ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка і впровадження веб-застосунку для електронної комерції на основі Django: Аналіз, проектування та реалізація системи продажу електроніки»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

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

_____ Сергій БОНДАРЕНКО
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-42
Сергій БОНДАРЕНКО

Керівник: Іван ШАХМАТОВ
науковий ступінь,
вчене звання

Рецензент: _____
науковий ступінь, Ім'я, ПРІЗВИЩЕ
вчене звання

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Бакалавр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС
_____ Каміла СТОРЧАК
«___»_____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бондаренку Сергію Юрійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка і впровадження веб-застосунку для електронної комерції на основі Django: Аналіз, проектування та реалізація системи продажу електроніки

керівник кваліфікаційної роботи Іван ШАХМАТОВ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно комунікаційних технологій від «_»___20___р. №

2. Строк подання кваліфікаційної роботи «19» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, проєкт Django, шаблони HTML та CSS, ORM-моделі.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз теоретичних принципів предметної області
Моделювання та проєктування інформаційної системи
Розробка веб-застосунку

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «21» лютого 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	10.04.2024-17.04.2024	
2	Вивчення теоретичних основ фреймворку Django	17.04.2024-20.04.2024	
3	Дослідження підходів та технологій для створення веб-застосунку	20.04.2024-24.04.2024	
4	Визначення та обґрунтування архітектурних рішень	24.04.2024-28.04.2024	
5	Реалізація застосунку	28.04.2024-05.05.2024	
6	Тестування та оцінка ефективності створеного застосунку	05.05.2024-08.05.2024	
7	Оформлення роботи: вступ, висновки, реферат	08.05.2024-09.05.2024	
8	Розробка демонстраційних матеріалів	09.05.2024-10.05.2024	

Здобувач вищої освіти _____ Сергій БОНДАРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Іван ШАХМАТОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 25 рис., 10 джерел.

Мета роботи – проєктування і розробка веб-застосунку для електронної комерції на основі Django.

Об'єкт дослідження – процес веб-розробки з застосуванням фреймворку Django.

Предмет дослідження – веб-застосунок для електронної комерції.

Короткий зміст роботи: У роботі було проаналізовано сферу електронної комерції. Досліджено підходи та технології веб-розробки. Було розглянуто теоретичні основи роботи Django. Створена основна структура проєкту. Розроблено та протестовано застосунок системи продажу електроніки за допомогою фреймворку Django.

КЛЮЧОВІ СЛОВА: DJANGO, AJAX, BOOTSTRAP, STRIPE, ORM, БАЗИ ДАНИХ, ШАБЛОНИ

ЗМІСТ

ВСТУП...……..……..6
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОІ ОБЛАСТІ……..…….8
1.1 Аналіз сфери електронної комерції……...…….8
1.2 Історія електронної комерції……..……9
1.3 Огляд Django……..….....10
1.4 Огляд компонентів Django…….....11
1.5 Особливості використання технології AJAX……...15
1.6 Особливості фреймворку Bootstrap…….....18
2 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ ВЕБ-ЗАСТОСУНКІВ НА ОСНОВІ DJANGO…….....21
2.1 Аналіз функціонування додатків Django……...…...21
2.2 Принципи проєктування баз даних……...21
2.3 Особливості роботи шаблонів Django……..26
2.4 Обробка платежів з використанням Stripe…….....30
2.5 Тестування застосунків з unittest……...32
3 РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ……...36
3.1 Підготовка середовища розробки……..36
3.2 Розробка моделей……41
3.3 Створення форм……..45
3.4 Розробка шаблонів та відображень……...48
3.5 Обробка електронних платежів зі Stripe……..51
3.6 Тестування веб-застосунку……55
ВИСНОВКИ……..58
ПЕРЕЛІК ПОСИЛАНЬ……60
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)……....61

ВСТУП

Актуальність: Популярність Django постійно зростає завдяки його розгалуженій екосистемі та багатій документації, що робить його привабливим вибором для розробників та організацій. Універсальність Django дозволяє розробникам працювати як над фронтендом, так і над бекендом додатків. Вбудовані функції, такі як інтерфейс адміністратора, автентифікація користувачів та ORM, прискорюють час розробки та зменшують кількість шаблонного коду. Django має активну спільноту, яка робить свій внесок в екосистему, розробляючи багаторазові пакети, плагіни та бібліотеки, а також надає широку підтримку через форуми, конференції та онлайн-ресурси.

Об'єкт дослідження – процес веб-розробки з застосуванням фреймворку Django.

Предмет дослідження – веб-застосунок для електронної комерції.

Мета роботи – проєктування та розробка веб-застосунку для електронної комерції з використанням Django.

Наукова новизна – використання Django та Ajax для розробки сучасного та ефективного веб-застосунку системи продажу електроніки, що дозволяє інтегрувати динамічні функції та покращити взаємодію з користувачем.

Практична значущість – Результати роботи можуть бути використані при розробці та проєктуванні інших веб-застосунків на основі Django, що підвищить їх ефективність та функціональність.

Методи дослідження – Аналіз літературних джерел, проєктування, системний аналіз, програмування та веб-розробка.

Апробація результатів:

Бондаренко С.Ю. «Оптимізація технологій IoT для "розумного будинку" та "розумного міста" з використанням стандарту 5G» Теза доповіді на I Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». – Київ, 28 листопада 2023 р.

Бондаренко С.Ю. «Особливості впровадження технологій штучного інтелекту у сфері IoT» Тези на V Міжнародній науково-технічній конференції «Сучасний стан та перспективи розвитку IoT». – Київ, 18 квітня 2024 р.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОІ ОБЛАСТІ

1.1 Аналіз сфери електронної комерції

Електронна комерція - це бізнес-модель, в якій комерційна діяльність здійснюється через електронні мережі, зокрема через Інтернет.

Залежно від взаємовідносин між залученими сторонами, електронну комерцію можна поділити на такі категорії:

— B2C (Business to Consumer) – товар продається покупцю напряму;

— B2B (Business to Business) – торгівля відбувається між бізнесами;

— C2C (Consumer to Consumer) – споживачі продають товари іншим споживачам.

Зростання електронної комерції супроводжується багатьма супутніми явищами, такими як автоматизація, діджиталізація, штучний інтелект, аналіз великих даних тощо. Розвиток Інтернету та сучасних технологій трансформував торгівлю до такої міри, що більшість транснаціональних потоків товарів проходять через онлайн-платформи.

Переваги електронної комерції:

— Розширює ринки збуту на національному та міжнародному рівні. Компанія може швидко знайти більше клієнтів, найкращих постачальників і найбільш підходящих бізнес-партнерів по всьому світу з мінімальними капіталовкладеннями.

— Часто надає дешевші продукти та послуги, дозволяючи споживачам легко порівнювати продукти, бренди та веб-сайти.

— Клієнти можуть заощадити час, купуючи те, що вони хочуть. Вони можуть легко переглядати велику кількість товарів одночасно і купувати те, що їм подобається.

— Покупець може залишати відгуки про товар і бачити, що купують інші, або переглядати відгуки інших покупців перед тим, як зробити остаточну покупку.

— Онлайн торгівля дозволяє споживачам подолати кордони і купувати товари будь-де в будь-який час.

— Вона приносить зручність покупцям, оскільки їм не потрібно виходити з дому, а достатньо лише переглянути веб-сайт в Інтернеті, особливо для купівлі товарів, які не продаються в найближчих магазинах.

Електронна комерція вже на шляху до того, щоб забезпечити більшу частину доходів від продажів у більшості галузей. Фахівці прогнозують, що протягом наступних двох років частка доходів від цифрових каналів перевищить цей поріг, відображаючи зростаючі переваги покупців B2C і B2B, які віддають перевагу онлайн-покупкам та взаємодії в Інтернеті. Більшість фахівців у сфері торгівлі спостерігають і передбачають подальше зростання попиту як з боку кінцевих споживачів, так і з боку внутрішніх команд, з якими вони працюють.

1.2 Історія електронної комерції

У 1969 році Доктор Джон Р. Гольц та Джеффрі Вілкінс заснували CompuServe, першу значну компанію електронної комерції, використовуючи комутоване з'єднання. Це стало першим випадком впровадження електронної комерції.

У 1979 році Майкл Олдріч винайшов електронні покупки (він також вважається засновником або винахідником електронної комерції). Для цього з'єднали комп'ютер для обробки транзакцій з модифікованим телевізором через телефонний зв'язок. Це було зроблено для передачі захищених даних.

У 1982 розвиток технологій, особливо в електроніці, призвів до запуску першої платформи електронної комерції на Бостонській комп'ютерній біржі.

У 1994 році Марк Андрессен і Джим Кларк представили веб-браузер Netscape Navigator. Він працював на платформі Windows.

1995 рік ознаменувався знаковою подією в історії електронної комерції, коли були запущені Amazon та eBay. Amazon заснував Джефф Безос, а eBay - П'єр Омідьяр.

У 1998 році PayPal запустив першу платіжну систему електронної комерції як інструмент для здійснення грошових переказів.

У 1999 році Alibaba запустила свою платформу для онлайн-покупок у 1999 році з капіталом понад 25 мільйонів доларів. Поступово вона перетворилася на гіганта електронної комерції.

У 2000 році Google запустив перший інструмент онлайн-реклами під назвою Google AdWords, щоб допомогти ритейлерам використовувати контекстну рекламу з оплатою за клік (PPC).

У 2005 році Amazon запустив членство в Amazon Prime, щоб допомогти клієнтам отримати безкоштовну дводенну доставку за річну плату.

У 2006 році Тобіас Лютке, Даніель Вейнанд і Скотт Лейк заснували компанію, яка зараз відома як Shopify, щоб спростити для власників бізнесу запуск інтернет-магазинів.

У 2012 році продажі електронної комерції вперше в історії перевищили $1 трлн.

У 2020 році Зростання електронної комерції посилюється через пандемію COVID-19, коли люди переходять на онлайн-покупки, особливо продуктів харчування, розваг тощо.

1.3 Огляд Django

Django - це фреймворк з відкритим вихідним кодом, написаний мовою Python. Він використовується для швидкої, прагматичної, підтримуваної та безпечної розробки веб-сайтів з чистим дизайном. Головне завдання Django - дозволити розробникам зосередитися на нових компонентах додатку, замість витрачання часу на повторну розробку існуючих.

Django робить процес розробки надзвичайно швидким, від ідеї до релізу, через запуск і виробництво. Завдяки прозорому, чистому коду розробка може бути ефективною та результативною.

Django використовують тисячі веб-сайтів, включаючи щоденні газети, соціальні мережі, сайти для обміну даними, великі фонди та некомерційні

організації. Оскільки Django спочатку розроблявся для використання в редакціях новин, не дивно, що такі великі щоденні видання, як Washington Post і The Guardian, покладаються на нього. Стартапи, такі як Eventbrite та Disqus, використовують Django для швидкого масштабування, а гіганти соціальних мереж, як Instagram та Pinterest, застосовують його для створення динамічних веб-додатків. Django є одним з найпопулярніших і найпоширеніших серед усіх Python-фреймворків.

Django дотримується підходу "батарейки в комплекті" і надає широкий спектр функціональних можливостей "з коробки", таких як інтегроване середовище модульного тестування, надійний шар ORM (Object Relational Mapping) і вбудований інтерфейс адміністратора (Django admin).

Django написаний з використанням принципу "Не повторюй себе", який сприяє повторному використанню коду та зменшенню його дублювання.

1.4 Огляд компонентів Django

Стандартний проєкт Django складається з таких файлів (Рис. 1.1):

— urls.py – маршрутизатор URL

— views.py – відображення

— models.py – ORM-моделі

— templates – шаблони HTML

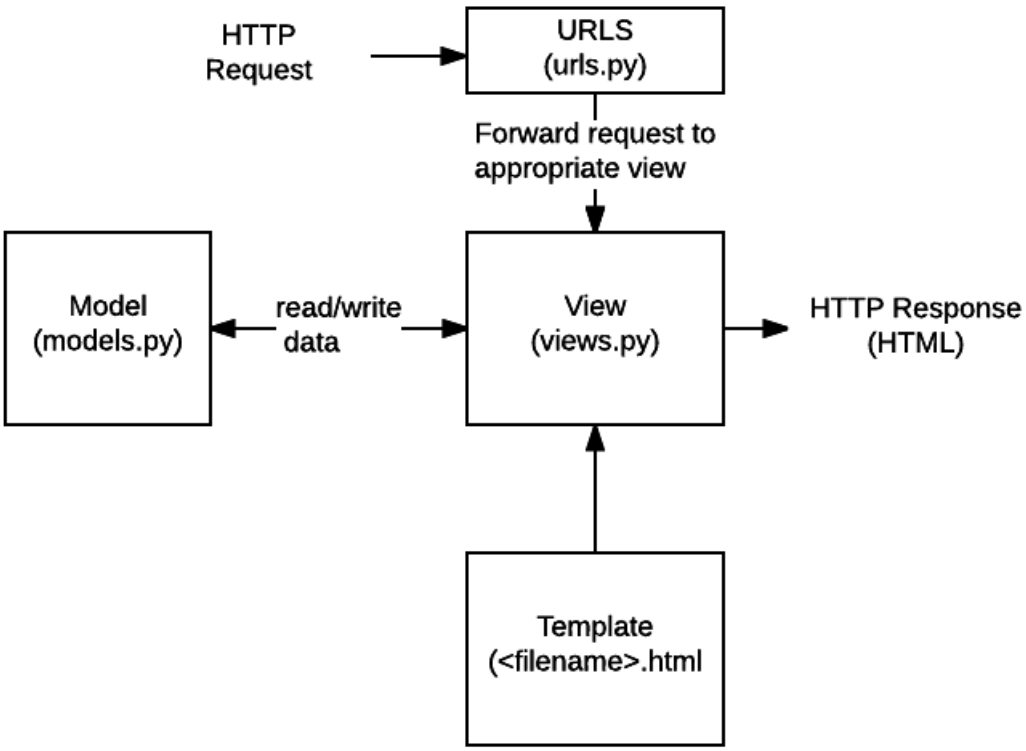


Рис. 1.1 Схема взаємодії файлів Django[4]

Маршрутизатор URL:

У Django маршрутизатор URL використовуються для зіставлення запитів користувача з відповідними відображеннями, які, в свою чергу, виводять правильні шаблони і дані. URL-шаблони у Django можуть бути визначені за допомогою регулярних виразів або простих рядків. Регулярні вирази забезпечують більш гнучкий і корисний спосіб пошуку URL, в той час як прості рядки корисні для точного збігу.

Відображення:

Відображення Django - це функції, які викликаються при запиті на певні URL-адреси. Вони діють як точка з'єднання між користувачем і внутрішньою логікою системи. Зазвичай, відображення кожного додатка визначаються в модулі views.py. Відображення приймають об'єкт запиту як аргумент. Цей об'єкт запиту містить інформацію про запит користувача. Шаблони URL-адрес записані у файлі urls.py, кожному виразу URL-адреси присвоюється функція (подання Django) з views.py, тому, коли робиться запит, ця функція отримує виклик разом з об'єктом HTTP-запиту. Таке зіставлення надає Django спосіб відповісти на URL-запит і

відобразити відповідь. Представлення визначатиме, який шаблон буде обробляти відповідь.

