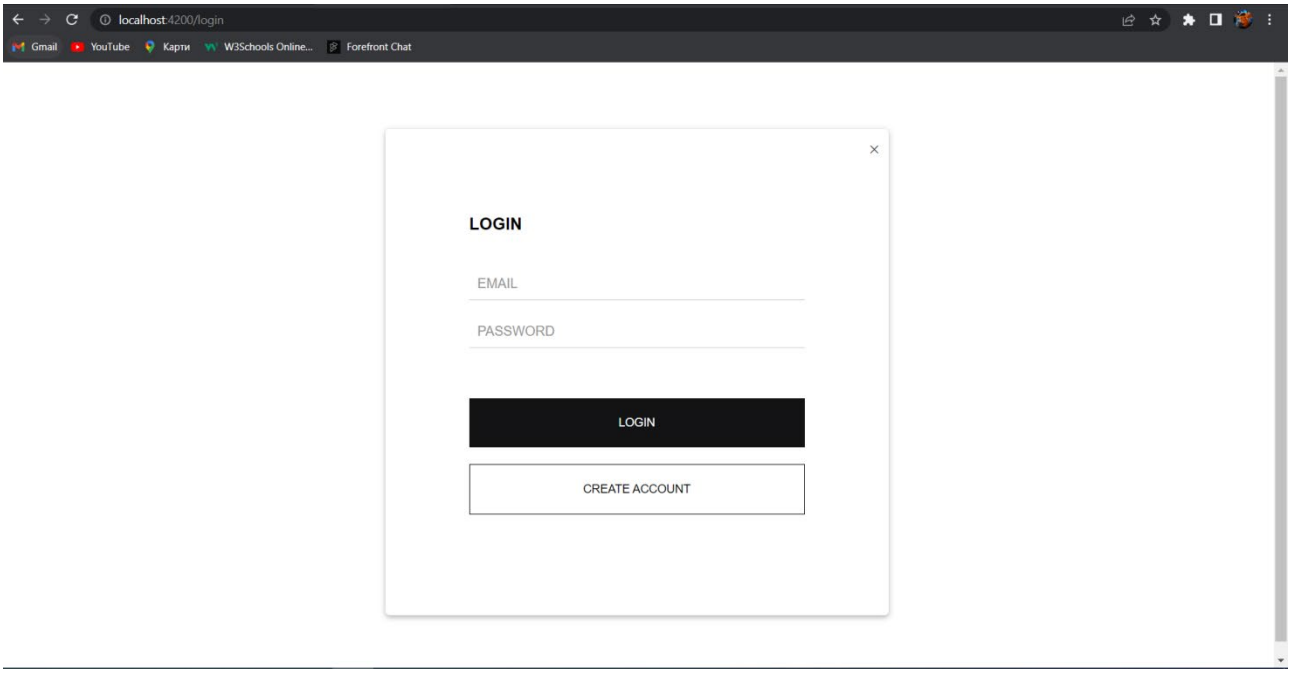

Рис. 2.23. Сторінка входу

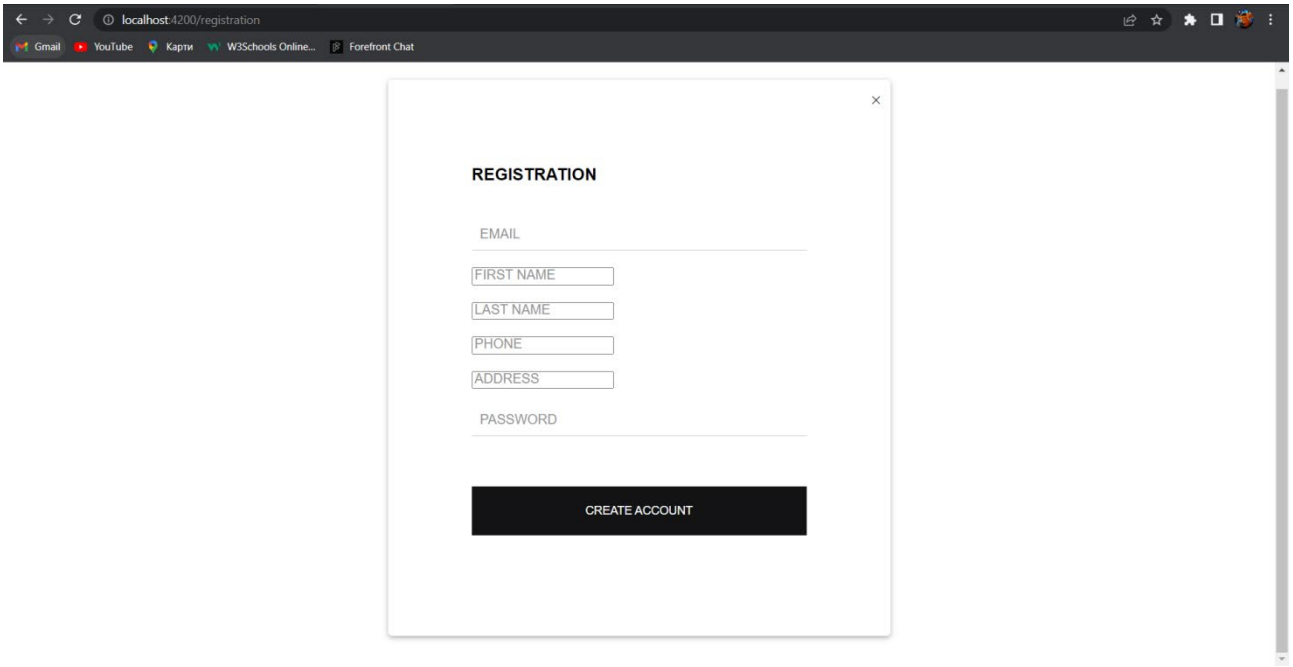

Рис. 2.24. Сторінка реєстрації

На рис. 2.25 зображено приклад оформлення замовлення, як можна побачити туди входить вартість продуктів та доставки, обрані користувачем продукти. Також замовлення редагувати, видаляючи продукти за допомогою кнопки “remove”. Після того, як користувач натисне на кнопку “payment”, його перенаправить на сторінку каталогу, а замовлення видалиться, тим самим симулюючи оплату замовлення.

