Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Дослідження ефективності використання фреймворків веб-розробки на продуктивність веб-додатку: порівняльний аналіз Django, Flask, та FastAPI
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-5

Солохін А. Є.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник проф. Смеляков С.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерних наук
Кафедра	Програмної інженерії
Рівень вищої освіти	другий (магістерський)
Спеціальність	121 – Інженерія програмного забезпечення
Тип програми	освітньо-наукова програма
Освітня програма	Інженерія програмного забезпечення

Курс 2 Группа ІПЗм-22-5 Семестр 2

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студенту Солохіну Артему Євгеновичу

1. Тема роботи: Дослідження ефективності використання фреймворків веб-розробки на продуктивність веб-додатку: порівняльний аналіз Django, Flask, та FastAPI

Затверджена наказом по університету від 29.03 2024р. № 250 Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2024

3. Вихідні дані до роботи *Огляд літератури за темою дослідження і аналіз стану проблеми і постановка задачі дослідження; Важливість використання Back-end фреймворків; Відмінності процесу розробки; Експериментальна частина; Висновки*

4. Перелік питань, що потрібно опрацювати в роботі: *огляд літератури та аналогів, теоретичні основи фреймворків ,відмінності процесу розробк, експериментальна частина*

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Видача завдання	29.04.2024	виконано
2	Аналіз предметної галузі	01.05.2024	виконано
3	Постановка задачі	02.05.2024	виконано
4	Експериментальні дослідження	02.05 – 20.05.24	виконано
5	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.05 – 22.05.24	виконано
6	Написання та оформлення статті та тез доповіді	20.05 – 23.05.24	виконано
7	Підготовка пояснювальної записки	01.05 – 26.05.24	виконано
8	Підготовка презентації та доповіді	26.05 – 2.05.24	виконано
9	Нормоконтроль	3.06 – 08.06.24	виконано
10	Рецензування	08.06 – 14.06.24	виконано
11	Занесення диплома в електронний архів	15.06.2024	виконано
12	Попередній захист	15.06.2024	виконано
13	Допуск до захисту у зав. кафедри	18.06.2024	виконано

Дата видачі завдання «29» березня 2024 р.

Студент _____ Солохін А. Є.

(підпис)

Керівник роботи_____ проф. Смеляков К.С.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 80 ст., 56 рис., 7 табл., 15 джерел.

ГНУЧКІСТЬ ФРЕЙМВОРКІВ, КРИТЕРІЇ ВИБОРУ ФРЕЙМВОРКУ, МАСШТАБОВАНІСТЬ, ОПТИМІЗАЦІЯ РЕСУРСІВ, ПРОДУКТИВНІСТЬ ВЕБ-ДОДАТКІВ, ШВИДКОДІЯ ВИКОНАННЯ, ТЕСТУВАННЯ ПРОДУКТИВНОСТІ, ЗБІР ТА АНАЛІЗ ДАНИХ, DJANGO, FASTAPI, FLASK.

Метою дослідження є аналіз і порівняння продуктивності зазначених фреймворків з метою надання розробникам та архітекторам відомостей, необхідних для обґрунтованого вибору фреймворку в залежності від конкретних вимог проекту.

Для досягнення поставленої мети використовуватиметься комплексний підхід, який включає в себе огляд літератури та аналогів, теоретичний аналіз основ фреймворків, розробку та реалізацію тестових сценаріїв, збір та аналіз даних за допомогою визначених критеріїв продуктивності.

Результатом дослідження буде комплексний порівняльний аналіз швидкодії виконання, масштабованості та ефективного використання ресурсів системи для фреймворків Django, Flask та FastAPI. Отримані висновки та рекомендації дозволять розробникам та архітекторам зробити обгрунтований вибір фреймворку в залежності від конкретних вимог та завдань проекту.

DATA COLLECTION AND ANALYSIS, DJANGO, EXECUTION SPEED, FASTAPI, FLASK, FRAMEWORK FLEXIBILITY, FRAMEWORK SELECTION CRITERIA, PERFORMANCE TESTING, RESOURCE OPTIMIZATION, SCALING, WEB APPLICATION PERFORMANCE.

The purpose of the study is to analyze and compare the performance of the specified frameworks in order to provide developers and architects with the

information necessary for a justified choice of the framework depending on the specific requirements of the project.

To achieve the goal, a comprehensive approach will be used, which includes a review of the literature and analogues, theoretical analysis of framework foundations, development and implementation of test scenarios, data collection and analysis using defined performance criteria.

The result of the research will be a comprehensive comparative analysis of execution speed, scalability and efficient use of system resources for Django, Flask and FastAPI frameworks. The obtained conclusions and recommendations will allow developers and architects to make an informed choice of the framework depending on the specific requirements and tasks of the project.

Я, Солохін Артем Євгенович, студент(ка) гр. ІПЗм-22-5, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності використання фреймворків веб-розробки на продуктивність веб-додатку: порівняльний аналіз Django, Flask, та FastAPI», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Розвиток веб-розробки востаннє десятиліття визначається не тільки розмаїттям інтернет-технологій, але й активним використанням фреймворків для створення веб-додатків. Django, Flask та FastAPI – це лише кілька представників цього роду інструментарію, які зайняли свої позиції в галузі веб-розробки.

З урахуванням постійно зростаючих вимог до продуктивності веб-додатків, виникає необхідність в об'єктивному порівнянні фреймворків з погляду їхнього впливу на ефективність розробки та функціональність створених додатків.

Мета даного дослідження – провести глибокий порівняльний аналіз фреймворків Django, Flask та FastAPI з точки зору їхнього впливу на продуктивність веб-додатків. В ході дослідження будуть визначені переваги та недоліки кожного фреймворку, а також надані практичні рекомендації для вибору найбільш підходящого інструменту в залежності від конкретних вимог та завдань розробки.

Для досягнення цієї мети будуть визначені критерії оцінки продуктивності, створені тестові сценарії, та проведений докладний аналіз результатів тестувань. Окрема увага буде приділена вивченню теоретичних аспектів роботи кожного фреймворку та інструментів оптимізації, які можуть впливати на продуктивність веб-додатків.

Ця робота не тільки спрямована на поглиблене вивчення вищезазначених фреймворків, але й слугує джерелом інформації для розробників та архітекторів веб-додатків, які стикаються з вибором інструменту для своїх проектів та прагнуть визначитися з найбільш оптимальним рішенням в конкретних умовах використання.