Відображення поділяються на такі типи:

— Відображення на основі функцій (Function-Based Views)

— Відображення на основі класів (Class-Based Views)

Відображення на основі функцій (FBV) є найпростішим типом відображень Django. Вони написані як функції Python, які приймають HTTP-запит на вхід і повертають HTTP-відповідь. FBV підходять для простих завдань, таких як рендеринг статичного контенту або проста обробка даних. Їх легко писати і розуміти, але для більш складних завдань вони можуть стати менш надійними.

Відображення на основі класів (CBV) - це набір вбудованих відображень, які використовуються для реалізації вибіркових стратегій перегляду, таких як get, post, put та delete. Замість того, щоб визначати функцію відображення, створюється клас, який наслідує один з вбудованих класів, наданих Django. Ці класи надають набір методів, які можна замістити, щоб налаштувати поведінку відображень.

ORM-моделі:

Django надає потужний та інтуїтивно зрозумілий ORM, який спрощує процес маніпулювання базами даних та полегшує створення додатків, що базуються на них. Django ORM працює шляхом перетворення класів Python в таблиці бази даних і об'єктів Python в записи бази даних.

Він надає високорівневий API, який абстрагується від складності SQL і дозволяє розробникам виконувати звичайні операції з базами даних, такі як створення, читання, оновлення та видалення записів, використовуючи методи та атрибути Python.

ORM використовує концепцію моделей, які є класами Python, що визначають структуру та поведінку таблиць бази даних. Кожен клас моделі представляє таблицю в базі даних, а його атрибути відповідають стовпчикам таблиці. Визначаючи моделі, ви можете створювати, запитувати та маніпулювати записами бази даних без написання SQL-запитів у явному вигляді (Рис. 1.2).

Основна перевага ORM - це швидка розробка. Використовуючи ORM, легше змінювати базу даних. У минулому веб-розробники також повинні були мати знання про бази даних. Мови програмування, що використовуються для веб-розробки, використовують класи та об'єкти для інтерпретації даних. Клас використовується для певної структури даних у веб-додатках. Потім така ж схема даних створюється в базі даних. Це завдання вимагає навичок і знання мови SQL.

Знання SQL також недостатньо, оскільки реалізації SQL відрізняються одна від одної в різних базах даних. Це стало складним і трудомістким завданням. Для її вирішення у веб-фреймворках було введено поняття ORM. ORM автоматично створюють схему бази даних з визначених класів/моделей. Вони генерують SQL з коду на Python для конкретної бази даних. ORM дозволяють розробнику будувати проект на одній мові, тобто на Python.

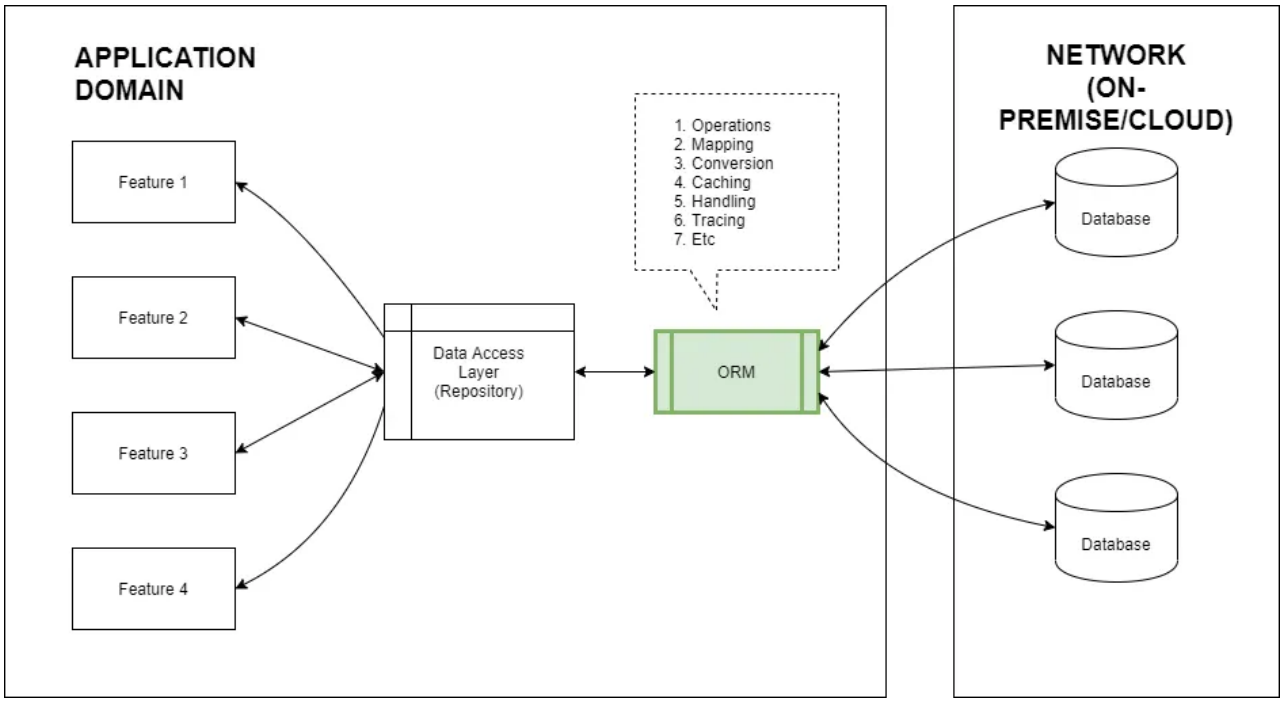


Рис. 1.2 Схема роботи ORM [5]

Шаблони HTML:

Шаблон складається зі статичних частин бажаного HTML-виводу, а також деякого спеціального синтаксису, що описує, як буде вставлено динамічний вміст. Щоб відокремити дизайн від Python-коду, використовується движок шаблонів Django . Він надає вбудовані теги і фільтри, які полегшують виконання типових повторюваних завдань, таких як форматування дат або генерація URL в додатках

Django. Шаблонний тег - це синтаксис, який дозволяє розробникам забезпечити необхідну логіку в процесі рендерингу.

1.5 Особливості використання технології AJAX

AJAX - це підхід до розробки веб-інтерфейсу користувача, який передбачає фоновий обмін даними між веб-браузером і ресурсами сервера. Він дозволяє оновлювати певні частини веб-сторінки асинхронно, тобто користувач може продовжувати взаємодіяти зі сторінкою, поки відбувається обмін даними з сервером у фоновому режимі. Це допомагає створити більш плавний і динамічний взаємодію з користувачем на веб-сайтах.

Технологія AJAX має такі переваги:

— Динамічне завантаження контенту – AJAX використовується для додавання динамічного контенту на веб-сторінки. Наприклад, він може впоратися з ситуацією, коли користувачу потрібно заповнити форму і просто оновити результат після відправлення форми.

— Відправлення та отримання даних у фоновому режимі – AJAX дозволяє сторінці взаємодіяти з сервером у фоновому режимі. Це дозволяє надсилати та отримувати дані непомітно для користувача. Коли користувач виконує якусь дію, дані можуть обмінюватися з сервером без перезавантаження сторінки.

— Швидкість і продуктивність – Збільшує швидкість роботи веб-додатків, обмінюючись даними без перезавантаження сторінки і дозволяючи користувачам отримувати швидші відповіді. Це дає помітну різницю, особливо в додатках, що працюють з великими обсягами даних.

— Перевірка форм і помилок – Коли користувач відправляє форму, запит на перевірку може бути відправлений на сервер через AJAX. Сервер перевіряє дані у формі і повертає тільки помилки або повідомлення про успішну транзакцію. Це дозволяє користувачеві бачити помилки безпосередньо у формі; сторінка не перезавантажується.

AJAX змінює традиційну робочу модель, за допомогою обміну мінімальною кількістю даних між веб-браузером і сервером. Це прискорює роботу веб-додатків. Він забезпечує відчуття, подібне до настільного комп'ютера, передаючи дані на веб-сторінках або дозволяючи відображати дані всередині існуючого веб-додатку. Він створює додатковий рівень, відомий як AJAX-движок, між веб-додатком і веб-сервером, завдяки якому можна здійснювати фонові виклики сервера за допомогою JavaScript і отримувати необхідні дані та оновлювати запитувану частину веб-сторінки, не викликаючи повного перезавантаження сторінки.

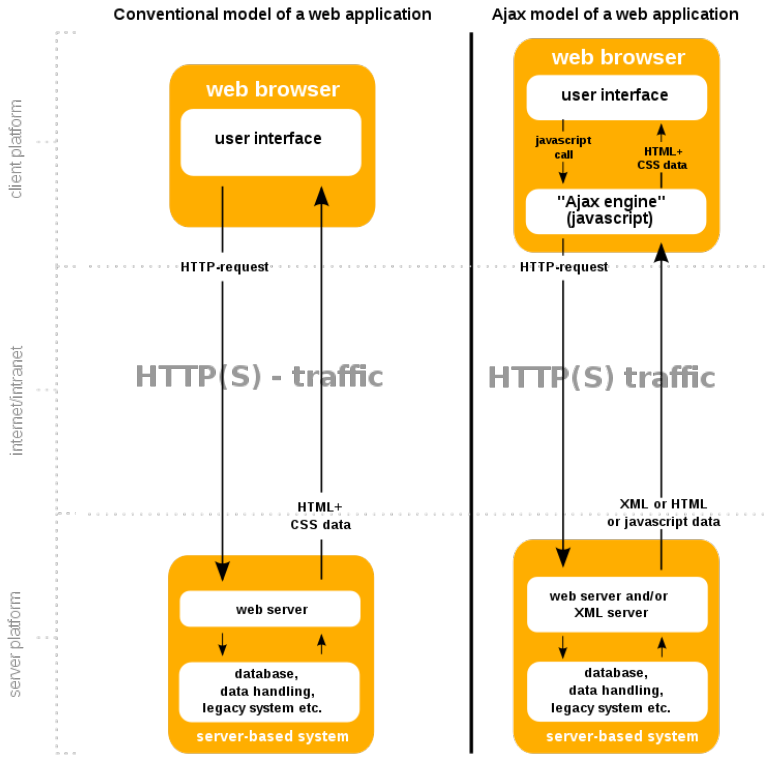


Рис. 1.3 Схема роботи Ajax[8]

AJAX поєднує в собі декілька веб-технологій, таких як JavaScript, DOM, JSON та XMLHttpRequest.

XMLHttpRequest (XHR) - це API, який використовується JavaScript для передачі і маніпулювання даними між клієнтом і сервером за допомогою HTTP. Цей об'єкт xhr містить різноманітні методи та утиліти, за допомогою яких ми можемо здійснювати AJAX-виклики. Він підтримує такі методи, як GET і POST для отримання даних або відправки даних на сервер. За допомогою об'єкта

XMLHttpRequest веб-додатки можуть отримувати дані з сервера, відправляти дані на сервер і динамічно оновлювати частини веб-сторінки, не перериваючи роботу користувача. Це вбудована функція в JS. Немає необхідності імпортувати будь-який фреймворк або бібліотеку.

JavaScript - це легка, кросплатформенна, однопотокова, інтерпретована компільована мова програмування. Вона добре відома завдяки розробці веб-сторінок, і багато небраузерних середовищ також використовують її. JavaScript є слаботипізованою мовою (динамічно типізованою). JavaScript можна використовувати як для клієнтських, так і для серверних розробок. JavaScript є як імперативною, так і декларативною мовою. JavaScript вважається легкою завдяки тому, що вона не використовує забагато ресурсів процесору, проста в реалізації, має мінімалістичний синтаксис і не має типів даних.

JSON або JavaScript Object Notation - це формат обміну даними, який відповідає синтаксису об'єктів JavaScript. JSON використовується для транспортування та відновлення даних. Формат JSON часто використовується для обміну даними між сервером і веб-сторінкою або для рендерингу в браузері. Незважаючи на те, що формат JSON створений на основі структури об'єктів JavaScript, він є виключно текстовим і може використовуватися незалежно від JavaScript. Json також використовується для серіалізації та десеріалізації типів даних в різних додатках, які створюються за допомогою різних мов програмування і працюють в дуже різних середовищах. JSON є популярним форматом для цього сценарію, оскільки він зручний для читання і його легко обробляти іншими системами.

DOM - це інтерфейс між HTML і JavaScript кодом, який перетворює структуру і вміст HTML-документа в об'єктну модель. DOM впорядковує елементи, з яких складається HTML-файл, і представляє їх у вигляді даних у деревовидній структурі, що забезпечує більш чітку навігацію і спрощує пошук окремих об'єктів. Дерево DOM - це ієрархічне представлення даних, в якому один елемент даних виступає в ролі кореня, а кожен вузол під ним з'єднаний лише з одним іншим вузлом. Ця структура містить пов'язані між собою елементи

сторінки (наприклад, заголовки, посилання, зображення). DOM пропонує веб-розробникам можливість створювати динамічний та інтерактивний контент для взаємодії з користувачем. Коли користувач натискає кнопку або надсилає форму, вміст сторінки може динамічно оновлюватися за допомогою JavaScript DOM. DOM лежить в основі сучасних веб-сайтів і додатків.

1.6 Особливості фреймворку Bootstrap

Bootstrap - це популярний фронтенд-фреймворк з відкритим вихідним кодом, створений для спрощення веб-дизайну. Він надає готові інструменти та рішення. Він містить набір синтаксису для дизайну шаблонів, покликаний полегшити процес розробки адаптивних веб-сайтів. Bootstrap створили в середині 2010 року Марк Отто, старший дизайнер в Twitter, і Джейкоб Торнтон, розробник в Twitter. Марк і Джейкоб розробляли внутрішні інструменти для підтримки узгодженості та підвищення ефективності веб-розробки в компанії. Їм спало на думку, що ці інструменти можуть бути корисними для ширшої спільноти веб-розробників, і в 2011 році народився Bootstrap. Відтоді Bootstrap став найпоширенішим CSS-фреймворком і другою за популярністю бібліотекою JavaScript в Інтернеті. В основі роботи Bootstrap лежать принципи адаптивного веб-дизайну.

Адаптивний веб-дизайн (АВД) - це підхід до веб-розробки, який забезпечує динамічну зміну зовнішнього вигляду веб-сайту залежно від розміру екрану та орієнтації пристрою, з якого він переглядається. АВД - це один з підходів до проблеми дизайну для безлічі пристроїв, доступних клієнтам, починаючи від крихітних телефонів і закінчуючи величезними настільними моніторами. Однаковий HTML-код відправляється на всі пристрої, а CSS змінює макет веб-сторінки відповідним чином. Замість того, щоб створювати окремий веб-сайт і відповідний код для девайсів усіх розмірів, єдина кодова база може обслуговувати користувачів з різними розмірами екранів. Елементи сторінки з адаптивним дизайном змінюються при збільшенні або зменшенні масштабу вікна перегляду. Адаптивний дизайн покладається на сітки на основі пропорцій для перестановки

вмісту та елементів дизайну. АВД має потенційні переваги над розробкою окремих сайтів для різних типів пристроїв. Використання єдиної кодової бази може пришвидшити розробку порівняно з розробкою 3 або 4 окремих сайтів, а також полегшити підтримку з часом, оскільки оновлювати потрібно один набір коду та контенту, а не 3 або 4. Якщо на ринку з'являється 5-дюймовий або 15-дюймовий пристрій, код може підтримувати нові пристрої. АВД не прив'язує дизайн до конкретного пристрою.

