ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Створення онлайн магазину на Python Django»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Владислав КУДЬ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КІД-41
Владислав КУДЬ
Ім'я, ПРІЗВИЩЕ
Керівник: Борис КОНДРАТЮК
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: Борис КОНДРАТЮК
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
_____Ім'я, ПРІЗВИЩЕ
«_____» _____ 2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кудь Владислав Дмитрович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Створення онлайн магазину на основі Python Django» керівник кваліфікаційної роботи Борис КОНДРАТЮК,

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «_____» _____2024р. № _____

2. Строк подання кваліфікаційної роботи «_____» _____ 2024р.
3. Вихідні дані до кваліфікаційної роботи: _____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. _____
2. _____
3. _____
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «_____» _____ 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	26.02-22.03.2024	
2	Обробка матеріалу	23.03.-05.04.2024	
3	Розробка теоретичної частини	06.04-10.04.2024	
4	Розробка проектної частини	11.04-30.04.2024	
5	Тестування та виправлення недоліків	01.05-03.05.2024	
6	Висновки за результатами аналізу	04.05-08.05.2024	
7	Оформлення документу	09.05-11.05.2024	
8	Розробка презентації	12.05-24.05.2024	
9	Здача в деканат	25.05-3.06.2024	

Здобувач(ка) вищої освіти _____ Владислав КУДЬ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Борис КОНДРАТЮК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 70 стор., 71 рис., 1 табл., 30 джерел.

Мета роботи – дослідження процесу створення веб-сайтів на сонові мови програмування Python з використанням фреймворку Django з метою розширення розуміння про веб-розробку.

Об'єкт дослідження – процес створення та програмного розвитку онлайн магазину, використовуючи програмні засоби на основі мови програмування Python та фреймворка Django.

Предмет дослідження – аналіз можливостей інструментів розробки, реалізація основних функціональних компонентів магазину, таких як моделі даних, форми зворотнього зв'язку, система авторизації та кошик для товарів, а також вивчення технічних аспектів впровадження та підтримки проекту.

Короткий зміст роботи: В сучасному світі веб-розробка набуває все більшого значення і стає областю з великою перспективою, особливо з урахуванням того, що різні галузі все більше залежать від інформаційних технологій, і цей тренд постійно зростає.

В ході дослідження було проаналізовано та випробувано процес створення онлайн магазину. Було досліджено можливості фреймворку Django для реалізації основних функціональних компонентів магазину. Крім того, в ході дослідження було проведено аналіз технічних аспектів розробки, зокрема вибір інтегрованих розробничих середовищ, розробку інтерфейсу користувача та забезпечення безпеки даних. Результатом дослідження став розроблений та протестований онлайн магазин, готовий до ефективного використання.

КЛЮЧОВІ СЛОВА: PYTHON, DJANGO, МОДЕЛЬ, АТРИБУТ.

ABSTRACT

Text part of the qualifying work for obtaining a bachelor's degree:

70 pages, 71 figures, 1 table, 30 sources.

The purpose of the work is to study the process of creating websites in the sleepy Python programming language using the Django framework in order to expand the understanding of web development.

The object of research is the process of creating and software development of an online store using software tools based on the Python programming language and the Django framework.

The subject of the study is the analysis of the possibilities of the development tools, the implementation of the main functional components of the store, such as data models, feedback forms, the authorization system and the shopping cart, as well as the study of the technical aspects of the implementation and support of the project.

Summary of the work: In today's world, web development is gaining more and more importance and is becoming an area with great prospects, especially considering that various industries are increasingly dependent on information technology, and this trend is constantly growing.

In the course of the study, the process of creating an online store was analyzed and tested. The possibilities of the Django framework for the implementation of the main functional components of the store were investigated. In addition, the study analyzed the technical aspects of development, including the selection of integrated development environments, user interface design, and data security. The result of the research was a developed and tested online store, ready for effective use.

KEYWORDS: PYTHON, DJANGO, MODEL, ATTRIBUTE.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра (магістра)

Направляється здобувач(ка) ___Кудь В. Д._____ до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю _____123 «Комп'ютерна інженерія»_____
(код, найменування спеціальності)
освітньо-професійної програми _____Комп'ютерні системи та мережі_____
(назва)
на тему: «_____Створення онлайн магазину на Python Django____».

Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ ____Бондарчук А. П._____
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач(ка): __Під час виконання бакалаврської роботи Кудь В. Д. показав хорошу теоретичну та практичну підготовку, знання матеріалу, вміння вирішувати самостійно питання і робити висновки. Зміст роботи відповідає завданню. Перелік використаних джерел свідчить про вміння розбиратись в наукових питаннях та застосовувати їх при дослідженнях._____

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача(ки) _Кудь В. Д._ на оцінку «_відмінно_» та присвоїти йому(їй) кваліфікацію __бакалавр з комп'ютерної інженерії__.

Керівник кваліфікаційної роботи _____ _Борис КОНДРАТЮК_
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«____» _____ 20___ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач(ка) _*Кудь В. Д.*_ допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою _____ _____ _____
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА

на кваліфікаційну бакалаврську (магістерську) роботу

здобувача(ки) вищої освіти ____Кудь Владислав Дмитрович_____

(прізвище, ім'я, по батькові)

на тему «__Створення онлайн магазину на Python Django_____»

Актуальність:

Веб-розробка стає все більш затребуваною та перспективною областю, особливо з урахуванням постійно зростаючої залежності різних галузей від інформаційних технологій. Мова програмування Python, разом з фреймворком Django, є інструментом для створення веб-додатків, що мають високу продуктивність і гнучкість.

Позитивні сторони.

1. Зміст бакалаврської роботи відповідає завданню. Проект, який виконав Кудь В. Д., показав високий рівень знань і готовність його до майбутньої роботи з фаху.
2. Достатньо успішні викладенні технічні питання.
3. Текст викладений грамотно, ясно, послідовно. Візуальний матеріал оформлений якісно. Широко використовується науково-технічна література.

Недоліки.

1. Недостатньо оформлена сторінка адміністратора.
2. В роботі недостатньо розкрито HTML структура проекту.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної бакалаврської (магістерської) роботи.

Висновок: *кваліфікаційна бакалаврська (магістерська) робота заслуговує оцінку "_Відмінно_", а здобувач(ка) _____Кудь Владислав Дмитрович__ заслуговує присвоєння кваліфікації: __бакалавр з комп'ютерної інженерії__*

Рецензент: _____ _____

науковий ступінь, вчене звання *підпис* *Ім'я, ПРІЗВИЩЕ*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CSS – (розшифровується як Cascading Style Sheets) спеціальна мова, яку використовують для графічного опису веб-сайтів.

HTTP — (розшифровується як HyperText Transfer Protocol) протокол передачі даних.

HTML – (розшифровується як HyperText Markup Language) стандартизована мова гіпертекстової розмітки документів для перегляду веб-сторінок в браузері.

ORM – (розшифровується як Object-Relational Mapping) технологія програмування, яка пов'язує бази даних із концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

WSGI – (розшифровується як Web Server Gateway Interface) є стандартом для спілкування між програмою Python, яка виконується на сервері, та веб-сервером, таким як Apache.

URL – адреса ресурсу в Інтернеті.

IDE – це система програмних засобів, що використовуються програмістами для розробки програмного забезпечення.

ПЗ – програмне забезпечення.

PEP8 – (розшифровується як Python Enhancement Proposal) загально прийнятий стиль коду на Python.

VS Code – Visual Studio Code.

Тег – елемент мови розмітки тексту.

ВСТУП

Актуальність теми. У сучасному світі веб-розробка стає все більш затребуваною та перспективною областю, особливо з урахуванням постійно зростаючої залежності різних галузей від інформаційних технологій. Мова програмування Python, разом з фреймворком Django, є інструментом для створення веб-додатків, що мають високу продуктивність і гнучкість.

Мета та завдання дослідження. Ця робота націлена на вивчення процесу створення веб-сайту з використанням HTML, CSS, JavaScript, Python та фреймворку Django. Основну увагу приділено дослідженню основних етапів розробки.

Об'єкт дослідження. Об'єктом дослідження є процес створення веб-сайту з використанням мов програмування, включаючи різні етапи процесу розробки та особливості інструментів, які застосовуються при цьому.

Предмет дослідження. Предметом дослідження є технології та методи, що використовуються при створені веб-сайтів, а також їх застосування для досягнення конкретних цілей у галузі веб-розробки.

Методи дослідження. Для досягнення поставлених цілей використовуються методи аналізу літератури, вивчення документації і веб-ресурсів з Python та Django, а також експериментальні методи, включаючи створення прототипів веб-додатків та аналіз їхньої продуктивності та функціональності.

Теоретичне та практичне значення отриманих результатів. Отримані результати дозволять не тільки глибше зрозуміти процес створення веб-сайтів, а й наддадуть практичні рекомендації для розробників-початківців, а також поради та пропозиції для подальшого розвитку та покращення веб-технологій.

Крім того, робота сприяє збагаченню наукового та практичного досвіду у галузі веб-розробки на основі сучасних технологій.

Структура роботи. Дипломна робота складається зі вступу, трьох розділів, поділених на підрозділи, висновків, додатків та списку використаних джерел з 30 найменувань.

1 ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

1.1 Вибір програмних засобів розробки

В цьому пункті розглянуто програмні засоби, які знадобилися для розробки веб-сайту: HTML5, мова розмітки CSS3, Bootstrap, JavaScript, фреймворк для розробки сайтів Python Django та середовище розробки мови Python – PyCharm.

Основна перевага такого підходу полягає у розподілі написання клієнтської частини від серверної. Отже, кожна окрема частина програми, створеної за допомогою Python, має одне призначення і може бути змінена незалежно, тобто без впливу на інші компоненти. Дизайнер може змінити HTML сторінки без внесення змін до коду, який відображає сторінку. Адміністратор бази даних може перейменувати таблицю та визначити ці зміни в одному місці, замість того, щоб шукати та вносити зміни до великої кількості файлів.

Для підготовки основи сайту у цій роботі використовується HTML5 (HyperText Markup Language – мова розмітки гіпертекстових документів), яка є останньою версією мови гіпертекстової розмітки і основою у створенні веб-сайтів. Вона пропонує багато можливостей для створення динамічних інтерфейсів і надає користувачам зручний та ефективний спосіб взаємодії з веб сторінками. Завдяки HTML5 розробник може формувати текст і вставляти зображення, а також створювати складні структури документів, такі як заголовки, підзаголовки, списки і таблиці. Крім того, ця мова надає можливість створювати посилання між різними документами, роблячи навігацію по сайту логічною і зручною для користувача.[1] Особливе значення має можливість

включення графічних і мультимедійних елементів. Це дозволяє створювати привабливі, інтерактивні веб-сторінки, які привертають увагу користувачів.

У п'ятій версії HTML розробники постаралися поєднати все інструменти, необхідні для створення професійних, сучасних та динамічних сайтів, що використовують найбільш поширені технології та відповідні до сучасних стандартів.

1.2 Python та фреймворк Django

Python – це багатофункціональна мова програмування, що поєднує в собі концепції імперативного, об’єктно-орієнтовного та функціонального програмування. Створений Гвідо ван Россумом у 1989 році, він сьогодні продовжує розвиватися та користується широкою популярністю. [3]

Рисунок 1.1 – Емблема Python

Переваги цієї мови програмування включають:

- відкрита розробка, що сприяє активній спільноті та постійному вдосконаленню мови;
- простота вивчення, особливо на ранніх етапах, завдяки чистому та зрозумілому синтаксису;
- синтаксичні особливості, які стимулюють створення коду, що читається;

- швидке прототипування та динамічна семантика, що прискорюють процес розробки;
- велика та доброзичлива спільнота, готова допомогти новачкам.
- широка бібліотека та розширення доступні для використання завдяки зручному механізму імпорту;
- добре продумані механізми модульності, що сприяють зручності використання;
- об’єктно-орієнтовний підхід, але не примушуючи його використання;
- інструменти для зручної та ефективної розробки веб-сайтів.

На основі цих преваг Python ідеально підходить для розробки як серверної, так і клієнтської частин веб-сайту.

Фреймворк – інструмент, який спрощує процес створення та використання веб-додатків. На мові програмування Python існує безліч відомих та високоякісних фреймворків. Вибір фреймворку слід здійснювати усвідомлено, оскільки це безпосередньо впливає на якість розроблюваного веб-магазину. Практично кожен фреймворк підходить для створення магазину.

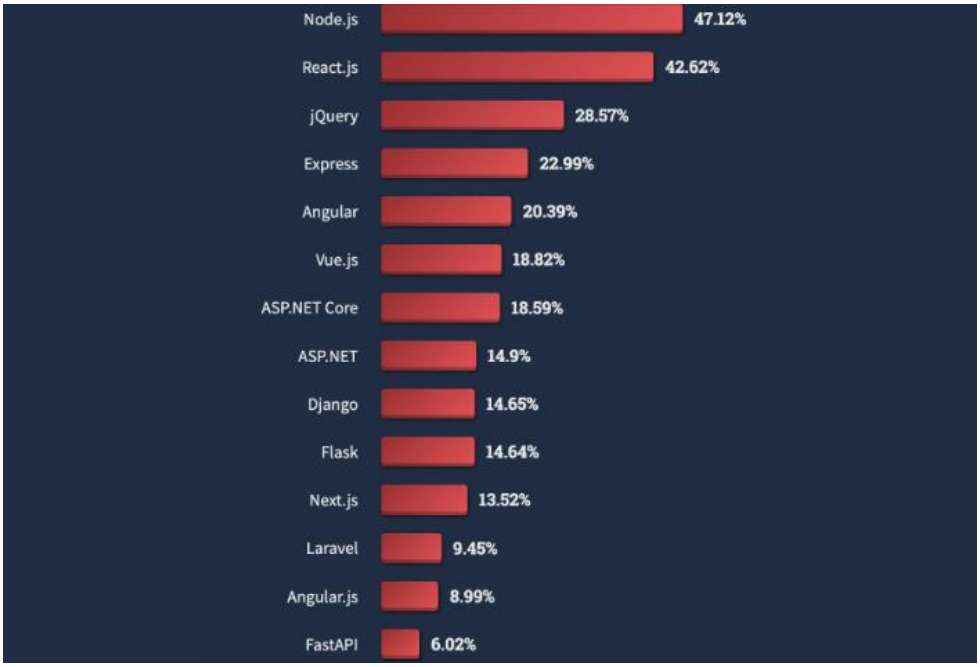


Рисунок 1.2 – Графік найпопулярніших фреймворків

В графіку [5] присутні 3 фреймворка Python: Django, Flask та FastAPI. Щоб обрати фреймворк, який буде найбільше підходити, порівняймо перші два.

Flask – це мікрофреймворк для створення веб-застосунків мовою програмування Python. Він має невелике ядро, яке легко розширити: це мікрофреймворк, який не містить ORM (Object Relational Manager) або подібних функцій.

Він має багато цікавих функцій, таких як маршрутизація URL-адреси, система шаблонів. Це структура веб-додатків WSGI [9]. На відміну від Django, Flask дуже сприяє принципам Python. Почати роботу з Flask легко, тому що він не вимагає тривалого навчання. Крім того, Flask дуже простий у використанні, що покращує читання коду.[7]

Ось кілька ключових особливостей Flask:

- надає мінімальний набір інструментів для створення веб-додатків, дозволяючи розробнику вибирати необхідні компоненти за необхідності. Це робить Flask гнучким та дозволяє використовувати лише ті функціональні можливості, які потрібні для конкретного проекту;
- володіє простим і інтуїтивно зрозумілим синтаксисом, що робить його ідеальним вибором для розробників-початківців і швидких прототипів;
- не нав'язує конкретні правила організації програми, дозволяючи розробнику вільно вибирати структуру проекту та використовувати сторонні бібліотеки та розширення на власний розсуд;
- Flask має широкий вибір розширень (Flask extensions), які додають додаткові функціональні можливості у додаток. Ці розширення дозволяють легко інтегрувати підтримку баз даних, автентифікацію, форми, асинхронні завдання та багато іншого.

Це невеликий фреймворк, але це не означає, що вся ваша програма повинна бути написана в одному файлі Python. Для великих проектів ви можете і повинні використовувати безліч файлів, щоб краще організувати код і впоратися зі складністю.

"Мікро" у назві Flask означає, що він простий у освоєнні, але при цьому гнучкий. Ви самі приймаєте рішення, наприклад, яку базу даних використовувати, чи потрібна вам ORM і таке інше. Фреймворк не нав'язує своїх рішень.

Flask є одним із найпопулярніших веб-фреймворків, що означає його актуальність та сучасність. Його функціональність легко розширити, і він масштабується для створення складних програм.

Django - це безкоштовний фреймворк для створення веб-застосунків на мові програмування Python, який заснований на шаблоні проектування MVC (Model-View-Controller). Проект підтримується програмою Django Software Foundation.[4]

Завдяки Django перетворення ідей на реальні веб-проекти займає лише кілька хвилин. Цей фреймворк доступний безкоштовно, що значно полегшує процес веб-розробки, дозволяючи розробникам зосередитися на дизайні та функціональності програми. Django став одним із найкращих фреймворків з моменту свого запуску в 2005 році, допомагаючи тисячам розробників виконувати роботу швидко та ефективно. Його спрощений підхід значно спрощує процес розробки веб-застосунків, пристосовуючись до складних викликів. Будучи розробленим для мови програмування Python, Django поєднує в собі простоту вивчення, гнучкість та зручність у роботі, що робить його одним із найпопулярніших інструментів для створення веб-проектів.

Веб-сайти, створені на Django, складаються з одного або декількох програм, які рекомендується розробляти як окремі та взаємозамінні. Це одна з ключових архітектурних відмінностей Django від деяких інших фреймворків, таких як Ruby on Rails. Одним з основних принципів Django є DRY (Don't Repeat Yourself).

На відміну від деяких інших фреймворків, обробники URL[10] Django налаштовуються явно з використанням регулярних виразів, а не генеруються автоматично зі структури моделей і контролерів.

Для взаємодії з базою даних Django використовує власний ORM (Object-Relational Mapping)[8], в якому модель даних описується з використанням класів Python, а потім автоматично генерується схема бази даних.

Архітектура програми Django слідує шаблону Model-Template-View (MTV), який розділяє логіку програми, інтерфейс користувача і доступ до даних.

- *модель* – це рівень доступу до даних, де програма взаємодіє з базами даних та іншими джерелами інформації;
- *шаблон* – це рівень, який визначає, як дані будуть представлені користувачеві;
- *подання (View)* – визначає, які дані мають бути представлені користувачеві.

У Django роль контролера, який визначає, який запит має бути спрямований до якогось подання, виконує інфраструктура фреймворку, використовуючи конфігурацію адрес URL.

Крім моделей, шаблонів та уявлень, Django надає вбудовану функціональність для керування адресами URL, автоматичного інтерфейсу адміністрування бази даних, кешування та багато іншого, що значно полегшує завдання веб-розробника. Аналогічно Python, Django надає сумісність із великою стандартною бібліотекою додаткових пакетів, які можна використовувати у своїх програмах без додаткового завантаження.

Рівень моделі Django обробляється в інфраструктурі Django як рівень доступу до даних. На цьому рівні міститься все, що стосується даних: налаштування з'єднання, параметри валідації, відносини тощо. Django спочатку включає підтримку PostgreSQL (переважної бази даних для творців Django), MySQL, SQLite та Oracle. Інформація про вибір бази даних зберігається у файлі налаштувань, і рівень моделі залишається одним і тим самим незалежно від вибраної бази даних.

Моделі Django можна розглядати як опис схеми таблиць бази даних, представлений у вигляді Python-коду. Django використовує модель для генерації та виконання інструкцій SQL у базі даних. Результат, що повертається базою даних, Django переводить у структуру даних Python, доступну для використання у Django. Однією з переваг такого підходу є можливість перемикання між різними СУБД, наприклад, перехід з MySql на PostgreSQL без зміни моделей програми.

Структура web-програми фреймворку Django виглядає наступним чином:

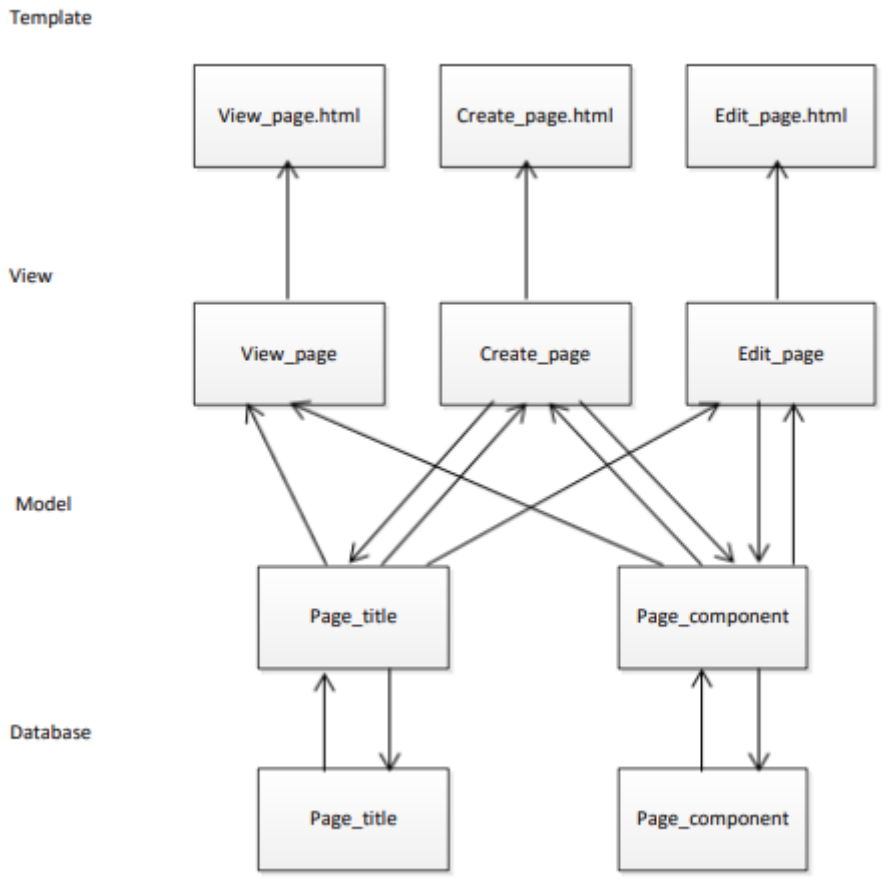


Рисунок 1.3 – Структура web-застосунку Django

Проаналізувавши плюси і мінуси цих двох фреймворків, я вирішив обрати Django для проекту, бо вважаю що він набагато краще за Flask.

1.3 Аналіз технічного завдання

Проаналізувавши потреби користувачів інтернету та наявні веб-сайти стало ясно, що найменше зустрічаються онлайн магазини, що продають кавомашини. Тому я вирішив зробити це темою мого веб-сайту.

Веб-сайт включає в себе такі функції:

- аутентифікацію користувачів з можливістю реєстрації нових користувачів;
- панель адміністратора сайту, де можна буде додавати, редагувати та керувати контентом;
- особа сторінка для менеджерів;
- обробка форм зворотнього зв’язку;
- зручний дизайн;
- кошик товарів;
- реалізація фільтрів для швидкого пошуку товару.