1 ОГЛЯД ЛІТЕРАТУРИ ТА АНАЛОГІВ

1.1 Огляд фреймворків веб-розробки

В сучасному світі веб-розробка не уявляється без використання фреймворків – спеціальних інструментів, які надають загальну структуру та ряд інструментів для ефективної розробки веб-додатків. У цьому розділі буде проведено огляд ключових фреймворків веб-розробки, зокрема Django, Flask та FastAPI.

Django є одним із найпопулярніших фреймворків для розробки веб-додатків на мові програмування Python. Він пропонує високорівневий інструментарій, вбудовану адміністративну панель, систему маршрутизації, та широкі можливості для роботи з базами даних. Django відомий своєю повнотою функціональності та конвенціями над конфігурацією.

Flask є легким та гнучким мікрофреймворком, також написаним на Python. Основна філософія Flask полягає в тому, що він надає лише базовий функціонал, дозволяючи розробникам обирати та доповнювати функціонал своїми компонентами. Це робить Flask відмінним вибором для тих, хто шукає простоту та гнучкість.

FastAPI – це новітній фреймворк для розробки веб-додатків на мові програмування Python, який активно використовує стандартні механізми Python, такі як аннотації типів та система автодокументації. Він славиться високою швидкодією завдяки використанню виключно асинхронного програмування.

Кожен з цих фреймворків має свої унікальні особливості, які можуть впливати на продуктивність веб-додатків. Django, як повноцінний фреймворк, надає готові рішення, тоді як Flask та FastAPI ставляться як більш легкі та гнучкі інструменти. Детальний огляд їхніх можливостей та впливу на розробку дозволить зрозуміти, яким чином ці фреймворки можуть відповісти на вимоги до продуктивності веб-додатків.

1.2 Історія розвитку Django, Flask та FastAPI

Django був випущений у 2005 році. Розробка цього фреймворку почалася у кінці 2003 року внутрішньою групою розробників в компанії Lawrence Journal-World. Їхня мета була створити простий інструмент для швидкої розробки новинних веб-сайтів. Django став відкритим проектом у 2005 році, і з тих пір має широкий вплив на галузь веб-розробки. Його назва походить від імені гітариста Джанго Рейнгольда [1].

Flask був вперше представлений у 2010 році. Армін Ронач, автор Flask, створив його як легкий та простий у використанні фреймворк. Він хотів, щоб Flask був настільки простим, щоб новачки могли вивчити його швидко, але при цьому був достатньо потужним для застосування в складних проектах. Flask став важливим гравцем у світі мікрофреймворків та здобув велику популярність завдяки своїй гнучкості та простоті [2].

FastAPI, випущений у 2018 році, є одним з новітніх учасників у світі фреймворків для веб-розробки. Розробка FastAPI була ініційована Себастьяном Раміресом під впливом сучасних тенденцій, таких як асинхронне програмування та автоматична генерація документації на основі аннотацій типів. Цей фреймворк отримав визнання через свою швидкодію, автоматизацію та сучасний підхід до розробки веб-додатків [3].

Історія кожного фреймворку відображає відповіді на змінюючіться потреби та технологічні тенденції в галузі веб-розробки. Django створений для повністю обґрунтованого та готового до використання підходу, Flask – для гнучкої та мінімалістичної розробки, а FastAPI – для високоефективних асинхронних додатків. Перегляд історії кожного фреймворку допомагає краще розуміти їхні особливості та напрями розвитку.

1.3 Переваги та недоліки кожного фреймворку

1.3.1 Django

Переваги Django [4]:

а) Батарейки в комплекті.

Django пишається собою як фреймворк з батарейками. Це означає, що він поставляється з багатьма речами з коробки, якими ви можете або не можете користуватися залежно від програми. Замість того, щоб писати власний код (потужність), вам просто потрібно імпортувати пакети, які ви хочете використовувати.

Це частина парадигми конвенцій щодо конфігурації, частиною якої є Django, і дозволяє вам використовувати рішення, реалізовані професіоналами світового рівня. Батареї Django охоплюють широкий спектр тем, зокрема:

- автентифікація з пакетом авторизації;
- взаємодія адміністратора з пакетом адміністратора;
- керування сеансами за допомогою пакета sessions;
- керування тимчасовими або сеансовими повідомленнями за допомогою пакета messages;
- створення xml карти сайту google за допомогою пакета sitemaps;
- спеціальні функції postgres із пакетом postgres;
- підключення до «типів» вмісту за допомогою структури типів вмісту.

б) Python.

Оскільки Django використовує Python, він використовує частину слави та потужності Python для власної вигоди. Python, мабуть, є однією з найпростіших, якщо не найпростіших, мов програмування для вивчення для початківців, а також вона досить популярна у вступних курсах інформатики в усьому світі. Опитування розробників Stackoverflow у 2017 році показало, що Python тепер більш поширений, ніж PHP, а робота на Python оплачується краще, ніж C# і C++.

в) Громада.

Спільнота Django є однією з найкращих у цьому, вони допомагають і активно працюють над тим, щоб зробити фреймворк більш зручним для початківців і стабілізувати фреймворк, додаючи нові функції. Документація Django є досить ретельною та корисною як окремий підручник, вона допоможе вам охопити різні функції, щоб ви могли використовувати її як основне джерело інформації.

г) Масштабований.

Більшість забудовників, думаючи про те, щоб вибрати рамковий план на майбутнє на свій вибір. Ось чому вибір масштабованого фреймворку є дуже важливим для багатьох, і Django є саме таким. Це дозволяє виконувати багато різних дій щодо масштабованості, наприклад запускати окремі сервери для бази даних, медіа та самої програми або навіть використовувати кластеризацію чи балансування навантаження для розподілу програми між кількома серверами.

д) Вбудований адмін.

Команда Django була дуже продуманою, коли вони створювали фреймворк, і вони дбали про задоволення користувачів і клієнтів. Нерозумно створювати власний інтерфейс адміністратора на сервері лише для того, щоб мати можливість керувати своїми даними за допомогою базових операцій CRUD. Ось чому Django пропонує адміністративний інтерфейс прямо з коробки, який є професійним і універсальним, відповідно до документів, які розробник тепер може розробляти з урахуванням презентації.