Компоненти Bootstrap включають:

— Адаптивна сітка: Бібліотека Bootstrap має потужну систему сітки, що є її головною перевагою і однією з найбільших переваг для веб-розробників. Ці сітки мають класи xs, sm, md і lg, кожен з яких визначається синхронно з роздільною здатністю пристрою. Все, що потрібно зробити розробнику - це застосувати ці класи, визначаючи видимість елемента в розмітці.

— Адаптивні зображення: Bootstrap має адаптивний CSS, що підлаштовується під настільні комп'ютери, телефони та планшети. Раніше зміна розміру зображень залежно від того, на якому пристрої їх переглядає користувач, була досить втомлюючою процедурою. Bootstrap автоматично змінює розмір зображень, коли до них додається клас "img-responsive", без потреби перемикатися між кодом і програмним забезпеченням для дизайну. Інші додаткові класи включають "img-circle" та "img-rounded".

— Попередньо розроблені компоненти: Однією з найзручніших переваг готової функціональності Bootstrap є широкий вибір компонентів, які можна додати на свою веб-сторінку. Для будь-якого стандартного елемента, який вам потрібен, ви можете перейти до їхньої бібліотеки і вибрати такі елементи, як випадаючі списки, панелі навігації, індикатори прогресу і мініатюри.

— JavaScript: JavaScript у Bootstrap, разом зі стандартною Java, пропонує безліч користувацьких плагінів, що дає більше можливостей для

створення інтерактивних елементів, таких як модальні спливаючі вікна, переходи та каруселі зображень. Маючи досвід роботи з JavaScript, можна використовувати його для інтеграції власних плагінів, що дасть вам змогу створювати надзвичайно інтерактивні веб-сайти.

— Документація: Доступна, зрозуміла документація є досить простою для початківців. На сайті Bootstrap описано і пояснено кожен фрагмент коду, включаючи приклади коду для базової реалізації. Він є послідовним і всеосяжним і пропонує багато швидких способів, вимагаючи лише базового розуміння HTML і CSS.

2 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ ВЕБ-ЗАСТОСУНКІВ НА ОСНОВІ DJANGO

2.1 Аналіз функціонування додатків Django

У Django додаток (app) - це автономний модуль або компонент веб-застосунку, який інкапсулює певну функціональність або набір пов'язаних функцій. Django заохочує розробників структурувати свої проекти, розбиваючи їх на декілька додатків, кожен з яких відповідає за певний аспект програми. Додатки Django призначені для багаторазового використання, що полегшує розробку складних веб-застосунків за допомогою модульного коду, який легко підтримується.

Ключові аспекти розробки додатків Django включають:

Модульність: Додатки на Django повинні бути високомодульними. Кожен додаток повинен мати певну мету або функціональність, наприклад, автентифікацію користувачів, управління блогом або каталогом продуктів. Модульність дозволяє логічно організувати код, уникнути повторень і полегшити тестування та налагодження.

Незалежність: Додатки Django є самодостатніми і не повинні мати сильної залежності від інших додатків в межах одного проекту. Таке розділення дозволяє вам підключати додатки до різних проектів за потреби.

Багаторазове використання: Після розробки додатку для одного проекту, ви можете повторно використовувати його в інших проектах. Це особливо корисно для поширених функцій, таких як автентифікація користувачів або система блогів.

2.2 Принципи проєктування баз даних

База даних (БД) - це впорядкована колекція структурованих даних, доступних та збережених в електронному вигляді. Вона створена для того, щоб швидко знаходити, додавати або видаляти великі об'єми даних. Бази даних

використовуються для широкого спектру послуг і додатків, таких як зберігання, управління та аналіз даних.

Реляційна база даних - це система зберігання даних, яка використовує реляційну модель. Ця модель базується на принципах організації даних у таблиці з рядками та стовпцями, де кожна частина даних зв'язана з іншими даними за допомогою ключів, створюючи мережу взаємопов'язаної інформації. Основне призначення цих баз даних - ефективно зберігати та організовувати структуровані дані.

Основними компонентами реляційної бази даних є:

— Таблиці: Таблиця - це набір пов'язаних даних, організованих у рядки та стовпці. Таблиці використовуються для зберігання даних у РБД.

— Поля: Поля - це окремі фрагменти даних у таблиці. Кожне поле містить певний тип даних, наприклад, текстовий рядок, числове значення або дату.

— Записи: Записи - це окремі рядки в таблиці, які містять пов'язані між собою дані. Кожен запис відповідає одному елементу або екземпляру даних, наприклад, конкретному клієнту, продукту або транзакції.

— Первинний ключ: Первинний ключ (Primary Key) - це унікальний ідентифікатор рядків у таблиці. Первинний ключ гарантує, що в таблиці немає дублікатів або відсутніх значень, і що до кожного запису можна легко отримати доступ і посилатися на нього в інших таблицях або запитах. Він використовується для забезпечення цілісності посилань і гарантує, що кожен запис може бути однозначно ідентифікований і знайдений. Первинний ключ також дозволяє створювати зовнішні ключі, тобто стовпці в інших таблицях, які посилаються на первинний ключ пов'язаної таблиці.

— Зовнішній ключ: Зовнішній ключ (Foreign Key) - це поле в одній таблиці, яке посилається на первинний ключ іншої таблиці. Зовнішній ключ забезпечує зв'язок між даними у двох таблицях. Зовнішній ключ встановлює зв'язок між даними у двох таблицях.

— Запити: Запит - це звернення до БД з метою отримання даних. Запити використовуються для пошуку та отримання певних даних з БД.

— Індекси: Індекс - це структура бази даних, яка допомагає прискорити виконання запитів, дозволяючи базі даних швидко знаходити певні записи.

— Відображення: Відображення - це запит, який при виконанні створює віртуальну таблицю. Використовуючи відображення , можна зберігати SQL-запити у вигляді шаблонів і отримувати дані з різних таблиць так, ніби вони надходять з однієї таблиці. Відображення забезпечують спосіб подання підмножини даних з БД у певному форматі.

Шаблон проектування описує загальний підхід до типових проблем проектування, з якими стикаються дизайнери або розробники при розробці коду схожого стилю в різних проектах або компонентах проекту.

Шаблони проектування для реляційних баз даних включають:

Наслідування з одною таблицею (Single Table Inheritance, STI) - це шаблон проектування, де одна таблиця бази даних використовується для зберігання декількох типів пов'язаних об'єктів, які мають спільні атрибути. STI - це техніка, яка використовується для представлення ієрархії класів в одній таблиці за допомогою стовпця, який вказує на тип кожного рядка.

Переваги наслідування з одною таблицею:

— Просто і легко в реалізації.

— Підтримує поліморфізм шляхом простої зміни типу рядка.

— Забезпечує швидкий доступ до даних, оскільки дані знаходяться в одній таблиці.

Недоліки наслідування з одною таблицею:

— Збільшує залежність в ієрархії класів, оскільки всі класи безпосередньо пов'язані з однією таблицею.

— Може призвести до зайвого простору в базі даних, якщо є багато нульових стовпців для атрибутів, які не є спільними для всіх підкласів.

— Може ускладнити логіку вказівки типу, якщо є значний перетин між підкласами.

Наслідування з таблицею для кожного класу (Class Table Inheritance, CTI) - це шаблон проектування, де кожен клас в ієрархії має власну таблицю в базі даних, а таблиці пов'язані зовнішніми ключами. CTI - це техніка, яка використовується для представлення ієрархії класів у декількох таблицях, використовуючи відносини успадкування між таблицями.

Переваги наслідування з таблицею для кожного класу:

— Зберігає цілісність та узгодженість даних завдяки використанню зовнішніх ключів та обмежень.

— Дозволяє уникнути зайвого простору в базі даних, зберігаючи лише необхідні атрибути для кожного підкласу.

— Дозволяє легко додавати нові підкласи шляхом створення нових таблиць.

Недоліки наслідування з таблицею для кожного класу:

— Ускладнює доступ до даних та маніпуляції з ними, вимагаючи об'єднання або злиття між таблицями.

— Знижує продуктивність і масштабованість через збільшення кількості запитів і об'єднань.

Модель сутність-атрибут-значення (Entity-attribute-value, EAV) - це шаблон проектування, в якому кожна сутність представлена набором пар «атрибут-значення», замість того, щоб мати фіксовану схему з наперед визначеними стовпцями. Реляційні бази даних базуються на жорсткій структурі, яка вимагає попереднього визначення атрибутів і типів даних кожної сутності. EAV - це метод, який використовується для гнучкого представлення сутностей з динамічними та змінними атрибутами за допомогою трьох таблиць: одна для сутностей, одна для атрибутів і одна для значень.

Переваги моделі сутність-атрибут-значення:

— Забезпечує гнучку і динамічну структуру, яка може вмістити будь-яку кількість і тип атрибутів для кожної сутності.

— Дозволяє легко додавати нові атрибути, вставляючи нові рядки в таблицю атрибутів.

— Підтримує розріджені дані, зберігаючи лише релевантні атрибути для кожної сутності.

Недоліки моделі сутність-атрибут-значення:

— Порушує нормальну форму і реляційну модель, зберігаючи дані в нетабличному форматі.

— Ускладнює доступ до даних і маніпуляції з ними, вимагаючи складних запитів і об'єднань між таблицями.

— Знижує продуктивність і масштабованість через збільшення розміру та кількості таблиць і індексів.

— Ускладнює перевірку та цілісність даних, оскільки зберігає значення у вигляді рядків без типів даних або обмежень.

Як і у випадку зі стовпцями в таблиці, поля в моделях Django відповідають за атрибути або властивості моделі. Конкретний тип кожного поля визначає, які дані воно може зберігати. Кожний тип полів має свій власний набір параметрів і поведінок.

Нижче наведено деякі з найпоширеніших типів полів:

— CharField: У цьому полі зберігаються символи або текстові рядки короткої та середньої довжини.

— IntegerField: Це поле використовується для зберігання цілих чисел.

— FloatField: Це поле використовується для зберігання чисел з плаваючою комою.

— BooleanField: Це поле використовується для зберігання логічних значень (тобто, True або False).

— Поле дати: Це поле використовується для зберігання дат.

— DateTimeField: Це поле використовується для зберігання дати і часу.

— TimeField: Це поле використовується для зберігання часу.

— TextField: Поле моделі TextField використовується для рядків великого розміру, наприклад, довгих абзаців. На відміну від CharField, максимальна довжина для рядків не задається.
— EmailField: EmailField - це спеціалізоване поле CharField, яке використовується для зберігання та перевірки адрес електронної пошти. Воно має вбудований механізм валідації, який гарантує, що збережені значення є дійсними адресами електронної пошти.
— URLField: Це поле використовується для зберігання URL-адрес.
— ForeignKey: Це поле використовується для визначення зв'язку «багато-до-одного» з іншою моделлю.
— ManyToManyField: Це поле використовується для визначення зв'язку «багато-до-багатьох» з іншою моделлю.
— OneToOneField: Це поле використовується для визначення зв'язку один-до-одного з іншою моделлю.
— FileField: Це поле використовується для завантаження файлів на сервер.
— ImageField: Це поле використовується для завантаження файлів зображень на сервер.

На додаток до цих типів полів, Django також надає поле вибору (ChoiceField), яке використовується для обмеження вибору, доступного для поля, фіксованим набором значень. Наприклад, поле вибору може бути використано для надання користувачеві випадаючого списку опцій для вибору при створенні або редагуванні записів.

2.3 Особливості роботи шаблонів Django

Шаблони Django - це HTML-файли, що визначають структуру та вміст веб-додатку. Вони виступають в якості візуального шару додатку, дозволяючи створювати веб-сторінки з динамічними елементами. Ці шаблони зазвичай використовуються з відображеннями та моделями Django для створення повноцінних веб-додатків.

Вони використовують мову шаблонів, яка дозволяє розробникам вставляти в HTML-сторінку динамічні дані, такі як змінні, цикли та умовні оператори. Система шаблонів Django надає вбудовані теги та фільтри, які спрощують виконання типових повторюваних завдань, таких як форматування дат або генерація URL-адрес у додатках Django. Шаблонний тег - це синтаксис, який дозволяє розробникам забезпечити довільну логіку в процесі рендерингу.

Шаблонні теги позначаються фігурними дужками зі знаками відсотків, наприклад, {% tag %}.Деякі завдання, що виконуються шаблонними тегами, включають: виведення контенту, отримання контенту з баз даних та інших носіїв інформації, виконання керуючих операцій, таких як оператори if та цикли for, генерування URL-адрес, форматування дат та інші складні функції, створені користувачем.

У Django є багато вбудованих шаблонних тегів, і нижче наведено деякі з найпоширеніших:

— Тег {% if %} дозволяє розробникам додавати до шаблонів умовну логіку. Він перевіряє, чи змінна має значення True, False, None або чи існує вона взагалі.

— Тег {% for %} дозволяє розробнику циклічно переглядати список елементів, наприклад, набір запитів з бази даних або список об'єктів, переданих з відображення.

— Тег {% include %} дозволяє включити один шаблон в інший.

— Тег {% extends %} дає команду Django успадкувати шаблон від іншого і перевизначити певні блоки вмісту.

— Тег {% load %} дозволяє розробникам завантажувати власні шаблонні теги, можливо, з іншої бібліотеки, подібно до використання імпорту в Python.

— Тег {% block %} визначає блок вмісту, який функція може перевизначити у дочірньому шаблоні.

— Тег {% csrf_token %} є важливим у Django для обробки форм. Він генерує приховане поле введення, що містить CSRF-токен, необхідний для безпечного надсилання форм.

— Тег {% url %} генерує абсолютне посилання на шлях (URL без доменного імені) на основі назви відображення або шаблону URL.

Окрім шаблонних тегів, у Django є також теги змінних. Теги змінних позначаються таким чином: {{ variable }} і використовуються для передачі виразу або змінної, визначеної з функції перегляду. Вони діють як посилання на значення або об'єкти, що передаються з серверного коду в движок шаблонів для рендерингу. Вони можуть представляти широкий спектр даних, включаючи рядки, числа, списки, словники і навіть складні об'єкти. Вони дозволяють відображати динамічний контент, отримуючи та відображаючи дані, що зберігаються у змінних.

Шаблонні теги та теги змінних можуть надавати додаткову функціональність при використанні фільтрів. Шаблонні фільтри Django використовуються для модифікації значень змінної або аргументів тегів, а також для їх перетворення і доповнення за необхідності. Шаблонні фільтри можна використовувати для форматування чисел, дат та інших типів даних або для виконання загальних операцій, таких як об'єднання або усікання рядків. Шаблонні фільтри застосовуються за допомогою вертикальної лінії всередині подвійних дужок {{ variable|filter }}. Фільтри також можна з'єднати в ланцюг, щоб виконати серію послідовних перетворень даних. Наприклад: {{ variable|filter_1|filter_2 }}.

Django постачається з великою кількістю вбудованих фільтрів шаблонів, які ви можете легко застосувати до вашого проекту:

Фільтр об'єднання:

Фільтр об'єднання використовується для об'єднання елементів списку з рядком і виводить рядок. Він працює подібно до функції str.join(list) у Python, яка повертає рядок, створений шляхом об'єднання елементів списку за допомогою заданого роздільника.