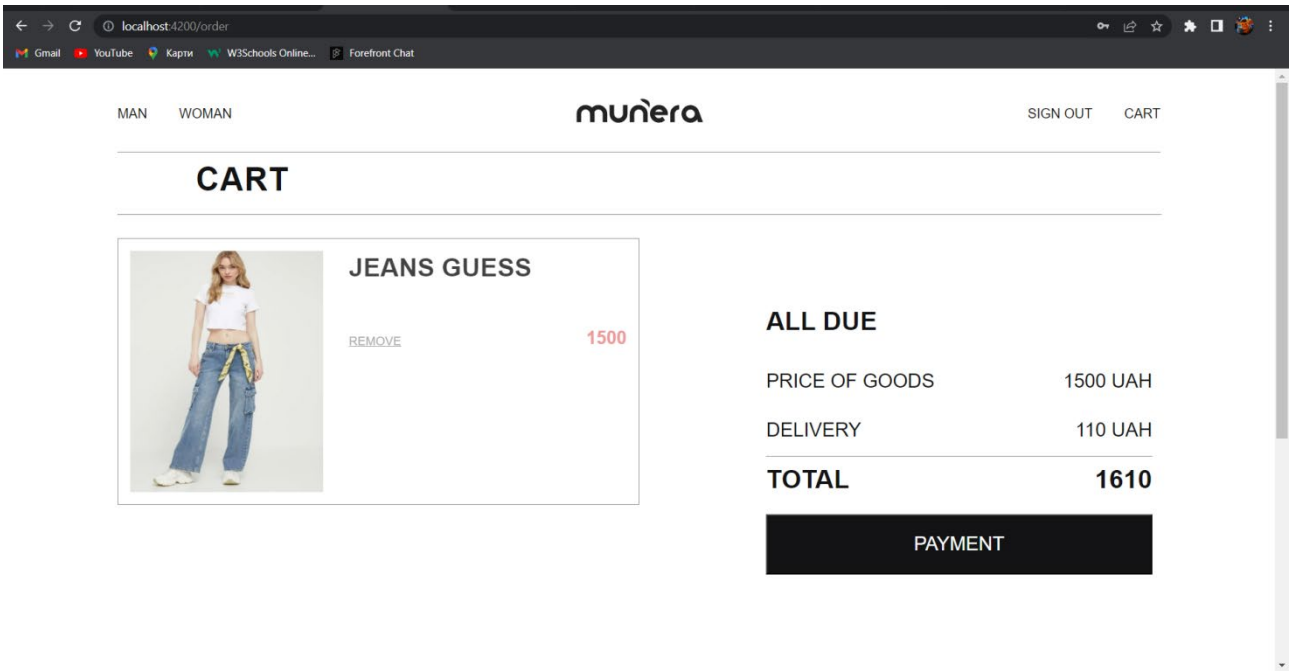

Рис. 2.25. Сторінка оплати

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

- передбачуване число операторів програми –3000;
- коефіцієнт складності програми – 1,3;
- коефіцієнт корекції програми в ході її розробки – 0,1;
- годинна заробітна плата програміста– 207,4 грн/год;
- коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі 1,25;
- коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1;
- вартість машино-години ЕОМ – 17 грн/год;
- кількість розробників – 1.

Згідно "Української спільноти програмістів (DOU)", за годину роботи Junior Python Software Engineer у 2022 році заробітна плата варіювалась від 700$ до 1300$ [19]. Враховуючи цей діапазон, середня зарплата програміста складала 1000$ на місяць. При курсі валют НБУ в кінці 2022 року, один американський долар еквівалентний 36,5 грн, що означає, що середня зарплата в гривнях становила 36 500 грн. При стандартному графіку роботи (176 годин на місяць), зарплата за годину складала приблизно 207,4 грн.

Оскільки розробка даної системи вимагає значної потужності комп'ютера для неперервної роботи серверу, зберігання та обслуговування великої кількості даних, розумним вибором буде оренда ноутбука.

Вартість оренди комп’ютера на протязі місяця складає 3000 грн. Враховуючи стандартний робочий графік 176 годин на місяць, вартість однієї машинної години роботи ЕОМ складатиме приблизно 17 грн.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \quad (3.2)$$

де q - передбачуване число операторів (3000);

C - коефіцієнт складності програми (1,4);

p - коефіцієнт корекції програми в ході її розробки (0,2).

За формулою (3.2) умовне число операторів в програмі становить:

$$Q = 3000 * 1{,}3 * (1 + 0{,}1) = 4290;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75..85)*k}, \quad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,25);

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 1 до 2 років він складає 1.

Збільшення витрат праці внаслідок недостатнього опису завдання будемо приймати не більше 50% (В = 1,25).