Недоліки Django:

Незважаючи на те, що Django є дивовижною структурою, є кілька недоліків, які можуть або не можуть бути для вас проблемою. По-перше, вказівка URL-адреси за допомогою регулярних виразів є непростим завданням, принаймні для початківців. Він також здається трохи роздутим для невеликих проектів, і деякі люди вважають його досить заповненим великими проектами, оскільки, наприклад, усі моделі включені в один файл. За замовчуванням повідомлення про помилки шаблону миттєво видаляються, тому, якщо ви цього не знаєте, ви можете витратити багато часу, намагаючись з’ясувати, що не так із програмою, або, що ще гірше, ви можете навіть не знати, що у вашій програмі є проблема. Це також впевнена структура, яка надає відчуття монолітності.

1.3.2 Flask

Переваги Flask [5]:

а) Масштабований.

Розмір - це все, і статус Flask як мікрофреймворк означає, що ви можете використовувати його для неймовірно швидкого розвитку технологічного проекту, наприклад веб-додатка. Якщо ви хочете створити програму, яка починається з малого, але має потенціал для швидкого розвитку та в напрямках, які ви ще не повністю опрацювали, тоді це ідеальний вибір. Його простота у використанні та невелика кількість залежностей дозволяють йому працювати безперебійно, навіть якщо він масштабується все більше і більше.

б) Гнучкий.

Це основна функція Flask і одна з його найбільших переваг. Перефразовуючи один із принципів дзен Python, простота краща за складність, тому що її можна легко переставляти та переміщувати.

Це не тільки корисно з точки зору того, що дозволяє вашому проекту легко рухатися в іншому напрямку, це також гарантує, що конструкція не зруйнується, коли частина буде змінена. Мінімальний характер Flask і його здатність до розробки невеликих веб-додатків означає, що він навіть більш гнучкий, ніж сам Django.

в) Легко вести переговори.

Як і в Django, можливість легко орієнтуватися є ключовою для того, щоб веб-розробники могли зосередитися лише на швидкому написанні коду, не загрузнувши. За своєю суттю мікрофреймворк простий для розуміння для веб-розробників, він не тільки економить час і зусилля, але й дає їм більше контролю над своїм кодом і тим, що можливо.

г) Легкий.

Коли ми використовуємо цей термін по відношенню до інструменту чи фреймворку, ми говоримо про його дизайн – є кілька складових частин, які потрібно зібрати та повторно зібрати, і він не покладається на велику кількість розширень, щоб функціонувати . Цей дизайн дає веб-розробникам певний рівень контролю.

Flask також підтримує модульне програмування, де його функціональні можливості можна розділити на кілька взаємозамінних модулів. Кожен модуль діє

як незалежний будівельний блок, який може виконувати одну частину функціональності. Разом це означає, що всі складові частини конструкції є гнучкими, рухливими та перевіряються самостійно.

д) Документація.

Дотримуючись власної теорії творця про те, що «гарний дизайн документації змушує вас справді писати документацію», користувачі Flask знайдуть здорову кількість прикладів і порад, упорядкованих у структурований спосіб. Це спонукає розробників використовувати фреймворк, оскільки вони можуть легко познайомитися з різними аспектами та можливостями інструменту. Ви знайдете документацію Flask на їх офіційному веб-сайті.

Недоліки Flask:

а) Не так багато інструментів.

Неминуче є деякі недоліки легкої природи цього мікрофреймворку. Головним з них є те, що на відміну від Django, у Flask відсутній великий інструментарій. Це означає, що розробникам доведеться вручну додавати розширення, наприклад бібліотеки. І, якщо ви додасте величезну кількість розширень, це може почати гальмувати саму програму через велику кількість запитів.

б) Важко ознайомитися з більшою програмою Flask.

Через те, що розробка веб-програми за допомогою Flask може потребувати різноманітних поворотів, веб-розробник, який приходить до проекту на півдорозі, може важко зрозуміти, як він був розроблений. Модульний характер мікрофреймворку, про який ми згадували раніше, може знову зацікавити програмістів, яким доведеться ознайомитися з кожною складовою частиною.

в) Витрати на технічне обслуговування.

Оскільки він настільки універсальний у плані того, з якими технологіями він може взаємодіяти, компанія, яка використовує Flask, часто несе додаткові витрати на підтримку цих технологій. Наприклад, якщо технологія, яка взаємодіє з вашим додатком Flask, застаріє або припиниться, то компанії доведеться з усіх

сил шукати нову сумісну. Чим складнішою стає програма, тим вищі потенційні витрати на обслуговування та впровадження.

1.3.3 FastAPI

Переваги FastAPI [6]:

- висока продуктивність: FastAPI – це сучасна структура, заснована на асинхронному програмуванні, відома своєю чудовою продуктивністю та низькою затримкою;
- автоматичне створення документації: FastAPI може створювати інтерактивну документацію на основі вашого коду, покращуючи розробку API та ефективність тестування;
- підтримка анотацій типів: FastAPI підтримує використання анотацій типів для покращення читабельності та зручності обслуговування коду;

Недоліки FastAPI:

- відносно новий проект: FastAPI є відносно новим і може не мати зрілих рішень і підтримки спільноти в певних сферах;
- крива навчання: для розробників, які не мають досвіду асинхронного програмування, FastAPI може мати крутішу криву навчання.

1.3.4 Підсумок порівняння

Вибір фреймворку залежить від вимог проекту, досвіду розробника та вподобань. Незалежно від обраної структури, необхідно враховувати такі фактори, як масштаб проекту, ефективність розробки, продуктивність і ремонтопридатність.

Flask підходить для невеликих проектів і розробників, яким потрібна гнучкість фреймворку, Django підходить для створення складних веб-додатків і CMS, а FastAPI підходить для створення високопродуктивних програм реального часу. Виходячи з конкретних вимог, раціональний вибір рамки сприятиме успіху розробки.

Python є потужною та популярною мовою, яка підходить для розробки різних веб-додатків. Його переваги включають легкість навчання, велику кількість бібліотек і фреймворків, сильну підтримку спільноти та ефективну швидкість розробки. Однак відносно нижча продуктивність Python, обмеження GIL, керування залежностями та відносно молода підтримка асинхронного програмування можуть створити певні обмеження для деяких програм.

Тому, вибираючи Python як мову веб-розробки, важливо враховувати вимоги та характеристики проекту, а також такі фактори, як технічна команда, доступний час і ресурси.

1.4 Постановка задачі

Мета цієї дипломної роботи полягає у проведенні порівняльного аналізу ефективності використання різних фреймворків веб-розробки на продуктивність веб-додатку.