Фільтр дати:

Це фільтр використовується для форматування дати. Синтаксис для використання фільтра дати наступний: {{ value | date: «D d M Y» }}. Якщо значення є об'єктом дати (наприклад, результат функції datetime.datetime.now()) або 2023-11-12T10:30:00.000123, на виході буде отримано рядок «Mon 12 Dec 2022». Можна також використовувати один із попередньо визначених форматів дати, таких як DATE_FORMAT, DATETIME_FORMAT, SHORT_DATE_FORMAT або SHORT_DATETIME_FORMAT, або користувацький формат, який використовує специфікатори формату, наприклад: {{ value|date: «SHORT_DATE_FORMAT» }}. На виході буде отримано рядок «12/11/2023».

Фільтр за замовчуванням:

Фільтр за замовчуванням використовується для встановлення значення за замовчуванням для змінної. Якщо змінна отримує значення False, буде використано аргумент за замовчуванням.

Синтаксис для використання фільтра за замовчуванням виглядає наступним чином: {{ value|default: "тут нічого немає" }}. Якщо значенням є порожній рядок, буде виведено «тут нічого немає».

Фільтр додавання:

Фільтр додавання використовується для додавання аргументу до значення. Це корисно для числових даних. Синтаксис для використання фільтра додавання такий: {{ value|add: "5" }}. Якщо значенням є 10, то на виході буде 15. Цей фільтр використовується для збільшення змінної у шаблонах Django шляхом додавання до змінної константного значення.

Фільтр вирізання:

Фільтр вирізання використовується для видалення всіх заданих аргументів з даного рядка. Аргумент, який передається у фільтр, буде вилучено зі результату. Синтаксис для використання фільтра з відсіканням наступний: {{ value|cut: " " }}. Якщо значенням є «String with spaces», результатом буде «Stringwithspaces».

2.4 Обробка платежів з використанням Stripe

Знання про обробку онлайн-платежів допомагають вибрати найкращий платіжний шлюз, процесор та торговий рахунок для бізнесу, а також краще зрозуміти те, навіщо потрібні рішення для захисту від шахрайства та повернень платежів.

Онлайн-платежі включають два основні етапи:

Авторизація картки:

Це єдиний етап, у якому безпосередньо бере участь клієнт. Клієнт надає дані своєї кредитної картки, заповнюючи форму онлайн-платежу. Форма зазвичай запитує ім'я на картці, тип картки, номер картки, CVV-код, термін дії та адресу власника картки. Після натискання кнопки "Відправити" інформація передається платіжним шлюзом емітенту кредитної картки для перевірки. Під час цього процесу перевіряється наявність достатніх коштів на картці покупця. Платіжний шлюз повертає повідомлення клієнту з підтвердженням або відхиленням покупки.

Відхилення покупки може бути пов'язано з тим, що клієнт помилився при введенні даних, або термін дії картки закінчився, або у власника картки недостатньо коштів для оплати покупки. У такому випадку покупка може вважатися підозрілою, і платіжний шлюз позначить її як потенційний ризик шахрайства. Якщо покупку схвалено, платіжний шлюз/сайт надсилає власнику картки автоматичне підтвердження з деталями покупки.

Урегулювання транзакції:

Це етап, під час якого обробляється покупка і зараховуються кошти на ваш рахунок. Платіжний шлюз виконав перший крок, а тепер ваш платіжна система виконує розрахунки за транзакцією.

Платіжна система надсилає дані про транзакцію до банку. (Залежно від вашої системи, це може відбуватися для кожної окремої покупки або в пакетному режимі в кінці робочого дня). Коли банк отримує повідомлення про транзакцію, він надсилає його до відповідної компанії-емітента кредитної картки (Visa, MasterCard, Discover тощо) і чекає на друге підтвердження від банку власника

картки. Після цього емітент надсилає вашому банку кредит, який банк використовує для зарахування платежу на ваш торговельний рахунок. Нарешті, з власника картки стягується плата, яка відображається у виписці з його кредитної картки.

Для обробки онлайн-платежів використовуються два ключові рішення: платіжний шлюз і платіжна система.

Платіжний шлюз дозволяє приймати онлайн та офлайн платежі, списуючи кошти з кредитної, дебетової чи подарункової картки клієнта, а також з його поточного рахунку. Цей платіжний шлюз невидимий для клієнтів і працює за лаштунками для обробки та перевірки платежів. Використовуючи безпечні методи зв'язку та токенізацію, платіжні шлюзи забезпечують зв'язок між інтернет-магазином і банком. Дані клієнта збираються, перевіряються, затверджуються, а потім платіж приймається і списується з рахунку клієнта.

Платіжна система з'єднує платіжний шлюз та торговий рахунок. Платіжна система відповідає за розрахунки за транзакціями, в підсумку гарантуючи, отримання оплати за покупку власника картки. Зазвичай платіжні системи працюють у партнерстві з платіжними шлюзами, щоб забезпечити безперебійну обробку транзакцій.

Stripe - це платформа для обробки платежів, яка надає підприємцям можливість приймати платежі від клієнтів через свій веб-сайт, платформу електронної комерції або додаток. За допомогою Stripe розробники можуть легко додати платіжну обробку до своїх додатків, написавши лише кілька рядків коду. Такі онлайн-платформи, як Shopify, Instacart та Flexport, використовують Stripe як основний платіжний процесор. Stripe працює як посередник між платіжними методами клієнтів і банківським рахунком бізнесу. Це гнучка платформа, яка підтримує безшовну інтеграцію без коду з будь-яким веб-сайтом або платформою, а також дає можливість створити власну точку продажу.

Stripe підтримує понад 100 способів оплати по всьому світу, включаючи:

— Кредитні картки, такі як Mastercard, Visa та Discover.

— Банківські перекази та попередньо авторизовані списання в доларах США, канадських доларах та багатьох інших валютах.
— Платформи типу "купи зараз, плати пізніше", такі як Affirm та Afterpay.
— Мобільні платіжні сервіси, такі як Apple Pay та Google Pay.
— Платформи онлайн-платежів, такі як PayPal.

2.5 Тестування застосунків з unittest

Тестування відіграє важливу роль у розробці програмного забезпечення, забезпечуючи якість та правильне функціонування додатків. Воно систематично оцінює програмні компоненти, модулі або системи для виявлення помилок, багів або відхилень від бажаної поведінки. Тестування допомагає зменшити ризики, підвищити надійність та покращити користувацький досвід. Основна мета тестування - виявити дефекти та розбіжності між очікуваними та фактичними результатами. Виконуючи тестові кейси, розробники перевіряють правильність, повноту та надійність програмного забезпечення. Тестування також допомагає виявити низьку продуктивність, вразливі місця в системі безпеки та проблеми сумісності.

У процесі розробки програмного забезпечення зазвичай використовуються різні типи тестів, зокрема:

— Модульні тести: Ці тести зосереджені на тестуванні окремих блоків або компонентів в умовах ізоляції.
— Інтеграційні тести: Ці тести перевіряють взаємодію та інтеграцію між декількома модулями.
— Системні тести: Ці тести оцінюють загальну функціональність і продуктивність повністю інтегрованої системи. Системне тестування виявляє проблеми в інтегрованих блоках системи та перевіряє дизайн всієї системи та поведінку відповідно до потреб кінцевого користувача.
— Приймальні тести: Ці тести перевіряють, чи працює система так, як того вимагає кінцевий користувач. Це тестування проводиться як різновид

тестування «чорного ящика», де користувачі, які беруть участь у тестуванні, визначають ступінь готовності системи.

Тестування може виконуватися вручну або за допомогою автоматизованих фреймворків та інструментів тестування. Автоматизованому тестуванню надають перевагу в сучасній розробці програмного забезпечення завдяки його ефективності, відтворюваності та масштабованості.
Воно дозволяє створювати набори тестів, які можуть виконуватися автоматично, що дає змогу розробникам виявляти проблеми на ранніх стадіях і оптимізувати процес розробки.

Тестування - це не окремий вид діяльності в розробці програмного забезпечення, а скоріше невід'ємна частина робочого процесу розробки. Воно повинно виконуватися безперервно протягом усього циклу розробки, починаючи з ранніх етапів збору вимог і проектування, через реалізацію і аж до розгортання та обслуговування.

Ітеративний підхід до тестування забезпечує своєчасне виявлення та виправлення дефектів, зменшуючи витрати та зусилля, пов'язані з переробкою та виправленням помилок.

Для організації та виконання тестів розробники використовують фреймворки модульного тестування, наприклад, unittest. Модуль unittest - це вбудований фреймворк тестування в Python, доступний як частина стандартної бібліотеки. Основна мета модуля unittest - полегшити створення варіантів тестування і наборів тестів для перевірки поведінки і працездатності окремих одиниць коду. Ці одиниці зазвичай є функціями, методами або класами, і мета модульного тестування - переконатися, що кожна з них поводиться належним чином і видає очікуваний результат. Модуль unittest пропонує підхід на основі класів для визначення варіантів тестування. Розробники створюють клас варіантів тестування, який успадковується від класу unittest.TestCase. У межах цього класу визначаються окремі методи тестування, де кожен метод представляє певний тестовий сценарій. Імена тестових методів починаються з префікса «test_», щоб їх міг виявити фреймворк тестування.

Unittest є кращим вибором для модульного тестування в Python за наступними причинами:

Включення до стандартної бібліотеки: unittest входить до стандартної бібліотеки Python, що робить його легкодоступним без необхідності додаткового встановлення.

Звичність та послідовність: unittest слідує принципам дизайну та шаблонам сімейства фреймворків для тестування xUnit. Розробники, які знайомі з іншими фреймворками xUnit, такими як JUnit в Java або NUnit в .NET, знайдуть unittest простим для розуміння і використання.

Підтримка спільноти та ресурси: unittest має велику та активну спільноту Python розробників, які використовують фреймворк протягом багатьох років. Це забезпечує безліч ресурсів, навчальних посібників, документації та онлайн-форумів, де розробники можуть шукати поради та обмінюватися знаннями.

Сумісність і переносимість: unittest сумісний з різними версіями Python, включаючи Python 2 та Python 3. Ця сумісність особливо цінна для проектів, які мають специфічні вимоги до версій або потребують підтримки сумісності зі старими версіями Python.

Інтеграція з інструментами та екосистемою: unittest без проблем працює з популярними IDE Python, надаючи такі функції, як виявлення тестів, виконання тестів та звітування про результати в середовищі IDE. unittest також підтримується засобами запуску тестів, системами безперервної інтеграції (CI) та інструментами покриття коду, що дозволяє легко інтегрувати модульні тести в робочий процес розробки.

Стабільність та надійність: Будучи основним компонентом Python, unittest використовує колективний досвід і знання основної команди розробників Python, що додатково підвищує його стабільність і надійність.

Визнання індустрією: Багато популярних бібліотек та фреймворків Python, таких як Django, використовують unittest для своїх потреб тестування. Широке визнання і використання unittest в індустрії підвищує довіру до нього і його надійність як фреймворку для модульного тестування.

Покриття коду - це показник того, яка частина коду в програмі виконується під час тестування. Існує декілька типів метрик покриття, зокрема покриття операторів, покриття гілок та покриття шляхів. Покриття тестів показує, наскільки ретельно була протестована програма. Програма з високим тестовим покриттям виконує більшість або всі шляхи коду під час тестування, що зменшує ймовірність помилок і покращує загальну якість програми. Крім того, покриття тестів може допомогти виявити ділянки коду, які можуть потребувати додаткового тестування, або ті, які складно протестувати і які можуть потребувати рефакторингу. Python надає декілька інструментів для вимірювання тестового покриття, включаючи вбудований модуль coverage.

Coverage.py - один з найпопулярніших інструментів покриття коду для Python. Він використовує інструменти аналізу коду, надані в стандартній бібліотеці Python, для вимірювання покриття. Coverage.py можна використовувати як з unittest, так і з Pytest.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

3.1 Підготовка середовища розробки

Перед початком створення веб-додатку Django, потрібно виконати кілька налаштувань:

Створення віртуального середовища для проєкту:

Віртуальне середовище ізолює проекти на Python. Воно дозволяє керувати залежностями та версіями Python для кожного проекту окремо, не впливаючи на інші проекти або системну збірку Python. Таким чином, можна створювати різні проекти з різними версіями бібліотек, і вони не будуть заважати один одному. Django або навіть Python з часом оновлюється, тому існує можливість мати декілька проектів, які використовують різні версії Django або Python. Крім того, для кожного проекту може знадобитися встановлення певних пакетів. Ці пакети також бувають різних версій, а отже, наявність віртуального середовища допоможе ізолювати пакети для використання лише у відповідному проекті.

venv - це вбудований модуль, доступний з Python 3.3 і вище. Це простий і легкий спосіб створення віртуального середовища.

Створити віртуальне середовище за допомогою venv можна так: python -m venv myenv

myenv - це назва віртуального середовища. Ви можете використовувати будь-яку іншу назву, але вона повинна складатися з малих літер і не містити пробілів.

Активація віртуального середовища:

Після створення віртуального середовища його потрібно активувати. Всередині каталогу є файл для активації. Щоб це зробити, виконайте наведену команду: .\venv\Scripts\activate. Одразу після активації у терміналі з'явився новий рядок на початку стандартного запрошення. Назву віртуального середовища буде взято у круглі дужки і розміщено перед рядком запиту вашого терміналу (Рис. 3.1).

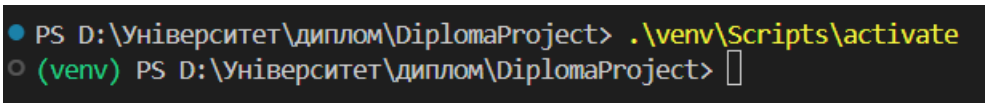


Рис. 3.1 Вигляд терміналу після активації віртуального середовища

Встановлення Django:

Тепер, коли ми налаштували наше віртуальне середовище, ми можемо перейти до встановлення фреймворку Django. Щоб встановити Django, виконайте команду: pip install django. Або скористайтеся наступною командою, щоб вказати версію: pip install django==4.1.13. pip - це інсталятор пакунків для Python, який рекомендується для встановлення Django.

Створення проєкту Django:

Згідно з документацією Django, проєкт - це пакет Python, тобто каталог коду, який містить всі налаштування для екземпляра Django. Він зазвичай включає конфігурацію бази даних, налаштування Django і конкретних додатків.

Щоб створити проєкт, використайте утиліту командного рядка django-admin. Ця команда згенерує файли, в яких можна налаштувати параметри бази даних, додати сторонні пакети тощо. Проект створюється за допомогою наступної команди: django-admin startproject project.

Де project - це назва проєкту, який ви створюєте. Ця дія створить новий каталог з назвою project. Усередині цього каталогу знаходиться файл з назвою manage.py. Цей файл є дуже важливим і буде використовуватися для виконання різних адміністративних завдань, таких як ініціалізація баз даних, створення баз даних, створення суперкористувача тощо. Усередині цього каталогу ви також знайдете підкаталог з такою ж назвою а, всередині підкаталогу знаходиться близько 5 різних файлів.

Запуск додатку Django:

Кожен проект, який ви створюєте за допомогою Django, може містити декілька додатків. Під час виконання команди startproject у попередньому розділі було створено програму керування, яка знадобиться для кожного проєкту. Тепер ми створимо додаток Django, який міститиме специфічну функціональність нашого веб-застосунку.

Тепер не потрібно використовувати утиліту командного рядка django-admin, замість неї скористаємося командою startapp з файлу manage.py: python manage.py startapp newapp.

Реєстрація додатку:

Тепер потрібно зареєструвати створений додаток у проєкті. Для цього треба відкрити файл settings.py (який знаходиться в папці проєкту). Тепер перейдемо до блоку INSTALLED_APPS і додамо до нього зверху або знизу назву додатку myapp, а решту коду залишимо без змін (Рис. 3.2).