2 ПІДГОТОВКА СЕРЕДОВИЩА РОЗРОБКИ

2.1 Вибір IDE

Для Python існує багато цікавих IDE. Одні з найпопулярніших це: PyCharm, Visual Studio Code, Spyder та Jupyter Notebook. Щоб їх порівняти я вирішив створити таблицю, з неї ми наглядно побачимо плюси и мінуси цих програм.[22]

Таблиця 2.1 – IDE для Python

IDE	**Підтримка та спільнота**	**Ціна**	**Платформа**	**Призначення**
PyCharm	Велика спільнота, багато плагінів	Версія Community – безкоштовна, версія Professional – платна	Windows, macOS, Linux	Веб-розробка, розробка ПЗ
Visual Studio Code	Велика спільнота, багато плагінів	Безкоштовна	Windows, macOS, Linux	Універсальне
Spyder	Активна спільнота, багато плагінів	Безкоштовна	Windows, macOS, Linux	Наукове програмування

Продовження таблиці 2.1 – IDE для Python

Jupyter Notebook	Активна підтримка, велика спільнота	Безкоштовна	Windows, macOS, Linux	Аналіз даних, наукові дослідження

З цієї таблиці бачимо, що підходять дві програми – PyCharm та Visual Studio Code. Давайте краще їх розглянемо.

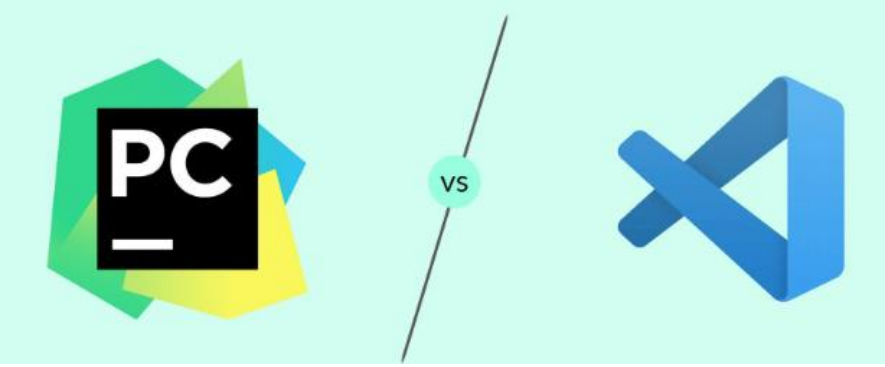


Рисунок 2.1 – Логотипи PyCharm та VS Code

Переваги PyCharm:

- PyCharm обробляє рутинні завдання, щоб ви могли зосередитися на важливих аспектах вашої роботи. PyCharm економить ваш час, усуваючи необхідність відволікатися від клавіатури для виконання більшості завдань;
- програма розуміє ваш код в деталях. Інтелектуальний аналіз коду забезпечує точне автозавершення, виявлення помилок і швидке їх виправлення, а також зручну навігацію по коду та інші корисні функції;
- допомагає писати простий в обслуговуванні, чистий код. IDE виконує контроль якості коду за допомогою перевірки відповідності PEP8, розумного рефакторингу, різних перевірок і допомоги в тестуванні;

- оскільки PyCharm був розроблений програмістами для програмістів, він містить все необхідне для продуктивної розробки на Python та активна підтримка зі сторони спільноти.

PEP 8, іноді згадуваний як PEP8 або PEP-8 – це документ, що містить рекомендації та найкращі практики написання коду на Python. Був написаний у 2001 році Гвідо ван Россумом, Баррі Уоршоу та Еліс Коглан. Основна мета PEP 8 – це покращити читабельність та узгодженість коду Python. PEP розшифровується як Python Enhancement Proposal, і існує багато PEP. Хоча ці документи в основному описують запропоновані нові можливості мови Python, деякі PEP зосереджені на дизайні та стилі і призначені для використання в якості ресурсу спільноти. PEP 8 є одним з таких стилістичних PEP.[24]

Як сказав Гвідо ван Россум: "Код читають набагато частіше, ніж пишуть". Можна витратити кілька хвилин або цілий день на написання коду для обробки автентифікації користувачів. Одного разу написаний, він навряд чи буде написаний знову. Однак, завжди доведеться його перечитувати. Цей шматок коду може залишитися частиною проекту, над яким йде робота. Кожного разу, коли програміст повертаєтеся до цього файлу, він повинен пам'ятати, що робить цей шматок коду і чому ви його написали.

Тому важлива читабельність. Буває важко згадати, що робить код через кілька днів або тижнів після того, як його написали.

Якщо програміст дотримується PEP 8, можете бути впевнені, що правильно назвали свої змінні. Ви будете знати, що додали достатньо пробілів, що полегшить відстеження логічних кроків коду. Це також полегшує коментування коду. Все це означає, що ваш код буде більш читабельним і на нього буде легше посилатися. Якщо ви початківець, дотримання правил PEP 8 зробить вивчення Python більш приємним заняттям

Якщо ви маєте великий досвід написання коду на Python, вам може знадобитися співпраця з іншими. Тут важливо писати читабельний код. Інші люди,

які ніколи не зустрічали вас і не бачили вашого стилю кодування, повинні мати можливість читати і розуміти ваш код. Наявність керівних принципів, яких ви дотримуєтесь і знаєте, полегшить іншим читання вашого коду.

Розглянемо переваги VS Code:

- підтримує широкий спектр мов програмування, включаючи Java, Python, C++ та JavaScript. Такі функції, як підсвічування синтаксису, завершення коду та специфічні для кожної мови інструменти, роблять його ідеальним для розробників, які працюють з декількома мовами;
- програма пропонує багато можливостей як потужний редактор коду. Основні функції включають інтеграцію з Git'ом, інструменти для налагодження та розширення, які дозволяють налаштовувати робочий процес;
- Visual Studio Code дуже добре налаштовується, дозволяючи користувачам адаптувати інтерфейс і комбінації клавіш до своїх уподобань. Це робить його ідеальним для розробників, які хочуть адаптувати середовище кодування до своїх конкретних потреб;
- програма має велику спільноту розробників, яка створює та підтримує розширення та плагіни, що додають нові можливості до редактора. Це означає, що ви можете знайти широкий спектр розширень, які допоможуть вам у вашому робочому процесі кодування;
- працює швидко та ефективно і займає мало місця. Це робить його ідеальним для розробників, яким потрібен редактор коду, що не сповільнює роботу комп'ютера.[11]

Обидва редактори, PyCharm та VS Code, мають свої переваги, які роблять їх популярними серед розробників. Я обрав PyCharm бо він призначений для розробки на Python, тобто, може запропонувати більш спеціалізовані інструменти та функції для роботи з мовою Python порівняно з VS Code, який підтримує ширший спектр мов. Також програма надає вбудовані інструменти для контролю якості коду, включаючи перевірку відповідності PEP8, що має велике значення для

Python розробників, та розумний рефакторинг. Це дозволяє писати чистіший, більш структурований код.

Хоча VS Code має багато можливостей, PyCharm зазвичай вважається більш потужним і гнучким середовищем розробки, особливо для великих і складних проектів. Тому, як ентузіаст Python і людина, яка шукає інструмент, що допоможе підвищити продуктивність і якість коду, я обираю PyCharm.

2.2 Початок створення проекту

Оскільки IDE дає вибір для створення проекту особисто для використання Django, я створюю проект з такими налаштуваннями (Встановлення фреймворку робить сама програма):

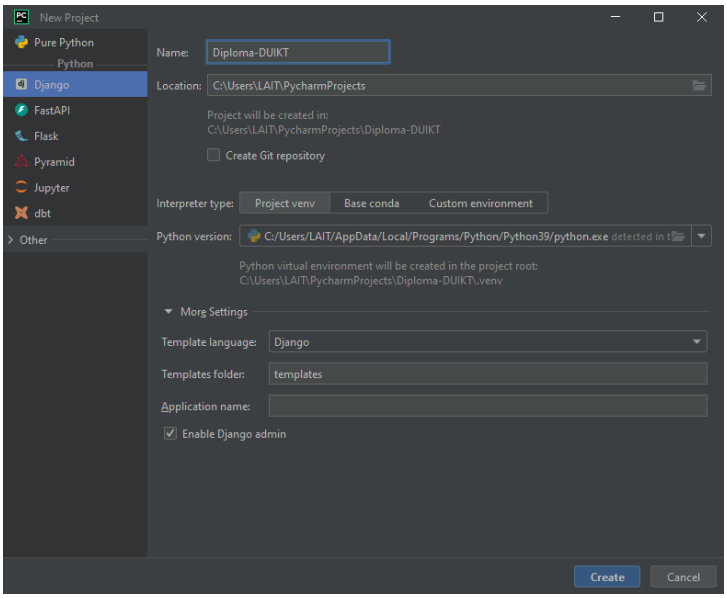


Рисунок 2.2 – Налаштування проекту

Після створення проекту буде виглядати так:

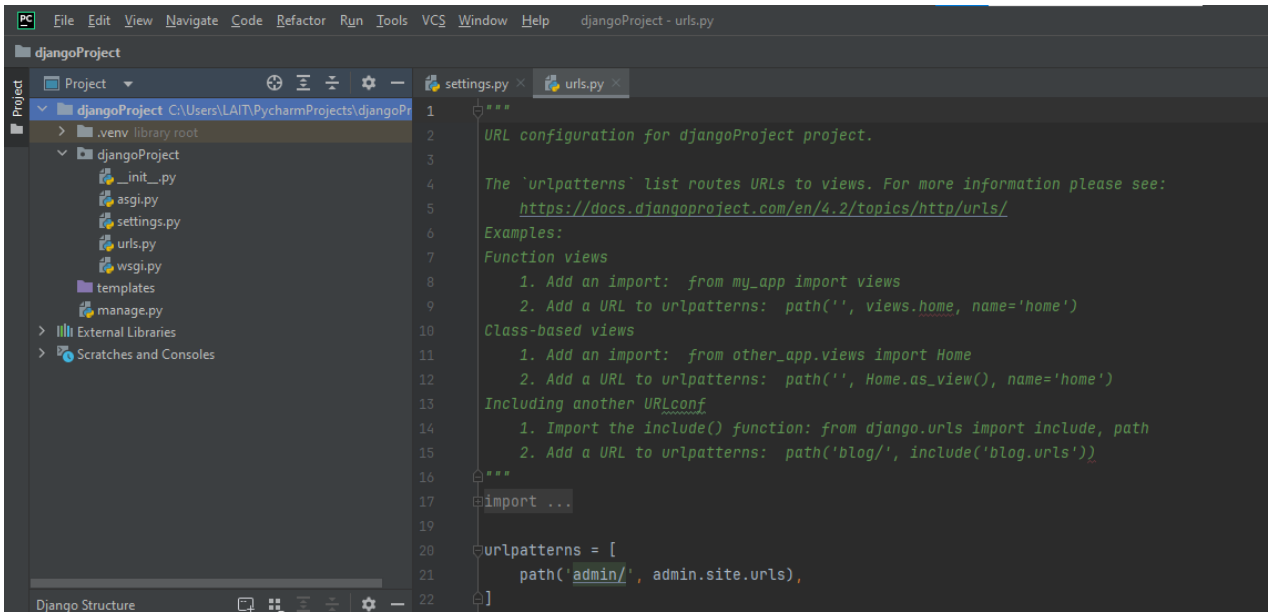


Рисунок 2.3 – Створений проект

Основна папка проекту мість такі файли як: ulrs.py, settings.py та папку «templates». Саме у цій папці ми будемо тримати наші HTML документи.

Час перевірити чи правильно створився проект. Для цього ми відкриваємо вкладку «Tools» і обираємо «Run manage.py Task». За допомогою manage.py можна взаємодіяти з проектом. У PyCharm запустити проект також можливо натиснувши на зелений трикутник у правому верхньому кутку програми. [6]

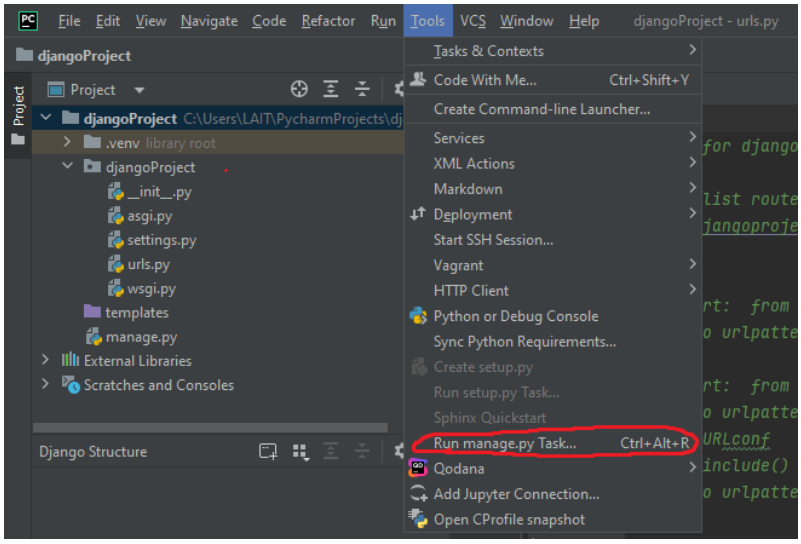


Рисунок 2.4 – manage.py

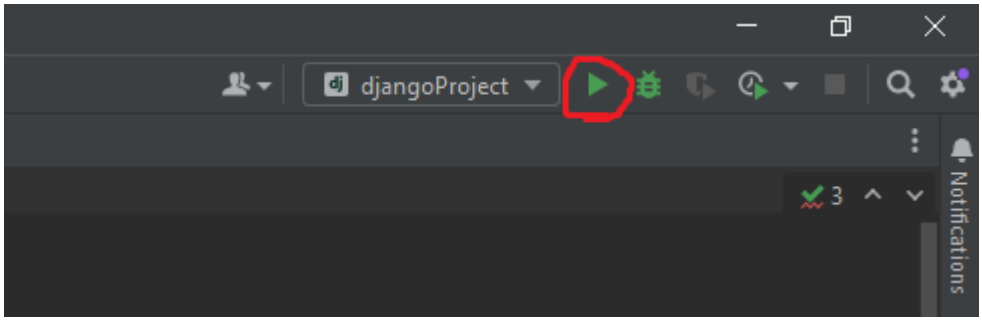


Рисунок 2.5 – Запуск з програми

Після однієї з цих дій, ми бачимо в командному рядку адресу нашого серверу:

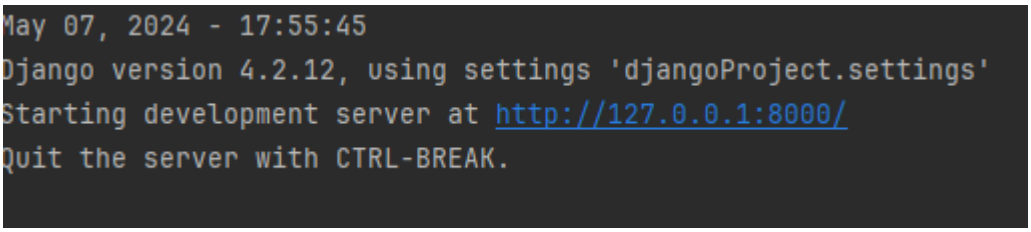


Рисунок 2.6 – Адреса серверу

Переходимо по цій адресі і якщо все було створено коректно, то бачиму таку сторінку в браузері:

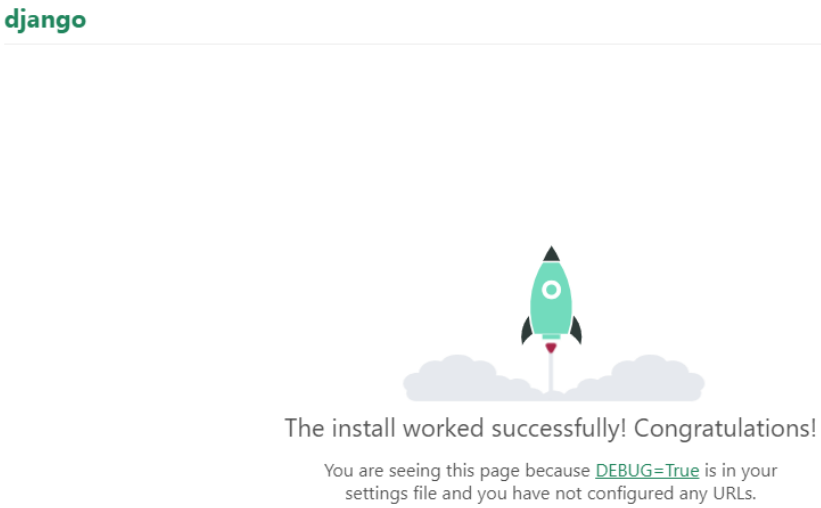


Рисунок 2.7 – Успішне встановлення Django

2.3 Вибір бази даних

База даних – це впорядкований набір структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді комп'ютерної системи. База даних зазвичай управляється системою управління базами даних (СУБД). Дані разом із СУБД, а також додатки, які з ними пов'язані, називаються системою баз даних, або, для стислості, просто базою даних.[17]

У найпоширеніших сучасних базах даних дані, зазвичай, зберігаються у вигляді рядків і стовпців, що утворюють таблиці. Цими даними можна легко керувати, змінювати, оновлювати, контролювати та організовувати. Більшість баз

даних використовують мову структурованих запитів(SQL) для опису та запитів до даних.

Бази даних значно еволюціонували з моменту їх появи на початку 1960-х років. Перші бази даних базувалися на різних моделях, таких як ієрархічні бази даних (на основі деревовидної моделі, яка дозволяла встановлювати зв'язки лише "один до багатьох") та мережеві бази даних (більш гнучка модель, яка дозволяла встановлювати множинні зв'язки).[15] Незважаючи на свою простоту, ці ранні системи були не гнучкими: реляційні бази даних стали популярними у 1980-х роках, а об'єктно-орієнтовані бази даних – у 1990-х роках. Зовсім недавно розвиток інтернету і потреба в аналізі неструктурованих даних призвели до появи баз даних NoSQL. Сьогодні хмарні та автономні бази даних відкривають нові можливості для збору, зберігання, використання та управління даними.

Можна подумати що електронні таблиці і бази даних не одне й те саме, але тут є свої нюанси. Бази даних та електронні таблиці (наприклад, Microsoft Excel) забезпечують зручний спосіб зберігання інформації. Основні відмінності між ними полягають у наступному:

- як дані зберігаються та обробляються;
- права доступу до даних;
- обсяг даних, що зберігаються.

Електронні таблиці спочатку були розроблені для одного користувача, і їхня функціональність відображає це. Вони найкраще підходять для одного користувача або невеликої кількості користувачів, яким не потрібно виконувати складні операції з даними. Бази даних, з іншого боку, призначені для зберігання організованої інформації, іноді великих обсягів інформації. Вони дозволяють багатьом користувачам отримувати доступ до даних і запитувати їх одночасно, швидко і безпечно, використовуючи складну логіку і мови запитів.

Існують різні типи баз даних. Вибір найкращої бази даних для конкретної компанії залежить від того, як компанія має намір використовувати дані. Вони

класифікуються за типом вмісту, наприклад, бібліографічні, повнотекстові, числові, графічні тощо. У сфері комп'ютерних технологій бази даних часто класифікують за організаційними підходами.[14]

До основних організаційних баз даних відносяться наступні:

- *Реляційні*. У цьому табличному підході дані реорганізуються різними способами і визначаються таким чином, щоб до них можна було отримати доступ. Реляційні бази даних складаються з таблиць. У таблицях дані розподіляються на заздалегідь визначені категорії. Кожна таблиця має стовпці з принаймні однією категорією даних і рядки з конкретними екземплярами даних категорії, визначеної в стовпці. Інформація в реляційній базі даних про конкретного клієнта організована в рядки, стовпці та таблиці. Вони індексуються для полегшення пошуку за допомогою SQL або NoSQL запитів. Реляційні бази даних використовують SQL у своїх користувацьких інтерфейсах і додатках. Нові категорії даних можна легко додавати до реляційної бази даних без модифікації існуючих додатків. Системи управління реляційними базами даних (СУБД) використовуються для зберігання, керування, запиту та отримання даних у реляційних базах даних. Як правило, СУБД надають користувачам можливість керувати доступом на читання/запис, визначати звітність та аналізувати використання. Деякі бази даних забезпечують атомарність, узгодженість, ізольованість, довговічність або сумісність з ACID для забезпечення узгодженості даних і завершення транзакцій.
- *Розподілені*. Бази даних, які зберігають записи та файли в декількох фізичних місцях. Обробка даних також розподілена і реплікується в різні місця мережі. Розподілені бази даних можуть бути однорідними, коли всі фізичні місця базуються на однаковому обладнанні і працюють під управлінням однієї операційної системи та додатків для роботи з базами даних. Вони також можуть бути гетерогенними. У таких випадках апаратне забезпечення, операційна система та програма для роботи з базами даних будуть відрізнятися залежно від місця розташування.

- *Хмарні*. Ці бази даних створюються в гібридній хмарі з публічних, приватних або віртуальних середовищ. Користувачі платять відповідно до обсягу пам'яті та пропускної здатності, які вони використовують. Вони також забезпечують масштабованість і високу доступність за вимоги. Ці бази даних можуть бути інтегровані з додатками, розгорнутими як програмне забезпечення, як послуга (SaaS).

- *NoSQL*. Бази даних NoSQL підходять для роботи з великими колекціями розподілених даних. Вони можуть вирішувати проблеми продуктивності великих даних краще, ніж реляційні бази даних. Вони також підходять для аналізу великих неструктурованих наборів даних і даних на віртуальних серверах у хмарі. Ці бази даних також називають нереляційними базами даних.

- *Об'єктно-орієнтовані*. Бази даних, які зберігають дані, створені за допомогою об'єктно-орієнтовних мов програмування. Акцент робиться на об'єктах, а не на діях, і на організації даних, а не на логіці. Наприклад, записи даних про зображення є об'єктами даних, а не алфавітно-цифровими.

- *Графові* бази даних. Графові бази даних зберігають дані у вигляді сутностей і зв'язків між сутностями. Ці бази даних часто використовуються для аналізу взаємозв'язків. Графові бази даних часто використовуються для аналізу даних про клієнтів, які взаємодіють з компанією на веб-сторінці або в соціальних мережах. Для аналізу цих баз даних використовують SPARQL, декларативну мову програмування і протокол для аназізу. SPARQL може виконувати всі аналізи, які може виконувати SQL, а також може використовуватися для семантичного аналізу і перевірки взаємозв'язків. Тому він корисний для виконання аналізу наборів даних, що містять як структуровані, так і неструктуровані дані.

Для нашого проекту найбільш підходить база даних SQLite. Це одна з найпопулярніших і найпростіших у використанні систем реляційних баз даних. Вона має багато особливостей порівняно з іншими реляційними базами даних. Багато великих міжнародних компаній, таких як Adobe, використовують SQLite як

формат файлів програми для свого продукту Photoshop Lightroom. Airbus, європейська багатонаціональна аерокосмічна компанія, використовує цю базу даних для свого програмного забезпечення для літаків сімейства A350 XWB.

Рисунок 2.8 – Логотип SQLite

SQLite – це вбудована безсерверна реляційна система керування базами даних. Це бібліотека з відкритим вихідним кодом, яка не потребує конфігурації чи встановлення. Вона також дуже зручна, оскільки її розмір становить менше 500КБ, що значно менше, ніж у інших систем управління базами даних.

У 2000 році Річард Хіпп розробив SQLite, щоб «не вимагати адміністрування» для роботи програми, а у серпні 2000 року було випущено SQLite 1.0 з менеджером баз даних GNU. В 2011 році Hipp оголосив про додавання інтерфейсу UNQl до бази даних SQLite для розробки UNQLite (документно-орієнтована база даних).