З урахуванням коефіцієнта кваліфікації k = 1, за формулою (3.3) отримуємо витрати праці на вивчення опису завдання:

$$t_u = \frac{4290*1,25}{85*1} = 63,1, \text{ людино-годин,}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20...25)*k}, \quad (3.4)$$

де Q – умовне число операторів програми (2047);

k – коефіцієнт кваліфікації програміста (1).

$$t_a = \frac{4290}{25*1} = 176,6, \text{ людино-годин,}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)*k}, \quad (3.5)$$

Маючи формулу (3.5) витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{4290}{25*1} = 176,6, \text{ людино-годин,}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)*k}, \quad (3.6)$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання за формулою (3.10):

$$t_{отл} = \frac{4290}{5*1} = 858, \text{ людино-годин,}$$

- за умови комплексного налагодження завдання:

$$t_{отл}^k = 1,5 * t_{отл}, \quad (3.7)$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання за формулою (3.12):

$$t_{\text{отл}}^{k} = 1{,}5 * 858 = 1287, \text{ людино-годин,}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \quad (3.8)$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20)*k}, \quad (3.9)$$

$t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{\text{до}} = 0{,}75 * t_{\text{др}}, \quad (3.10)$$

Маючи формули (3.8), (3.9) та (3.10) обчислюємо витрати праці на документацію:

$$t_{\text{др}} = \frac{4290}{20*1} = 214{,}5, \text{ людино-годин,}$$

$$t_{\text{до}} = 0{,}75 * 214{,}5 = 160{,}87, \text{ людино-годин,}$$

$$t_{\text{д}} = 214{,}5 + 160{,}87 = 375{,}37, \text{ людино-годин,}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 63{,}1 + 176{,}6 + 176{,}6 + 858 + 375{,}37 = 1654{,}67,$$

людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ КПО включають витрати на заробітну плату виконавця програми ЗЗП і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$\text{К}_{\text{ПО}} = \text{З}_{\text{ЗП}} + \text{З}_{\text{МВ}}, \text{ грн.} \quad (3.11)$$

Заробітна плата виконавців визначається за формулою:

$$\text{З}_{\text{ЗП}} = t * \text{С}_{\text{ПР}}, \text{ грн,} \quad (3.12)$$

де t - загальна трудомісткість, людино-годин (1654,67);

$\text{С}_{\text{ПР}}$ - середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 187,9 грн / год, за формулою (3.12) отримуємо:

$$З_{ЗП} = 1654{,}67 * 207{,}4 = 343\ 178, \text{грн},$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} * С_{Мч}, \text{грн}, \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год (409 год);

$С_{Мч}$ - вартість машино-години ЕОМ, грн/год (28,7 грн/год).

Підставивши в формулу (3.13) відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$$З_{МВ} = 858 * 17 = 14\ 586, \text{грн},$$

Звідси, за формулою (3.11), витрати на створення програмного продукту:

$$К_{ПО} = 343\ 178 + 14\ 586 = 357\ 764, \text{грн},$$

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k * F_p}, \text{міс}, \quad (3.14)$$

де B_k - число виконавців (1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p =176 годин).

За формулою 3.27 очікуваний період створення програмного забезпечення:

$$T = \frac{1654{,}67}{1*176} \approx 9{,}4 \text{ міс}.$$

Висновки: Вартість розробки даної інформаційної системи складає 357,764 грн, і ця сума не включає будь-які додаткові витрати.

Враховуючи отримані дані, очікуваний час розробки інформаційної системи становить приблизно 1654,67 години або 9,4 місяці з урахуванням стандартного робочого графіку. Цей результат залежить від кваліфікації розробника і включає час на дослідження, розробку концепції, програмування, тестування та документацію.

ВИСНОВКИ

У рамках даної кваліфікаційної роботи поставлено завдання розробити веб-орієнтований додаток для інтернет-магазину з продажу одягу.

Актуальність теми ще більше посилюється у зв'язку зі світовою пандемією COVID-19 та війну, яка відбувається в Україні. Умови, створені пандемією та війною, значно змінили підходи до торгівлі та споживання. Зокрема, обмеження, введені для боротьби з COVID-19, призвели до зростання популярності електронної комерції, оскільки багато людей шукають способи здійснювати покупки онлайн, уникати фізичного контакту та забезпечувати безпеку для себе та своєї родини. У той же час, війна призвела до обмежень руху товарів та послуг, що робить електронну комерцію ще більш важливою для забезпечення доступу до необхідних товарів та послуг навіть у важких умовах. Інтернет-магазини можуть стати важливим засобом забезпечення необхідних товарів та послуг, а також сприяти збереженню бізнесів та забезпеченню економічного розвитку в умовах конфліктів та війни. Тому, розробка ефективної та зручної web-орієнтованої системи для інтернет-магазину одягу стає ще більш актуальною, оскільки вона дозволить забезпечити безперебійну торгівлю та задовольнити потреби як покупців, так і продавців навіть у складних геополітичних та економічних умовах.

Додаток створений за допомогою HTML5/CSS3 та фреймворків Django та Angular.

Практичне значення отриманих результатів полягає у можливості використовувати розроблений веб-додаток для запуску та експлуатації реального інтернет-магазину з продажу одягу. Додаток надає користувачам зручний спосіб перегляду та вибору товарів, здійснення покупок і оплати.

У процесі виконання даного проекту були розглянуті та вирішені наступні завдання:

- проведення аналізу існуючих аналогів;
- створення дизайну;

– проектування бази даних;

– розробка веб-додатку, який реалізує основні функції.