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

Рис. 3.2 Блок INSTALLED_APPS у файлі settings.py

Створення папок static та templates:

Тепер треба створити дві дуже важливі папки, в яких буде зберігатися більшість наших файлів:

— Папка "static" буде містити статичні файли (наприклад, текстові файли, файли CSS або JavaScript і т.д...).

— Папка "templates" буде містити динамічні файли (наприклад, html-файли, код сайту та ін.).

Тепер треба прокрутити файл settings.py до кінця і вписати в нього наступний код:

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

У тому ж файлі settings.py знаходимо TEMPLATES, у цій секції шукаємо 'DIRS':[], і в порожніх дужках вводимо наступний код:

'DIRS': [os.path.join(BASE_DIR, 'templates')],

Міграція бази даних:

Кожного разу, коли ми створюємо новий проект Django, для нас автоматично налаштовується сайт адміністратора. Однак, для того, щоб він працював, нам необхідно налаштувати базу даних. За замовчуванням Django використовує базу даних sqlite. Нам потрібно привести цю базу даних у робочий стан. Ми зробимо це шляхом міграції бази даних. Для міграції бази даних ми скористаємося файлом manage.py. Щоб скористатися цим файлом, необхідно ввести команду, яка починається з python, потім `manage.py, а потім ім'я команди, яку ви хочете виконати. Команда для міграції виглядає наступним чином:

python manage.py migrate

Повністю налаштований проєкт Django показаний на Рис. 3.1.

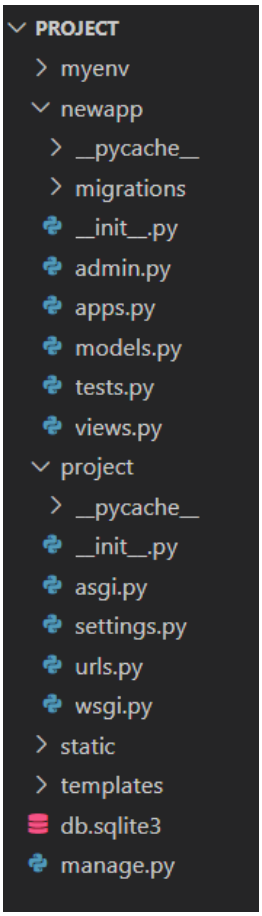


Рис. 3.3 Папка проєкту після початкового налаштування

Створення суперкористувача:

Тепер, коли база даних налаштована, нам потрібно створити суперкористувача, щоб отримати доступ до адміністративної частини сайту. Це

робиться за допомогою команди manage.py. Команда для створення суперкористувача виглядає наступним чином:

python manage.py createsuperuser

Після цього вам буде запропоновано ввести дані для суперкористувача. Ці дані включають: Ім'я користувача, Адреса електронної пошти, Пароль, Пароль (ще раз). Після того, як буде введено всі вищезазначені дані, суперкористувача буде успішно створено.

Запуск сервера розробки:

Ще однією командою manage.py є runserver, яка запускає сервер розробки, щоб ми могли попередньо переглянути проект, над яким працюємо.

Тому, щоб запустити сервер, виконаємо наведену нижче команду:

python manage.py runserver

За замовчуванням ця команда запускає локальний сервер розробки на порту 8080 (Рис. 3.4). Це означає, що тепер можна отримати доступ до веб-сайту за адресою https://127.0.0.0:8000/.

Веб-сайт буде доступний до тих пір, поки працює сервер. Як тільки сервер закривається, доступ до сайту буде втрачено, і сервер доведеться запускати заново. Щоб вийти з сервера, досить натиснути комбінацію клавіш ctrl + c.

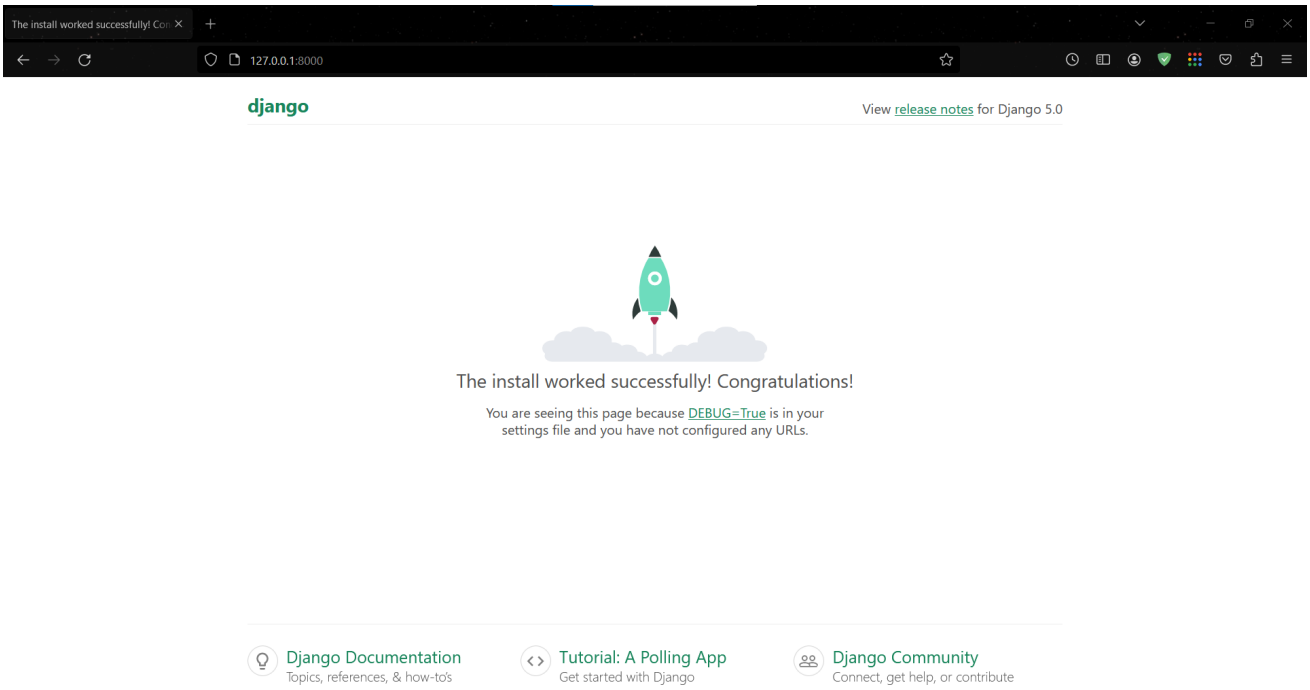


Рис. 3.4 Стандартна сторінка сервера Django

3.2 Розробка моделей

Моделі Django - це класи Python, що визначають структуру та поведінку даних. Кожна модель представляє таблицю в базі даних, а атрибут моделі - поле в таблиці. Модель Django є підкласом django.db.models.Model, і кожне поле відповідає стовпцю бази даних. Django надає абстрактний API бази даних, який дозволяє створювати, отримувати, оновлювати та видаляти записи в таблицях. Модель визначається у файлі models.py. Наприклад, створимо модель Category з полями name і slug (Рис. 3.5 - 3.6).

```python
class Category(models.Model):
    name = models.CharField(max_length=255, db_index=True)
    slug = models.SlugField(max_length=255, unique=True)

    class Meta:
        verbose_name_plural = 'categories'

    def get_absolute_url(self):
        return reverse('store:category_list', args=[self.slug])

    def __str__(self):
        return self.name
```

Рис. 3.5 Модель Category

Клас Meta визначає метадані для моделі. Він дозволяє вказати різні опції, які контролюють поведінку моделі, такі як назва таблиці бази даних, впорядкування результатів запитів, унікальні обмеження тощо. Опції verbose_name і verbose_name_plural дозволяють вказати зрозумілі для користувача імена моделі. verbose_name використовується для одиничних об'єктів, тоді як verbose_name_plural використовується для колекцій об'єктів.

Метод __str__ визначає представлення моделі, яке відображається в інтерфейсі адміністратора Django та в оболонці Django.

Метод get_absolute_url() встановлює канонічну URL-адресу для моделі. Це необхідно при використанні функції reverse(). Це також правильний спосіб посилатися на модель у шаблонах замість того, щоб жорстко кодувати їх.

Name	Type	NN	PK	AI	U	Default	Check	Collation	Foreign Key
id	integer	☑	☑	☑	☐				
name	varchar(255)	☑	☐	☐	☐				
slug	varchar(255)	☑	☐	☐	☑				

Рис. 3.6 Таблиця створена на основі моделі Category

Django автоматично додає поле id як первинний ключ до всіх моделей Django, щоб зробити пошук записів простішим та ефективнішим.

Наступна модель визначає таблицю Product з 14 полями: title, category, brand, added_by, description, image, price, in_stock, is_active, created_at, updated_at, slug, objects, products (Рис. 3.7).

```python
class Product(models.Model):
    title = models.CharField(max_length=255)
    category = models.ForeignKey(Category, related_name='product', on_delete=models.CASCADE)
    brand = models.CharField(max_length=255, default='unspecified')
    added_by = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name='product_creator')
    description = models.TextField(blank=True)
    image = models.ImageField(upload_to='images/', default='images/default.png')
    price = models.DecimalField(max_digits=9, decimal_places=2)
    in_stock = models.BooleanField(default=True)
    is_active = models.BooleanField(default=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    slug = models.SlugField(max_length=255)
    objects = models.Manager()
    products = ProductManager()

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

    def get_absolute_url(self):
        return reverse('store:product_detail', args=[self.slug])

    def __str__(self):
        return self.title
```

Рис. 3.7 Модель Product

Поля title і brand - це CharField з максимальною довжиною 255 символів. Поле brand має значення за замовчуванням "unspecified". Поля category та added_by є зовнішніми ключами, які посилаються на інші моделі. Category, представляє категорію товару та settings.AUTH_USER_MODEL представляє користувача, який додав товар до бази даних. Аргумент on_delete визначає, що відбувається, коли категорія, до якої належить товар, видаляється з бази даних. У цьому випадку ми використовуємо CASCADE, що означає, що всі товари в цій категорії також будуть видалені. Поле description - це текстове поле, яке може зберігати будь- який обсяг тексту. Поле image - це поле, в якому зберігається

зображення товару або зображення за замовчуванням, якщо його не було надано. Поле price - це поле, яке зберігає ціну товару у вигляді десяткового числа з фіксованою точністю. Поля in_stock і is_active - це логічні поля, які зберігають значення True або False. Поля created_at та updated_at зберігають дату і час створення та оновлення товару. Поле slug зберігає коротку назву товару, яка буде використана в його URL-адресі. Воно містить лише цифри, літери, дефіси та підкреслення. Поля objects і products є менеджерами моделей Django.

Менеджер моделей - це клас, який забезпечує інтерфейс для роботи з моделлю. Він використовується для запитів до бази даних. За замовчуванням Django автоматично створює менеджер для кожної моделі під назвою objects, але можна створювати власні менеджери. Наприклад, можна створити менеджер для моделі Product, який повертатиме лише активні продукти (Рис. 3.8).

```python
class ProductManager(models.Manager):
    def get_queryset(self):
        return super(ProductManager, self).get_queryset().filter(is_active=True)
```

Рис. 3.8 Менеджер моделі Product

На рисунку 3.9 показано модель UserBase, яка містить дані користувачів.

```python
class UserBase(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(_('email address'), unique=True)
    username = models.CharField(max_length=150, unique=True)
    firstname = models.CharField(max_length=150, blank=True)
    about = models.TextField(_('about'), max_length=500, blank=True)
    country = CountryField()
    phone_number = models.CharField(max_length=15, blank=True)
    address_line_1 = models.CharField(max_length=150, blank=True)
    address_line_2 = models.CharField(max_length=150, blank=True)
    city = models.CharField(max_length=150, blank=True)
    is_active = models.BooleanField(default=False)
    is_staff = models.BooleanField(default=False)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    objects = CustomAccountManager()
    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']

    class Meta:
        verbose_name = "Accounts"
        verbose_name_plural = "Accounts"

    def email_user(self, subject, message, from_email=None, *args, **kwargs):
        send_mail(subject, message, from_email, [self.email], **kwargs)

    def __str__(self):
        return self.username
```

Рис. 3.9 Модель UserBase

Django має вбудовану систему авторизації з використанням моделі User для керування обліковими записами. Однак, модель User за замовчуванням може не відповідати багатьом вимогам до реальних додатків. Стандартна форма UserCreationForm не підтримує створення різних типів користувачів. Вона призначена для створення екземплярів стандартної моделі користувача, яка зазвичай включає такі поля, як ім'я користувача, електронна пошта та пароль. Припустимо, нам потрібно створити різні типи користувачів з різними атрибутами або ролями у додатку Django. У такому випадку потрібно налаштувати реєстрацію, створивши власні форми та представлення.

Існує два основних способи створення користувацької моделі користувача у Django: AbstractUser та AbstractBaseUser.

Клас AbstractUser має 11 полів. Ви можете додати нові поля, змінити або видалити існуючі. Поля 'username' та 'email' є спеціальними, причому 'username' є унікальним.

Нижче наведено початкові поля класу AbstractUser, які за замовчуванням має клас User:

— id

— password

— last_login

— is_superuser

— username (унікальне значення)

— first_name

— last_name

— email

— is_staff

— is_active

— date_joined

Клас AbstractBaseUser має 3 поля. Ви можете додавати нові поля, змінювати або видаляти існуючі.

Нижче наведено початкові поля класу AbstractBaseUser:

— id

— password

— last_login

AbstractUser - це повна модель користувача, яку можна успадкувати і додати власні поля профілю. Поля, які ви можете додати до моделі користувача, включають дату народження, місцезнаходження. У результаті ви отримаєте всі поля, моделі за замовчуванням, а також ті, що були додані вами.

AbstractBaseUser містить лише функцію автентифікації: решту полів потрібно вказати під час створення підкласу. Ви маєте вказати йому, яке поле буде представляти ім'я користувача, які поля є обов'язковими, і як керувати користувачами.

3.3 Створення форм

Для збору інформації про користувачів, що зберігається в базі даних, ми використовуємо форми Django. Django пропонує різноманітні типи полів для форм. Форми Django дозволяють створювати форми за кілька рядків коду, що є їх основною перевагою.