Ось чому я обрав SQLite:

- це програмне забезпечення з відкритим вихідним кодом. Після встановлення не потрібна ніяка ліцензія;

- вона є безсерверною, оскільки не потребує жодних інших серверних процесів чи систем;
- база даних є гнучкою і дозволяє працювати з декількома базами даних одночасно в одному сеансі;
- крос-платформа СУБД і працює на всіх платформах, включаючи macOS і Windows;
- не потребує конфігурації. Непотрібно ніякої конфігурації або адміністрування.

3 РОЗРОБКА ФУНКЦІОНАЛЬНОСТІ ОНЛАЙН МАГАЗИНУ

3.1 Початок роботи

Почнемо з шаблону. Я створив шаблон нашого сайту, який складається з HTML, CSS та JS файлів і зображень. Усі вони – статичний ресурс. Це означає те, що змінення контенту на сайті можливе лише через змінення цих файлів. Наша задача проаналізувати що нам потрібно зберігати в базі даних, що буде залишатись без змін, що буде підтягуватися з бази, збереження форм у базі з можливістю перегляду їх менеджером та наповнення реальними даними.

HTML шаблон складається з секцій, у нас це: top bar, header, hero, about, main (why us, products, gallery, review, contact us) та footer. І ми розуміємо, що буде 3 типи користувачів з різним рівнем доступу: звичайний користувач, менеджер і адміністратор. Подивившись на сторінку, можна зрозуміти, що змінюватись буде тільки секція «main» і інші секції будуть не змінними. Тому ми помічаємо цю секцію як «Content» і переміщаємо його в окремий файл і пишемо, що цей файл розширяє основний шаблон.

«Content» складається з секцій why us, products, gallery, review та contact us і вони будуть знаходитися в окремих HTML файлах(попередньо позначивши, що вони включаються в «Content»), що буде нам давати змогу безболісно видаляти, додавати нові секції та легше редагувати їх.[25]

CSS, JS файли і зображення є також статикою, що на сервері має знаходитися в окремій папці «static». Тому ми переміщаємо всю статику у цю папку, а основний файл «index.html» – у папку templates.[13]

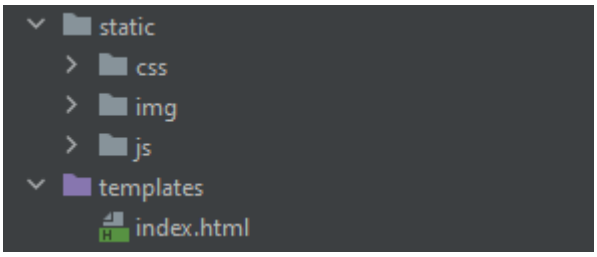


Рисунок 3.1 – Створені директорії

Але ще нам треба підключити папку «static» нашому HTML-документу. Тому заходимо в шаблон і пишем наступну команду угорі документа.

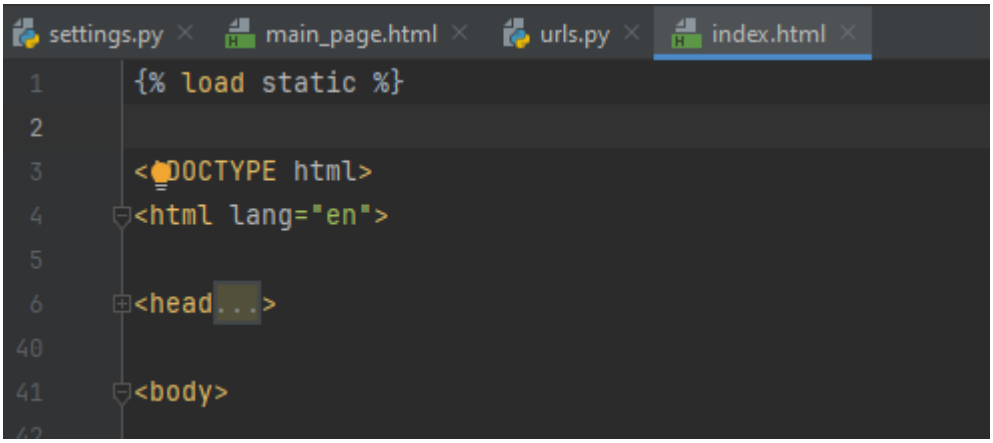


Рисунок 3.2– Підключення статичних файлів

Після чого, прописуємо статику для всіх шляхів до CSS та JS файлів і зображень, які не будуть змінюватись через базу даних.

```
<!-- Favicons -->
<link href={% static 'img/favicon.png' %} rel="icon">
<link href={% static 'img/apple-touch-icon.png' %} rel="apple-touch-icon">

<!-- Google Fonts -->
<link href="https://fonts.googleapis.com/css?family=Poppins:300,300i,400,400i,600,600i,700,700i|Satisfy|Comic+Neue:300

<!-- Template Main CSS File -->
<link href={% static 'css/style.css' %}  rel="stylesheet">
```

Рисунок 3.3 – Шлях до статичних файлів

І щоб створити блок «Content» як ми планували треба створити застосунок, що буде працювати з цим блоком. Реалізувати це можна через команду python manage.py startapp *назва*. Я назвав його main_page.

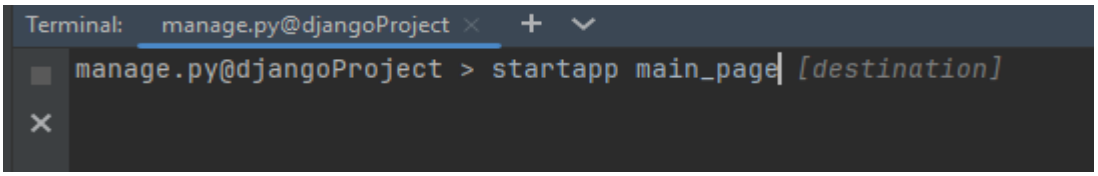


Рисунок 3.4 – Команда створення застосунку

Створилася директорія одразу з файлами. Але перед тим як працювати з ними треба занести назву у файл settings.py щоб все коректно працювало.

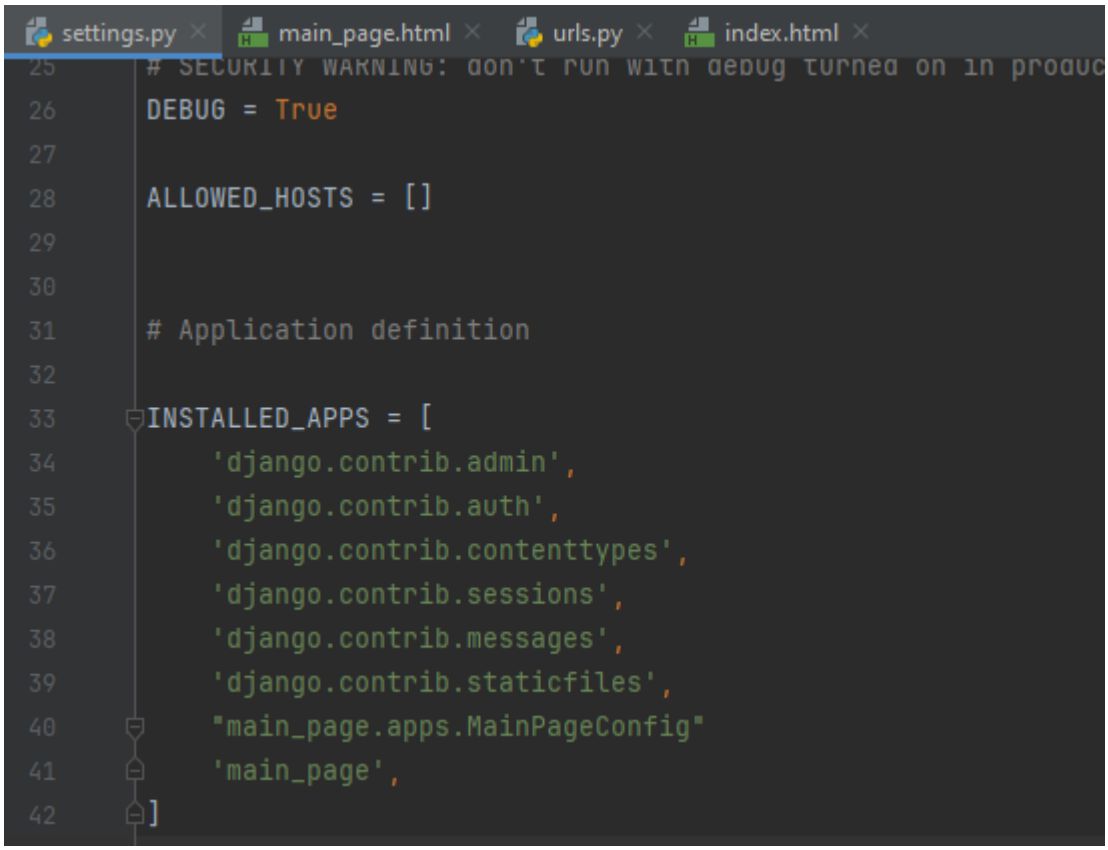


Рисунок 3.5 – settings.py

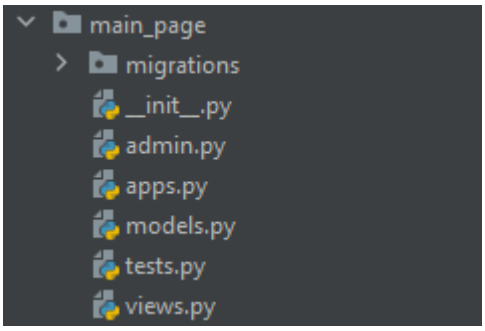


Рисунок 3.6 – Файли директорії main_page

Тепер ми можемо реалізувати нашу задумку, що писали на початку розділу. Для цього, в директорії main_page/templates сторюємо HTML-документ з назвою «main» і в ньому пишемо команду що він доповнює основний шаблон і містить блок «Content»

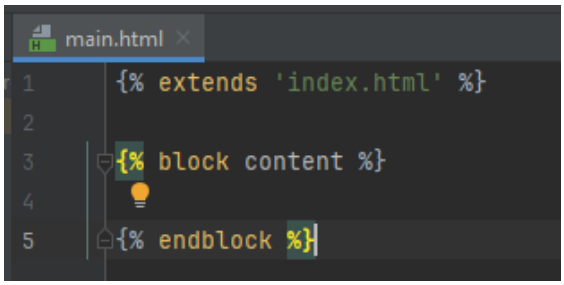


Рис 3.7 – Вигляд main.html

Переходимо у основний документ. Звідти копіюймо і вирізаємо весь тег «main», а замість нього пишемо команду, що буде перенаправляти на документ, який створили раніше. В цей документ, в блок «Content» вставляємо скопійований тег «main».

Рисунок 3.8 – Основний HTML-документ

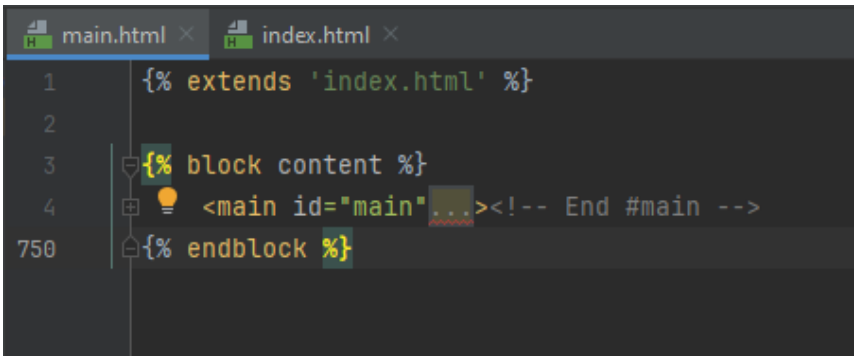


Рисунок 3.9 – Дочірній HTML-документ

Ми рознесли шаблони. Тепер треба додати шлях до сайту і панелі адміністратора.

Відкриваємо файл «urls.py», основної директорії. Багато в чому urls.py є точкою входу в проект – зазвичай це перший файл, який ми відкриваємо, коли знайомимося з проектом Django. Це все одно, що прочитати мапу перед тим, як досліджувати місто. В основному, urls.py містить кореневі налаштування URL і URL Conf для всього проекту.

Це список шаблонів Python, присвоєних глобальній змінній urlpatterns. Вхідні URL-адреси порівнюються з кожним шаблоном зверху вниз. Пошук зупиняється при першому збігу, і запит надсилається до відповідного подання. Ми вписуємо туди такі рядки:

```
17  from django.contrib import admin
18  from django.urls import path, include
19
20  urlpatterns = [
21      path('admin/', admin.site.urls),
22
23      path('', include( arg: 'main_page.urls', namespace='main_page'))
24  ]
25
```

Рисунок 3.10 – Посилання на основну сторінку

Ця команда при запиті перенаправляє користувача на файл у нашому застосунку, який буде відкриватися. Також ми додали параметр «namespace», щоб нам було простіше звертатися до посилання і назвали ми його так як і директорію, на яку ми перенаправляємось щоб не заплутатися в подальших назвах.

В директорії main_page створюємо файл «urls.py». В ньому нам необхідно підключити всі модулі та дати ім’я застосунку так як у нас він буде великий і будуть різні користувачі та посилання, то нам потрібно буде потім пересуватися між ними.

Щоб уникнути великої кількості кодингу і спростити собі роботу ми будемо додавати імена для наших майбутніх застосунків.

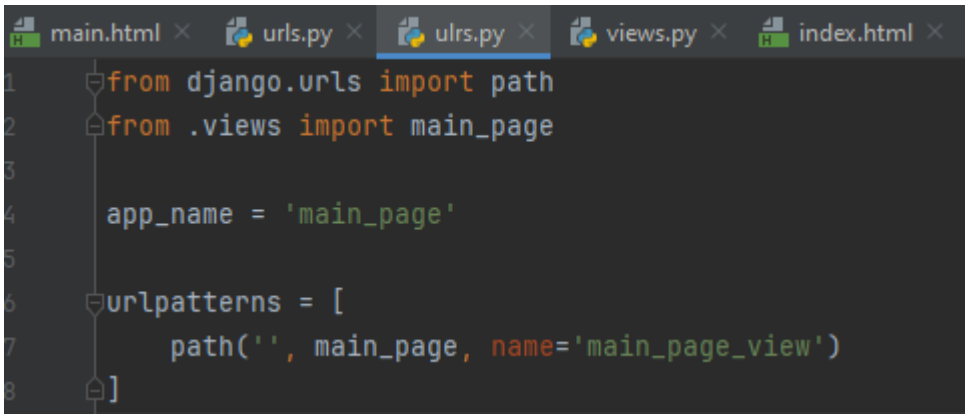


Рисунок 3.11 – main_page.urls.py

Тепер переходимо у views.py. Функція перегляду, або скорочено view – це функція Python, яка приймає веб-запит і повертає веб-відповідь. Цією відповіддю може бути HTML-вміст веб-сторінки, перенаправлення, помилка 404, XML-документ або зображення і т.д. Це може бути що завгодно. Саме подання містить будь-яку логіку, необхідну для повернення відповіді. Цей код може бути де завгодно на шляху Python. Це як "магія", так би мовити. Для того, щоб розмістити код десь, загально прийнятою практикою є розміщення файлу з назвою views.py в каталозі проекту або додатку. У ній ми створюємо функцію, що буде виводити дані користувачу.[16]

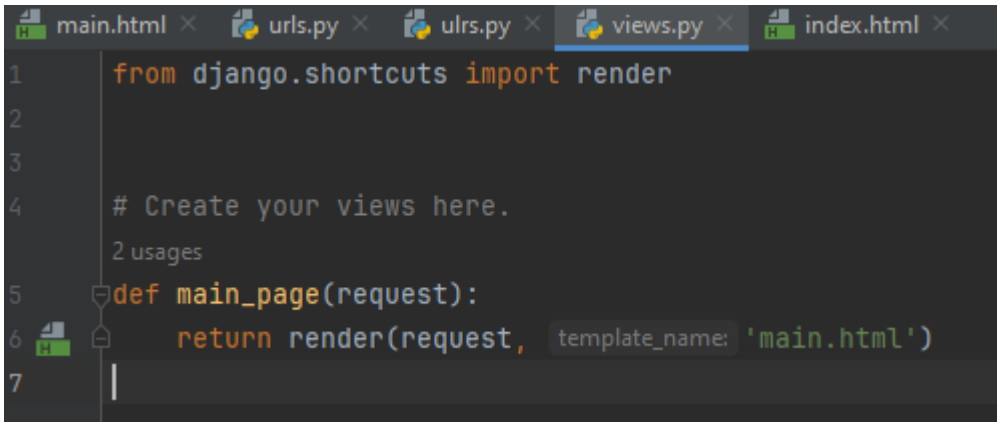


Рисунок 3.12 – views.py

Запустивши проект, ми побачимо, що статика не підгружається, а тільки HTML–документ. Медіа файли не грузить бо ми їх будемо брати з бази даних(окрім деяких, що не потрібно змінювати) і тому це вже буде обробка «MEDIA». Це все тому що ми маємо прописати статику в налаштуваннях. Після цих дописів все запустилось коректно:

```
118  # Static files (CSS, JavaScript, Images)
119  # https://docs.djangoproject.com/en/4.2/howto/static-files/
120
121  STATIC_URL = 'static/'
122  STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
123
124  STATICFILES_DIRS = (
125      os.path.join(BASE_DIR, 'static'),
126  )
```

Рисунок 3.13 – це settings.py

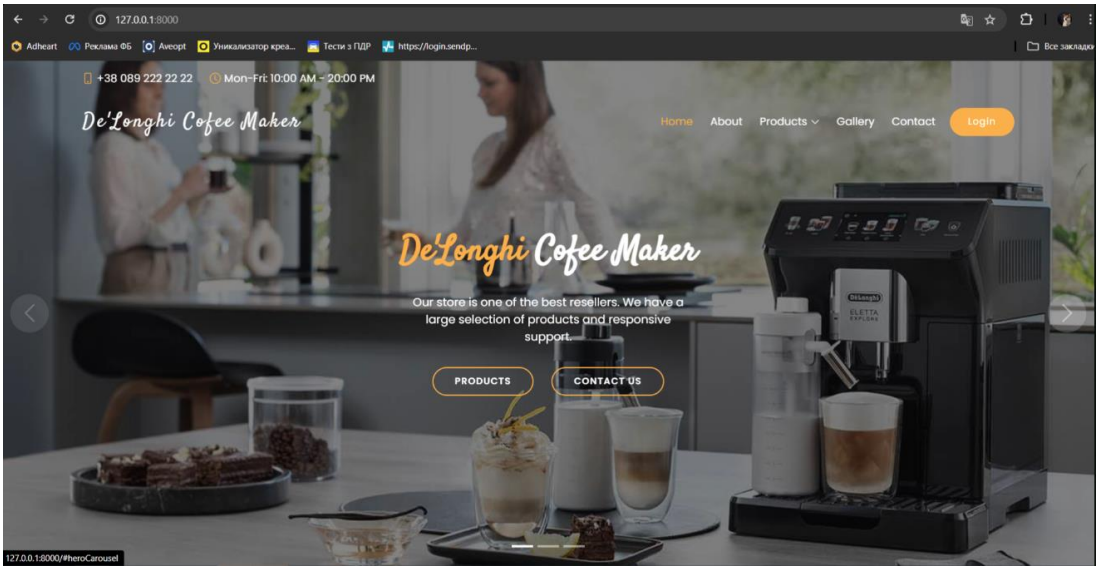


Рисунок 3.14 – Начальний вигляд сайту

Для кращої роботи з проектом та орієнтації в ньому розділимо секції з «Content» на окремі файли. Створимо наступні файли у папці «templates» директорії main_page:

- cart.html – Кошик товарів
- whyus.html – розпис чому варто обрати нас
- product.html – частина сайту, де будуть всі продукти, що ми продаємо
- gallery.html – галерея товарів користувачів
- reviews.html – відгуки користувачів
- contactform.html – форма для отримання відповідей на запитання або порад

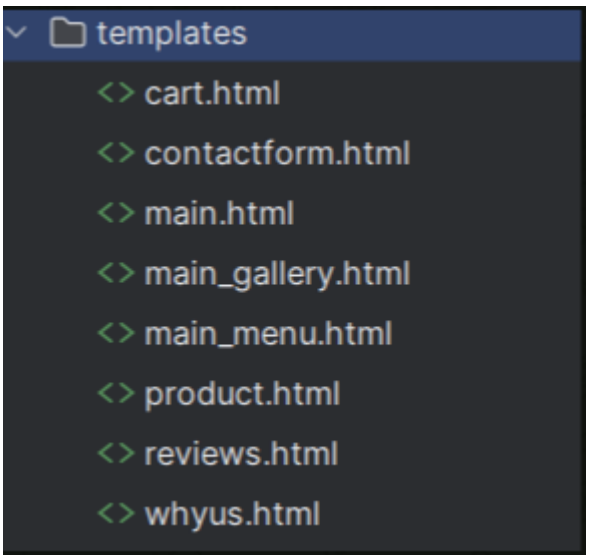
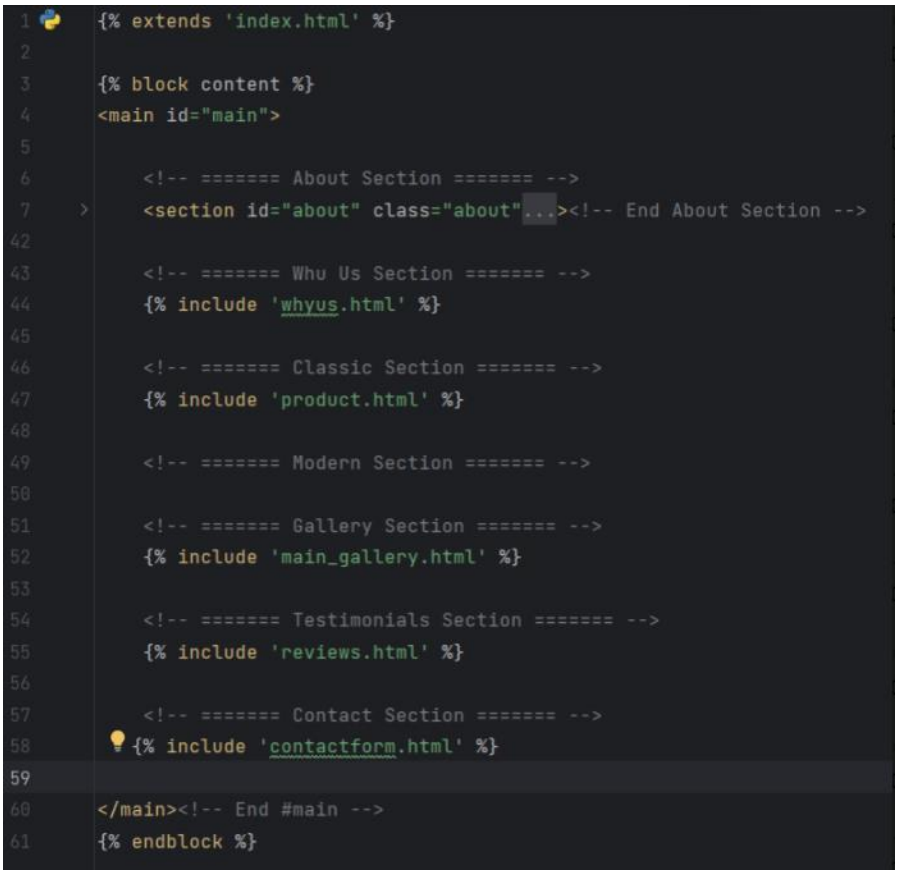


Рисунок 3.15 – Розділ на секції

Щоб файли коректно працювали з main.html треба в ньому замість секуцій написати наступну команду: «*{% include «назва файлу секції» %}*». Так робимо з кожною секцією. Тепер файл main.html виглядає так:



```
{% extends 'index.html' %}

{% block content %}
<main id="main">

    <!-- ======= About Section ======= -->
    <section id="about" class="about"...><!-- End About Section -->

    <!-- ======= Whu Us Section ======= -->
    {% include 'whyus.html' %}

    <!-- ======= Classic Section ======= -->
    {% include 'product.html' %}

    <!-- ======= Modern Section ======= -->

    <!-- ======= Gallery Section ======= -->
    {% include 'main_gallery.html' %}

    <!-- ======= Testimonials Section ======= -->
    {% include 'reviews.html' %}

    <!-- ======= Contact Section ======= -->
    {% include 'contactform.html' %}

</main><!-- End #main -->
{% endblock %}
```

Рисунок 3.16 – main.html

3.2 Створення моделей

Для створення моделей для бази даних нам потрібно у створеній директорії відкрити файл models.py. Models.py – одна з найважливіших концепцій фреймворку Django. Вона дозволяє повністю визначити базу даних нашого веб-додатку, дозволяючи нам:

- оголосити таблиці та поля бази даних;
- визначити всі мета дані бази даних;
- визначати поведінку кожного поля.