Для досягнення цієї мети необхідно вирішити наступні завдання:

1. Огляд теоретичних аспектів фреймворків Django, Flask та FastAPI. Вивчити архітектуру та основні принципи роботи кожного з фреймворків, а також дослідити переваги та недоліки кожного фреймворка.
2. Розробка тестових веб-додатків на основі кожного з фреймворків. Створити три аналогічні за функціоналом веб-додатки.
3. Проведення експериментальних досліджень для оцінки продуктивності. Визначити критерії для оцінки продуктивності веб-додатків та провести серію тестів для кожного веб-додатку, використовуючи обрані критерії.
4. Розробка рекомендацій щодо вибору фреймворка для різних типів веб-додатків. На основі проведеного аналізу надати рекомендації щодо використання того чи іншого фреймворка в залежності від конкретних вимог.
5. Оформлення результатів дослідження та висновки. Узагальнити отримані результати та зробити висновки щодо ефективності використання Django, Flask та FastAPI для розробки веб-додатків.

,

2 ТЕОРЕТИЧНІ ОСНОВИ ФРЕЙМВОРКІВ

2.1 Основні принципи роботи фреймворків

Django, як високорівневий фреймворк для веб-розробки, базується на кількох ключових принципах, які допомагають забезпечити консистентність, ефективність та гнучкість в розробці веб-додатків.

На найвищому рівні Django – це структура Model-View-Controller або MVC.

MVC – це шаблон проектування програмного забезпечення, метою якого є розділення веб-програми на три взаємопов’язані частини:

Модель, яка забезпечує інтерфейс з базою даних, що містить дані програми;

Перегляд, який вирішує, яку інформацію надати користувачеві, і збирає інформацію від користувача; і

Контролер, який керує бізнес-логікою програми та діє як інформаційний посередник між моделлю та представленням.

Django використовує дещо іншу термінологію у своїй реалізації MVC (див. рис. 2.1).

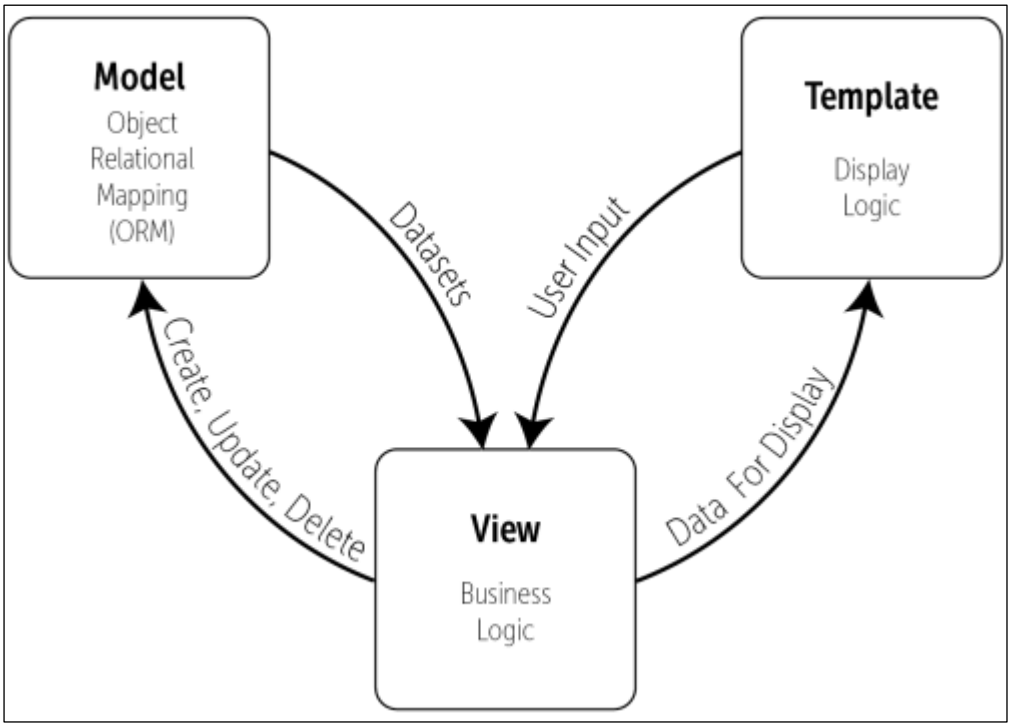


Рисунок 2.1 – Графічний опис шаблону Model-Template-View (MTV) у Django (за даними [7])

Функціонально модель однакова. Об’єктно-реляційне відображення Django (ORM – докладніше про ORM пізніше) забезпечує інтерфейс до бази даних програми;

Шаблон забезпечує логіку відображення та є інтерфейсом між користувачем і вашою програмою Django; і

Представлення керує більшою частиною обробки даних програм, логікою програми та обміном повідомленнями.

Шаблон дизайну MVC використовується як для настільних, так і для веб-додатків протягом багатьох років, тому існує велика кількість варіацій на цю тему, з яких Django не є винятком. Якщо ви бажаєте глибше заглибитися в шаблон проектування MVC, просто майте на увазі, що люди можуть бути дуже захоплені тим, що є різною інтерпретацією того самого.

Flask, як мікрофреймворк, ставить за основу простоту та гнучкість, дозволяючи розробникам вибирати та інтегрувати саме ті компоненти, які необхідні для їхнього проекту. Основні принципи роботи Flask включають.

Мінімалізм та Гнучкість.

Flask прагне до мінімалізму, надаючи лише базовий функціонал, необхідний для роботи. Це дозволяє розробникам вибирати компоненти та розширення згідно з конкретними потребами проекту.

Маршрутизація та Відображення.

Flask використовує декоратори для визначення URL-шляхів та пов'язаних із ними функцій (відображень).

Простота організації маршрутів робить код більш зрозумілим та легко розширюваним.

Шаблонізація та Jinja2.

Flask використовує шаблонізатор Jinja2 для відокремлення логіки та представлення.

Jinja2 дозволяє використовувати шаблони та вставляти змінні, щоб створювати HTML-сторінки.

Необов'язкове Використання ORM.

Flask не накладає конкретного ORM та бази даних. Розробник може вибрати ORM за своїм вибором або навіть обирати не використовувати його взагалі.

Розширюваність через Розширення.

Flask підтримує концепцію розширень, які дозволяють розробникам додавати функціонал та інтегрувати сторонні компоненти легко.

Інтеграція з WSGI та Веб-серверами.

Flask побудований на основі WSGI (Web Server Gateway Interface), що робить його сумісним з різними веб-серверами.

Дозволяє легко переносити та розгортати додатки.

Необов'язкове Використання Статичних Файлів.