Django ModelForm автоматично генерується на основі моделі. Вона включає всі поля моделі та використовується для створення, оновлення та видалення екземплярів. Ми можемо створювати ModelForm на основі моделі. Приклад ModelForm, яка базується на моделі Django User представлений на рисунку 3.10.

```python
class RegistrationForm(forms.ModelForm):
    username = forms.CharField(label='Ім`я користувача', min_length=4, max_length=50, help_text='Обов`язкове поле для входу')
    email = forms.EmailField(max_length=100, help_text='Required', error_messages={'required': 'Вибачте, вам потрібен email'})
    repeat_password = forms.CharField(label='Повторити пароль', widget=forms.PasswordInput)
    password = forms.CharField(label='Пароль', widget=forms.PasswordInput)

    class Meta:
        model = UserBase
        fields = ('username', 'email',)

    def clean_username(self):
        username = self.cleaned_data['username'].lower()
        r = UserBase.objects.filter(username=username)
        if r.count():
            raise forms.ValidationError("Це ім'я вже існує")
        return username

    def clean_password(self):
        cd = self.cleaned_data
        if cd['password'] != cd['repeat_password']:
            raise forms.ValidationError("Паролі не співпадають")
        return cd['repeat_password']

    def clean_email(self):
        email = self.cleaned_data['email']
        if UserBase.objects.filter(email=email).exists():
            raise forms.ValidationError('Ця email адреса вже зайнята')
        return email

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['username'].widget.attrs.update({'class': 'form-control mb-3', 'placeholder': 'Ім`я користувача'})
        self.fields['email'].widget.attrs.update({'class': 'form-control mb-3', 'placeholder': 'Email', 'name': 'email', 'id': 'id_email'})
        self.fields['password'].widget.attrs.update({'class': 'form-control mb-3', 'placeholder': 'Пароль'})
        self.fields['repeat_password'].widget.attrs.update({'class': 'form-control', 'placeholder': 'Повторіть пароль'})
```

Рис. 3.10 Форма регістрації користувача

Цей код визначає форму Django з назвою RegistrationForm за допомогою класу forms.ModelForm. Форма має чотири поля: username, email, password і repeat_password. Клас Meta використовується для визначення деяких властивостей форми, таких як модель, яку буде використано для форми і поля, які буде включено до форми. Властивість model класу Meta визначає модель, на якій базується форма, в даному випадку це модель UserBase. Властивість fields класу Meta визначає, які саме поля з моделі слід включити у форму. У цьому випадку ми додаємо поля username та email з моделі UserBase. Використовуючи ModelForm, поля форми автоматично генеруються на основі полів моделі та правил валідації. Це полегшує створення і підтримку форм, які відповідають полям відповідної моделі. Після відправки форми ми можемо перевірити вхідні дані методами clean_username, clean_password і clean_email. Django постачається з вбудованими відображеннями та формами автентифікації, які забезпечують базову функціональність входу та автентифікації для веб-додатків. Однак іноді нам може знадобитися налаштувати процес автентифікації відповідно до наших специфічних вимог, наприклад, використати інший бекенд автентифікації або

додати додаткові поля до форми входу в систему. Першим кроком є створення власної форми автентифікації, яка успадковує вбудовану форму AuthenticationForm Django. Потім ми можемо додати до форми власні поля і валідацію. Приклад форми автентифікації зображений на рисунку 3.11.

```
class UserLoginForm(AuthenticationForm):
    username = forms.CharField(widget=forms.TextInput(attrs={'class': 'form-control mb-3', 'placeholder': 'Email', 'id': 'login-username'}))
    password = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-control', 'placeholder': 'Пароль', 'id': 'login-pwd'}))
```

Рис. 3.11 Форма автентифікації користувача

Функція скидання пароля є поширеною опцією у багатьох застосунках, що дозволяє користувачам легко відновлювати свої паролі, якщо вони їх забули. Ми скористаємося вбудованим класом автентифікації Django для обробки процесу скидання пароля і налаштуємо стандартні відображення автентифікації відповідно до наших потреб.

Зазвичай скидання пароля відбувається у такі кроки:

1. Користувач надсилає свою електронну адресу.
2. На пошту користувача надсилається лист із посиланням для скидання пароля.
3. Користувач натискає на посилання і потрапляє на форму, де він може ввести новий пароль.
4. Після введення нового пароля користувач отримує повідомлення про успішне відновлення пароля.

Реалізація цього функціоналу в проєкті Django полегшить користувачам відновлення доступу до своїх акаунтів у випадку, якщо вони забудуть свої паролі.

Django надає вбудований клас автентифікації, який спрощує роботу з функцією скидання паролю. Імпортувавши відображення автентифікації з модуля django.contrib.auth, ми можемо швидко налаштувати необхідні URL-адреси для скидання пароля. Ці відображення керують логікою кожного кроку процесу скидання пароля, що значно полегшує нашу роботу. Для того, щоб отримати електронну пошту користувача для процесу скидання пароля, нам потрібно створити форму, де користувач може вказати свою електронну адресу.

Django надає клас PasswordResetForm, який ми можемо використати для створення цієї форми (Рис. 3.12).

```
class CustomPasswordResetForm(PasswordResetForm):
    email = forms.EmailField(max_length=254, widget=forms.TextInput(attrs={'class': 'form-control mb-3', 'placeholder': 'Email', 'id': 'form-email'}))

    def clean_email(self):
        email = self.cleaned_data['email']
        user = UserBase.objects.filter(email=email)

        if not user:
            raise forms.ValidationError('Цю адресу не було знайдено')

        return email
```

Рис. 3.12 Форма скидання пароля

Після того, як користувач введе свою адресу електронної пошти у формі зміни пароля, ми повинні надіслати йому листа з посиланням для зміни пароля. Django надає стандартний шаблон для листів для скидання пароля, але ми можемо налаштувати його відповідно до дизайну нашого проекту. Ми також налаштуємо параметри електронної пошти у файлі settings.py Django, щоб вказати провайдера електронної пошти та дані для автентифікації. Це гарантує успішну відправку листів для скидання пароля.

Коли користувач натискає на посилання для зміни пароля, надіслане на його електронну пошту, він потрапляє на сторінку форми, де він може ввести новий пароль. Django надає клас PasswordResetConfirmView, який обробляє цей крок процесу. Створивши шаблон форми підтвердження скидання пароля і зв'язавши його з класом PasswordResetConfirmView, ми можемо відобразити форму і обробити відправку форми в наших відображеннях.

3.4 Розробка шаблонів та відображень

У моделі MVC шаблон є рівнем відображення. Цей рівень взаємодіє з користувачем і надсилає запити до відображень. Шаблони Django працюють з HTML/CSS/JS, обробляючи всю візуалізацію веб-сторінок. Окрім базових конструкцій HTML/CSS/JS, Django має мову шаблонів для відображення динамічних даних.

Щоб налаштувати шаблони Django, перейдіть до settings.py і оновіть параметр DIRS у TEMPLATES, вказавши шлях до папки templates. Зазвичай,

папка templates створюється і зберігається в папці проєкту, де знаходиться файл manage.py. Ця папка містить всі шаблони для додатків Django.

```
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',
                'store.context_processors.categories',
                'basket.context_processors.basket',
            ],
        },
    },
]
```

Рис. 3.13 Налаштування шаблонів Django

Базовий шаблон містить спільний код для всіх HTML-файлів. Він повинен містити такі елементи, як заголовок сайту або логотип, тобто все, що завжди буде відображатися незалежно від того, на якій сторінці ви знаходитесь. Коли ми вставляємо {% block content %} в наш html, ми визначаємо, що цей блок може бути перевизначеним з дочірнього класу, і те, що знаходиться всередині нього, має бути замінено на те, що з'явиться у відповідних тегах дочірнього класу. Як бачимо, ми визначили тут два блоки: content і title. Кожен блок завершується тегом {% endblock %}.

Наступним шаблоном, який ми будемо писати, буде шаблон home.html (Рис. 3.14). Цей шаблон успадкує весь код від нашого базового шаблону і замінить блоки, встановлені в базовому шаблоні. Щоб успадкувати базовий шаблон, нам потрібно включити наступний рядок на початку нашого коду: {% extends 'base.html' %}

```
{% extends "./base.html" %}
{% load static %}
{% block title %}Домашня сторінка{% endblock title %}
{% block content %}
<main>
    <div class="album py-5 bg-light">
        <div class="container">
          <div class="pb-3 h5">Всі товари</div>
          <div class="row row-cols-1 row-cols-sm-2 row-cols-md-5 g-3">
            {% for product in products %}
            <div class="col">
              <div class="card shadow-sm">
                <img class="img-fluid" alt="Responsive image" src="{{ product.image.url }}">
                <div class="card-body">
                  <p class="card-text">
                    <a class="text-dark text-decoration-none" href="{{ product.get_absolute_url }}">{{ product.title }}</a>
                  </p>
                  <div class="d-flex justify-content-between align-items-center">
                    <small class="text-muted">{{ product.price }} ₴</small>
                  </div>
                </div>
              </div>
            </div>
            {% endfor %}
          </div>
        </div>
      </div>
</main>
{% endblock content %}
```

Рис. 3.14 Приклад коду з шаблону home.html

В архітектурі MVC відображення відповідає за представлення даних користувачам. У фреймворку Django представлення - це функції або класи Python, які отримують веб-запит і повертають відповідь. Відповідь може бути HTTP-відповіддю, HTML-шаблоном або переадресацією. Відображення містять логіку, яка необхідна для повернення інформації у відповідь користувачеві у будь-якій формі. Логіка, яка працює з відображеннями, міститься у файлі views.py у додатку Django. У відображеннях також можуть оброблятися HTML-шаблони. Шаблони дозволяють розробникам розділити логіку представлення та бізнес-логіку. Приклад відображення, яке обробляє HTML-шаблон показаний на рисунку 3.15.

```
def product_all(request):
    products = Product.products.all()
    return render(request, 'store/home.html', {'products': products})
```

Рис. 3.15 Функція відображення домашньої сторінки

У цьому прикладі функція про відображення приймає об'єкт запиту як аргумент і повертає HTTP-відповідь з шаблоном home.html. Для передачі даних до шаблону використовується словник products.

3.5 Обробка електронних платежів зі Stripe

Stripe надає API для прийому платежів, керування підписками та виплат на банківські рахунки або картки. Stripe підтримує різні способи оплати, включаючи кредитні картки, банківські перекази, Apple Pay та Google Pay. Розробники обирають Stripe через його безпеку. Він надійно обробляє ключові платіжні дані, гарантуючи, що конфіденційна інформація ніколи не потрапить на ваш сервер.

Інтеграція Stripe починається зі створення акаунта. Цей акаунт буде нашим шлюзом для використання інструментів обробки платежів Stripe, доступу до даних про транзакції та налаштування платіжних параметрів. Stripe проводить користувача через налаштування акаунта. Це включає надання інформації про бізнес.

Необхідно буде заповнити наступні деталі:

Тип бізнесу та деталі: Потрібно вказати, чи працює ваша компанія як фізична особа, партнерство, корпорація тощо. Також треба вказати назву компанії та, за його наявності, веб-сайт.

Банківські реквізити: Вказуємо дані свого банківського рахунку, на який Stripe зарахує кошти.

Після перевірки інформації, Stripe активує акаунт. Тепер можна налаштовувати платіжне середовище.

Після налаштування акаунта Stripe наступним важливим кроком в інтеграції Stripe із застосунком є отримання ключів API. Ці ключі дозволяють застосунку безпечно взаємодіяти з серверами Stripe, полегшуючи обробку платежів, управління клієнтами та інші транзакції.

Заходимо до Панелі управління Stripe за допомогою облікових даних. Після входу слід знайти розділ " Developers" у верхній навігаційній панелі або меню панелі інструментів. Натиснувши на нього, ми отримаємо доступ до розділу для розробників, який містить документацію, довідник по API і ключі API.

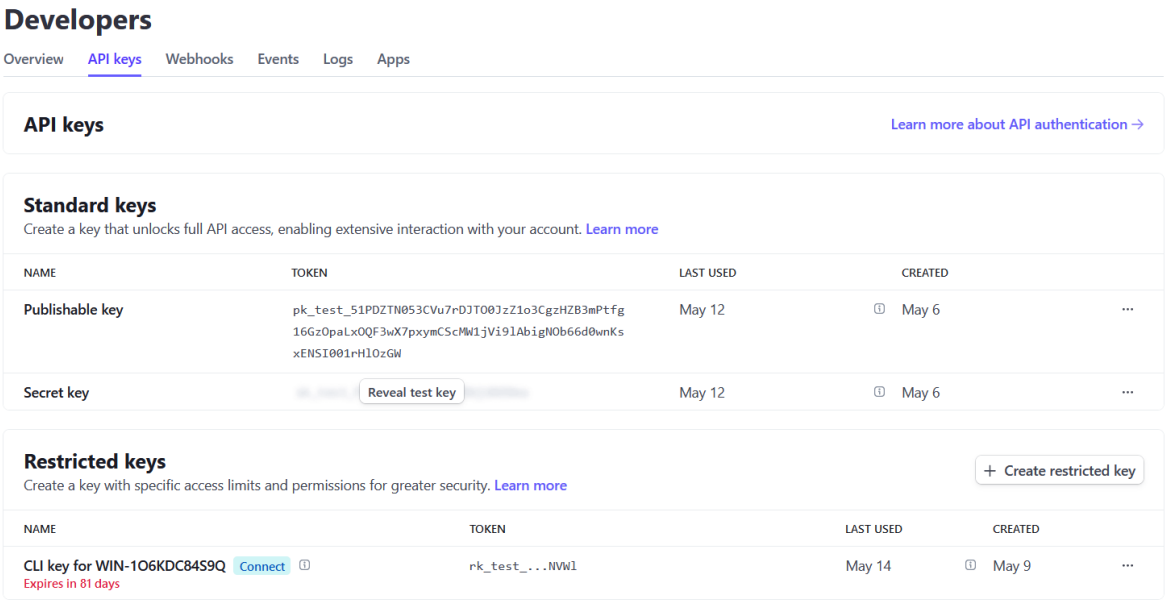


Рис. 3.16 Розділ ключів API на сторінці розробника Stripe

У розділі "Developers" можна знайти вкладку "API keys". У цій вкладці відобразяться ключі API, необхідні для інтеграції.

Stripe пропонує два типи ключів:

Відкритий ключ: Цей ключ використовується у зовнішньому інтерфейсі застосунку, наприклад, на веб-сторінках або в мобільних додатках, для токенізації платіжної інформації. Він ідентифікує акаунт на серверах Stripe, але не має дозволу на списання коштів.

Секретний ключ: Цей ключ використовується на сервері. З його допомогою можна виконувати практично будь-які операції в акаунті Stripe, включаючи списання та повернення коштів, тому його варто тримати в безпеці і ніколи не передавати і не виставляти в загальний доступ. Stripe надає окремі набори ключів для тестування та реального середовища. Тестові ключі мають префікс pk_test_ і sk_test_, тоді як робочі ключі використовують pk_live_ і sk_live_. На етапі розробки та інтеграції будуть використовуватися тестові ключі, що гарантує безпечну імітацію транзакцій без переміщення реальних грошей. Коли застосунок почне приймати реальні платежі, слід перейти на робочі ключі. Секретний ключ потрібно тримати в таємниці і зберігати тільки в надійних місцях. Відкритий ключ призначений для публічного доступу, але секретний ключ ніколи не повинен бути видимим у клієнтському коді або сховищах. Отримання та

керування ключами API - це простий процес, але він життєво необхідний для безпеки та функціональності інтеграції зі Stripe.

Після отримання ключів API наступним кроком в інтеграції Stripe у додаток або веб-сайт буде вибір найбільш підходящого методу інтеграції для потреб проєкту. Stripe пропонує безліч варіантів, кожен з яких відповідає різним рівням кастомізації, простоти використання та трудомісткості розробки.

Методи інтеграції Stripe включають:

Елементи Stripe:

Елементи Stripe - це готові компоненти інтерфейсу користувача для створення платіжних форм. Вони оптимізують процес оформлення замовлення і можуть бути стилізовані під ваш сайт. Елементи обробляють конфіденційну платіжну інформацію, токенізуючи реквізити карток без передачі на ваш сервер.

Stripe Checkout:

Stripe Checkout - це платіжна сторінка на хостингу Stripe для перенаправлення клієнтів. Вона безпечна, проста у впровадженні і підтримує мобільні пристрої. Вона обробляє все, від збору платіжної інформації до токенізації.