У розділі, присвяченому економіці, була визначена трудомісткість розробленої інформаційної системи, яка становить 1654,67 людино-годин. Крім того, було розраховано вартість роботи з розробки програмного забезпечення, яка складає 357 764 гривень, а також був визначений час, необхідний для його розробки, який становить 9,4 місяців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. E-Commerce, або Електронна комерція [Електроний ресурс] – Режим доступу: https://fractus.com.ua/uk/blog/korysni-statti/prodazhi/e-commerce-abo elektronna-komerciya/ (дата звернення 15.04.2023)
2. Магазин одягу “JollyBell” [Електроний ресурс] – Режим доступу: https://jolybell.com/ (дата звернення 15.04.2023)
3. Магазин одягу “Answear” [Електроний ресурс] – Режим доступу :https://answear.ua/cart (дата звернення 15.04.2023)
4. Магазин одягу “Cropp” [Електроний ресурс] – Режим доступу: https://www.cropp.com/ua/uk/ (дата звернення 15.04.2023)
5. Етапи створення сайту [Електроний ресурс] – Режим доступу: https://webcase.com.ua/uk/blog/iz-chego-sostoit-razrabotka-sajta/#f6 (дата звернення 18.04.2023)
6. M. Nayeem Abdullah. E Commerce Web Platform Development. 2017. DOI [Електроний ресурс] – Режим доступу / URL: http://dspace.uiu.ac.bd/bitstream/handle/52243/98/M.Nayeem%20Abdullah_012153015_MSCSE.pdf?sequence=1&isAllowed=y (дата звернення 18.04.2023)
7. Ecommerce Security: Importance, Issues & Protection Measures [Електроний ресурс] – Режим доступу: https://www.getastra.com/blog/knowledge-base/ecommerce-security/ (дата звернення 20.04.2023)
8. 10 Must-Haves of an eCommerce Website in 2022 [Електроний ресурс] – Режим доступу: https://www.envato.com/blog/ecommerce-website-features/ (дата звернення 20.04.2023)
9. Велика кількість бібліотек, сувора динамічна типізація та проста логіка. Розробники — про переваги та недоліки Python [Електроний ресурс] – Режим доступу: https://dou.ua/lenta/articles/pros-and-cons-of-python/ (дата звернення 25.04.2023)

10. Angular vs Vue.js: що обрати [Електроний ресурс] – Режим доступу: https://wearecommunity.io/communities/epamlive/articles/1017 (дата звернення 25.04.2023)
11. What is PostgreSQL? [Електроний ресурс] – Режим доступу: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/ (дата звернення 25.04.2023)
12. MVC Design Pattern [Електроний ресурс] – Режим доступу: https://www.geeksforgeeks.org/mvc-design-pattern/ (дата звернення 30.04.2023)
13. Web Application Architecture: The Latest Guide 2022 [Електроний ресурс] – Режим доступу: https://www.clickittech.com/devops/web-application-architecture/ (дата звернення 04.05.2023)
14. Get To Know The Benefits Of VS As A Code Editor [Електроний ресурс] – Режим доступу: https://thegcpgurus.com/get-to-know-the-benefits-of-vs-as-a-code-editor/ (дата звернення 05.05.2023)
15. Introducing WebStorm: The IDE that Makes JavaScript Coding Easier [Електроний ресурс] – Режим доступу: https://www.aztek.co.il/2022/10/06/introducing-webstorm-the-ide-that-makes-javascript-coding-easier/ (дата звернення 05.05.2023)
16. What is PyCharm? Features, Advantages & Disadvantages [Електроний ресурс] – Режим доступу: https://hackr.io/blog/what-is-pycharm (дата звернення 05.05.2023)
17. API TESTING POSTMAN VS REST ASSURED: ADVANTAGES AND CHALLENGES [Електроний ресурс] – Режим доступу: https://www.testrigtechnologies.com/api-testing-postman-vs-rest-assured-advantages-and-challenges/ (дата звернення 06.05.2023)
18. What is Figma and its advantages? [Електроний ресурс] – Режим доступу: https://myuxacademy.com/what-is-figma/ (дата звернення 21.04.2023)
19. Середня заробітна плата Junior Python Software Developer за грудень 2022 року [Електроний ресурс] – Режим доступу:

https://jobs.dou.ua/salaries/?period=202212&position=Junior%20SE&technology=Python (дата звернення 21.05.2023)

20. Вартість оренди системного блоку та монітору на місяць [Електроний ресурс] – Режим доступу: https://chip-chip.com.ua/rent (дата звернення 21.05.2023)

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Бекенд частина

Налаштування Django проєкту clothing_store/clothing_store/settings.py

```
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-8xcn(wf@k-6o)$qhr@6^a^b%s0zy(%l)om$q+lv=e5vbe^-+wi'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',
    'rest_framework',
    'rest_framework_simplejwt',
    'products',
    'customer',
    'order'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
```

```
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'clothing_store.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'clothing_store.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'Eshop',
        'USER': 'postgres',
        'PASSWORD': 'admin',
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

```
# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

# Uploaded files (PDFS, DOCS, Images)

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

CORS_ALLOWED_ORIGINS = [
```

```
    'http://localhost:4200'
]

CORS_ALLOW_METHODS = (
    "DELETE",
    "GET",
    "OPTIONS",
    "PATCH",
    "POST",
    "PUT",
)

CORS_ALLOW_HEADERS = (
    "accept",
    "authorization",
    "content-type",
    "user-agent",
    "x-csrftoken",
    "x-requested-with",
)

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}


from datetime import timedelta


SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=5),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": False,
    "UPDATE_LAST_LOGIN": False,

    "ALGORITHM": "HS256",
    "SIGNING_KEY": SECRET_KEY,
    "VERIFYING_KEY": "",
    "AUDIENCE": None,
```

```
    "ISSUER": None,
    "JSON_ENCODER": None,
    "JWK_URL": None,
    "LEEWAY": 0,

    "AUTH_HEADER_TYPES": ("Bearer",),
    "AUTH_HEADER_NAME": "HTTP_AUTHORIZATION",
    "USER_ID_FIELD": "id",
    "USER_ID_CLAIM": "user_id",
    "USER_AUTHENTICATION_RULE":
"rest_framework_simplejwt.authentication.default_user_authentication_rule",

    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
    "TOKEN_TYPE_CLAIM": "token_type",
    "TOKEN_USER_CLASS": "rest_framework_simplejwt.models.TokenUser",

    "JTI_CLAIM": "jti",

    "SLIDING_TOKEN_REFRESH_EXP_CLAIM": "refresh_exp",
    "SLIDING_TOKEN_LIFETIME": timedelta(minutes=30),
    "SLIDING_TOKEN_REFRESH_LIFETIME": timedelta(days=1),