Flask не накладає конкретних обмежень на роботу зі статичними файлами, дозволяючи розробникам вибирати та організовувати їх за власними правилами.

Вбудовані Інструменти безпеки.

Flask включає вбудовані інструменти для захисту від атак, таких як Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF).

Flask ставить на свободу розробників, дозволяючи їм будувати власну архітектуру за конкретними потребами проекту. Це зробило його популярним в середовищі розробників, які шукають гнучкий та простий у використанні інструмент для створення веб-додатків.

FastAPI, як сучасний фреймворк для веб-розробки на мові програмування Python, має свої унікальні принципи та особливості, які роблять його ефективним та високопродуктивним. Основні принципи роботи FastAPI включають:

Асинхронність.

FastAPI активно використовує асинхронне програмування, що дозволяє обробляти багато запитів одночасно та підвищує швидкодію застосунку.

Використання асинхронності робить FastAPI ідеальним для високонавантажених додатків.

Автоматична Документація.

FastAPI автоматично генерує документацію API на основі аннотацій типів та метаданих.

Використання Swagger та ReDoc дозволяє зручно переглядати та тестувати API.

Аннотації Типів Даних.

Використання аннотацій типів для параметрів та результатів дозволяє FastAPI автоматично генерувати схеми та перевіряти дані в runtime.

Це сприяє автоматизації та зменшенню кількості помилок в коді.

Заголовки та Кукізи.

FastAPI надає спеціальні об'єкти для роботи з HTTP-заголовками та кукізами, що дозволяє легко та безпечно взаємодіяти з ними.

Залежності та Впорядковані Виклики.

Використання системи залежностей та впорядкованих викликів дозволяє визначати та управляти порядком виконання функцій, які обробляють запити.

Валідація та Схеми Даних.

FastAPI використовує підходи OpenAPI та JSON Schema для валідації та документації даних, які надходять та повертаються API.

Використання Стандартних Інструментів Python.

FastAPI активно використовує стандартні інструменти Python, такі як типи даних та аннотації, а їхній високий рівень взаємодії з типами дозволяє розробникам писати більш безпечний та зрозумілий код.

Швидкодія та Висока Продуктивність.

З використанням асинхронності, компіляції до статичного коду та інших оптимізацій, FastAPI забезпечує високу продуктивність для веб-додатків.

Ці принципи дозволяють FastAPI надавати високопродуктивний та зручний для використання інструмент для створення веб-додатків з високопродуктивним API.

2.2 Архітектура Django, Flask та FastAPI

2.2.1 Архітектура Django

Архітектура Django відповідає структурі MVT (див. рис. 2.2). У MVT M означає модель, V означає представлення, а T означає шаблони. Модель – це

структура зберігання даних у базі даних, представлення – це функція Python, яка використовується для обробки веб-запиту, а шаблон містить статичний вміст, наприклад HTML, CSS і Javascript.

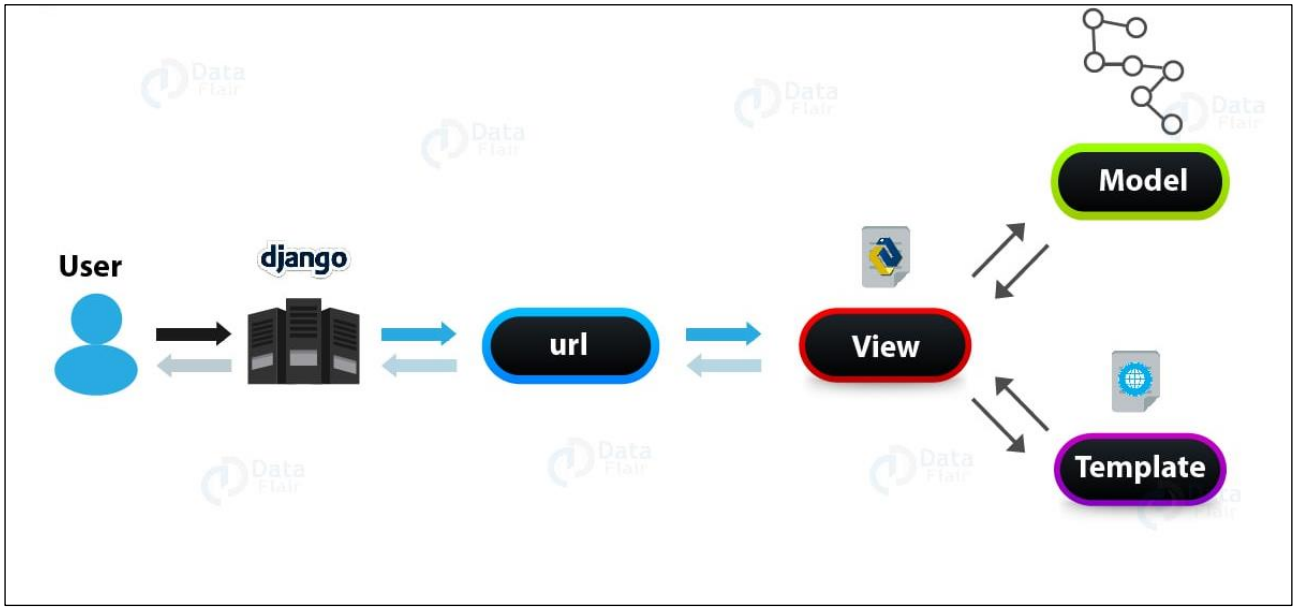


Рисунок 2.2 – Архітектура Django (за даними [8])

Програми Django використовують об’єкти Python, які називаються моделями, для доступу та керування даними. Ці моделі визначають внутрішню структуру даних, що зберігаються, як-от типи полів та їх максимальний розмір, значення за замовчуванням, текст міток для форм тощо. Створення моделей не залежить від того, яку базу даних ви використовуєте. Ви можете використовувати будь-яку базу даних. Вам не потрібно безпосередньо спілкуватися з базою даних. Вам потрібно лише створити структуру моделі та інший код, а Django подбає про брудну роботу спілкування з базою даних.

Функція перегляду в Django – це функція Python, яка приймає веб-запит і надає веб-відповідь. Відповіддю у представленнях Django може бути все, що може показати веб-браузер, включаючи HTML веб-сторінки, перенаправлення, XML-документ, помилку 404, зображення тощо. Представлення в Django є частиною інтерфейсу користувача в Програма Django, оскільки вони повертають веб-сторінки, які містять HTML/CSS/JAVASCRIPT у шаблоні Django.