Нам знадобиться створити клас «Категорії», в якому будуть атрибути: name, is_visible та position. Атрибут name буде мати тип даних CharField. Це тип даних, який зберігає символи або текстові рядки короткої або середньої довжини і тому підходить для зберігання атрибутів, таких як імена і т.д. CharField має параметр max_length, який ми вказуємо на 50. Також вказуємо параметр унікалькість, щоб не можна було створити однакові категорії і так як це основне поле в класі, то ми включаємо індексацію. Індексація – це метод структури даних для швидкого пошуку записів у базі даних.

Атрибуту «is_visible» вказуємо тип даних BooleanField – це поле true/false. За замовчуванням ми ставимо true, бо тих категорій, що буде видно, завжди буде більше ніж тих, що будуть приховані. Поле «position» буде допомагати нам у пошуку продуктів, що належать до однієї категорії і буде мати тип даних PositiveSmallIntegerField з параметром унікальності. Цей тип даних обмежує набір тільки додатними значеннями від 0 до 32767.

Після цього нам потрібно написати метод «str». Він буде відображати дані на сторінці адміністратора і для більш легкого аналізу даних ми припишемо відображення назви категорії, її позиції в меню та потім чи активна категорія. Останнім прописуємо клас «Meta». Цей клас використовуються для зміни поведінки моделі, такі як ordering та verbose_name. Нам цікавий параметр ordering, бо він буде визначати поле, по якому ми будемо сортувати. Вказуємо поле «position». Після всіх дій наша модель буде виглядати так:

```python
from django.db import models


4 usages
class Category(models.Model):
    name = models.CharField(max_length=50, unique=True, db_index=True)
    is_visible = models.BooleanField(default=True)
    position = models.PositiveSmallIntegerField(unique=True)

    def __str__(self):
        return f'{self.name}| Visible:{self.is_visible}| Pos:{self.position}'

    class Meta:
        ordering = ('position', )
```

Рисунок 3.17 – Перша модель

Тепер ми відкриваємо «manage.py» і прописуємо команду «makemigrations». Після чого у командному рядку ми бачимо, що було знайдено створену модель і з’явилась директорія migrations, де прописаний скрипт, що створить у базі даних таблицю. Для цього ми прописаємо команду «migrate», що буде означати для Python’а «Створи таблицю».

```
manage.py@djangoProject > makemigrations
C:\Users\LAIT\PycharmProjects\djangoProject\.ve
Tracking file by folder pattern:  migrations
Migrations for 'main_page':
  main_page\migrations\0001_initial.py
    - Create model Category

Following files were affected
 C:\Users\LAIT\PycharmProjects\djangoProject\ma
Process finished with exit code 0
```

Рисунок 3.18 – «makemigrations»

```
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0011_update_proxy_permissions... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying main_page.0001_initial... OK
Applying sessions.0001_initial... OK
```

Рисунок 3.19 – «migrate»

Далі ми реєструймо модель в admin.py. admin.py – це скрипт, який створить необхідну структуру директорій та файли для нас. Вставимо команду в кінці цього файлу, щоб зареєструвати модель. Цей код просто імпортує моделі і реєструє її за допомогою виклику admin.site.register.

```python
from django.contrib import admin
from .models import Category


# Register your models here.
admin.site.register(Category)
```

Рисунок 3.20 – admin.py

Тепер треба створили superuser’a. Це обліковий запис з високим ступенем привілеїв, які в основному використовуються для адміністрування. Створюється він за допомогою команди «createsuperuser» у manage.py. Далі нас попросять вписати ім’я користувача, електрону пошту та пароль. Після вводу всіх даних успішно був створений супер користувач. Запускаємо код, за заходимо в панель адміністратора.

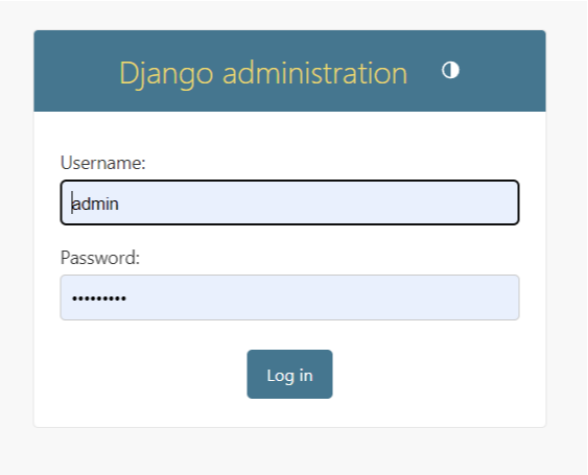


Рисунок 3.21 – Вхід

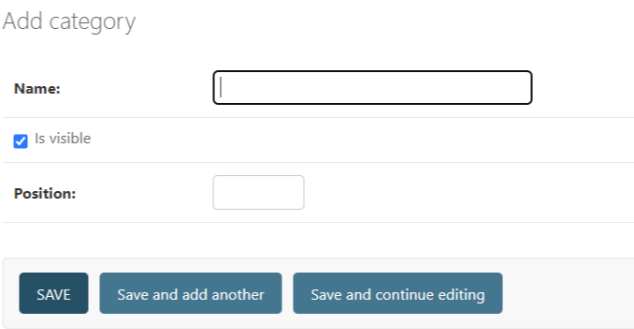


Рисунок 3.22 – Додавання категорії

Так як каво машини на сайті будуть ділитися на 4 категорії, то я в таблиці 4 записи.

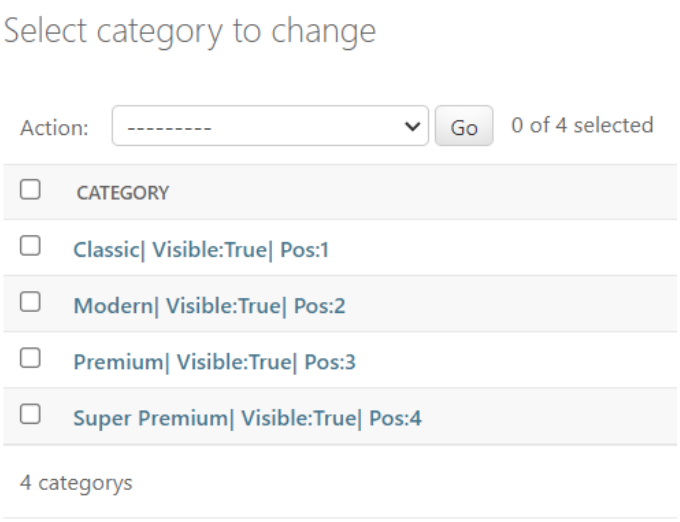


Рисунок 3.23 – Всі категорії

Як ми бачимо на рисунку 3.23, сортування по позиції чудово працює. Воно придає охайності панелі адміністрування.

Тепер створимо клас для самих кофе машин. Він буде називатися «Products» і буде включати в себе наступні атрибути: name, price, description, short_description, photo, in_stock, category, position та is_visible. Атрибути name, position і is_visible та in_stock ми вписуємо з таким самим чином як і в минулому класі(in_stock має такі самі значення, що й is_visible).

Атрибут price буде мати тип даних DecimalField, що дасть змогу вводити тільки цифри. Значення для атрибуту ми впишемо max_digits=8. Це буде значити, що цінник не буде перевищувати 6 символів, значення decimal_places=2, буде значити, що після точки можливо ввести тільки 2 символа.

Атрибут description матиме TextField. Це велике текстове поле для великого тексту. TextField зазвичай використовується для зберігання абзаців та інших текстових даних. Для рядків найчастіше використовують або CharField, або TextField. Поле моделі CharField використовується для невеликих і середніх текстів і дозволяє вказати максимальну довжину рядка, залежно від обмежень, дозволених базою даних. На відміну від CharField, TextField не має ніяких обмежень на

кількість символів. Основна відмінність між ними полягає в розмірі рядка, що зберігається в базі даних. CharField зберігає невеликі або середні рядк из максимальним обмеженням, визначеним у більшості баз даних. Інавпаки, TextField використовується для великих рядків без максимального обмеження. В нашому випаду опис продукту містить велику кулькість тексту, тому цей атрибут найкраще нам підходить, але ми обмежмо його значенням max_length до 500 символів. А ось під атрибут short_description ми використаємо CharField з обмеженням у 150 символів.[28]

Атрибут category буде мати тип даних ForeignKey та буде посилатися на таблицю «Category». У Django є три основні типи зв'язків: «один-до-одного», «багато-до-багатьох» і «багато-до-багатьох». Зв'язок «багато-до-одного» – тип зв'язку, коли кілька записів в одній таблиці пов'язані з одним записом в іншій таблиці. Припустимо, нашій базі даних є дві таблиці: «Відділі» і «Працівник». Зв'язок між «Відділом» і «Працівником» - це зв'язок один-до-багатьох:

- у відділі є багато працівників, кожен з яких належить до одного відділу;
- зв'язок між працівником і відділом – зв'язок «багато-до-одного».

Це можна проілюструвати на діаграмі нижче:

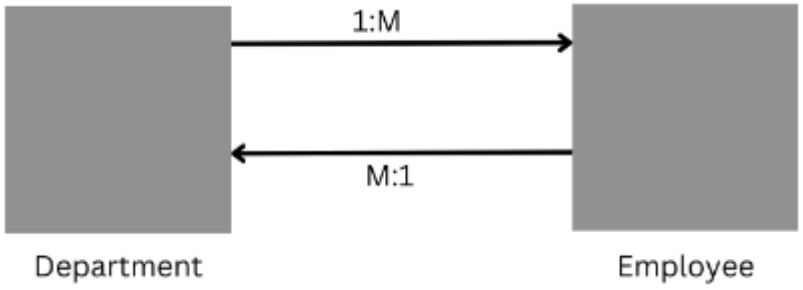


Рисунок 3.24 – Зв’язок «багато-до-одного»

Наприклад, бухгалтерський відділ може мати багато співробітників (один до багатьох), і всі ці співробітники належать до бухгалтерського відділу (багато до

одного). ForeignKey використовується для з’єднання двох таблиць і встановлення зв’язку «багато до одного».

В цей атрибут ми введемо значення «on_delete=models.CASCADE». Це означає, що коли видалиться категорія, товари, що знаходились в ній не видаляться, а значення «категорія» в них стане «null».

Для атрибуту photo ми маємо спеціальне поле, називається ImageField. Поле ImageField – це поле FileField, і завантаження обмежується лише форматами зображень. Перед завантаженням файлу необхідно вказати ряд налаштувань, щоб гарантувати безпечне зберігання файлу і можливість його зручного пошуку. Типовим віджетом форми для цього поля є ClearableFileInput, але ми його не будемо використовувати. На додаток до спеціальних атрибутів, доступних для поля FileField, поле ImageField також має атрибути висоти і ширини. Але ми будемо використовувати більш професійну бібліотеку – ImageKit.

ImageKit – це програма Django для обробки зображень. Потрібні чорно-білі версії завантажених користувачем зображень? Якщо вам потрібно створити окреме зображення з вашої програми, вам потрібен ImageKit. Вона постачається з низкою інструментів для обробки зображень для типових завдань, таких як зміна розміру, обрізання та покращення зображення.

В ImageField ми вказуємо в параметр upload_to функцію (яку ми припишемо в цьому класі, тільки вище), що буде зберігати розширення файлу і замість назви файлу буде формувати унікальне ім’я файлу. Формування унікальної послідовності дуже низька. Ці всі дії гарантують нам, що під час залиття одного з тогож замого фото на 2 продукти, при видалені одного з них, фото іншого продукту залишиться, а не видалиться разом з першим.

Потім нам потрібно зробити однаковий розмір у зображень. Тут в роботу вступає ImageKit. За допомогою типу даних ImageSpecField ми зможемо прописати висоту и ширину зображення, що буде взяте з ImageField і також зробити його більш якісним.[26]

Оформлюємо метод «str» прописуючи туди ім’я, ціну, наявність і чи видно продукт. Наш клас має такий вигляд:

Рисунок 3.25 – Таблиця для продуктів

Коли ми вже завантажуємо фотографії, то це вже медіа інформація і нам потрібно «сказати» коду де ця інформація має зберігатись. Прийнято, що медіа-файли завжди зберігаються у папці media. Створюємо її і потім відкриваємо файл settings.py, де вказуємо, там де ми додавали статику, що медіа зберігається в папці media і додаємо шлях до неї.

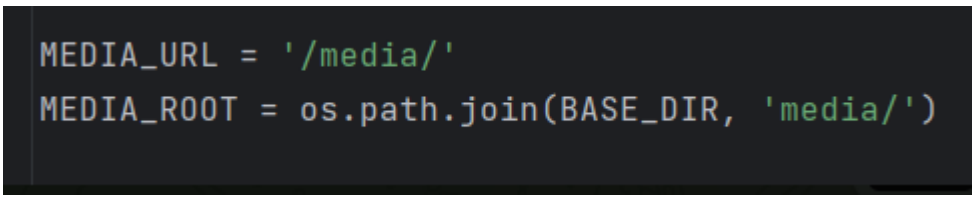


Рисунок 3.26 – Реєстрація media

Після чого, нам потрібно прописати команди «makemigrations» та «migrate» у manage.py і зареєструвати в admin.py. Також треба записати в значенні «context» в views.py.

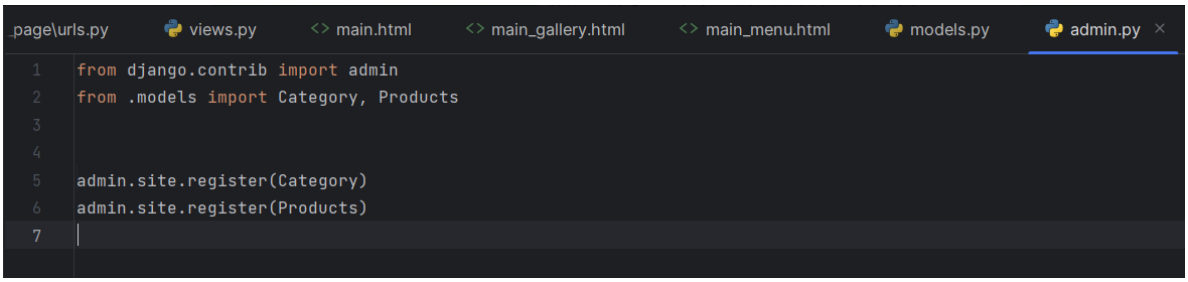


Рисунок 3.26 – admin.py

Тепер запускаємо проект, заходим в панель адміністратора і перевіряємо що все працює правильно.

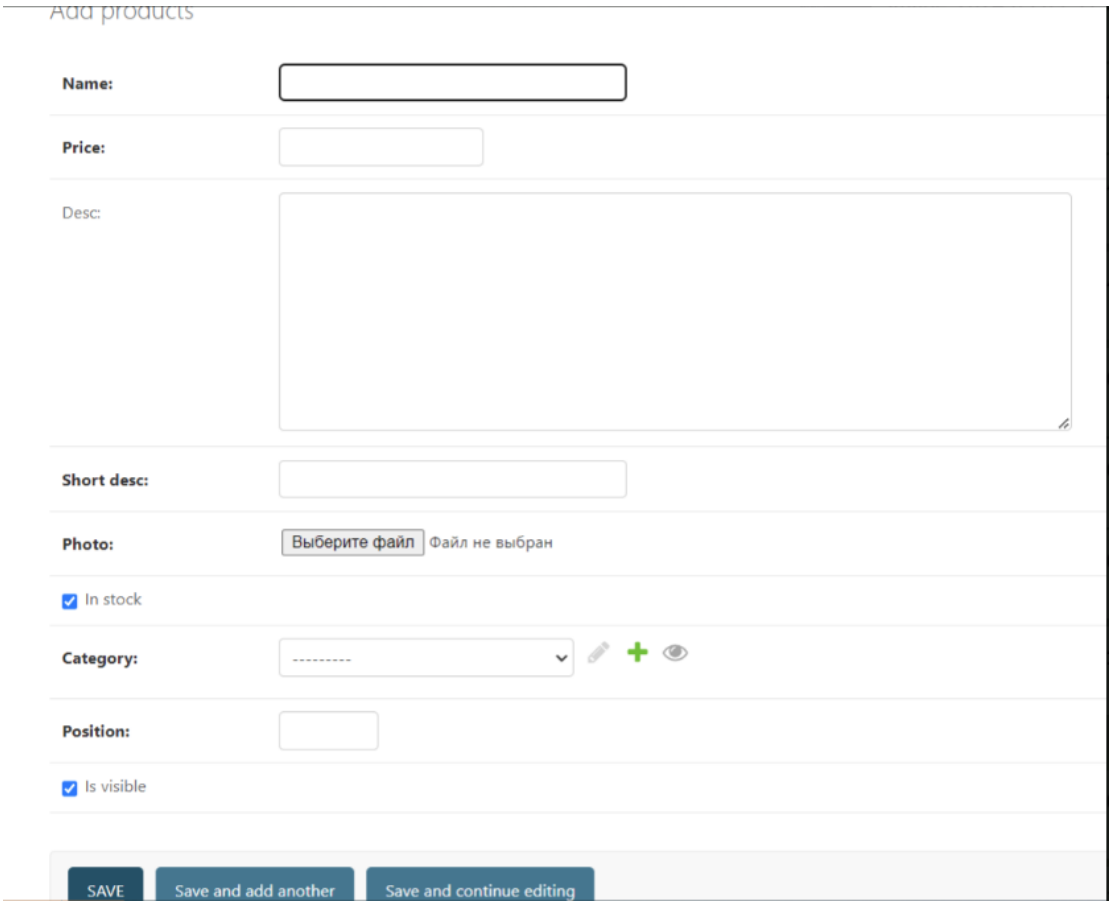


Рисунок 3.28 – Таблиця продуктів в панелі адміністратора

Таблиця заповнюється і дані відображаються успішно. інших секцій. Зображення зберігаються в папці з особою назвою.

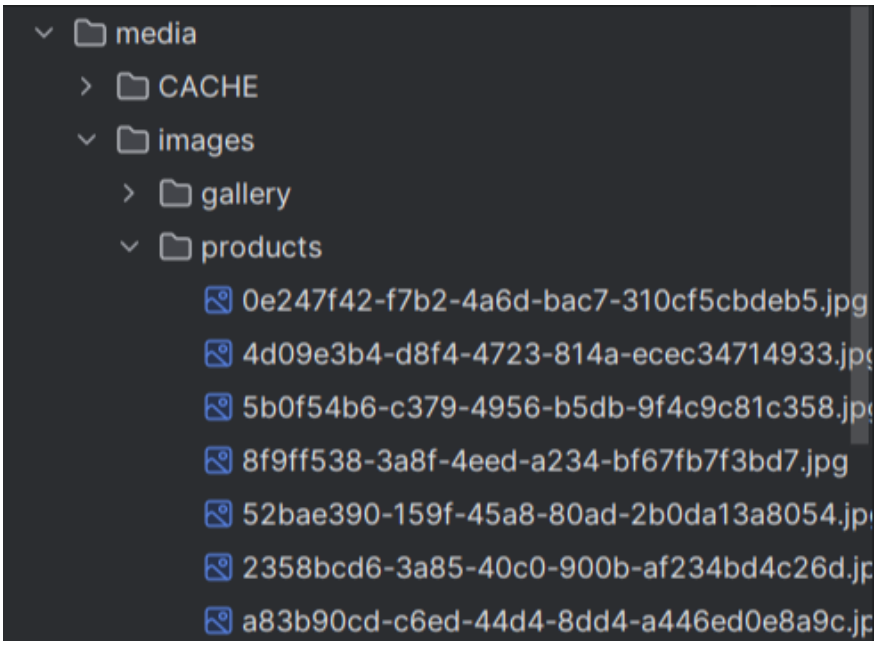


Рисунок 3.28 – Збереження медіа

Створимо таблиці для інших секцій.

Рисунок 3.29 – Інші секції main_page

Надалі, щоб це всі дані виводилися на сайті нам треба додати їх до HTML-документу. Реалізовувати це ми будемо через тег «for» в самому документі, аби уникнути великої коду в документі. Щоб цей тег працював у HTML потрібно записати його в такому вигляді – {% тег %}. Це шаблонний тег. Використовується

він для вставки тегів через пробіл. Ще приклади включають extend, include і load. Вони часто розширюють, вставляють або надають логічну функціональність (наприклад, if-умови або цикли). Цей тег потребую обов'язкового закриття, наприклад {% endfor %}, {% endif %} і т.д.. Визивати дані з моделі ми будемо за допомогою синтаксису {{ код }}. Це синтаксис для шаблонних змінних. Використовується для інтерполяції змінних, оголошених як вбудовані параметри, або ваших власних змінних, створених за допомогою будь-якої кількості методів (модель, контекст перегляду тощо).[29] Додаємо в наш HTML-документ дані моделі «Products».

```
<div class="row">
  <div class="col-lg-3">
    <ul class="nav nav-tabs flex-column">
        <li class="nav-item">
            {% for item in products %}
                {% if item.category.name == 'Classic' %}
                {% if forloop.first %}
                    <a class="nav-link active show" data-bs-toggle="tab" href="#tab-{{ item.id }}">{{ item.name }}</a>
                {% else %}
                    <a class="nav-link" data-bs-toggle="tab" href="#tab-{{ item.id }}">{{ item.name }}</a>
                {% endif %}
                {% endif %}
            {% endfor %}
    </li>
    </ul>
```

Рисунок 3.30 – Список продуктів в категорії

При натисканні на назву продукту буде показуватися інформація про нього: назва, ціна, опис, додаємо кнопку «додати в кошик» для авторизованих користувачів (про створення кошику товарів та реєстрацію користувачів розповім пізніше) та зображення товару.

```
<div class="col-lg-9 mt-4 mt-lg-0">
  <div class="tab-content">
      {% for item in products %}
          {% if item.category.name == 'Classic' %}
          {% if forloop.first %}
            <div class="tab-pane active show" id="tab-{{ item.id }}">
              <div class="row">
                <div class="col-lg-8 details order-2 order-lg-1">
                    <a href="#"><h3>{{ item.name }}</h3></a>
                  <p class="fst-italic">$ {{ item.price }}</p>
                    <p>{{ item.desc }}</p>
                    {% if request.user.is_authenticated %}
                      <a class="book-a-table-btn scrollto" href="{% url 'cart:add_to_cart' item.id %}">Add To Cart</a>
                    {% else %}
                      <a href="login/" class="book-a-table-btn scrollto">Login to buy</a>
                    {% endif %}
                </div>
                <div class="col-lg-4 text-center order-1 order-lg-2">
                  <img src={{ item.photo_thrumbnail.url }} alt="" class="img-fluid">
                </div>
              </div>
            </div>
          {% else %}
```

Рисунок 3.31 – Показ інформації з моделі

Запускаємо проект та перевіряємо чи все відображається коректно.