    "TOKEN_OBTAIN_SERIALIZER": "rest_framework_simplejwt.serializers.TokenObtainPairSerializer",
    "TOKEN_REFRESH_SERIALIZER": "rest_framework_simplejwt.serializers.TokenRefreshSerializer",
    "TOKEN_VERIFY_SERIALIZER": "rest_framework_simplejwt.serializers.TokenVerifySerializer",
    "TOKEN_BLACKLIST_SERIALIZER": "rest_framework_simplejwt.serializers.TokenBlacklistSerializer",
    "SLIDING_TOKEN_OBTAIN_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenObtainSlidingSerializer",
    "SLIDING_TOKEN_REFRESH_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenRefreshSlidingSerializer",
}

AUTH_USER_MODEL = 'customer.Customer'
```

Маршрути всього проєкту clothing_store/clothing_store/urls.py

```
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('products.urls')),
```

```
    path('api/', include('order.urls')),
    path('api/', include('customer.urls')),
    path('api/token/', views.TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', views.TokenRefreshView.as_view(), name='token_refresh')
]
```

Реєстрація моделі Customer в бд clothing_store/customer/admin.py

```
from django.contrib import admin

from .models import Customer

admin.site.register(Customer)
```

Модель таблиці Customer clothing_store/customer/models.py

```
from django.contrib.auth.models import AbstractUser, PermissionsMixin, UserManager
from django.db import models


class Customer(AbstractUser):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    phone = models.CharField(max_length=10)
    email = models.EmailField(unique=True)
    address = models.CharField(blank=True)
    password = models.CharField(max_length=100)
    username = None

    objects = UserManager()
    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []
```

Серіалайзер для Customer clothing_store/customer/serializers.py

```
from rest_framework import serializers

from customer.models import Customer


class CustomerSerializer(serializers.ModelSerializer):
```

```
    class Meta:
        model = Customer
        fields = ['first_name', 'last_name', 'phone', 'email', 'password']
        # extra_kwargs = {'password': {'write_only': True}}

    def create(self, validated_data) -> Customer:
        password = validated_data.pop('password')
        user = self.Meta.model(**validated_data)
        user.set_password(password)
        user.save()
        return user

    def update(self, instance, validated_data):
        password = validated_data.pop('password', None)
        if password is not None:
            instance.set_password(password)

        return super().update(instance, validated_data)
```

Ендпоінти для Customer clothing_store/customer/urls.py

```
from django.urls import path, include
from rest_framework import routers
from .views import CustomerViewSet, UpdateUserView



urlpatterns = [
    path('user/signup', CustomerViewSet.as_view({'post': 'user_create'})),
    path('user/update/<int:pk>', UpdateUserView.as_view({'put': 'update'}))
]
```

Відображення для Customer clothing_store/customer/views.py

```
from django.http import JsonResponse
from rest_framework import viewsets
from rest_framework.exceptions import PermissionDenied
from rest_framework.permissions import IsAuthenticated

from .models import Customer
from .serializers import CustomerSerializer
```

```
class UpdateUserView(viewsets.ModelViewSet):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer
    permission_classes = (IsAuthenticated, )

    def update(self, request, *args, **kwargs):
        user = self.get_object()
        if request.user != user:
            raise PermissionDenied()
        return super().update(request, partial=True)


class CustomerViewSet(viewsets.ModelViewSet):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer

    def user_create(self, request):
        user = request.data
        serializer = self.get_serializer(data=user)
        if not serializer.is_valid():
            return JsonResponse(serializer.errors, status=400)
        serializer.save()
        return JsonResponse(serializer.data)
```

Реєстрація моделі Order в БД clothing_store/customer/admin.py

```
from django.contrib import admin

from .models import Order

admin.site.register(Order)
```

Модель Order clothing_store/order/models.py

```
import datetime

from django.db import models

from customer.models import Customer
```

```
from products.models import Products


class Order(models.Model):
    product = models.ForeignKey(Products, on_delete=models.CASCADE)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    quantity = models.IntegerField(default=1)
    price = models.IntegerField()
    date = models.DateTimeField(default=datetime.datetime.today())
    status = models.BooleanField(default=False)

    class Meta:
        unique_together = ('product', 'customer')
```

Order серіалайзер clothing_store/order/serializers.py

```
from rest_framework import serializers
from .models import Order


class OrderSerializer(serializers.ModelSerializer):

    class Meta:
        model = Order
        fields = '__all__'
```

Order маршрут clothing_store/order/urls.py

```
from django.urls import path, include
from rest_framework import routers

from .views import OrderViewSet

router = routers.DefaultRouter()
router.register(r'order', OrderViewSet)

urlpatterns = [
    path('', include(router.urls))
]
```

Order відображення clothing_store/order/views.py

```
import jwt
from rest_framework import viewsets, status
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from clothing_store.settings import SECRET_KEY

from .models import Order
from .serializers import OrderSerializer

class OrderViewSet(viewsets.ModelViewSet):
    queryset = Order.objects.all()
    serializer_class = OrderSerializer
    permission_classes = (IsAuthenticated, )

    def update(self, request, *args, **kwargs):
        return super().update(request, partial=True)

    def get_queryset(self):
        queryset = Order.objects.all()
        token = self.request.headers['Authorization']
        token = token[len("Bearer "):]
        decoded_token = jwt.decode(jwt=token, key=SECRET_KEY, algorithms=["HS256"])
        user_id = decoded_token['user_id']
        if user_id:
            queryset = queryset.filter(customer_id=user_id)
        return queryset

    def create(self, request, *args, **kwargs):
        # import pdb; pdb.set_trace()
        token = request.headers['Authorization']
        token = token[len("Bearer "):]
        decoded_token = jwt.decode(jwt=token, key=SECRET_KEY, algorithms=["HS256"])
        payload = decoded_token['user_id']
        order = request.data
        order['customer'] = payload
        serializer = self.get_serializer(data=order)
        serializer.is_valid(raise_exception=True)
        self.perform_create(serializer)
        headers = self.get_success_headers(serializer.data)
        return Response(serializer.data, status=status.HTTP_201_CREATED, headers=headers)
```

Product реєстрація моделей clothing_store/products/admin.py