Шаблон у Django містить статичний вміст проекту Django, наприклад Html, CSS і Javascript, а також зображення, яке використовується в проекті. Ми можемо встановити шлях до шаблону Django з файлу setting.py проекту Django.

2.2.2 Архітектура Flask

З архітектурної точки зору Flask Monitoring Dashboard використовує патерн "Шари". Існують 3 шари: шар даних, шар логіки, шар презентації. Кожен з них детально розглядається в цьому розділі.

На наведеній нижче діаграмі показано (див. рис. 2.3), як FMD взаємодіє з відстежуваною програмою. Ми вважали, що програма використовує базу даних та веб-інтерфейс, але ці компоненти не є обов'язковими. Крім того, FMD DB може бути такою ж, як БД програми.

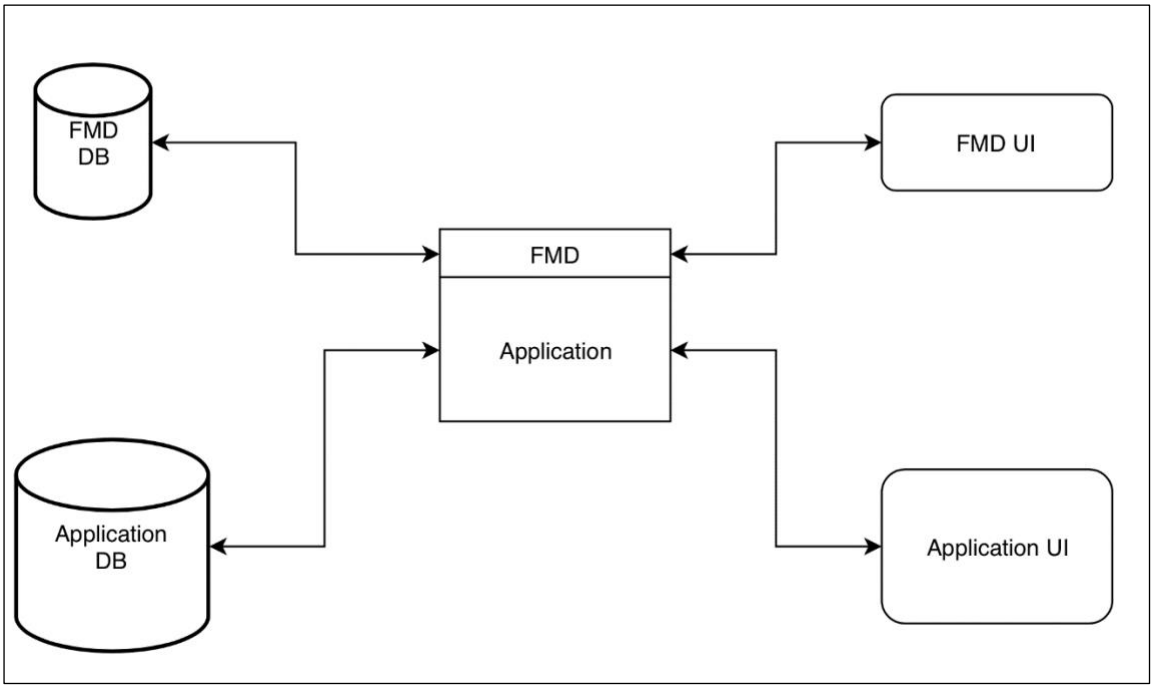


Рисунок 2.3 – Архітектура Flask (за даними [9])

Архітектура Flask є гнучкою і не обмежує розробників у конкретних шаблонах організації коду. Основні компоненти архітектури включають маршрутизацію, шаблонізацію, модель (яку розробник може обрати за власним вибором), відображення (або контролери, які обробляють запити та повертають

відповіді), інтерфейси зовнішніх сервісів через HTTP або інші протоколи, а також розширення та middleware для розширення функціоналу.

Хоча Flask дозволяє створювати додатки згідно з власними вподобаннями, у великих проектах може виникнути необхідність в структуризації коду. Розробники можуть обирати різні шаблони організації коду, такі як "Blueprints" або застосовувати підходи, що виникають з патернів проектування, для покращення організації та підтримки масштабування проектів.

2.2.3 Архітектура FastAPI

З архітектурної точки зору FastAPI використовує сучасний та ефективний дизайн, базуючись на ключових принципах для створення надійного фреймворку для веб-розробки (див. рис. 2.4).