Рисунок 3.32 – Відображення даних з бази даних

Все коректно відображається. Інформацію до бази даних я ввів швидко, бо коли буде проводиться залиття сайту на домен, у нас данні видаляться, запишуться пусті таблиці. Тому вводити дані потрібно бути заново.

Для всіх інших секцій проводимо таку ж процедуру, починаючи зі створення моделі і закінчуючи додаванням відображення даних в HTML-документі.

3.3 Створення форми зворотнього зв’язку

Приступимо до створення таблиці, яка буде записуватися через форму. Називатися вона буде «ContactForm». Через цю форму ми будемо отримувати відгук, скаргу або питання від користувачів.

Коли користувач заходить до нас на сторінку, то ми маємо йому надіслати пусту форму, бо він її ще не заповнював. Тобто, він має бачити що форма є. Далі, коли користувач вирішить заповнити форму, він її заповнює і нажимає на кнопку «Send Message» і всі дані, що будуть заповнені в полях форми, переходять до нас на сервер в базу даних. Це буде POST запит.

POST – це метод HTTP, призначений для надсилання великих обсягів даних з певного ресурсу на сервер. Більшість HTML форм працює з цим методом запитів. Зазвичай він надсилає відносно невеликі обсяги даних до отримувача. Цей метод дозволяє надсилати дані масово через окреме з'єднання зі скриптом обробки. Це означає, що дані, надіслані за допомогою методу POST, не з'являться в URL-адресі, оскільки параметри не надсилаються разом з URI.[18]

Формат HTTP POST вимагає наявності заголовка HTTP, за яким слідує порожній рядок і тіло запиту. POST-запити можна використовувати для відправки веб-форм або завантаження файлів, але важливо переконатися, що вони відповідають формату, використовуваному програмою-одержувачем.

До цього ми обробляли тільки GET запити. Простими словами, GET-запит – це метод отримання даних з джерела даних за допомогою Інтернету. Метод GET-запиту є дуже поширеним методом HTTP-запитів. Незважаючи на використання великих літер, "GET" не є абревіатурою. Однак, простий спосіб зрозуміти метод GET-це уявити його як "GETing" даних з джерела.[19]

Всі ці дані переходять до нас на сервер і будуть записані в «словник». Далі ми маємо у views.py визначити який тип даних прийшов на сервер. Дані про метод запиту знаходиться у «request» «Requests» - бібліотека python для надсилання HTTP-запитів. І якщо дані, що ввів користувач, правильні (відповідають типу поля) – ми їх будемо записувати в базу даних. Ось які дані повинен ввести користувач у нашу форму:

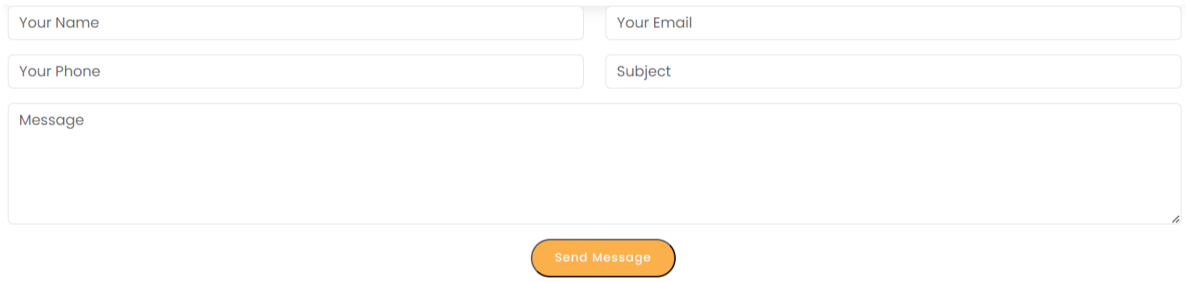


Рисунок 3.33 – Вигляд форми

Створимо модель для цієї форми і дані в такому вигляді будуть надсилатися нам у базу даних. Прописуємо атрибути name, phone, email, subject, message, is_processed і created_at.

Арибут name та is_processed (стандартне значення буде False) заповнюємо як ми робили раніше.

Subject і message будуть мати тип даних CharField і TextField відповідно. Перший атрибут буде мати максимальну кількість символів 150, а другий 500.

Переходимо до самого цікавого. Атрибут phone буде мати тип даних CharField з максимальною кількістю символів 15. Але нам треба зробити так, щоб

це поле користувачі заповнювали правильно, для цього ми вводимо validator. Це об'єкти, що викликаються, які приймають значення і видають повідомлення ValidationError, якщо значення не відповідає певним критеріям. Він корисний для повторного використання логіки перевірки між полями різних типів. За критерій ми створимо значення «mobile_regex». Це буде регулярний вираз, де в нас присутня перевірка: спочатку йде група символів, це будуть 3 цифри, далі може бути пробіл чи « – » і потім повторюється 3 цифри, і потім ще 4 цифри. Додаємо повідомлення до користувача що телефон має формат «xxx xxx xxxx». Для коректної роботи треба імпортувати з бібліотеки validator RegexValidator.[20]

Також нам потрібно розуміти коли прийшла заявка, з цим впорається атрибут created_at, який буде мати тип даних DateTimeField. Це поле дати і часу, яке зберігає дату, представлені як дата і час Python. Як випливає з назви, це поле зберігає об'єкт типу дата-час, створений у Python. Воно буде мати параметр «auto_now_add», що автоматично встановлює поточне значення поля при першому створенні об'єкта. Корисно для створення міток часу. Потрібно звернути увагу, що поточна дата використовується завжди. Це не просто значення за замовчуванням, яке можна змінити. Якщо встановити значення для цього поля під час створення об'єкта, воно буде проігноровано.[21]

Атрибут email буде мати тип даних CharField з максимальною довжиною в 50 символів і буде мати валідатор, який буде перевіряти наявність «@» в полі. Якщо цього символа не буде, то нам видають помилку «Incorrect email».

Формувати сортування ми будемо по полю «created_at» з символом (-), щоб всі нові відповіді на форму зберігали зверху. Прописуємо метод «str» і робимо міграцію за допомогою команд «makemigrations» і «migrate». Додаємо модель в admin.py.

```
9 usages
class ContactForm(models.Model):
    2 usages
    def validate_email(self):
        if "@" in self:
            return self
        else:
            raise ValidationError('Incorrect email')

    mobile_regex = RegexValidator(regex=r'^(\d{3}[- ]?){2}\d{4}$', message='Phone format xxx xxx xxxx')

    name = models.CharField(max_length=50, unique=True, db_index=True)
    phone = models.CharField(max_length=15, validators=[mobile_regex])
    email = models.CharField(max_length=50, validators=[validate_email])
    subject = models.CharField(max_length=150)
    message = models.TextField(max_length=500, blank=True)

    is_processed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

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

    def __str__(self):
        return f'{self.name} | Subject: {self.subject[:30]} | Processed: {self.is_processed} | {self.created_at} |'
```

Рисунок 3.34 – Вигляд моделі форми

Тепер на потрібно змусити форму на сторінці працювати. В Django ми це можемо наступним чином: нам необхідно в наш застосунок «main_page» додати це один спеціальний файл, який буде називатися «forms.py». У нас є спеціальний модуль, який позволяє нам працювати з формами – django.forms. Імпортуємо його разом з моделлю форми. В Django зв’язок між формою і базу даних дуже добре налагоджений. Якщо прийшов об’єкт форми і він зв’язаний з конкретною моделлю, і при коректності даних у формі моментально проводиться запис в базу даних. Вам не приходиться робити запис даних через інші команди, фреймворк робить це за нас. [27]

Створюємо в цьому файлі клас форми і при створенні класу ми маємо від чогось успадковуватися. В models.py ми успадковувалися від models.Model, а тут ми у нас є 2 варіанти – це forms.Form forms.ModelForm. Різниця між ними в тому, що перший варіант ми використовуємо коли нам потрібно звірити дані, котрі він ввів, наприклад, логін на сайт. А другий варіант ми використовуємо якщо нам потрібно виконувати зв’язок форми з таблицею в базі даних. Тут ми маєм успадковуватися від ModelForm. В Meta класі нам потрібно встановити зв’язок з

моделлю, прописуємо що форма зв’язана з моделлю, яка називається «ContactForm». Крім цього ми маємо вказати кортеж, який буде містити строки, назви полів, що приходять від користувача. У нас це name, email, phone, subject та message. Кортежі – це незмінний тип даних у Python, який використовується для зберігання впорядкованих послідовностей елементів.

Далі відкриваємо views.py і на початку функції «main_page» прописуємо перевірку істинності: якщо request має метод «POST» і він надійшов від форми, то ми перевіряємо форму на правильність заповнення і якщо все правильно – зберігаємо її в базі даних. Потім ініціалізуємо нашу форму и додаємо контекст до неї.

Наступним кроком буде створення HTML-документу contactform.html, додаємо шлях до нього на нашій головній сторінці. В цьому файлі прописуємо тег форми з методом «POST», вписуємо {{% csrf_token %}}. Прописувати це потрібно завжди, коли є метод «POST» для уникнення підробки міжсайтових запитів. І додаємо поля до форми з допомогою синтаксису {{ }}.

```
<form id='cont' method="post" role="form">
    {% csrf_token %}
    <div class="row">
        <div class="col-md-6 form-group">
          {{ contactform.name }}
        </div>
        <div class="col-md-6 form-group mt-3 mt-md-0">
          {{ contactform.email }}
        </div>
    </div>
    <p></p>
    <div class="row">
        <div class="col-md-6 form-group">
          {{ contactform.phone }}
        </div>
        <div class="col-md-6 form-group mt-3 mt-md-0">
          {{ contactform.subject }}
        </div>
    </div>
    <div class="form-group mt-3">
        {{ contactform.message }}
    </div>
    <p></p>
    <div class="text-center"><button class="book-a-table-btn scrollto" type="submit">Send Message</button></div>
</form>
```

Рисунок 3.35 – contactus.html

І далі ми будемо оформлювати поля форми у файлі forms.py. У класі, що ми створили, прописуємо поля форми. В нашій формі всі поля будуть мати тип даних CharField і для кожного прописуємо свій ліміт по символам. Після ліміту нам потрібно вказати, шо це віджети Textinput, а поле «message» буде мати віджет Textarea, бо містить воно може містити велику кількість символів. У кожного віджета вводимо параметр «атрибути» і додаємо туди властивості, що будуть у поля форми. Наприклад, для поля «name» це буде: 'name': "name", 'class': "form-control", 'id': "name", 'placeholder': "Your Name", 'required': "required". Таким чином ми додали аргументи полю.

```
class FContactForm(forms.ModelForm):
    name = forms.CharField(max_length=50,
                           widget=forms.TextInput(attrs={
                               'type': "text",
                               'name': "name",
                               'class': "form-control",
                               'id': "name",
                               'placeholder': "Your Name",
                               'required': "required"
                           }))
    email = forms.CharField(max_length=50,
                            widget=forms.TextInput(attrs={
                               'type': "email",
                               'class': "form-control",
                               'name': "email",
                               'id': "email",
                               'placeholder': "Your Email",
                               'required': "required"
                            }))
    phone = forms.CharField(max_length=15,
                            widget=forms.TextInput(attrs={
                               'type': "text",
                               'class': "form-control",
                               'name': "phone",
                               'id': "phone",
                               'placeholder': "Your Phone",
                               'required': "required"
                            }))
    subject = forms.CharField(max_length=150,
                              widget=forms.TextInput(attrs={
```

Рисунок 3.36 – forms.py

Запускаємо проект та перевіряємо працездатність форми:

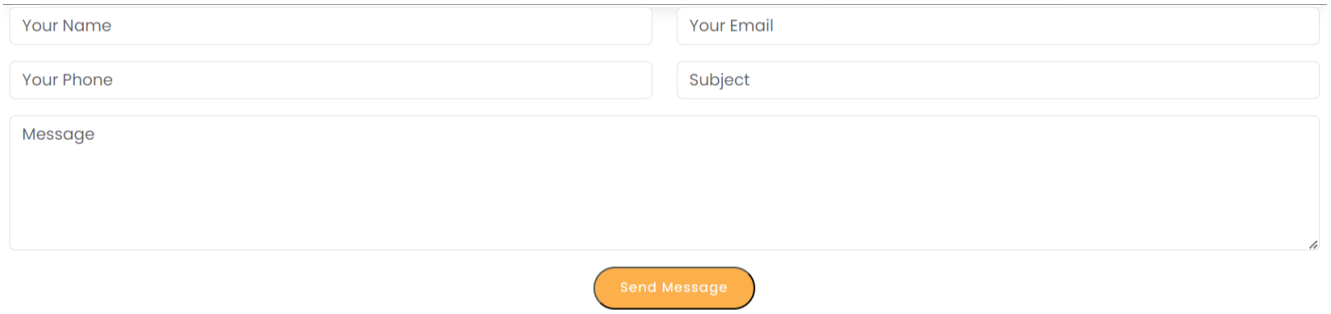


Рисунок 3.37 – Форма на сайті

Стилі і поля для форми відображаються нормально, заповнемо форму і подивимось чи зберігається інформація в базі даних.

Рисунок 3.38 – Форма користувача в базі даних

Всі дані успішно зберіглись в базі даних. Ми все зробили правильно.

Буде не правильно робити так, щоб менеджер роздивлявся відповіді на форму через панель адміністратора, бо він автоматично має доступ до ролей користувачів і інших полей, що не потрібні йому. По суті, ми не маємо допускати менеджера і користувачів до панелі адміністратора. Нам треба для менеджера створити окрему сторінку, де ми будемо відображали всі заявки, що приходять через форму і коли менеджер опрацює заявку, він буде нажимати на кнопку, яка буде відправляти заявки в «оброблені».

Так як у нас зараз з’являються різні ролі, то сторінку менеджера ми не маємо показувати звичайному користувачу, тому нам буде потрібно скривати частину сайту від них.

На початку нам потрібно реалізувати можливість менеджера кудись заходити. Для цього треба створити новий застосунок і називатись він буде «manager». Реалізується це через команду startapp manager і обов’язково реєструємо застосунок в settings.py.

Рисунок 3.39 – Створення застосунку

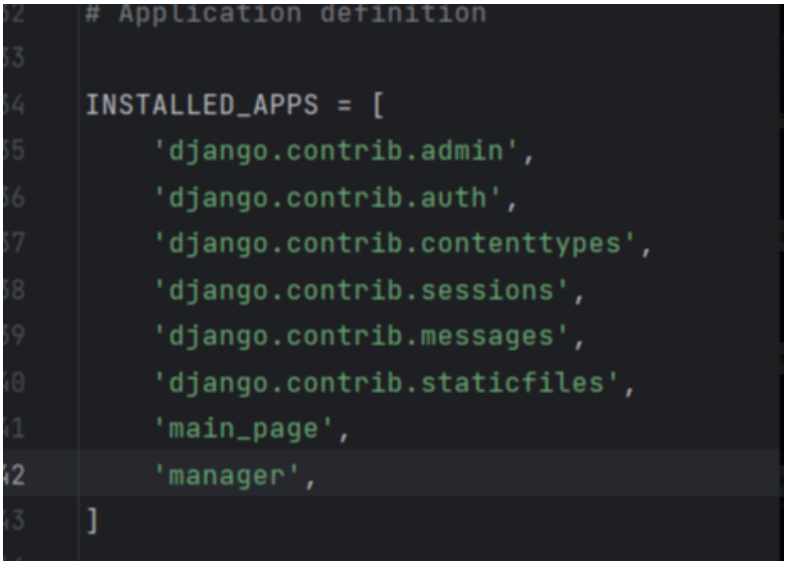


Рисунок 3.40 – Реєстрація застосунку

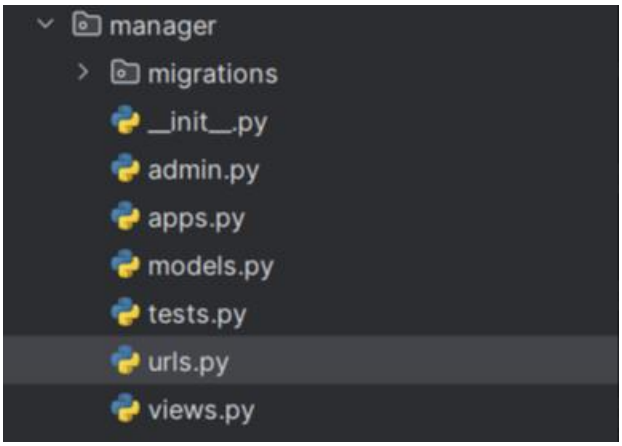


Рисунок 3.41 – Створена директорія manager

І в urls.py основної директорії прописуємо шлях до сторінки менеджера.

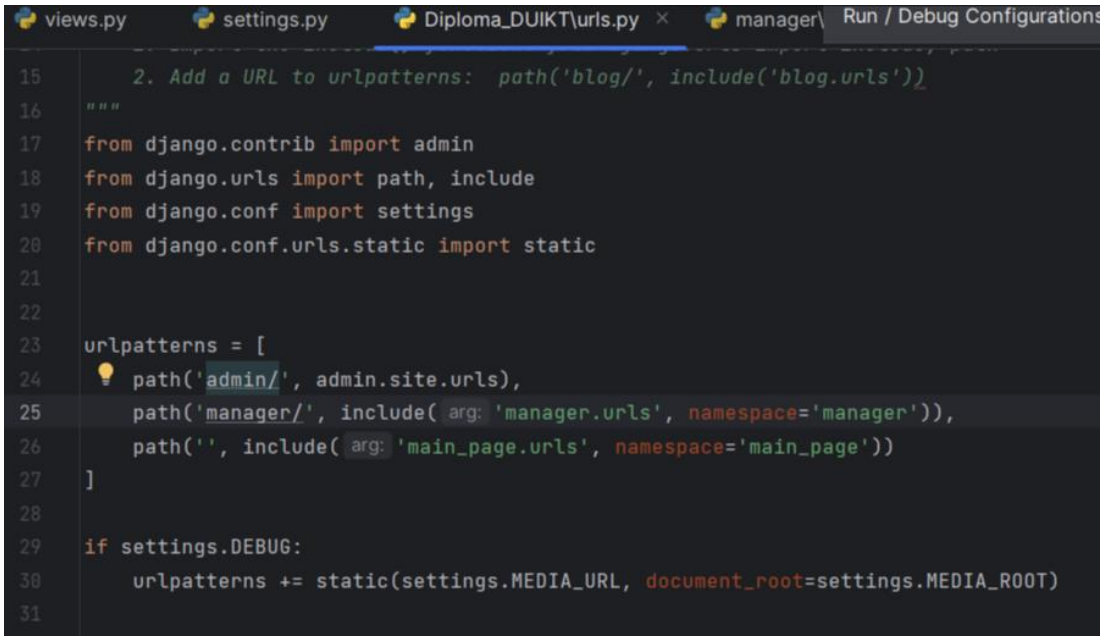


Рисунок 3.42 – Шлях до сторінки менеджера

Тепер нам треба перейти в urls.py директорії менеджера і прописати ім’я застосунку та шляхи до сторінки менеджера. Першу прописуємо назву функції, що буде виводити всі заявки користувачів, які не виконані, коли менеджер заходить на свою сторінку. Після цього переходимо в файл views.py директорії менеджера і там прописуємо імпорт моделі форми з main_page та цю функцію. Вона буде

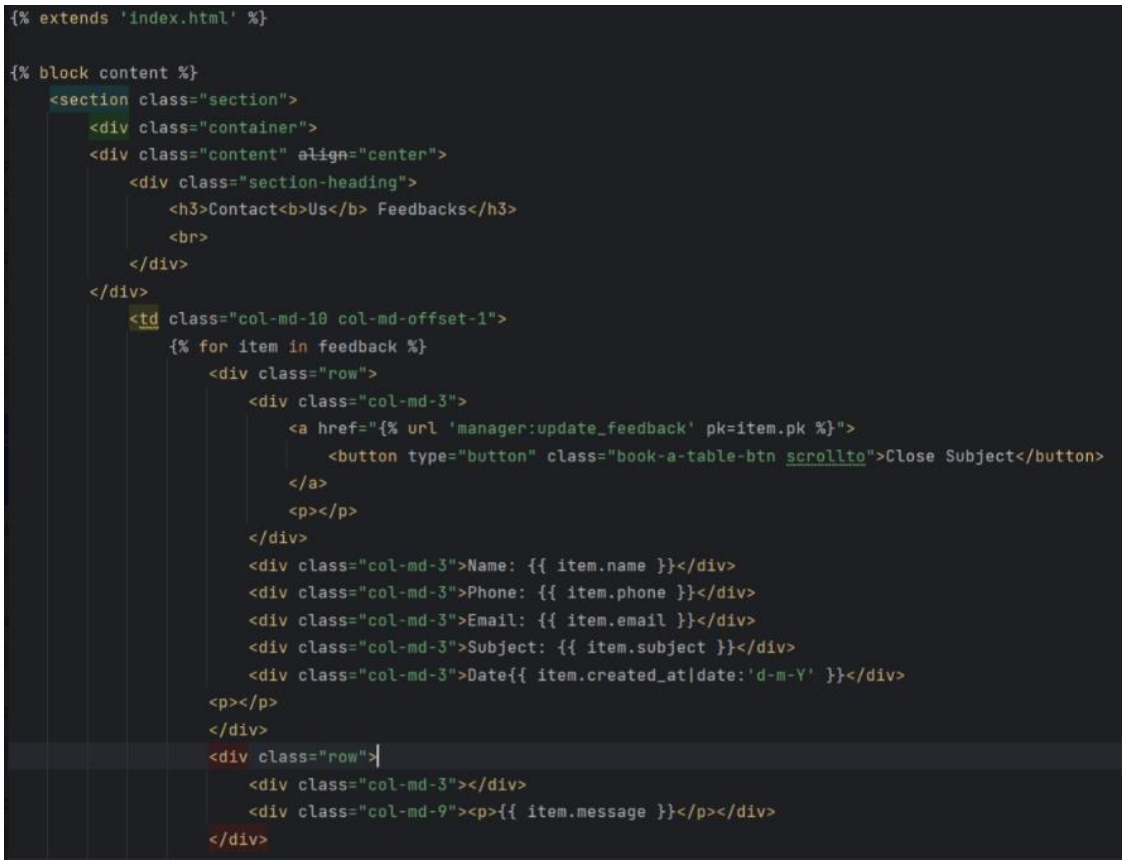
фільтрувати заявки по тому, чи не виконані вони і заносити їх у список. Щоб не засмічувати базу даних, всі заявки які були виконані ми будемо видаляти. Потім готовий список ми рендеримо у HTML-документ, який буде називатися «feedback_list». Також у цьому файлі пропишемо функцію, яка буде перевіряти чи є користувач менеджером і імпорт «user_passes_test» куди в аргумент запишемо цю функцію. Таким чином, звичайний користувач не зможе зайти на сторінку з заявками. Декоратор user_passes_test виконує перенаправлення, коли виклик повертає false. Це виклик, який отримує об'єкт User і повертає True, якщо користувачеві дозволено переглядати сторінку. Переходимо у HTML-документ feedback_list і прописуємо код сторінки.

```
@user_passes_test(is_manager)
def feedback_list(request):
    if request:
        cleaner = ContactForm.objects.filter(is_processed=True)
        cleaner.delete()

    feedback = ContactForm.objects.filter(is_processed=False)
    return render(request, template_name: 'feedback_list.html', context={'feedback': feedback})
```

Рисунок 3.43 – Функція відображення списку заяв

```
def is_manager(user):
    return user.groups.filter(name='manager').exists()
```

Рисунок 3.44 – Перевірка користувача



```
{% extends 'index.html' %}

{% block content %}
    <section class="section">
        <div class="container">
        <div class="content" align="center">
            <div class="section-heading">
                <h3>Contact<b>Us</b> Feedbacks</h3>
                <br>
            </div>
        </div>
            <td class="col-md-10 col-md-offset-1">
                {% for item in feedback %}
                    <div class="row">
                        <div class="col-md-3">
                            <a href="{% url 'manager:update_feedback' pk=item.pk %}">
                                <button type="button" class="book-a-table-btn scrollto">Close Subject</button>
                            </a>
                            <p></p>
                        </div>
                        <div class="col-md-3">Name: {{ item.name }}</div>
                        <div class="col-md-3">Phone: {{ item.phone }}</div>
                        <div class="col-md-3">Email: {{ item.email }}</div>
                        <div class="col-md-3">Subject: {{ item.subject }}</div>
                        <div class="col-md-3">Date{{ item.created_at|date:'d-m-Y' }}</div>
                    <p></p>
                    </div>
                    <div class="row">
                        <div class="col-md-3"></div>
                        <div class="col-md-9"><p>{{ item.message }}</p></div>
                    </div>
```

Рисунок 3.45 – HTML-документ feedback_list

В документі ми попередньо прописали кнопку, яка закриває форму з ініціалізацією по pk (primary key). Первинний ключ – це спеціальне поле в таблицях SQL, яке однозначно ідентифікує кожен запис у таблиці. Як правило, ці поля використовуються для зберігання унікальних ідентифікаторів об'єктів, перелічених у таблиці, наприклад, це може бути ID клієнта чи товару. Зробимо запит на цю функцію в urls.py і пропишемо її в views.py.

```
5   app_name = 'manager'
6
7   urlpatterns = [
8       path('', feedback_list, name='manager_view'),
9       path('feedback/update/<int:pk>/', update_feedback, name='update_feedback'),
10  ]
11
```

Рисунок 3.46 – Запит на функцію.