```
from django.contrib import admin
from .models import ProductCategory, Products, SubProductCategory



class CategoryAdmin(admin.ModelAdmin):
    list_display = ('name', 'category_id')

class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'sub_category_id')

admin.site.register(ProductCategory)
admin.site.register(Products, ProductAdmin)
admin.site.register(SubProductCategory, CategoryAdmin)
```

Product моделі clothing_store/products/models.py

```
from django.db import models



class ProductCategory(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name



class SubProductCategory(models.Model):
    name = models.CharField(max_length=100)
    category_id = models.ForeignKey(ProductCategory, on_delete=models.CASCADE, default=1)

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



class Products(models.Model):
    name = models.CharField(max_length=255)
    desc = models.TextField(default='', blank=True, null=True)
    main_image = models.ImageField(upload_to='uploads/mainImages', blank=True)
    add_image1 = models.ImageField(upload_to='uploads/addImages', blank=True)
    add_image2 = models.ImageField(upload_to='uploads/addImages', blank=True)
    add_image3 = models.ImageField(upload_to='uploads/addImages', blank=True)
    price = models.IntegerField(default=0)
```

```
    include_model = models.BooleanField(default=False)
    sub_category_id = models.ForeignKey(SubProductCategory, on_delete=models.CASCADE, default=1)

    def __str__(self):
        return f"{self.name} {self.sub_category_id}"
```

Product серіалайзер clothing_store/products/serializers.py

```
from rest_framework import serializers

from .models import Products, ProductCategory, SubProductCategory


class ProductCategorySerializer(serializers.ModelSerializer):

    class Meta:
        model = ProductCategory
        fields = '__all__'


class ProductSubCategorySerializer(serializers.ModelSerializer):

    class Meta:
        model = SubProductCategory
        fields = '__all__'


class ProductSerializer(serializers.ModelSerializer):

    class Meta:
        model = Products
        fields = '__all__'
```

Product шляхи clothing_store/products/urls.py

```
from django.urls import path
from .views import ProductCategoryViewSet, ProductViewSet, ProductSubCategoryViewSet


urlpatterns = [
    path('categories/', ProductCategoryViewSet.as_view({'get': 'list'})),
    path('categories/subcategories/', ProductSubCategoryViewSet.as_view({'get': 'list'})),
    path('products/', ProductViewSet.as_view({'get': 'list'})),
```

```
    path('products/<int:pk>', ProductViewSet.as_view({'get': 'retrieve'})),
    path('products/all', ProductViewSet.as_view({'get': 'get_products'}))
]
```

Product відображення clothing_store/products/views.py

```
from django.core.exceptions import ObjectDoesNotExist
from django.http import JsonResponse
from rest_framework import viewsets

from .models import Products, ProductCategory, SubProductCategory
from .serializers import ProductSerializer, ProductCategorySerializer, ProductSubCategorySerializer


class ProductCategoryViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = ProductCategory.objects.all()
    serializer_class = ProductCategorySerializer


class ProductSubCategoryViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = SubProductCategory.objects.all()
    serializer_class = ProductSubCategorySerializer

    def list(self, request):
        if category_id := request.query_params.get('category_id'):
            sub_category = self.queryset.filter(category_id=category_id)
            serializer = self.get_serializer(sub_category, many=True)
            return JsonResponse(serializer.data, safe=False)

        return JsonResponse({'error': 'category_id is required'}, status=400)

class ProductViewSet(viewsets.ModelViewSet):
    queryset = Products.objects.all()
    serializer_class = ProductSerializer

    def list(self, request):
        if sub_category_id := request.query_params.get('sub_category_id'):
            products = self.queryset.filter(sub_category_id=sub_category_id)
            serializer = self.get_serializer(products, many=True)
            for obj in serializer.data:

                obj['main_image'] = obj['main_image'][len("http://127.0.0.1:8000"):]
                print(obj['main_image'])
```

```
            return JsonResponse(serializer.data, safe=False)

        return JsonResponse({'error': 'sub_category_id is required'}, status=400)

    def retrieve(self, request, pk):
        try:
            product = self.queryset.get(id=pk)
        except ObjectDoesNotExist:
            return JsonResponse({'error': 'product not found'}, status=404)

        serializer = self.get_serializer(product)
        product = serializer.data

        product['main_image'] = product['main_image'][len("http://localhost:8000"):]
        product['add_image1'] = product['add_image1'][len("http://localhost:8000"):]
        product['add_image2'] = product['add_image2'][len("http://localhost:8000"):]
        product['add_image3'] = product['add_image3'][len("http://localhost:8000"):]

        return JsonResponse(product)

    def get_products(self, request):
        serializer = self.get_serializer(self.queryset, many=True)
        for obj in serializer.data:
            obj['main_image'] = obj['main_image'][len("http://127.0.0.1:8000"):]

        return JsonResponse(serializer.data, safe=False)
```

Фронтенд частина

src/app/auth/interceptor/token.interceptor.ts

```
import {Injectable} from "@angular/core";
import {HttpErrorResponse, HttpEvent, HttpHandler, HttpInterceptor, HttpRequest} from "@angular/common/http";
import {catchError, concatMap, Observable, switchMap, take, throwError} from "rxjs";
import {AuthService} from "../services/auth.service";
import {SessionStorageService} from "../services/storage.service";