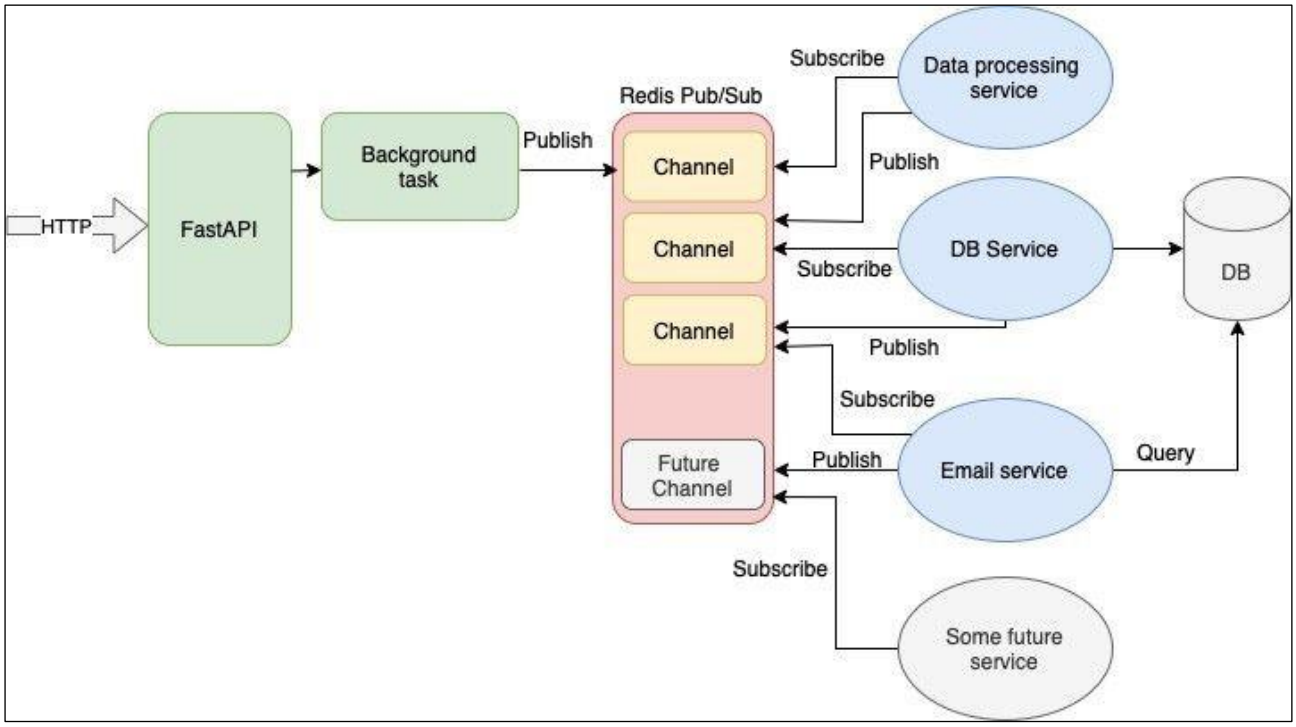


Рисунок 2.4 – Архітектура FastAPI (за даними [10])

Архітектура FastAPI впливає на асинхронне програмування, підказки типів та автоматичну документацію OpenAPI. Нижче наведено основні аспекти архітектури FastAPI:

Асинхронний Дизайн.

FastAPI розроблено з урахуванням асинхронного програмування, використовуючи ключові слова async та await. Це дозволяє обробляти кілька запитів одночасно і робить його придатним для високопродуктивних застосунків.

Система Внедрення Залежностей.

У FastAPI існує вдосконалена система внедрення залежностей. Залежності, такі як підключення до бази даних чи налаштування, можуть бути вбудовані в функції маршрутів, роблячи код більш модульним і тестовим.

Pydantic для Валідації Даних.

FastAPI широко використовує Pydantic для валідації даних. Моделі Pydantic використовуються для оголошення структур даних, виконання валідації введення та автоматичної генерації документації OpenAPI. Це забезпечує консистентність даних та зменшує ризик помилок.

Автоматична Документація OpenAPI.

Однією з основних особливостей FastAPI є здатність автоматично генерувати документацію OpenAPI та JSON Schema. Це базується на типових підказках, наданих у сигнатурі функцій та моделях Pydantic.

Маршрутизація та Внедрення Залежностей.

FastAPI використовує підхід, заснований на декораторах, для визначення маршрутів, аналогічно Flask. Проте він розширює це, безперешкодно інтегруючись із системою внедрення залежностей. Залежності можна оголошувати в функціях маршрутів, і FastAPI відповідає за їхнє забезпечення під час виконання.

Засоби Безпеки.

FastAPI включає засоби безпеки з коробки, включаючи захист від типових вразливостей веб-додатків. Він обробляє валідацію введення, санітарізацію та захист від загроз безпеки, таких як SQL-ін'єкції, міжсайтові скриптові атаки (XSS) та підроблення запитів міжсайтових атак (CSRF).

Підтримка OAuth2 та JWT.

FastAPI надає вбудовану підтримку OAuth2 та JSON Web Tokens (JWT), що дозволяє розробникам легко захищати свої API та реалізовувати механізми авторизації.

Фонові Завдання та Роботи.

FastAPI підтримує фонові завдання та роботи, які можуть виконуватися асинхронно. Це корисно для обробки завдань, які можна відкласти, таких як відправка електронної пошти чи обробка даних поза циклом основних запитів-відповідей.

Підтримка WebSocket.

FastAPI включає підтримку WebSocket, що дозволяє розробникам створювати додатки реального часу, обробляючи з'єднання WebSocket поруч з HTTP-запитами.

Інтеграція із Стандартними Типами Python.

FastAPI безперешкодно інтегрується зі стандартними типами та бібліотеками Python, включаючи підтримку різних типів даних, таких як списки, словники та користувацькі типи.

Загалом, архітектура FastAPI призначена для забезпечення високої продуктивності та ефективного досвіду розробки, з акцентом на типовій безпеці, автоматичній документації та підтримці сучасних практик веб-розробки.

3 ВІДМІННОСТІ ПРОЦЕСУ РОЗРОБКИ

3.1 Етапи типового процесу розробки

Типовий процес розробки сайтів ділиться на декілька складових. Одна з найважливіших складових – це архітектура, яка визначає загальний вигляд та функціональність будь-якої структури або системи [10]. Наступна складова являє собою шаблони сторінок сайту, наприклад, шаблон для сторінок всіх користувачів. Далі, використовуючи створені шаблони, створюються сторінки сайту, наповнюються контентом та реалізується базовий функціонал деяких елементів. Після чого, за допомогою маршрутизації, можна розробити навігацію між сторінками. Далі додається функціонал взаємодії бази даних з сайтом, наприклад, вхід до системи, отримання контенту з бази даних, його додавання, видалення чи зміна. В цьому допоможуть складові представлення, форм та підключення до БД. Схему процесу показано на рисунку 3.1.

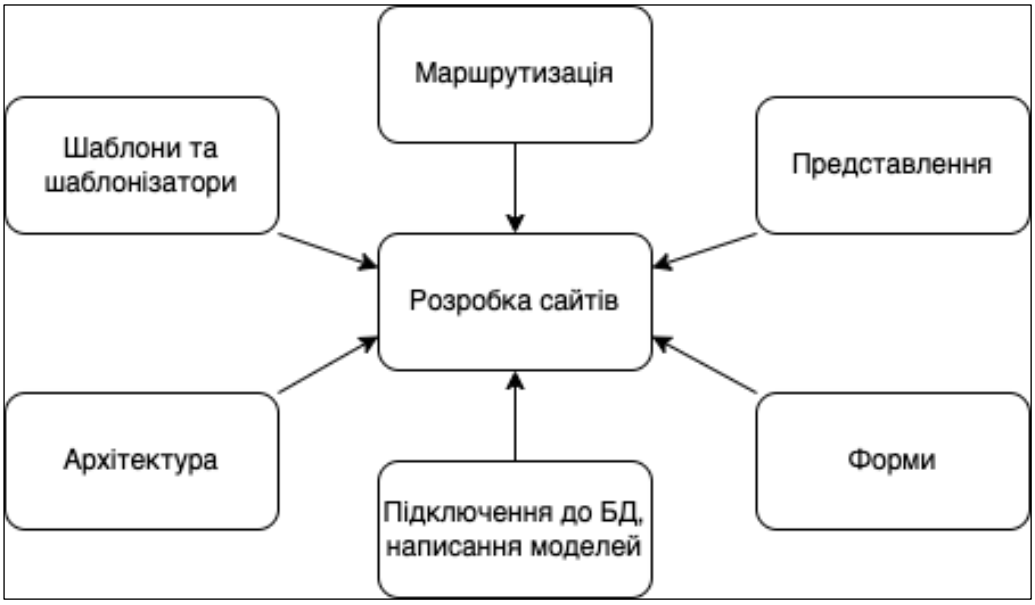


Рисунок 3.1 – Схема типового процесу розробки сайтів

3.2 Спосіб підключення фреймворку Django

Першим кроком у створенні проекту з використанням Django є його налаштування та створення нового проекту. Існує кілька способів, запропонованих в офіційній документації (див. рис. 3.2), але ми зосередимося на

найпопулярнішому і найпростішому методі – використанні інструменту pip, який служить для встановлення Python-пакетів [11].