В шляху ми вказали, що з ним буде передаватися параметр pk, по якому ми будемо находити заявку, яку треба зробити виконаною.

```
@user_passes_test(is_manager)
def update_feedback(request, pk):
    ContactForm.objects.filter(pk=pk).update(is_processed=True)
    return redirect('manager:manager_view')

```

Рисунок 3.47 – Функція оновлення статусу заявок

Запускаємо проект і перевіряємо чи все працює.

Рисунок 3.48 – Показ заявки зі сторінки менеджера

При натисканні на кнопку форма успішно видаляється. Тепер потрібно реалізувати систему авторизації користувачів

3.4 Авторизація користувачів та створення кошику для товарів

Система авторизації користувачів буде окремим застосунком, який ми створюємо за допомогою команди «startapp *назва*». Ми назвемо її «account». Далі

переходимо в файл ulrs.py основної директорії і створюємо нові шляхи, які будуть називатися «login/», «logout/», «registration/» та автоматично придумаємо назви функцій, які будуть відповідати дії і робимо імпорт цих функцій. Виглядатиме це таким чином:

```
from account.views import login_view, logout_view, registration_view


urlpatterns = [
    path('login/', login_view, name='login'),
    path('logout/', logout_view, name='logout'),
    path('registration/', registration_view, name='registration'),
```

Рисунок 3.49 – Створення шляхів для аккаунтів

Далі створюємо файл forms.py в директорії «account» і в ньому імпортуємо модуль «forms». Щодо автентифікації користувачів, в Django вже це вбудовано, тому ми просто імпортуємо модулі «authenticate» та «get_user_model». Використовують «authenticate()» для перевірки набору облікових даних. Вона приймає облікові дані як аргументи ключових слів, «username» і «password» для випадку за замовчуванням, звіряє їх з базою даних користувачей і повертає об'єкт «User», якщо облікові дані дійсні. Якщо облікові дані не дійсні повертається «None». Прописуємо функцію, яка буде отримувати модель користувача і починаємо писати клас форми логіну. Успадковуватись клас буде від forms.Form, бо ніякі дані ми не додаємо в базу даних, а тільки з нею звіряємо (Роз’яснення про це було в підрозділі 3.3). В нас буде 2 поля «username» та «password» і вони обидва будуть CharField, бо це текстові поля. Додаємо до них аргумент «віджет». У першого поля він буде TextInput, а у другого PasswordInput.[23]

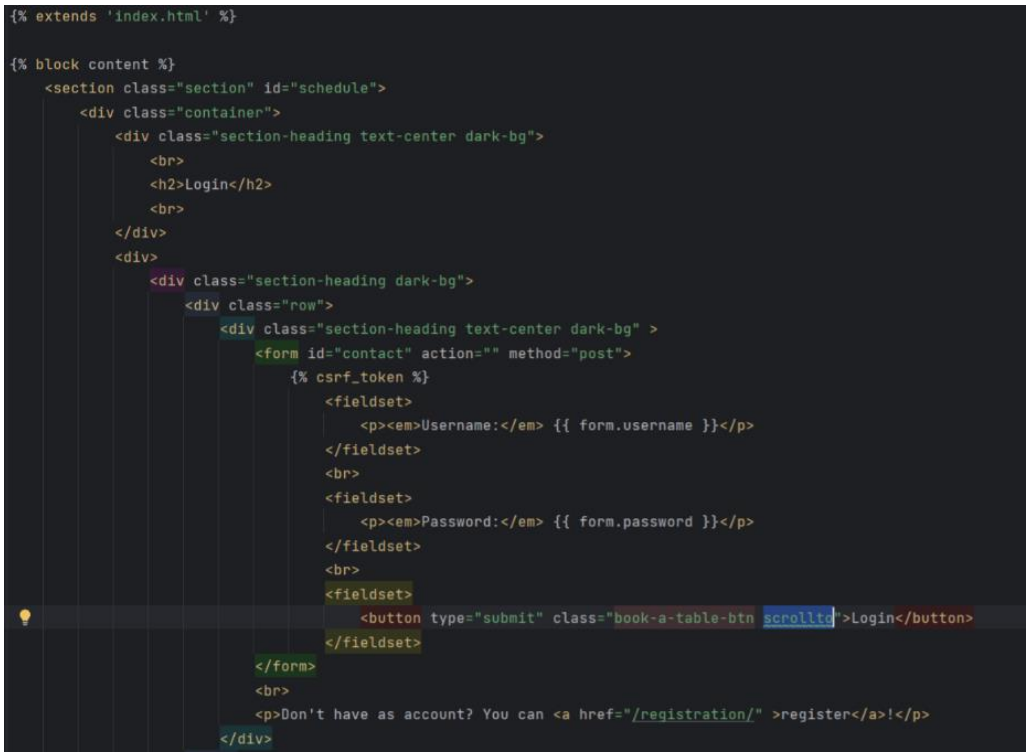
Коли ми говоримо про Django і про форми, то у класа «Forms» є також стандартні методи. І є один метод, який називається «clean». Він відповідає за перевірку даних і якщо вони коректні – ці дані поміщаються в словник який

називається «cleaned_data» і потім з цього словника формується запит до бази даних. Тому зараз нам необхідно попрацювати з цим методом, бо в ньому ми будемо отримувати «username» та «password», які ввів користувач. В методі «clean» ми отримуємо ім’я користувача зі словника «cleaned_data» з методом «get», так само робимо з паролем. І тепер наша задача перевірити чи ввів взагалі ім’я і пароль. Робимо це за допомогою «if» і якщо вони є, мені потрібно провести автентифікацію користувача. Автентифікація проводиться за допомогою модуля «authenticate» в який ми вписуємо ім’я користувача і пароль, що взяли з словника «cleaned_data». Далі ми перевіряємо чи є такий користувач і чи співпадають паролі. Якщо одна з цих умов неправдиві, то ми видаємо помилку з повідомленням, що некоректно був введений пароль чи логін. Але якщо ми пропишемо помилку як завжди, то вона відобразиться тільки в консолі сервера, тому ми пишемо цю помилку через forms.ValidationError і тоді користувач побачить помилку у своєму браузері.

```python
from django import forms
from django.contrib.auth import authenticate, get_user_model


User = get_user_model()


2 usages
class LoginForm(forms.Form):
    username = forms.CharField(widget=forms.TextInput())
    password = forms.CharField(widget=forms.PasswordInput())

    def clean(self, *args, **kwargs):
        username = self.cleaned_data.get('username')
        password = self.cleaned_data.get('password')

        if username and password:
            user = authenticate(username=username, password=password)
            if not user or not user.check_password(password):
                raise forms.ValidationError('Incorrect login or password')
        return super().clean()
```

Рисунок 3.50 – Форма логіну

Створимо сторінку «login.html» та пропишемо в ній поля для нашої форми.



```
{% extends 'index.html' %}

{% block content %}
    <section class="section" id="schedule">
        <div class="container">
            <div class="section-heading text-center dark-bg">
                <br>
                <h2>Login</h2>
                <br>
            </div>
            <div>
                <div class="section-heading dark-bg">
                    <div class="row">
                        <div class="section-heading text-center dark-bg" >
                            <form id="contact" action="" method="post">
                                {% csrf_token %}
                                    <fieldset>
                                        <p><em>Username:</em> {{ form.username }}</p>
                                    </fieldset>
                                    <br>
                                    <fieldset>
                                        <p><em>Password:</em> {{ form.password }}</p>
                                    </fieldset>
                                    <br>
                                    <fieldset>
                                        <button type="submit" class="book-a-table-btn scrollto">Login</button>
                                    </fieldset>
                            </form>
                            <br>
                            <p>Don't have as account? You can <a href="/registration/" >register</a>!</p>
                        </div>
```

Рисунок 3.51 – login.html

Далі реалізуємо функцію виходу з аккаунту. Це дуже просто, через виклик стандартної функції «logout», куди ми передаємо request і після цього перенаправляємо користувача на головну сторінку нашого сайту.

Переходимо до створення форми реєстрації нових користувачів. Повертаємось до файлу forms.py і починаємо створювати новий клас для реєстрації. Успадковуватись він буде від «forms.ModelForm», бо дані ми будемо вносити в базу даних. В цьому класі прописуємо клас Meta, де ми говоримо що будемо успадковуватися від моделі User і додаємо 3 атрибути: «username», «password» та «password2». Вони всі будуть мати тип даних «CharField». Перший атрибут буде мати віджет «TextInput», а останні два – «PasswordInput». Ми вводимо два рази пароль для того, щоб користувач був впевнений, що пароль він ввів коректний.

Ми вже працювали з методом «clean», який дозволяє нам перед тим, як щось робити з базою даних, перевірити що прийшло нас нам вхід. У випадку коли користувач ввів свої данні в поля, коли слоїть задача працювати тільки з одним

полем, наприклад, нас цікавить щоб «password2» дорівнював «password». Ми прописуємо функцію «clean_password2» і тут ми одержуємо дані з «cleaned_data». Перевіряємо чи ці поля співпадають і при позитивному результаті ми повинні з цього методу повернути значення, яке буде записуватись в базу даних. В іншому випадку користувачу висвітиться помилка з повідомленням, що паролі не співпадають.

```
class RegistrationForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('username', )

    username = forms.CharField(widget=forms.TextInput())
    password = forms.CharField(widget=forms.PasswordInput())
    password2 = forms.CharField(widget=forms.PasswordInput())

    def clean_password2(self, *args, **kwargs):
        data = self.cleaned_data
        if data['password'] == data['password2']:
            return data['password2']
        return forms.ValidationError('Password doesn`t match')
```

Рисунок 3.52 – Форма реєстрації

Створимо сторінку «registration.html» та пропишемо тут поля для нашої форми.

```
        <h2>Registration</h2>
    </div>
    <div>
        <div class="section-heading dark-bg">
            <div class="row">
                <div class="section-heading text-center dark-bg" >
                    <form id="contact" action="" method="post">
                        {% csrf_token %}
                            <fieldset>
                                <p><em>Username:</em>  {{ form.username }}</p>
                            </fieldset>
                            <br>
                            <fieldset>
                                <p><em>Password:</em>  {{ form.password }}</p>
                            </fieldset>
                            <br>
                            <fieldset>
                                <p><em>Confirm Pass:</em> {{ form.password2 }}</p>
                            </fieldset>
                            <br>
                            <fieldset>
                                <button type="submit" id="form-submit" class="book-a-table-btn scrollto">Register</button>
                            </fieldset>
                    </form>
                    <br>
                    <p>Have as account? You can <a href="/login/" >login</a>!</p>
```

Рисунок 3.53 – registration.html

Тепер переходимо в файл views.py директорії «account» і починаємо прописувати ці функції.

Спочатку пропишемо функцію логіну. Імпортуємо нашу форму. В функції ініціалізуємо форму та перевіримо її на правильність і після цього виводимо рендер сторінки «login.html». Тепер запускаємо проект і спробуємо залогінитись на сайт.

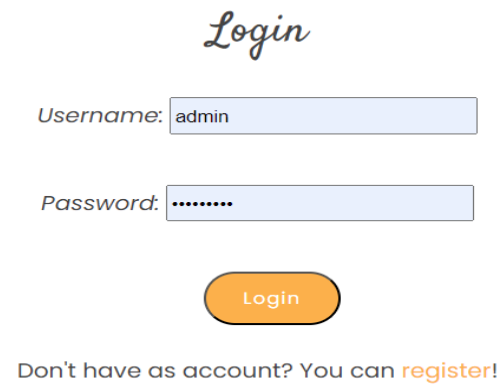


Рисунок 3.52 – Сторінку входу в аккаунт

Наступним кроком буде створення форми реєстрації нових користувачів. Імпортуємо форму реєстрації і ініціалізуємо її в функції. Перевіряємо форму на правильність, якщо нас все влаштовує – нам необхідно виконати декілька дій. Ми маємо начебто і зберегти інформацію в базу даних, але при цьому ми маємо виконати певні дії з цими даними, тому в методі «save», що зберігає форму, ми ставимо аргумент «commit=False» щоб дані поки не потрапили до бази даних. І тепер нам необхідно встановити для користувача пароль, що пройшов перевірку на схожість. Тільки після цього ми зберігаємо дані в базу даних і перенаправляємо користувача на сторінку, де буде написано що реєстрація пройшла успішно. Якщо форма не правильна – користувач отримує помилку і перенаправляється на сторінку реєстрації.[29] Наш файл тепер має такий вигляд:

```
    def login_view(request):
        form = LoginForm(request.POST or None)
        next_get = request.GET.get('next')

        if form.is_valid():
            username = request.POST.get('username')
            password = request.POST.get('password')

            user = authenticate(username=username.strip(), password=password.strip())
            login(request, user)
            next_post = request.POST.get('next')
            return redirect(next_get or next_post or '/')

        return render(request, template_name: 'login.html', context={'form': form})


    2 usages
    def logout_view(request):
        logout(request)
        return redirect('/')


    2 usages
    def registration_view(request):
        form = RegistrationForm(request.POST or None)

        if form.is_valid():
            new_user = form.save(commit=False)
            new_user.set_password(form.cleaned_data['password'])
            new_user.save()
            return render(request, template_name: 'registration_done.html', context={'user': new_user})
        return render(request, template_name: 'registration.html', context={'form': form})
```

Рисунок 3.53 – account.views.py

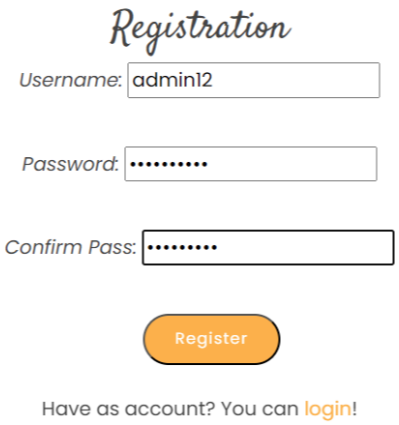


Рисунок 3.54 – Сторінка реєстрації

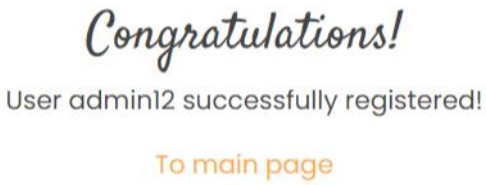


Рисунок 3.55 – Реєстрація успішна

Після створення реєстрації користувачів наступним кроком є створення кошика, в якому користувачі можуть вибрати товари, які вони хочуть придбати. Кошик дозволяє користувачам обирати продукти, які вони хочуть, і тимчасово зберігати їх під час перегляду сайту, поки не буде зроблено замовлення. Додавання кошика на веб-сторінку є дуже важливим кроком при створенні веб-сайту електронної комерції або будь-якого іншого додатку, який передбачає здійснення онлайн-транзакцій. Кошики дозволяють користувачам збирати та керувати продуктами, які вони хочуть купити, перш ніж перейти до оформлення замовлення.

Спочатку створимо застосунок з ім’ям «cart» і зареєструймо його як ми це робили раніше. У файлі main_page.models.py ми створюємо клас кошику до якого будуть такі атрибути: «product», «quantity», «user» та «date_added».

Атрибут «product» буде мати тип даних «ForeignKey», який буде успадковуватися від класу «Products». Тобто, цей атрибут буде приймати всі дані продукту, який обере користувач.

«Quantity» буде мати тип даних PositiveSmallIntegerField, в якому ми виставляємо стандартне значення 0, а максимальне число 10.

У атрибута «user» теж буде тип даних «ForeignKey» і успадковуватися від моделі всіх користувачів.

І останній атрибут «date_added» буде мати тип «DateTimeField». Він буде записувати дату і час коли було створене замовлення.

Виконуємо сортування по даті і робимо створення таблиці (міграції). Після цього не забуваємо зареєструвати модель в admin.py.

```
class CartItem(models.Model):
    product = models.ForeignKey(Products, on_delete=models.CASCADE)
    quantity = models.PositiveSmallIntegerField(default=0, validators=[MaxValueValidator(10), MinValueValidator(0)])
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self. product.name} | {self.quantity}'

    class Meta:
        ordering = ('date_added', )
```

Рисунок 3.56 – Модель кошику

Тепер відкриваємо файл views.py директорії «cart» та створимо такі функції:

- view_cart(request): ця функція відображає вміст кошика. Він отримує елементи кошика, пов'язані з поточним користувачем, обчислює загальну вартість елементів у кошику і створює шаблон, який відображає елементи кошика разом із загальною вартістю.

- add_to_cart(request, product_id): ця функція запускається, коли користувач натискає кнопку "Додати в кошик" для товару. Вибраний товар додається до кошика користувача. Якщо товар вже є в кошику, його кількість збільшується. Користувач буде перенаправлений на сторінку кошика, де буде показано оновлений вміст кошика.

- remove_from_cart(request, item_id): ця функція видаляє товар з кошика. Вона приймає ідентифікатор товару як параметр, знаходить відповідний товар у кошику і видаляє його з бази даних. Після видалення користувач буде перенаправлений на подання кошика, де буде показано оновлений кошик.

```python
from django.shortcuts import render, redirect
from main_page.models import Products, CartItem


1 usage
def view_cart(request):
    cart_items = CartItem.objects.filter(user=request.user)
    total_price = sum(item.product.price * item.quantity for item in cart_items)
    return render(request, template_name: 'cart.html', context: {'cart_items': cart_items,
                                        'total_price': total_price})


1 usage
def add_to_cart(request, product_id):
    product = Products.objects.get(id=product_id)
    cart_item, created = CartItem.objects.get_or_create(product=product,
                                                        user=request.user)
    cart_item.quantity += 1
    cart_item.save()
    return redirect('cart:view_cart')


1 usage
def remove_from_cart(request, item_id):
    cart_item = CartItem.objects.get(id=item_id)
    cart_item.delete()
    return redirect('cart:view_cart')
```

Рисунок 3.57 – Файл views.py директорії «cart»

Наступним кроком ми додаємо шляхи в файл urls.py. Файл буде мати наступний вигляд:

```python
import ...


app_name = 'cart'

urlpatterns = [
    path('', views.view_cart, name='view_cart'),
    path('add/<int:product_id>/', views.add_to_cart, name='add_to_cart'),
    path('remove/<int:item_id>/', views.remove_from_cart, name='remove_from_cart'),
```

Рисунок 3.58 – Шляхи до кошику

Тепер створимо HTML-документ cart.html. В ньому пропишемо вигляд сайту і підставимо відображення нашого кошику.

```
<div class="content" align="center">
    <table class="cart">
        <thead>
        <tr>
            <th>Image</th>
            <th>Product</th>
            <th>Quantity</th>
            <th>Remove</th>
            <th>Unit price</th>
            <th>Price</th>
        </tr>
        </thead>
        <tbody>
        {% for item in cart_items %}
                <tr>
                    <td>
                        <a href="{{ product.get_absolute_url }}">
                            <img src="
                                {% if item.product.photo %}
                                    {{ item.product.photo_thrumbnail.url }}
                                {% else %}
                                    <p>No image</p>
                                {% endif %}" alt="" width="120px" height="120px">
                        </a>
                    </td>
                    <td>{{ item.product.name }}</td>
                    <td>{{ item.quantity }}</td>
                    <td class="num">${{ item.product.price }}</td>
                    <td><a href="{% url 'cart:remove_from_cart' item.id %}">Remove</a></td>
                </tr>
            {% empty %}
```

Рисунок 3.59 – cart.html

```
                {% empty %}
                    <div class="container">
                        <p>Cart is empty!</p>
                    </div>
                {% endfor %}
                <tr class="total">
                    <td>Total</td>
                    <td colspan="4"></td>
                    <td class="num">${{ total_price }}</td>
                </tr>
                </tbody>
            </table>
                <div class="content" align="right   ">
                    <br>
                    <a href="{% url 'main_page:main_page_view' %}" class="book-a-table-btn scrollto">Main page</a>
                    <br>
                    <br>
                    <a href="{% url 'cart:delete_cart' %}" class="book-a-table-btn scrollto">Checkout</a>
                </div>
            </div>
        </div>
    </section>
{% endblock %}
```

Рисунок 3.60 – cart.html (2 частина)

Залишилося запустити проект і перевірити робото спроможність нашого кошику.

Рисунок 3.61 – Кошик товарів

Все працює добре. Товари додаються і видаляються. Наш сайт повністю готовий, тільки залишилося вигрузити його на домен.

ВИСНОВКИ

У цій роботі було розглянено і проведено процес створення веб-сайту за допомогою HTML, CSS, JavaScript, Python та фреймворку Django. Основна увага була приділена аналізу основних етапів створення веб-сайту.

Об'єктом дослідження став процес створення веб-сайту з використанням різних мов програмування, в основному Python та фреймворку Django, включаючи ключові етапи розробки та характеристики інструментів, що використовуються в цьому процесі.

Предметом дослідження стали прийоми та методи, що використовуються для створення веб-сайтів та їх застосування для досягнення конкретних цілей у сфері веб-розробки.

Для досягнення поставлених цілей було використано метод аналізу літератури з вивченням документації Python та Django, а також експериментальний метод, що полягає у створенні прототипів веб-додатків та аналізі їхньої продуктивності та функціональності.