Мобільні SDK:

Stripe пропонує SDK для iOS та Android для інтеграції платежів у мобільні додатки. Ці SDK надають компоненти та інтерфейси, адаптовані до мобільного середовища, забезпечуючи зручність та безпеку платежів.

Елементи Stripe надають більше можливостей для кастомізації, а Stripe Checkout - найшвидший спосіб почати роботу. Мобільні SDK потрібні для інтеграції в нативні додатки.

Інтеграція Stripe у додаток або веб-сайт вимагає додавання бібліотеки Stripe до проєкту. Ця бібліотека надає функції для взаємодії з API Stripe.
Для інтеграції у веб-додаток використовують Stripe.js для створення токенів або PaymentIntent та серверні бібліотеки Stripe для бекенд-операцій. Потрібно додати тег скрипта Stripe.js до HTML сторінки, на якій ви хочете обробляти платежі (Рис.

3.17). Це необхідно для використання елементів Stripe або безпечної обробки платіжної інформації.

```
<script src="https://js.stripe.com/v3/"></script>
```

Рис. 3.17 Тег скрипта Stripe.js

Цей рядок завантажує бібліотеку Stripe.js, роблячи об'єкт Stripe доступним у JavaScript-коді, який ми можемо використовувати для ініціалізації елементів Stripe або прямих викликів API Stripe. Для бекенд-операцій потрібно встановити бібліотеку Stripe на Python, набравши pip install stripe у вашому терміналі.

Впровадження збору платежів є важливим етапом інтеграції Stripe у застосунок або веб-сайт. Цей процес включає в себе налаштування системи для збору платіжних реквізитів клієнтів і обробки транзакцій. Приклади інтеграції веб-додатків, що демонструють, як збирати платежі за допомогою Stripe представлений на рисунку 3.18.

```
var stripe = Stripe('pk_test_51PDZTN053CVu7rDJTO0JzZ1o3CgzHZB3mPtfg16GzOpaLxOQF3wX7pxymCScMW1jVi9lAbigNOb66d0wnKsxENSI001rHlOzGW');

var elem = document.getElementById('submit');
clientsecret = elem.getAttribute('data-secret');
var elements = stripe.elements();
var style = {
    base: {
        color: "#000",
        lineHeight: '2.4',
        fontSize: '16px'
    }
};
var card = elements.create("card", {style: style});
card.mount("#card-element");

card.on('change', function(event){
var displayError = document.getElementById('card-errors')
if (event.error) {
    displayError.textContent = event.error.message;
    $('#card-errors').addClass('alert alert-info');
} else {
    displayError.textContent = '';
    $('#card-errors').removeClass('alert alert-info');
}
});
```

Рис. 3.18 Код форми створення платежів

На стороні сервера потрібно використати токен з фронтенду для створення платежу (Рис. 3.19).

```python
@login_required
def BasketView(request):
    basket = Basket(request)
    total = str(basket.get_total_price())
    total = total.replace('.', '')
    total = int(total)
    print('total')
    stripe.api_key = 'sk_test_51PDZTN053CVu7rDJDLBY9xQzr7Kea8DHHdk6F1eGueXYRQUp92GbnBuBD6k2hXEbfwBEBl3KRKRuZZft0QXGKqXh008JwfgTZk'
    intent = stripe.PaymentIntent.create(
        amount=total,
        currency='uah',
        metadata={'userid': request.user.id}
    )
    return render(request, 'store/payment/detail.html', {'client_secret': intent.client_secret})

@csrf_exempt
def stripe_webhook(request):
    payload = request.body
    event = None

    try:
        event = stripe.Event.construct_from(json.loads(payload), stripe.api_key)
    except ValueError as e:
        print(e)
        return HttpResponse(status=400)

    if event.type == 'payment_intent.succeeded':
        payment_confirmation(event.data.object.client_secret)
    else:
        print('Unhandled event type {}'.format(event.type))

    return HttpResponse(status=200)

def order_placed(request):
    basket = Basket(request)
    basket.clear()
    return render(request, 'store/payment/orderplaced.html')
```

Рис. 3.19 Обробка платежів на стороні сервера

3.6 Тестування веб-застосунку

Модульне тестування забезпечує надійність та функціональність Django-додатків. Написання та організація модульних тестів дозволяє перевірити поведінку окремих компонентів, таких як моделі, представлення та форми.

Тестування моделей перевіряє їх коректність. Воно включає в себе тестування різних аспектів, включаючи перевірку полів, взаємодію з базами даних і методи моделі. Тести моделей переконують, що дані працюють правильно.

Тестування відображень перевіряє їх поведінку. Воно включає тестування статусу відповіді, вмісту та будь-якої пов'язаної з ним бізнес-логіки. Шляхом написання тестів відображень можна переконатися, що вони виводять правильні шаблони, повертають очікувані HTTP-відповіді та належним чином обробляють вхідні дані.

Тестування форм перевіряє їх поведінку та валідацію. Воно включає в себе тестування відправлення форм, валідацію полів і обробку помилок. Шляхом

написання тестів форм можна переконатися, що форми коректно обробляють дані, введені користувачем, і повертають відповідні повідомлення про помилки, коли це необхідно.

Django надає інструменти для написання ефективних модульних тестів. Ці інструменти допомагають створювати тестові кейси та перевіряти код.

Клас TestCase надає основу для написання тестових кейсів. Він налаштовує чисту базу даних для кожного тесту, надає специфічні для тесту твердження і обробляє загальні операції тестування, такі як запуск методів setUp() і tearDown().

Клас Django Client дозволяє імітувати HTTP-запити і тестувати поведінку відображень. За допомогою класу Client можна робити запити, перевіряти коди статусу та вміст відповідей, обробляти файли cookie та сесії, тощо.

Інтеграційне тестування дозволяє перевірити взаємодію між різними компонентами застосунку Django, такими як подання, шаблони та базові дані. Моделюючи реальні сценарії, можна переконатися, що ці компоненти працюють разом, як очікується.

Перед тестуванням відображень і шаблонів налаштуйте реалістичні тестові дані. Використовуйте Django ORM для заповнення тестової бази даних даними (Рис. 3.20).

```
from importlib import import_module
from django.conf import settings
from django.test import TestCase, Client
from account.models import UserBase
from django.urls import reverse
from store.models import Category, Product
from django.http import HttpRequest
from store.views import product_all



class TestViewResponses(TestCase):
    def setUp(self):
        self.c = Client()
        UserBase.objects.create(username='user')
        Category.objects.create(name='test', slug='test')
        Product.objects.create(category_id=1, title='test', added_by_id=1, slug='test', price='20.00', image='test')
```

Рис. 3.20 Налаштування даних за допомогою функції setUp()

Клас Django Client використовується для імітації HTTP-запитів та взаємодії з відображеннями. Він дозволяє виконувати GET, POST та визначати статус і вміст відповіді.

```
def test_url_allowed_hosts(self):
    response = self.c.get('/')
    self.assertEqual(response.status_code, 200)

def test_product_detail_url(self):
    response = self.c.get(reverse('store:product_detail', args=['test']))
    self.assertEqual(response.status_code, 200)

def test_category_detail_url(self):
        response = self.c.get(reverse('store:category_list', args=['test']))
        self.assertEqual(response.status_code, 200)
```

Рис. 3.21 Тестування HTTP запитів за допомогою класу Client

Для перевірки шаблонів використовуйте движок сесій Django для рендерингу з тестовими даними (Рис. 3.22).

```
def test_homepage_html(self):
    request = HttpRequest()
    engine = import_module(settings.SESSION_ENGINE)
    session_key = None
    request.session = engine.SessionStore(session_key)
    response = product_all(request)
    html = response.content.decode('utf8')
    self.assertEqual(response.status_code, 200)
    self.assertIn('<title>Домашня сторінка</title>', html)
```

Рис. 3.22 Тестування шаблону HTML за допомогою движка сесій Django

ВИСНОВКИ

У роботі проаналізовано сферу електронної комерції. Розмір ринку електронної комерції зростає дуже швидкими темпами. Ця тенденція, ймовірно, збережеться і в найближчі роки, оскільки компанії розуміють, що підтримувати інтернет-магазин дешевше і простіше, ніж звичайний. Компанії мають бути готовими до змін у способах здійснення покупок і зосередитися на створенні зручного в користуванні та інтерактивного онлайн-простору. Крім того, з'являється все більше платформ та інструментів, які допомагають бізнесу розпочати свою діяльність у сфері електронної комерції.

Розглянуто теоретичні основи Django, AJAX, Bootstrap, Stripe та принципи проєктування моделей ORM, шаблонів HTML і сесій AJAX. Впровадження AJAX у застосунку Django може значно покращити взаємодію з користувачем, зробивши його більш адаптивним та інтерактивним. Використовуючи jQuery і вбудовані в Django класи відображень і форм, можна створити динамічні елементи, які оновлюються без перезавантаження всієї сторінки. Bootstrap - це безкоштовний фронтенд-фреймворк, який сьогодні популярний серед розробників, особливо тих, хто працює у сфері веб-дизайну. Він дозволяє легко інтегрувати багато корисних функцій, які збагачують взаємодію з користувачем, без необхідності писати їх з нуля. Stripe - провідна платформа для онлайн-платежів, що пропонує бізнесу безпечне, гнучке та адаптивне рішення для обробки платежів. Завдяки широкому набору функцій та зручному інтерфейсу Stripe дозволяє компаніям створювати безперебійний платіжний досвід та розширити свою діяльність на міжнародний рівень.

Досліджено проєктування баз даних з моделями ORM, обробку онлайн-платежів та тестування застосунків з unittest. Моделі Django слугують основою взаємодії з базами даних у проектах Django. Їх гнучкість та можливості, такі як зв'язки між моделями, успадкування та менеджери моделей, дозволяють легко створювати масштабовані веб-додатки. Модульне тестування - це ключовий компонент розробки програмного забезпечення. Воно перевіряє, чи працює код

так, як заплановано. Створюючи чіткі та ефективні модульні тестові кейси, розробники можуть виявляти проблеми на ранніх стадіях розробки, заощаджуючи цінний час і ресурси.

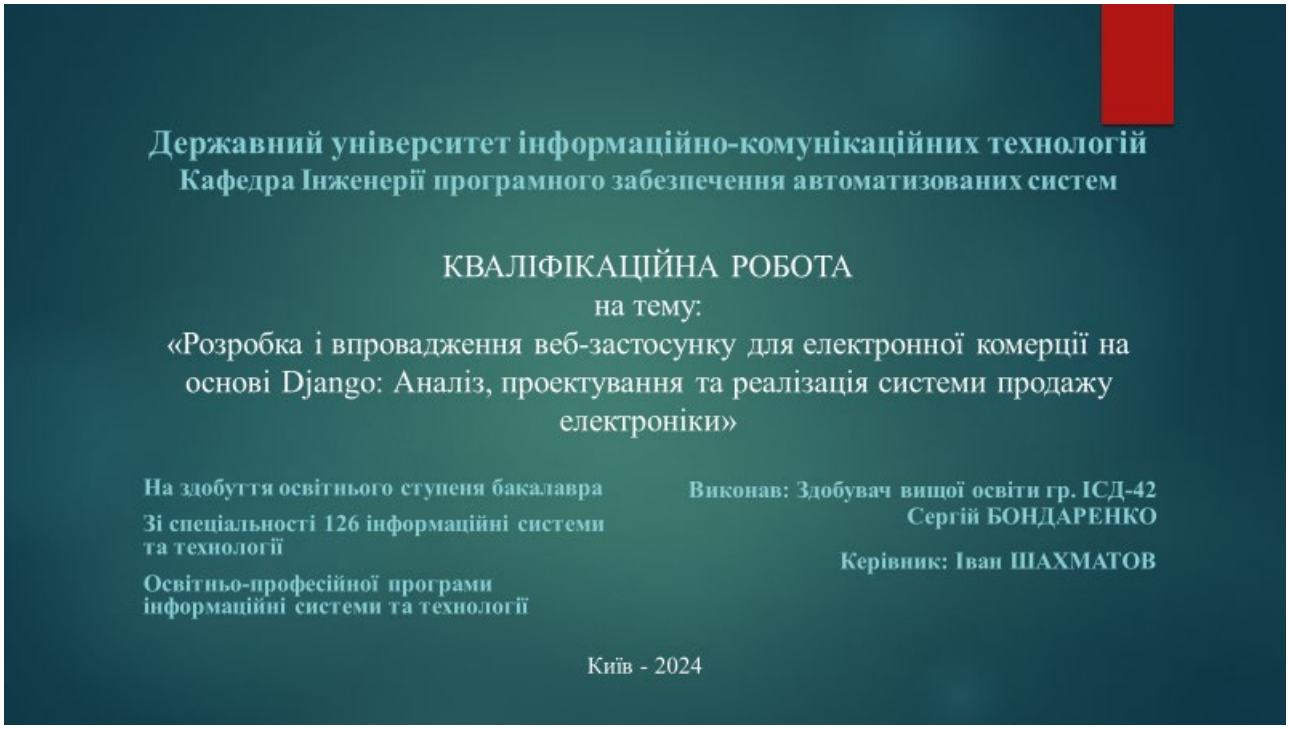
Створено структуру проєкту, базу даних, HTML-шаблони та форми для взаємодії з користувачем, налаштовано систему онлайн-оплати з Stripe.

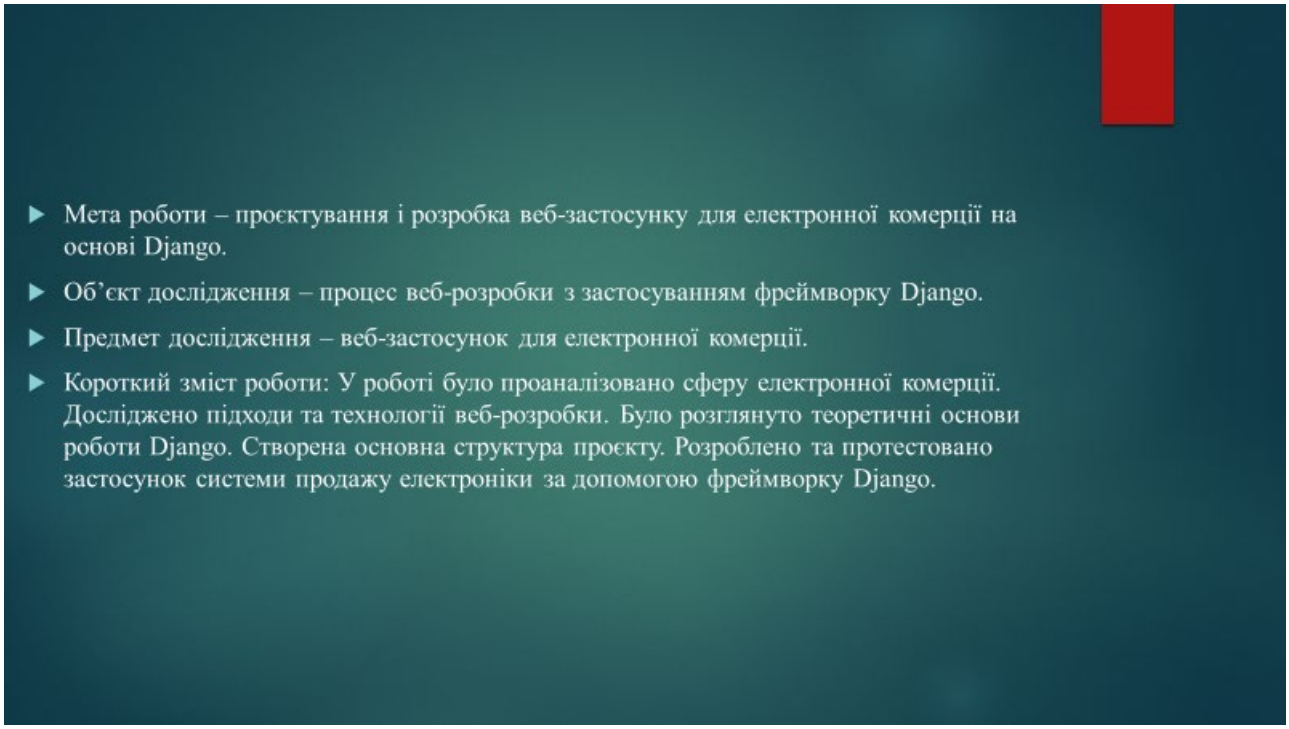
Розроблено та протестовано застосунок для продажу електроніки з використанням Django.