@Injectable({
  providedIn: 'root'
})
export class AuthInterceptorService implements HttpInterceptor {
  constructor(private authFacae: AuthService, private sessionStorage: SessionStorageService) { }
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
```

```
    console.log('intercepting', request)

    let token = this.sessionStorage.getToken();
    console.log('token goes', token)
    if (token) {
      console.log('token', token)
      request = request.clone({
        setHeaders: { Authorization: `${token}` }
      });

    }

    return next.handle(request).pipe(
      catchError((err) => {
        if (err instanceof HttpErrorResponse) {
          if (err.status === 401) {
            console.log('loging out in interceptor')
            this.authFacae.logout()
          }
        }
        return throwError(err);
      })
    )

  }
}
```

src/app/auth/services/auth.service.ts

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpErrorResponse} from "@angular/common/http";
import {SessionStorageService} from "./storage.service";
import {BehaviorSubject, catchError, first, map, Observable, of, switchMap, throwError} from "rxjs";
import {User} from "../../User";
import {Router} from "@angular/router";
import {AuthResponse} from "../../AuthResponse";

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private isAuthorized$$ = new BehaviorSubject<boolean>(false);
```

```
public isAuthorized$ = this.isAuthorized$$.asObservable();
public token$: Observable<string> = new Observable<string>();

constructor(private httpClient: HttpClient, private sessionStorageService: SessionStorageService,
          private router: Router) {
  this.token$ = of(sessionStorageService.getToken())
}

register(first_name:string, last_name:string, phone:string,
        email:string,   address: string, password: string,): Observable<string> {
  return this.httpClient.post<User>('http://localhost:8000/api/user/signup', {
    "email": email,
    "password": password,
    "first_name": first_name,
    "last_name": last_name,
    "phone": phone,
    "address": address
  })
    .pipe(
      switchMap(data => {
        return this.httpClient.post<AuthResponse>('http://localhost:8000/api/token/', {
          "email": email,
          "password": password
        })
          .pipe(map(token => {
            this.sessionStorageService.setToken(token.access);
            this.isAuthorized$$.next(true);
            this.isAuthorized$ = this.isAuthorized$$.asObservable();
            return token.access;
          }))
      }));
}

logout() {
  this.sessionStorageService.deleteToken();
  this.router.navigate(['login'])
}

login(password: string, email: string): Observable<string> {
  return this.httpClient.post<AuthResponse>('http://localhost:8000/api/token/', {
    "email": email,
    "password": password
```

```
  })
    .pipe(map(data => {
      console.log('login response', data.access)
      this.sessionStorageService.setToken(data.access);
      return data.access
    }),
    catchError(this.handleError));
 }

 handleError(error: HttpErrorResponse) {
  console.log(error.error)
  console.log(error);
  return throwError(error.error);
 }

}
```

src/app/auth/services/storage.service.ts

```
import {Inject, Injectable} from '@angular/core';
import {WindowToken} from "../../app.module";

@Injectable({
 providedIn: 'root'
})
export class SessionStorageService {

 constructor(@Inject(WindowToken) private window: Window) { }

 public setToken(token: string) {
  this.window.localStorage.setItem('token', 'Bearer '.concat(token));
 }

 public deleteToken() {
  this.window.localStorage.removeItem('token');
 }

 public getToken():string {
  console.log('token', this.window.localStorage.getItem('token'))
  return this.window.localStorage.getItem('token') === undefined ? 'undefined' :
   this.window.localStorage.getItem('token') as string;
 }
}
```