Результатом дослідження стало поглиблення знань про процес розробки веб-сайтів та важливість використання різних інструментів і технік для досягнення успішних результатів. Також було визначено найкращі підходи та методи для розробки веб-додатків з використанням цих технологій.

Всі завдання та цілі були успішно досягнуті. Дослідження стало важливим кроком у розумінні процесу створення веб-сайту та використання різних методів для досягнення цілей веб-розробки.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Учасники проектів Вікімедіа. HTML – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/HTML (дата звернення: 01.05.2024).

2. Учасники проектів Вікімедіа. CSS – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/CSS (дата звернення: 02.05.2024).

3. Учасники проектів Вікімедіа. Python – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Python (дата звернення: 02.05.2024).

4. Учасники проектів Вікімедіа. Django – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Django (дата звернення: 02.05.2024).

5. Топ 7 фреймворків Python для розробки веб застосунків dev.ua. URL: https://dev.ua/ru/news/top-7-freimvorkov-python-dlya-razrabotki-veb-prilozhenii (дата звернення: 04.05.2024).

6. Веб-застосунок на Django з нуля. Tproger. URL: https://tproger.ru/translations/create-your-first-django-app (дата звернення: 04.05.2024).

7. What is Flask Python - Python Tutorial. Learn Python Programming - Python Tutorial. URL: https://pythonbasics.org/what-is-flask-python/ (дата звернення: 06.05.2024).

8. Учасники проектів Вікімедіа. Об'єктно-реляційне відображення – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/ORM (дата звернення: 06.05.2024).

9. Contributors to Wikimedia projects. Web Server Gateway Interface - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface (date of access: 06.05.2024).

10. Contributors to Wikimedia projects. URL – Вікіпедія. Вікипедия – вільна енциклопедія. URL: https://ru.wikipedia.org/wiki/URL (дата звернення: 06.05.2024).

11. Mir M. A. What are the advantages and disadvantages of using Visual Studio Code or Atom?. Medium. URL: https://medium.com/@ssc.ahmed.926748/what-are-

the-advantages-and-disadvantages-of-using-visual-studio-code-or-atom-d3132bf1af85 (дата звернення: 06.05.2024).
12. Django documentation | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.0/ (дата звернення: 07.05.2024)
13. How to manage static files (e.g. images, JavaScript, CSS) | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/4.2/howto/static-files/ (дата звернення: 07.05.2024).
14. Gillis A. S., Lutkevich B., Hughes A. What is a database (DB)? | Definition from TechTarget. Data Management. URL: https://www.techtarget.com/searchdatamanagement/definition/database#:~:text=A%20database%20is%20information%20that,data,%20financials%20and%20product%20information. (дата звернення: 07.05.2024).
15. Alexey Anokhin. Еволюція баз даних. Researchgate.net. URL: https://www.researchgate.net/publication/266375158_Evolucia_baz_dannyh (дата звернення: 07.05.2024).
16. Writing views | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.0/topics/http/views/ (дата звернення: 08.05.2024).
17. Що таке база даних? - Blogchain. Український Блог про Сучасні Інформаційні Технології Світу | IT - BLOG Blogchain. URL: https://blogchain.com.ua/shcho-take-baza-danykh/ (дата звернення: 08.05.2024).
18. HTTP Request Methods | W3Schools Online Web Tutorials. URL: https://www.w3schools.com/tags/ref_httpmethods.asp (дата звернення: 09.05.2024).
19. What Is the Difference Between GET and POST Methods? | Baeldung on Computer Science. Baeldung on Computer Science. URL: https://www.baeldung.com/cs/http-get-vs-post (дата звернення: 09.05.2024).
20. RegexValidator | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.0/ref/validators/#:~:text=A%20RegexValidator%20searches%20the%20provided,when%20a%20match%20is%20found. (дата зверення: 09.05.2024).

21. DateTimeField - Django Models - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/datetimefield-django-models/ (дата звернення: 10.05.2024).
22. Кращі IDE для Python в 2024 році - Блог Mate academy. Блог Mate academy. URL: https://mate.academy/blog/python/ide-for-python-2024/ (дата звернення: 10.05.2024).
23. Using the Django authentication system | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.0/topics/auth/default/ (дата звернення: 11.05.2024).
24. Real Python. How to Write Beautiful Python Code With PEP 8 – Real Python. Python Tutorials – Real Python. URL: https://realpython.com/python-pep8/#:~:text=PEP%208,%20sometimes%20spelled%20PEP8,and%20consistency%20of%20Python%20code. (дата звернення: 11.05.2024).
25. Александрович Д. В. Django 2.1. Практика создания веб-сайтов на Python. Russia : БХВ-Петербург, 2019. 672 с.
26. Вивчаємо Python : навч. Посіб 1 том. Наук. Світ, 2024. 764 с.
27. Вивчаємо Python : навч. Посіб 2 том. Наук. Світ, 2024. 294 с
28. Мова програмування Python для інженерів і науковців: Навчальний посібник. Ivano-Frankivsk, Ukraine : ІФНТУНГ, 2019. 275 с.
29. Beerbohm M., Mohmmed M. Django: Django, Web Framework for Python. Independently Published, 2019.
30. Jordon W. Python Django Web Development: The Ultimate Django Web Framework Guide for Beginners. Independently Published, 2019.

Додаток А

Програмний код проекту

Файл «Dimploma_DUIKT/urls.py»

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from account.views import login_view, logout_view, registration_view


urlpatterns = [
    path('login/', login_view, name='login'),
    path('logout/', logout_view, name='logout'),
    path('registration/', registration_view, name='registration'),
    path('admin/', admin.site.urls),

    path('cart/', include('cart.urls')),

    path('manager/', include('manager.urls', namespace='manager')),

    path('', include('main_page.urls', namespace='main_page'))
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Файл «Dimploma_DUIKT/settings.py»

```
from pathlib import Path
import os


BASE_DIR = Path(__file__).resolve().parent.parent

CART_SESSION_ID = 'cart'

SECRET_KEY = ''

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',
    'main_page',
    'manager',
    'account',
    'imagekit',
    'cart',
]
```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'Diploma_DUIKT.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates']
        ,
        '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 = 'Diploma_DUIKT.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

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',
    },
]

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True
```

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Файл «main_page/views.py»

```
from django.shortcuts import render, redirect
from .models import Category, Products, WhyUs, Reviews, AdBlock, Gallery
import random
from .forms import FContactForm


def main_page(request):
    if request.method == 'POST':
        contactform = FContactForm(request.POST)
        if contactform.is_valid():
            contactform.save()
            return redirect('/')

    contactform = FContactForm()
    categories = Category.objects.filter(is_visible=True).order_by('position')
    products = Products.objects.filter(is_visible=True).order_by('position')
    whyus = WhyUs.objects.filter(is_active=True).order_by('position')
    reviews = Reviews.objects.filter(is_visible=True).order_by('is_visible')
    adblock = AdBlock.objects.filter(is_active=True).order_by('is_active')
    gallery = list(Gallery.objects.filter(is_visible=True).order_by('number'))
    random.shuffle(gallery)

    return render(request, 'main.html', context={
        'categories': categories,
        'products': products,
        'whyus': whyus,
        'reviews': reviews,
        'contactform': contactform,
        'adblock': adblock,
        'gallery': gallery,
    })
```

Файл «main_page/urls.py»

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


app_name = 'main_page'

urlpatterns = [
    path('', main_page, name='main_page_view')
]
```

Файл «main_page/models.py»

```
from django.db import models
from uuid import uuid4
from imagekit.models import ImageSpecField
from imagekit.processors import ResizeToFill
from os import path
from django.core.validators import MaxValueValidator, MinValueValidator,
RegexValidator
from django.core.exceptions import ValidationError
from django.contrib.auth.models import User


class Category(models.Model):
    name = models.CharField(max_length=50, unique=True, db_index=True)
    is_visible = models.BooleanField(default=True)
    position = models.PositiveSmallIntegerField(unique=True)

    def __str__(self):
        return f'{self.name}| Visible:{self.is_visible}| Pos:{self.position}'

    class Meta:
        ordering = ('position', )


class Products(models.Model):
    def get_file_name(self, filename: str):
        ext = filename.strip().split('.')[-1]
        filename = f'{uuid4()}.{ext}'
        return path.join('images/products', filename)

    name = models.CharField(max_length=70, unique=True, db_index=True)
    price = models.DecimalField(max_digits=8, decimal_places=2)
    desc = models.TextField(max_length=600, blank=True)
    short_desc = models.CharField(max_length=200)
    photo = models.ImageField(upload_to=get_file_name, default='null')
    photo_thrumbnail = ImageSpecField(source='photo',
processors=[ResizeToFill(800, 822)],
                                      format='JPEG', options={'quality': 80})
    in_stock = models.BooleanField(default=True)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    position = models.PositiveSmallIntegerField()
    is_visible = models.BooleanField(default=True)

    def __str__(self):
        return f'{self.name}: {self.price} | In Stock: {self.in_stock} | Visible:
{self.is_visible}'


class WhyUs(models.Model):
    position = models.PositiveSmallIntegerField()
    title = models.CharField(max_length=50, unique=True, db_index=True)
    desc = models.TextField(max_length=300, blank=False)
    is_active = models.BooleanField(default=True)

    def __str__(self):
        return f'Title: {self.title} | Pos: {self.position}| Active:
{self.is_active}'


class Reviews(models.Model):
    name = models.CharField(max_length=50, unique=True, db_index=True)
    job = models.CharField(max_length=70)
```

```
    stars = models.PositiveSmallIntegerField(default=1,
validators=[MaxValueValidator(6), MinValueValidator(1)])
    speech = models.TextField(max_length=200)
    is_visible = models.BooleanField(default=True)

    def __str__(self):
        return f'{self.name}: rate: {self.stars} | Visible: {self.is_visible}'


class ContactForm(models.Model):
    def validate_email(self):
        if "@" in self:
            return self
        else:
            raise ValidationError('Incorrect email')

    mobile_regex = RegexValidator(regex=r'^(\d{3}[- ]?){2}\d{4}$', message='Phone
format xxx xxx xxxx')

    name = models.CharField(max_length=50, unique=True, db_index=True)
    phone = models.CharField(max_length=15, validators=[mobile_regex])
    email = models.CharField(max_length=50, validators=[validate_email])
    subject = models.CharField(max_length=150)
    message = models.TextField(max_length=500, blank=True)

    is_processed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

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

    def __str__(self):
        return f'{self.name} | Subject: {self.subject[:30]} | Processed:
{self.is_processed} | {self.created_at} |'


class AdBlock(models.Model):
    def get_file_name_ad(self, filename: str):
        ext = filename.strip().split('.')[-1]
        filename = f'{uuid4()}.{ext}'
        return path.join('images/ads', filename)

    ad_title = models.CharField(max_length=60, unique=True, db_index=True)
    ad_desc = models.TextField(max_length=400)
    photo = models.ImageField(upload_to=get_file_name_ad)
    button1 = models.CharField(max_length=25)
    button1_url = models.CharField(max_length=150)
    button2 = models.CharField(max_length=25)
    button2_url = models.CharField(max_length=150)
    is_active = models.BooleanField(default=True)

    def __str__(self):
        return f'{self.ad_title}: {self.ad_desc[:20]} | Active: {self.is_active}'


class Gallery(models.Model):
    def get_file_name_gallery(self, filename: str):
        ext = filename.strip().split('.')[-1]
        filename = f'{uuid4()}.{ext}'
        return path.join('images/gallery', filename)

    number = models.PositiveIntegerField(unique=True)
    photo_gal = models.ImageField(upload_to=get_file_name_gallery)
```

```
    photo_thumbnail = ImageSpecField(source='photo_gal',
processors=[ResizeToFill(800, 600)],
                                     format='JPEG', options={'quality': 70})
    is_visible = models.BooleanField(default=True)

    def __str__(self):
        return f'{self.number} | Visible: {self.is_visible}'


class CartItem(models.Model):
    product = models.ForeignKey(Products, on_delete=models.CASCADE)
    quantity = models.PositiveSmallIntegerField(default=0,
validators=[MaxValueValidator(10), MinValueValidator(0)])
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self. product.name} | {self.quantity}'

    class Meta:
        ordering = ('date_added', )
```

Файл «main_page/forms.py»

```
from django import forms
from .models import ContactForm


class FContactForm(forms.ModelForm):
    name = forms.CharField(max_length=50,
                           widget=forms.TextInput(attrs={
                               'type': "text",
                               'name': "name",
                               'class': "form-control",
                               'id': "name",
                               'placeholder': "Your Name",
                               'required': "required"
                           }))
    email = forms.CharField(max_length=50,
                            widget=forms.TextInput(attrs={
                               'type': "email",
                               'class': "form-control",
                               'name': "email",
                               'id': "email",
                               'placeholder': "Your Email",
                               'required': "required"
                            }))
    phone = forms.CharField(max_length=15,
                            widget=forms.TextInput(attrs={
                               'type': "text",
                               'class': "form-control",
                               'name': "phone",
                               'id': "phone",
                               'placeholder': "Your Phone",
                               'required': "required"
                            }))
    subject = forms.CharField(max_length=150,
                              widget=forms.TextInput(attrs={
                               'type': "text",
                               'class': "form-control",
                               'name': "subject",
```

```
                                'id': "subject",
                                'placeholder': "Subject",
                                'required': "required"
                            }))
    message = forms.CharField(max_length=500,
                            widget=forms.Textarea(attrs={
                                'class': "form-control",
                                'name': "message",
                                'rows': "5",
                                'placeholder': "Message",
                                'required': "required"
                            }))

    class Meta:
        model = ContactForm
        fields = ('name', 'email', 'phone', 'subject', 'message')
```

Файл «main_page/admin.py»

```
from django.contrib import admin
from .models import (Category, Products, WhyUs,
                     Reviews, ContactForm, AdBlock,
                     Gallery, CartItem)


admin.site.register(Category)
admin.site.register(Products)
admin.site.register(WhyUs)
admin.site.register(Reviews)
admin.site.register(ContactForm)
admin.site.register(AdBlock)
admin.site.register(Gallery)
admin.site.register(CartItem)
```

Файл «main_page/templates/main.html»

```
{% extends 'index.html' %}

{% block content %}
<main id="main">

    <!-- ======= About Section ======= -->
    <section id="about" class="about">
      <div class="container-fluid">

        <div class="row">

          <div class="col-lg-5 align-items-stretch video-box" style='background-
image: url("static/img/aboutus.jpg");'>
            <a
href="https://www.youtube.com/watch?v=HZmeiEnLHYE&ab_channel=WilliamsSonoma"
class="venobox play-btn mb-4" data-vbtype="video" data-autoplay="true"></a>
          </div>

          <div class="col-lg-7 d-flex flex-column justify-content-center align-
items-stretch">

            <div class="content">
                <h3><strong>The De' Longhi Group</strong> is one of the world's
leading players in small domestic appliances associated with the world of coffee,
```

```
the kitchen, air conditioning, and home care.</h3>
              <p>
                The Group's products are sold in more than 120 markets around the
globe. Every year, its community of over 10.000 employees contributes to launching
products that are increasingly innovative and tailored to consumers' needs. In
2021 the Group generated €3221,6 million in revenue.
              </p>
              <p class="fst-italic">
                Our business model
              </p>
              <ul>
                <li><i class="bx bx-check-double"></i> We design products and
experiences.</li>
                <li><i class="bx bx-check-double"></i> From the best raw materials
to the best products</li>
                <li><i class="bx bx-check-double"></i> Test after test, quality
and safety.</li>
              </ul>
              <p>
                 We do only the best for our consumers. You won't go wrong if
<span>you choose us!</span>
              </p>
            </div>

          </div>

        </div>

      </div>
    </section><!-- End About Section -->

    <!-- ======= Whu Us Section ======= -->
    {% include 'whyus.html' %}

    <!-- ======= Classic Section ======= -->
    {% include 'product.html' %}

    <!-- ======= Modern Section ======= -->

    <!-- ======= Gallery Section ======= -->
    {% include 'main_gallery.html' %}

    <!-- ======= Testimonials Section ======= -->
    {% include 'reviews.html' %}

    <!-- ======= Contact Section ======= -->
   {% include 'contactform.html' %}

</main><!-- End #main -->
{% endblock %}
```

Файл «main_page/templates/reviews.html»

```
<section id="testimonials" class="testimonials">
      <div class="container position-relative">

        <div class="testimonials-slider swiper" data-aos="fade-up" data-aos-
delay="100">
          <div class="swiper-wrapper">
            {% for item in reviews %}
              <div class="swiper-slide">
```

```
                <div class="testimonial-item">
                  <h3>{{ item.name }}</h3>
                  <h4>{{ item.job }}</h4>
                  <div class="stars">
                      {% with ''|center:item.stars as range %}
                          {% for star in range %}
                              <i class="bi bi-star-fill"></i>
                          {% endfor %}
                      {% endwith %}
                  </div>
                  <p>
                    <i class="bx bxs-quote-alt-left quote-icon-left"></i>
                    {{ item.speech }}
                    <i class="bx bxs-quote-alt-right quote-icon-right"></i>
                  </p>
                </div>
              </div>
            {% endfor %}
          </div>
          <div class="swiper-pagination"></div>
        </div>

      </div>
    </section>
```

Файл «manager/urls.py»

```
from django.urls import path
from .views import feedback_list, update_feedback


app_name = 'manager'

urlpatterns = [
    path('', feedback_list, name='manager_view'),
    path('feedback/update/<int:pk>/', update_feedback, name='update_feedback'),
]
```

Файл «manager/views.py»

```
from django.shortcuts import render, redirect
from main_page.models import ContactForm
from django.contrib.auth.decorators import login_required, user_passes_test


# Create your views here.
def is_manager(user):
    return user.groups.filter(name='manager').exists()


@login_required(login_url='/login/')
@user_passes_test(is_manager)
def feedback_list(request):
    if request:
        cleaner = ContactForm.objects.filter(is_processed=True)
        cleaner.delete()

    feedback = ContactForm.objects.filter(is_processed=False)
    return render(request, 'feedback_list.html', context={'feedback': feedback})
```

```
@login_required(login_url='/login/')
@user_passes_test(is_manager)
def update_feedback(request, pk):
    ContactForm.objects.filter(pk=pk).update(is_processed=True)
    return redirect('manager:manager_view')
```

Файл «manager/feedback_list.html»

```
{% extends 'index.html' %}

{% block content %}
    <section class="section">
        <div class="container">
        <div class="content" align="center">
            <div class="section-heading">
                <h3>Contact<b>Us</b> Feedbacks</h3>
                <br>
            </div>
        </div>
            <td class="col-md-10 col-md-offset-1">
                {% for item in feedback %}
                    <div class="row">
                        <div class="col-md-3">
                            <a href="{% url 'manager:update_feedback' pk=item.pk
%}">
                                <button type="button" class="book-a-table-btn
scrollto">Close Subject</button>
                            </a>
                            <p></p>
                        </div>
                        <div class="col-md-3">Name: {{ item.name }}</div>
                        <div class="col-md-3">Phone: {{ item.phone }}</div>
                        <div class="col-md-3">Email: {{ item.email }}</div>
                        <div class="col-md-3">Subject: {{ item.subject }}</div>
                        <div class="col-md-3">Date: {{ item.created_at|date:'d-m-
Y' }}</div>
                    <p></p>
                    </div>
                    <div class="row">
                        <div class="col-md-3"></div>
                        <div class="col-md-9"><p>{{ item.message }}</p></div>
                    </div>
                    <br>
                {% endfor %}
            </div>
        </div>
    </section>
{% endblock %}
```

Файл «account/views.py»

```
from django.shortcuts import render, redirect
from .forms import LoginForm, RegistrationForm
from django.contrib.auth import authenticate, login, logout


# Create your views here.
def login_view(request):
```

```
    form = LoginForm(request.POST or None)
    next_get = request.GET.get('next')

    if form.is_valid():
        username = request.POST.get('username')
        password = request.POST.get('password')

        user = authenticate(username=username.strip(), password=password.strip())
        login(request, user)
        next_post = request.POST.get('next')
        return redirect(next_get or next_post or '/')

    return render(request, 'login.html', context={'form': form})


def logout_view(request):
    logout(request)
    return redirect('/')


def registration_view(request):
    form = RegistrationForm(request.POST or None)

    if form.is_valid():
        new_user = form.save(commit=False)
        new_user.set_password(form.cleaned_data['password'])
        new_user.save()
        return render(request, 'registration_done.html', context={'user':
new_user})
    return render(request, 'registration.html', context={'form': form})
```

Файл «account/forms.py»

```
from django import forms
from django.contrib.auth import authenticate, get_user_model


User = get_user_model()


class LoginForm(forms.Form):
    username = forms.CharField(widget=forms.TextInput())
    password = forms.CharField(widget=forms.PasswordInput())

    def clean(self, *args, **kwargs):
        username = self.cleaned_data.get('username')
        password = self.cleaned_data.get('password')

        if username and password:
            user = authenticate(username=username, password=password)
            if not user or not user.check_password(password):
                raise forms.ValidationError('Incorrect login or password')
        return super().clean()


class RegistrationForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('username', )

    username = forms.CharField(widget=forms.TextInput())
```

```
    password = forms.CharField(widget=forms.PasswordInput())
    password2 = forms.CharField(widget=forms.PasswordInput())

    def clean_password2(self, *args, **kwargs):
        data = self.cleaned_data
        if data['password'] == data['password2']:
            return data['password2']
        return forms.ValidationError('Password doesn`t match')
```

Файл «account/login.html»

```
{% extends 'index.html' %}

{% block content %}
    <section class="section" id="schedule">
        <div class="container">
            <div class="section-heading text-center dark-bg">
                <br>
                <h2>Login</h2>
                <br>
            </div>
            <div>
                <div class="section-heading dark-bg">
                    <div class="row">
                        <div class="section-heading text-center dark-bg" >
                            <form id="contact" action="" method="post">
                                {% csrf_token %}
                                    <fieldset>
                                        <p><em>Username:</em> {{ form.username
}}</p>
                                    </fieldset>
                                    <br>
                                    <fieldset>
                                        <p><em>Password:</em> {{ form.password
}}</p>
                                    </fieldset>
                                    <br>
                                    <fieldset>
                                        <button type="submit" class="book-a-table-
btn scrollto">Login</button>
                                    </fieldset>
                            </form>
                            <br>
                            <p>Don't have as account? You can <a
href="/registration/" >register</a>!</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </section>
{% endblock %}
```

Файл «account/registation.html»

```
{% extends 'index.html' %}

{% block content %}
    <section class="section" id="schedule">
```

```
        <div class="container">
            <div class="section-heading text-center dark-bg">
                <br>
                <h2>Registration</h2>
            </div>
            <div>
                <div class="section-heading dark-bg">
                    <div class="row">
                        <div class="section-heading text-center dark-bg" >
                            <form id="contact" action="" method="post">
                                {% csrf_token %}
                                    <fieldset>
                                        <p><em>Username:</em>  {{ form.username
}}</p>
                                    </fieldset>
                                    <br>
                                    <fieldset>
                                        <p><em>Password:</em>  {{ form.password
}}</p>
                                    </fieldset>
                                    <br>
                                    <fieldset>
                                        <p><em>Confirm Pass:</em> {{
form.password2 }}</p>
                                    </fieldset>
                                    <br>
                                    <fieldset>
                                        <button type="submit" id="form-submit"
class="book-a-table-btn scrollto">Register</button>
                                    </fieldset>
                            </form>
                            <br>
                            <p>Have as account? You can <a href="/login/"
>login</a>!</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </section>
{% endblock %}
```