ПЕРЕЛІК ПОСИЛАНЬ

1. What Is E-Commerce? Definition, Types & Getting Started [Електронний ресурс] — Режим доступу: https://www.forbes.com/advisor/business/what-is-ecommerce/
2. 35 E-Commerce Statistics of 2024 [Електронний ресурс] — Режим доступу: https://www.forbes.com/advisor/business/ecommerce-statistics/
3. A Brief History of Ecommerce (and a Look at the Future) [Електронний ресурс] — Режим доступу: https://www.thefulfillmentlab.com/blog/history-of-ecommerce
4. Django documentation [Електронний ресурс] — Режим доступу: https://docs.djangoproject.com/en/5.0/
5. Django introduction [Електронний ресурс] — Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction
6. The Essentials of ORM Framework in Your Software Development [Електронний ресурс] — Режим доступу: https://medium.com/@mikependon/the-essentials-of-orm-framework-in-your-software-development-837131efd91b
7. AJAX Introduction [Електронний ресурс] — Режим доступу: https://www.w3schools.com/xml/ajax_intro.asp
8. AJAX [Електронний ресурс] — Режим доступу: https://uk.wikipedia.org/wiki/AJAX
9. Get started with Bootstrap [Електронний ресурс] — Режим доступу: https://getbootstrap.com/docs/5.3
10. What Is a Payment Processor? Top 5 Payment Processors (2024) [Електронний ресурс] — Режим доступу: https://www.shopify.com/blog/payment-processors#

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)





Огляд Django

- Django - це фреймворк з відкритим вихідним кодом написаний мовою Python, який використовується для швидкої розробки, прагматичних, підтримуваних і безпечних веб-сайтів з чистим дизайном.
- Головне завдання фреймворку Django - дозволити розробникам сконцентруватися на нових компонентах додатку, замість того, щоб витрачати час на вже розроблені.
- Зараз Django використовують тисячі веб-сайтів, від щоденних газет до соціальних мереж і сайтів для обміну даними, від великих фондів до некомерційних організацій.

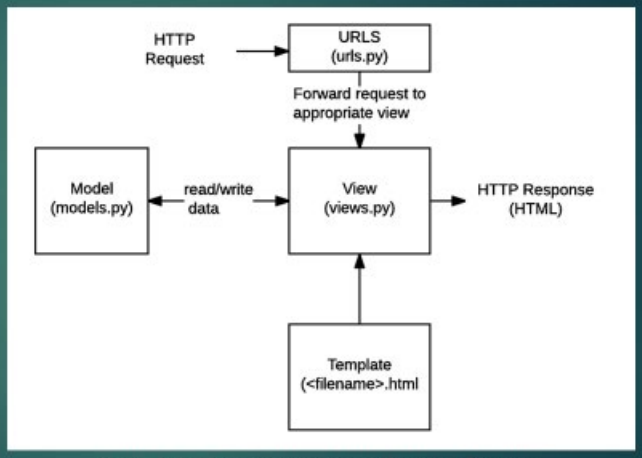


Рис. 1.1 Схема взаємодії файлів Django

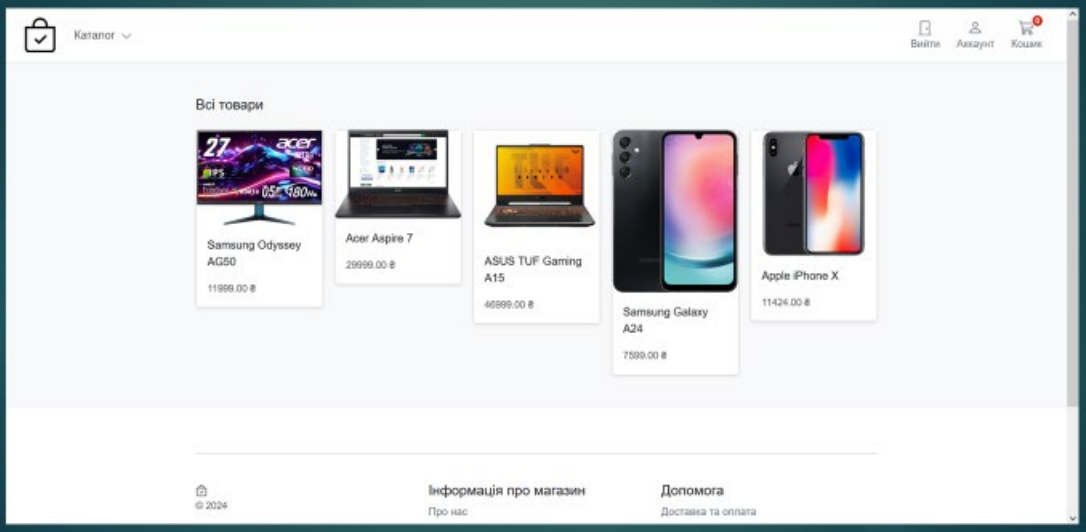
Особливості використання технології AJAX

- AJAX - це підхід до розробки веб-інтерфейсу користувача, який передбачає фоновий обмін даними між веб-браузером і ресурсами сервера. Він дозволяє оновлювати певні частини веб-сторінки асинхронно, тобто користувач може продовжувати взаємодіяти зі сторінкою, поки відбувається обмін даними з сервером у фоновому режимі. Це допомагає створити більш плавний і динамічний взаємодію з користувачем на веб-сайтах.
- AJAX використовується для додавання динамічного контенту на веб-сторінки. Наприклад, він може впоратися з ситуацією, коли користувачу потрібно заповнити форму і просто оновити результат після відправлення форми.
- AJAX дозволяє сторінці взаємодіяти з сервером у фоновому режимі. Це дозволяє надсилати та отримувати дані непомітно для користувача. Коли користувач виконує якусь дію, дані можуть обмінюватися з сервером без перезавантаження сторінки.

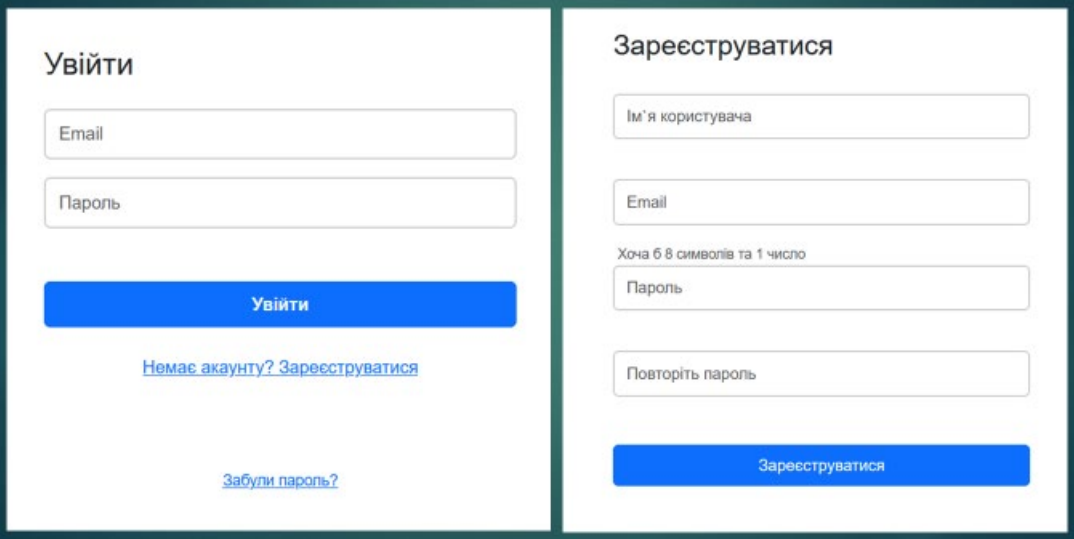
Особливості фреймворку Bootstrap

- Bootstrap - це популярний фронтенд-фреймворк з відкритим вихідним кодом, створений для спрощення веб-дизайну шляхом надання готових інструментів та рішень. Він розроблений для полегшення процесу веб-розробки адаптивних веб-сайтів, надаючи колекцію синтаксису для дизайну шаблонів.
- Bootstrap був створений в середині 2010 року Марком Отто, старшим дизайнером в Twitter, і Джейкобом Торнтоном, розробником в Twitter. Марк і Джейкоб розробляли внутрішні інструменти для підтримки узгодженості та підвищення ефективності веб-розробки в компанії. Їм спало на думку, що ці інструменти можуть бути корисними для ширшої спільноти веб-розробників, і в 2011 році народився Bootstrap.
- Відтоді Bootstrap став найпоширенішим CSS-фреймворком і другою за популярністю бібліотекою JavaScript в Інтернеті. В основі роботи Bootstrap лежать принципи адаптивного веб-дизайну.

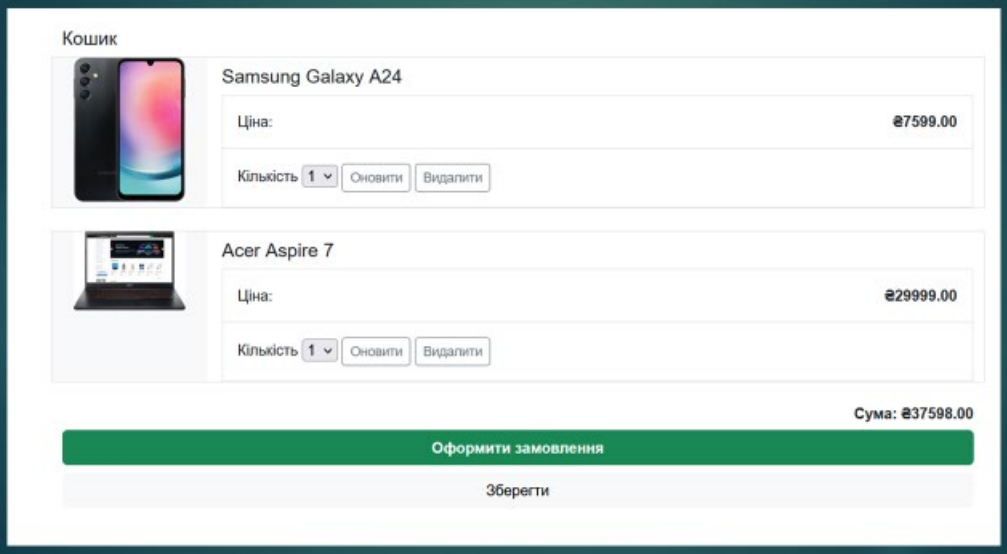
Головна сторінка веб-додатку



Панелі реєстрації та входу



Кошик



Приклади коду

```
{% extends "./base.html" %}
{% load static %}
{% block title %}Домашня сторінка{% endblock title %}
{% block content %}
<main>
    <div class="album py-5 bg-light">
        <div class="container">
          <div class="pb-3 h5">Всі товари</div>
          <div class="row row-cols-1 row-cols-sm-2 row-cols-md-5 g-3">
            {% for product in products %}
            <div class="col">
              <div class="card shadow-sm">
                <img class="img-fluid" alt="Responsive image" src="{{ product.image.url }}">
                <div class="card-body">
                  <p class="card-text">
                    <a class="text-dark text-decoration-none" href="{{ product.get_absolute_url }}">{{ product.title }}</a>
                  </p>
                  <div class="d-flex justify-content-between align-items-center">
                    <small class="text-muted">{{ product.price }} ₴</small>
                  </div>
                </div>
              </div>
            </div>
            {% endfor %}
          </div>
        </div>
      </div>
</main>
{% endblock content %}
```

Приклади коду

```
{% extends "../../store/base.html" %}
{% block title %}Вхід{% endblock title %}
{% block content %}
<div class="container-fluid">
    <div class="row no-gutter">
        <div class="col-md-12">
            <div class="login d-flex align-items-center py-5">
                <div class="container">
                    <div class="row">
                        <div class="col-12 col-lg-6 mx-auto">
                            <form class="account-form p-4 rounded" action="{% url 'account:login' %}" method="post">
                                {% csrf_token %}
                                <h1 class="mb-4">Увійти</h1>
                                {% if form.errors %}
                                <div class="alert alert-primary" role="alert">Помилка: Неправильний Email або пароль!</div>
                                {% endif %}
                                {{ form.username }}
                                {{ form.password }}
                                <div class="d-grid gap-2">
                                    <input type="hidden" name="next" value="{{ next }}">
                                    <button class="btn btn-primary py-2 mb-4 mt-5 fw-bold" type="submit" value="log in">Увійти</button>
                                </div>
                                <p class="text-center pb-3">
                                    <a href="{% url "account:register" %}">Немає акаунту? Зареєструватися!</a>
                                </p>
                            </form>
                            <p class="small text-center pt-4">
                                <a href="{% url "account:password_reset" %}">Забули пароль?</a>
                            </p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock content %}
```

```
{% extends "../../store/base.html" %}
{% load static %}
{% block title %}Register{% endblock title %}

{% block content %}
<div class="container-fluid">
    <div class="row no-gutter">
        <div class="col-md-12">
            <div class="login d-flex align-items-center py-5">
                <div class="container">
                    <div class="row">
                        <div class="col-12 col-lg-7 mx-auto">
                            <form class="account-form p-4 rounded col-lg-10 mx-auto" action="." method="post">
                                {% csrf_token %}
                                <h1 class="mb-2">Зареєструватися</h1>
                                {% if form.errors %}
                                    <p>В роботі веб-застосунку виникли помилки:</p>
                                    {% for field in form %}
                                        {% if field.errors %}
                                            <div class="alert alert-primary" role="alert">
                                                {{ field.label }}: {{ field.errors|striptags }}
                                            </div>
                                        {% endif %}
                                    {% endfor %}
                                {% endif %}
                                {{ form.username }}
                                {{ form.email }}
                                <small class="form-text text-muted mb-4 small">
                                    Вона є в комплекті та 1 числа
                                </small>
                                {{ form.password }}
                                {{ form.repeat_password }}
                                <button class="btn btn-primary btn-block py-2 mb-4 mt-5 fw600 w-100" type="submit">Зареєструватися</button>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock content %}
```

Висновки

- У роботі було проаналізовано сферу електронної комерції.
- Досліджено підходи та технології веб-розробки.
- Було розглянуто теоретичні основи роботи Django.
- Створена основна структура проєкту.
- Розроблено та протестовано застосунок системи продажу електроніки за допомогою фреймворку Django.

Дякую за увагу!