src/app/feature/catalog/catalog.component.ts

```
import {Component, Input, OnInit} from '@angular/core';
import {exhaustMap, map, Observable, switchMap, tap, zip, of} from "rxjs";
import {Product} from "../../Product";
import {Subcategory} from "../../Subcategory";
import {CatalogService} from "../../services/catalog.service";
import {NONE_TYPE} from "@angular/compiler";
import {ActivatedRoute, Router} from "@angular/router";
import {SessionStorageService} from "../../auth/services/storage.service";
import {AuthService} from "../../auth/services/auth.service";

@Component({
  selector: 'app-catalog',
  templateUrl: './catalog.component.html',
  styleUrls: ['./catalog.component.css']
})
export class CatalogComponent implements OnInit {

  @Input()
  category: number = 2;

  categoryName: string = '';

  subcategory?:number;

  getByIdMode: boolean = false;
  authenticated: boolean = false;
  products$: Observable<Product[]> = new Observable<Product[]>();

  subcategories$: Observable<Subcategory[]> = new Observable<Subcategory[]>();
  observables: Observable<AsyncData> = new Observable<AsyncData>();
  constructor(private catalogService: CatalogService, private router: Router,
              private activatedRoute: ActivatedRoute, private storageService: SessionStorageService, private authService:
AuthService) {
    console.log('token', this.storageService.getToken())
    let token = this.storageService.getToken()
    this.authenticated = this.storageService.getToken() != null
  }

  ngOnInit() {
    this.activatedRoute.params.subscribe(params => {
```

```
  let categoryParam = params['categoryId']
  let subcategoryParam = params['subcategoryId']
  if(categoryParam) {
    this.subcategory = subcategoryParam;
    this.category = categoryParam;
    this.getByIdMode = true;
  }
})

if(this.getByIdMode) {
  this.observables = this.catalogService.gettSubcategories(this.category)
    .pipe(switchMap(categories => {
      // @ts-ignore
      this.categoryName = categories.at(0).name
      // @ts-ignore
      return this.catalogService.getAllProductsBySubcategory(this.subcategory)
        .pipe(switchMap(products => {

          return of({subcategories: categories, products: {
              modelProducts: products.filter(product => product.include_model),
              nonModelProducts: products.filter(product => !product.include_model)
            }});
        }))
    }))
} else {

  this.observables = this.catalogService.gettSubcategories(this.category)
    .pipe(switchMap(categories => {
      // @ts-ignore
      this.categoryName = categories.at(0).name
      // @ts-ignore
      return this.catalogService.getAllProductsBySubcategory(categories.at(0).id)
        .pipe(switchMap(products => {

          return of({
            subcategories: categories, products: {
              modelProducts: products.filter(product => product.include_model),
              nonModelProducts: products.filter(product => !product.include_model)
            }
          });
        }))
    }))
```

```
  }
}

getAll(id:number) {
  this.getObservables(id)
}

getObservables(categoryId:number){
  this.observables = this.catalogService.gettSubcategories(this.category)
    .pipe(switchMap(categories => {
      // @ts-ignore
      this.categoryName = categories.filter(category => category.id === categoryId).at(0).name
      // @ts-ignore
      return this.catalogService.getAllProductsBySubcategory(categoryId)
        .pipe(switchMap(products => {
          return of({subcategories: categories, products: {

              modelProducts: products.filter(product => product.include_model),
              nonModelProducts: products.filter(product => !product.include_model)
            }});
        }))
    }))
}

changeCategory(category:number) {
  this.category = category;
  this.observables = this.catalogService.gettSubcategories(this.category)
    .pipe(switchMap(categories => {
      // @ts-ignore
      this.categoryName = categories.at(0).name
      // @ts-ignore
      return this.catalogService.getAllProductsBySubcategory(categories.at(0).id)
        .pipe(switchMap(products => {

          return of({subcategories: categories, products: {
              modelProducts: products.filter(product => product.include_model),
              nonModelProducts: products.filter(product => !product.include_model)
            }});
        }))
    }))
}
```

```
  navigateToProduct(id: number) {
    this.router.navigate(['/product/' + id.toString()])
  }
  navigateToLanding() {
    this.router.navigate(['/landing'])
  }
  navigateToLogin(){
    this.router.navigate(['/login'])
  }
  navigateToCart(){
    this.router.navigate(['/order'])
  }
  logout() {
    this.authService.logout()
  }
  login() {
    this.router.navigate(['/login'])
  }
}


class AsyncData {
  subcategories: Subcategory[];
  products: Products;

  constructor(subcategories: Subcategory[], products: Products) {
    this.products = products;
    this.subcategories = subcategories;
  }
}

class Products {
  modelProducts: Product[];
  nonModelProducts: Product[];

  constructor(modelProducts:Product[],
              nonModelProductss: Product[]) {
    this.modelProducts = modelProducts;
    this.nonModelProducts = nonModelProductss;
  }
}
```

src/app/feature/catalog/catalog.module.ts

```
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';
import {CatalogComponent} from './catalog.component';
import {HttpClientModule} from "@angular/common/http";
import {CatalogRoutingModule} from "./catalog-routing.module";


@NgModule({
  declarations: [
    CatalogComponent
  ],
  imports: [
    CommonModule,
    HttpClientModule,
    CatalogRoutingModule
  ],
  exports: [CatalogComponent]
})
export class CatalogModule {
}
```

src/app/feature/catalog/catalog-routing.module.ts

```
import { NgModule } from '@angular/core';
import {RouterModule, Routes} from "@angular/router";
import {CatalogComponent} from "./catalog.component";

const routes: Routes = [
  {
    path: '',
    component: CatalogComponent
  }
];


@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class CatalogRoutingModule { }
```

src/app/feature/login/login.component.ts

```
import { Component } from '@angular/core';
import {FormBuilder, FormControl, FormGroup, Validators} from "@angular/forms";
import {AuthService} from "../../auth/services/auth.service";
import {Router} from "@angular/router";

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  // @ts-ignore
  formGroup: FormGroup;
  constructor(private fb: FormBuilder,private authService: AuthService,  private router: Router,) {
    this.initForm()
  }
  initForm(): void {
    this.formGroup = new FormGroup({
      email: new FormControl("", [Validators.required]),
      password: new FormControl("", [Validators.required])
    });
  }

  get email(){
    return this.formGroup.get('email')
  }

  get password(){
    return this.formGroup.get('password')
  }
  login() {
    console.log('is form valid', this.formGroup.valid)
    if(this.formGroup.valid) {
      this.authService.login(
        this.formGroup.controls["password"].value,
        this.formGroup.controls["email"].value
      ).subscribe(result => {
        console.log('login')
        if(result){
          this.redirectToLanding()
        }
      })
```

```
    }
  }
  navigateToCreate() {
    this.router.navigate(['/registration'])
  }
  private redirectToLanding() {
    this.router.navigate(['/landing'])
  }
  navigateToLanding() {
    this.router.navigate(['/landing'])
  }
}
```

src/app/feature/landing/landing.module.ts

```
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';
import {LandingComponent} from './landing.component';
import {HttpClientModule} from "@angular/common/http";
import {LandingRoutingModule} from "./landing-routing.module";


@NgModule({
  declarations: [
    LandingComponent
  ],
  imports: [
    CommonModule,
    HttpClientModule,
    LandingRoutingModule
  ],
  exports: [LandingComponent]
})
export class LandingModule {
}
```

src/app/feature/landing/landing-routing.module.ts

```
import { NgModule } from '@angular/core';
import {RouterModule, Routes} from "@angular/router";
import {LandingComponent} from "./landing.component";

const routes: Routes = [
```

```
  {
    path: '',
    component: LandingComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class LandingRoutingModule { }
```

Лістинг login, order, product, registr, services буде знаходитись на електронному носії. Верстка усього проєкту також буде знаходитись на електронному носії.

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Сердюк_122_19_2_Диплом.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Сердюк_122_19_2_Диплом.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
diploma.rar	Архів. Містить коди програми
Презентація	
Презентація_Сердюк.ppt	Презентація кваліфікаційної роботи.