Файл «account/registration_done.html»

```
{% extends 'index.html' %}

{% block content %}
    <section class="section" id="schedule">
        <div class="container">
            <div class="section-heading text-center dark-bg">
                <br>
                <h2>Congratulations!</h2>
            </div>
            <div>
                <div class="section-heading dark-bg">
                    <div class="row">
                        <div class="section-heading text-center dark-bg" >
                            <p>User {{ user.username }} successfully
registered!</p>
                            <p><a href="/">To main page</a></p>
                        </div>
```

```
                    </div>
                </div>
            </div>
        </div>
    </section>
{% endblock %}
```

Файл «cart/views.py»

```
from django.shortcuts import render, redirect
from main_page.models import Products, CartItem


def view_cart(request):
    cart_items = CartItem.objects.filter(user=request.user)
    total_price = sum(item.product.price * item.quantity for item in cart_items)
    return render(request, 'cart.html', {'cart_items': cart_items,
                                         'total_price': total_price})


def add_to_cart(request, product_id):
    product = Products.objects.get(id=product_id)
    cart_item, created = CartItem.objects.get_or_create(product=product,
                                                        user=request.user)
    cart_item.quantity += 1
    cart_item.save()
    return redirect('cart:view_cart')


def remove_from_cart(request, item_id):
    cart_item = CartItem.objects.get(id=item_id)
    cart_item.delete()
    return redirect('cart:view_cart')


def delete_cart(request):
    CartItem.objects.all().delete()
    return redirect('main_page:main_page_view')
```

Файл «cart/urls.py»

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


app_name = 'cart'

urlpatterns = [
    path('', views.view_cart, name='view_cart'),
    path('add/<int:product_id>/', views.add_to_cart, name='add_to_cart'),
    path('remove/<int:item_id>/', views.remove_from_cart,
name='remove_from_cart'),
    path('delete/', views.delete_cart, name='delete_cart'),
]
```

Файл «cart/cart.html»

```
{% extends 'index.html' %}
{% load static %}
{% block content %}
    <section class="section">
        <div class="container">
        <div class="content" align="center">
            <table class="cart">
                <thead>
                <tr>
                    <th>Image</th>
                    <th>Product</th>
                    <th>Quantity</th>
                    <th>Remove</th>
                    <th>Unit price</th>
                    <th>Price</th>
                </tr>
                </thead>
                <tbody>
                {% for item in cart_items %}
                        <tr>
                            <td>
                                <a href="{{ product.get_absolute_url }}">
                                    <img src="
                                        {% if item.product.photo %}
                                            {{ item.product.photo_thrumbnail.url
}}
                                        {% else %}
                                            <p>No image</p>
                                        {% endif %}" alt="" width="120px"
height="120px">
                                </a>
                            </td>
                            <td>{{ item.product.name }}</td>
                            <td>{{ item.quantity }}</td>
                            <td class="num">${{ item.product.price }}</td>
                            <td><a href="{% url 'cart:remove_from_cart' item.id
%}">Remove</a></td>
                        </tr>
                    {% empty %}
                        <div class="container">
                            <p>Cart is empty!</p>
                        </div>
                {% endfor %}
                <tr class="total">
                    <td>Total</td>
                    <td colspan="4"></td>
                    <td class="num">${{ total_price }}</td>
                </tr>
                </tbody>
            </table>
                <div class="content" align="right   ">
                    <br>
                    <a href="{% url 'main_page:main_page_view' %}" class="book-a-
table-btn scrollto">Main page</a>
                    <br>
                    <br>
                    <a href="{% url 'cart:delete_cart' %}" class="book-a-table-btn
scrollto">Checkout</a>
                </div>
            </div>
        </div>
    </section>
{% endblock %}
```

Файл «templates/index.html»

```
{% load static %}

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">

  <title>Delicious Bootstrap Template - Index</title>
  <meta content="" name="description">
  <meta content="" name="keywords">

  <!-- Favicons -->
  <link href= {% static "img/favicon.png" %} rel="icon">
  <link href= {% static "img/apple-touch-icon.png" %} rel="apple-touch-icon">

  <!-- Google Fonts -->
  <link
href="https://fonts.googleapis.com/css?family=Poppins:300,300i,400,400i,600,600i,7
00,700i|Satisfy|Comic+Neue:300,300i,400,400i,700,700i" rel="stylesheet">

  <!-- Vendor CSS Files -->
  <link href= {% static "vendor/animate.css/animate.min.css" %} rel="stylesheet">
  <link href= {% static "vendor/bootstrap/css/bootstrap.min.css" %}
rel="stylesheet">
  <link href= {% static "vendor/bootstrap-icons/bootstrap-icons.css" %}
rel="stylesheet">
  <link href= {% static "vendor/boxicons/css/boxicons.min.css" %}
rel="stylesheet">
  <link href= {% static "vendor/glightbox/css/glightbox.min.css" %}
rel="stylesheet">
  <link href= {% static "vendor/swiper/swiper-bundle.min.css" %} rel="stylesheet">

  <!-- Template Main CSS File -->
  <link href= {% static "css/style.css" %} rel="stylesheet">

  <!-- ======================================================== -->
</head>

<body>

  <!-- ======= Top Bar ======= -->
  <section id="topbar" class="d-flex align-items-center fixed-top topbar-
transparent">
    <div class="container-fluid container-xl d-flex align-items-center justify-
content-center justify-content-lg-start">
      <i class="bi bi-phone d-flex align-items-center"><span>+38 089 222 22
22</span></i>
      <i class="bi bi-clock ms-4 d-none d-lg-flex align-items-center"><span>Mon-
Fri: 10:00 AM - 20:00 PM</span></i>
    </div>
  </section>

  <!-- ======= Header ======= -->
  <header id="header" class="fixed-top d-flex align-items-center header-
transparent">
    <div class="container-fluid container-xl d-flex align-items-center justify-
content-between">

      <div class="logo me-auto">
```

```
        <h1><a href="/">De'Longhi Cofee Maker</a></h1>
      </div>

      <nav id="navbar" class="navbar order-last order-lg-0">
        <ul>
            {% if 'manager' in user.groups.all.0.name %}
          <li><a class="nav-link scrollto active" href="{% url
'manager:manager_view' %}">FeedBack</a></li>
            {% else %}
          <li><a class="nav-link scrollto active" href="#hero">Home</a></li>
          <li><a class="nav-link scrollto" href="#about">About</a></li>
          <li class="dropdown"><a href="#classic"><span>Products</span> <i
class="bi bi-chevron-down"></i></a>
            <ul>
              <li><a class="nav-link scrollto" href="#classic">Classic</a></li>
              <li><a class="nav-link scrollto" href="#modern">Modern</a></li>
              <li><a class="nav-link scrollto" href="#premium">Premium</a></li>
              <li><a class="nav-link scrollto" href="#super-premium">Super
Premium</a></li>
            </ul>
          </li>
          <li><a class="nav-link scrollto" href="#gallery">Gallery</a></li>
          <li><a class="nav-link scrollto" href="#contact">Contact</a></li>
            {% endif %}
        </ul>
        <i class="bi bi-list mobile-nav-toggle"></i>
      </nav><!-- .navbar -->
    <div class="cart">
        {% if request.user.is_authenticated %}
            <a href="{% url 'cart:view_cart' %}" class="book-a-table-btn
scrollto">Cart</a>
            <a class="book-a-table-btn scrollto">{{ user }}</a>
            <a href="/logout/" class="book-a-table-btn scrollto">Log Out</a>
        {% else %}
            <a href="/login/" class="book-a-table-btn scrollto">Login</a>
        {% endif %}
    </div>
    </div>
  </header><!-- End Header -->

  <!-- ======= Hero Section ======= -->
  <section id="hero">
    <div class="hero-container">
      <div id="heroCarousel" data-bs-interval="5000" class="carousel slide
carousel-fade" data-bs-ride="carousel">

        <ol class="carousel-indicators" id="hero-carousel-indicators"></ol>

        <div class="carousel-inner" role="listbox">

          <!-- Slide 1 -->
          <div class="carousel-item active" style="background-image: url({% static
"img/slide/slide1.jpg" %});">
            <div class="carousel-container">
              <div class="carousel-content">
                <h2 class="animate__animated animate__fadeInDown"><span>De'Longhi
</span> Cofee Maker</h2>
                <p class="animate__animated animate__fadeInUp">Our store is one of
the best resellers. We have a large selection of products and responsive
support.</p>
                <div>
                  <a href="#classic" class="btn-menu animate__animated
animate__fadeInUp scrollto">Products</a>
```

```
                <a href="#cont" class="btn-book animate__animated
animate__fadeInUp scrollto">Contact Us</a>
              </div>
            </div>
          </div>
        </div>

        <!-- Slide 2 -->
        <div class="carousel-item" style="background-image: url({% static
'img/slide/slide2.jpg' %});">
          <div class="carousel-container">
            <div class="carousel-content">
              <h2 class="animate__animated animate__fadeInDown">Quality for
centuries</h2>
              <p class="animate__animated animate__fadeInUp">We are famous for
the quality of our product. If something goes wrong, we will definitely find a
solution that will make you happy.</p>
              <div>
                <a href="#classic" class="btn-menu animate__animated
animate__fadeInUp scrollto">Products</a>
                <a href="#cont" class="btn-book animate__animated
animate__fadeInUp scrollto">Contact Us</a>
              </div>
            </div>
          </div>
        </div>

        <!-- Slide 3 -->
        <div class="carousel-item" style="background-image: url({% static
'img/slide/slide3.jpg' %});">
          <div class="carousel-container">
            <div class="carousel-content">
              <h2 class="animate__animated animate__fadeInDown">Support</h2>
              <p class="animate__animated animate__fadeInUp">Our support is
available seven days a week. Leave us a request and we will contact you within 10
minutes.</p>
              <div>
                <a href="#classic" class="btn-menu animate__animated
animate__fadeInUp scrollto">Products</a>
                <a href="#cont" class="btn-book animate__animated
animate__fadeInUp scrollto">Contact Us</a>
              </div>
            </div>
          </div>
        </div>

      </div>

      <a class="carousel-control-prev" href="#heroCarousel" role="button" data-
bs-slide="prev">
        <span class="carousel-control-prev-icon bi bi-chevron-left" aria-
hidden="true"></span>
      </a>

      <a class="carousel-control-next" href="#heroCarousel" role="button" data-
bs-slide="next">
        <span class="carousel-control-next-icon bi bi-chevron-right" aria-
hidden="true"></span>
      </a>

    </div>
  </div>
</section>
```

```
  <!-- End Hero -->

  {% block content %}
  {% endblock %}

  <!-- ======= Footer ======= -->
  <footer id="footer">
    <div class="container">
      <h3>De'Longhi Coffe Maker</h3>

      <div class="social-links">
        <a href="https://twitter.com/" class="twitter"><i class="bx bxl-
twitter"></i></a>
        <a href="https://www.facebook.com/" class="facebook"><i class="bx bxl-
facebook"></i></a>
        <a href="https://www.instagram.com/" class="instagram"><i class="bx bxl-
instagram"></i></a>
      </div>
      <div class="copyright">
        &copy; Copyright <strong><span>Kud Vladyslav</span></strong>. This site
was made for bachelor's work (DUIKT).
      </div>
      <div class="credits">
        Designed by <a>Kud Vladyslav</a>
      </div>
    </div>
  </footer><!-- End Footer -->

  <a href="#" class="back-to-top d-flex align-items-center justify-content-
center"><i class="bi bi-arrow-up-short"></i></a>

  <!-- Vendor JS Files -->
  <script src={% static "vendor/bootstrap/js/bootstrap.bundle.min.js" %}></script>
  <script src={% static "vendor/glightbox/js/glightbox.min.js" %}></script>
  <script src={% static "vendor/isotope-layout/isotope.pkgd.min.js" %}></script>
  <script src={% static "vendor/swiper/swiper-bundle.min.js" %}></script>
  <script src={% static "vendor/php-email-form/validate.js" %}></script>

  <!-- Template Main JS File -->
  <script src={% static "js/main.js" %}></script>

</body>

</html>
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра
Комп'ютерної інженерії

Презентація бакалаврської роботи на тему:
«Створення онлайн магазину на основі Python Django»

ВИКОНАВ: СТУДЕНТ ГРУПИ КІД-41
КУДЬ ВЛАДИСЛАВ ДМИТРОВИЧ
КЕРІВНИК РОБОТИ:
КОНДРАТЮК БОРИС ОЛЕКСАНДРОВИЧ
АСИСТЕНТ КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

1

Слайд 1

Мета, об'єкт і предмет дослідження та актуальність теми

Мета роботи – створення веб-сайту на основі мови програмування Python з використанням фреймворку Django.

Об'єкт дослідження – процес створення веб-сайту з використанням мов програмування, включаючи різні етапи процесу розробки та особливості інструментів, які застосовуються при цьому

Предмет дослідження – створення веб-сайтів, а також технології та методи, що використовуються при цьому, їх застосування для досягнення поставлених цілей у галузі веб-розробки.

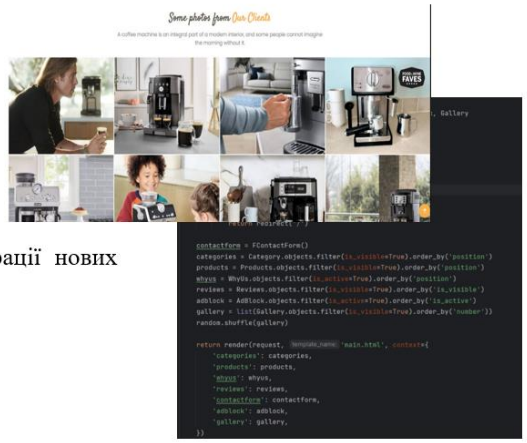
Актуальність теми – у сучасному світі веб-розробка стає все більш затребуваною та перспективною областю, особливо з урахуванням постійно зростаючої залежності різних галузей від інформаційних технологій. Мова програмування Python, разом з фреймворком Django, є інструментом для створення веб-додатків, що мають високу продуктивність і гнучкість.

2

Слайд 2

Задачі даної роботи:

1. Розробка дизайну
2. Реалізація фільтрів для швидкого пошуку товару
3. Створення зручної панелі адміністратора сайту, де можна буде додавати, редагувати та керувати контентом
4. Аутентифікацію користувачів з можливістю реєстрації нових користувачів
5. Обробка форм зворотнього зв'язку
6. Унікальна сторінка для менеджерів
7. Кошик товарів



3

Слайд 3

Використовувані інструменти та засоби розробки:

1. Python
2. Django
3. Bootstrap
4. JavaScript

4

Слайд 4

Python - високорівнева мова програмування загального призначення.

Був створений у 1991 році голландським математиком Гвідо ван Россумом. З самого початку це був відкритий для спільноти проект.

Де використовується Python

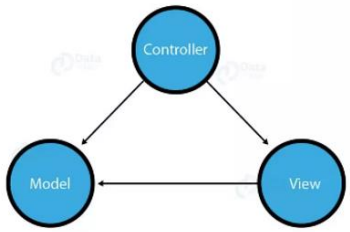
Веб-розробка та хмарні технології	Наукоємні технології та машинне навчання	Скриптова мова ігор
Instagram, openstack, YouTube, Dropbox	NASA, Los Alamos National Laboratory EST.1943	EVE Online, Mount&Blade, Sid Meier's Civilization IV

5

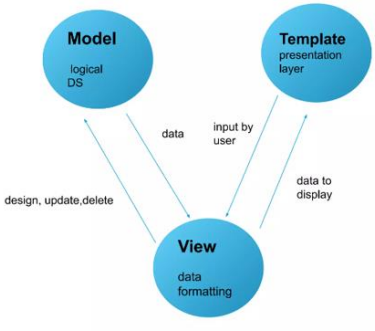
Слайд 5

Коротко про django

Django — вільний фреймворк для веб-застосунків мовою Python, що використовує шаблон проектування MVC (Model-View-Controller).



MVC

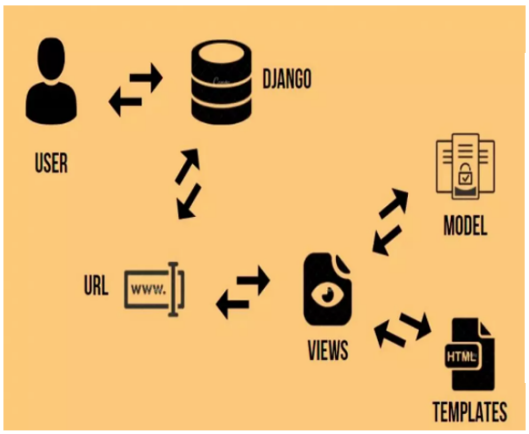


MVT

6

Слайд 6

Як відбувається весь процес?



1.Користувач надсилає URL-запит на ресурс до Django

2. Потім платформа Django шукає URL-ресурс

3. Якщо шлях URL-адреси пов'язаний із "View", тоді викликається саме ця "View"

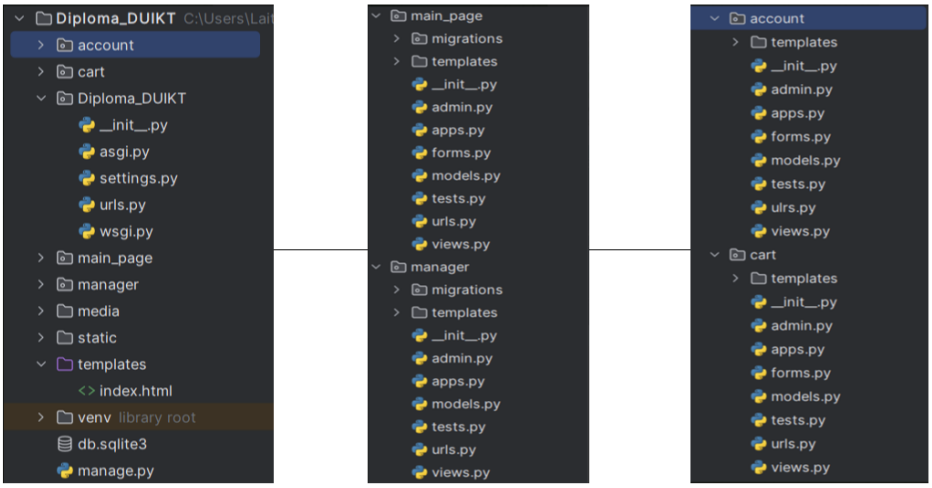
4. "View" буде взаємодіяти з моделлю та отримувати відповідні дані з бази даних

5. Потім "View" відображає відповідний шаблон разом із отриманими даними для користувача.

7

Слайд 7

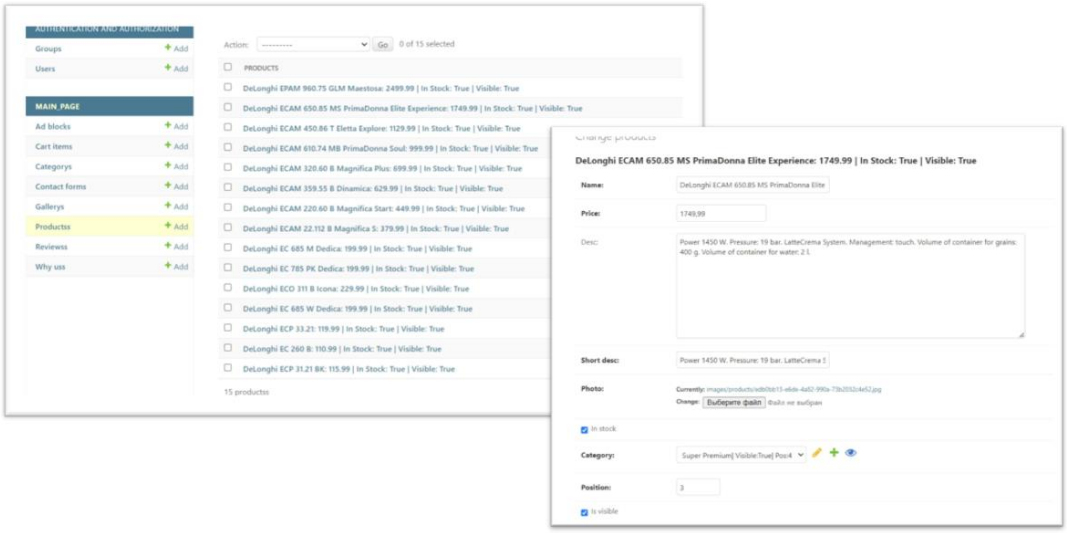
Структура файлів розробленого веб-застосунку



8

Слайд 8

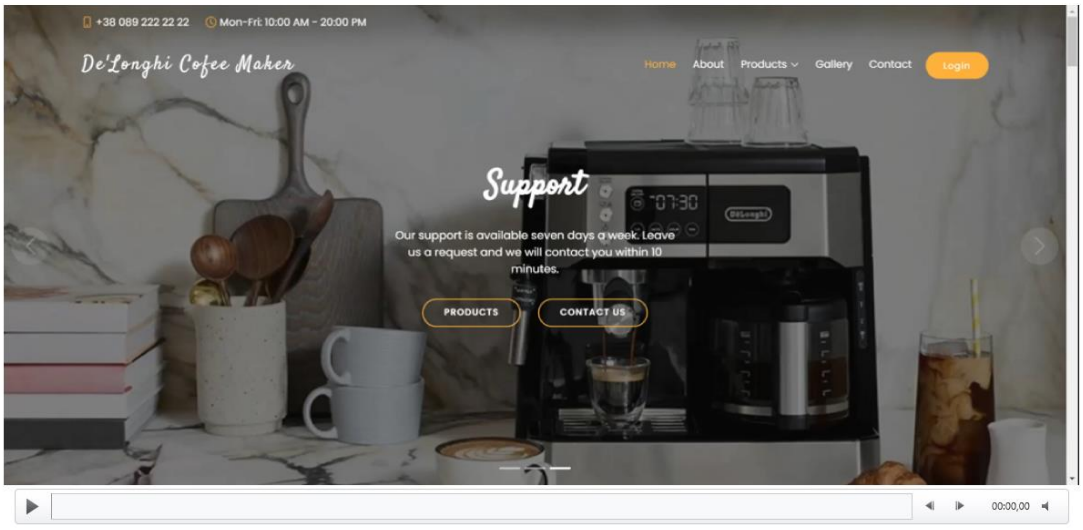
Панель адміністратора та редагування продукту



9

Слайд 9

Головна сторінка



10

Слайд 10

Висновки

У цій роботі було розглянено і проведено процес створення веб-сайту за допомогою Python та фреймворку Django. Основна увага була приділена аналізу основних етапів створення веб-сайту.

Використано метод аналізу літератури з вивченням документації Python та Django, а також експериментальний метод, що полягає у створенні прототипів веб-додатків та аналізі їхньої продуктивності та функціональності.

Всі завдання та цілі були успішно досягнуті. Дослідження стало важливим кроком у розумінні процесу створення веб-сайту та використання різних методів для досягнення цілей веб-розробки.

11

Слайд 11

Дякую за увагу

12

Слайд 12

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Владислав КУДЬ

Назва: Створення онлайн магазину на основі Python Django
Координатор: Борис КОНДРАТЮК
Підрозділ: ННІІТ

Коефіцієнт подібності 1:4.3

Коефіцієнт подібності 2:1.2

Тривога: 12

Після аналізу Звіту подібності констатую наступне:

☐ виявлені в роботі запозичення є сумлінними і не мають ознак плагіату. Тому робота визнається самостійною і допускається до захисту;

☐ виявлені в роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на доопрацювання;

☐ виявлені в роботі запозичення є недобросовісними і мають ознаки плагіату або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. У зв'язку з чим, робота не допускається до захисту.

……. ……

Дата *Підпис Наукового керівника*