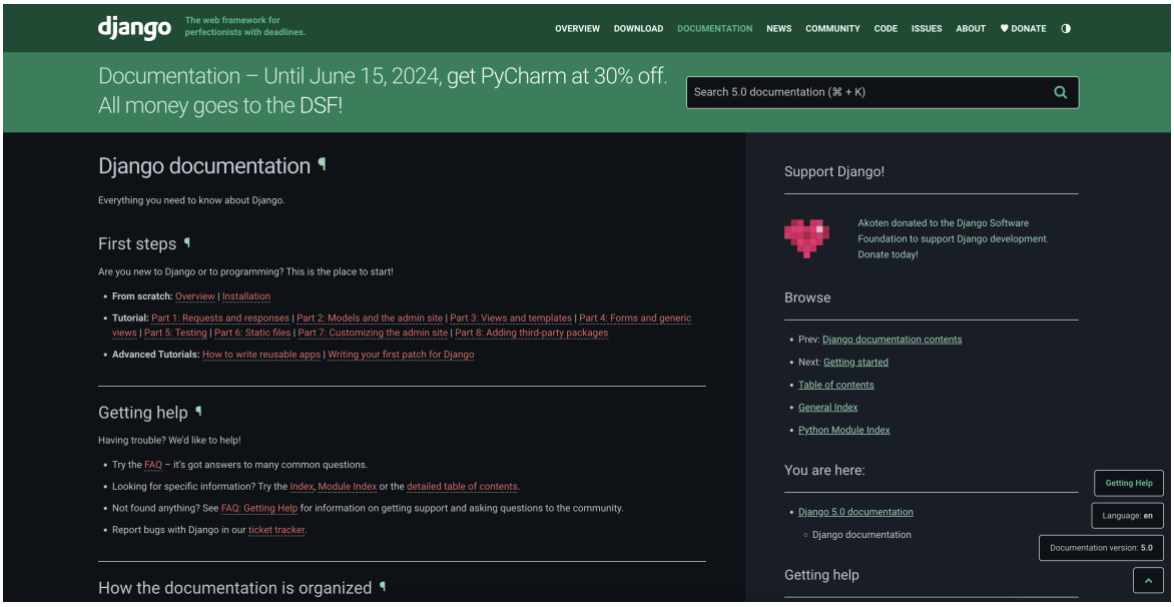


Рисунок 3.2 – Сторінка документації Django

Django – це високорівневий веб-фреймворк для Python, призначений для швидкої розробки та чистого, прагматичного дизайну. Він має безліч вбудованих функцій для створення веб-додатків, таких як ORM для роботи з базами даних, система шаблонів для створення динамічних HTML-сторінок, маршрутизація URL, система аутентифікації і багато іншого. Django значно спрощує процес розробки складних веб-додатків, забезпечуючи високу продуктивність і безпеку.

Раніше веб-розробка з використанням Python була досить складною і вимагала багато налаштувань. Django спрощує цей процес, пропонуючи набір інструментів і бібліотек для швидкого і ефективного створення веб-додатків. З Django можна розробляти складні та інтерактивні веб-сайти з меншими витратами часу і коду порівняно з іншими фреймворками або чистим Python.

Таким чином, Django розширює можливості Python для веб-розробки, подібно до того, як Node.js розширив можливості JavaScript, дозволяючи запускати його на сервері.

pip – менеджер пакетів, що входить до складу Python. Встановлення необхідних пакетів здійснюється за допомогою команди в терміналі: pip install

<packagename>. Усі доступні для встановлення пакети та їх короткий опис можна знайти на PyPI (Python Package Index) (рис 3.3). Пошук пакетів здійснюється командою: pip search <packagename> [12].

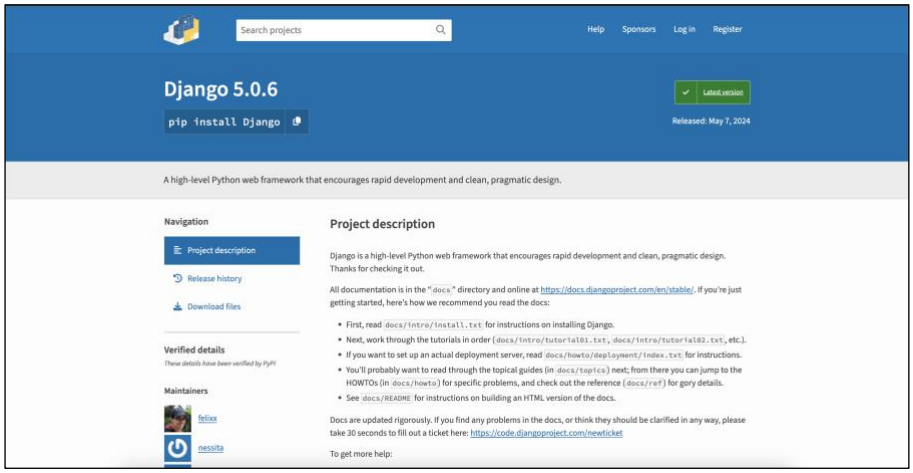


Рисунок 3.3 – Сторінка Django на PyPI

Для початку роботи потрібно встановити актуальну версію Python з офіційного сайту. Після встановлення Python, pip буде автоматично доданий до вашої системи.

На macOS термінал можна відкрити за допомогою Spotlight Search (Cmd + Space), ввівши "Terminal" і натиснувши Enter (рис 3.4). Розглянемо процес встановлення Django в проект. Спочатку потрібно перейти до офіційної документації Django і знайти розділ "Getting Started", де вказана команда для встановлення фреймворку, pip install Django.

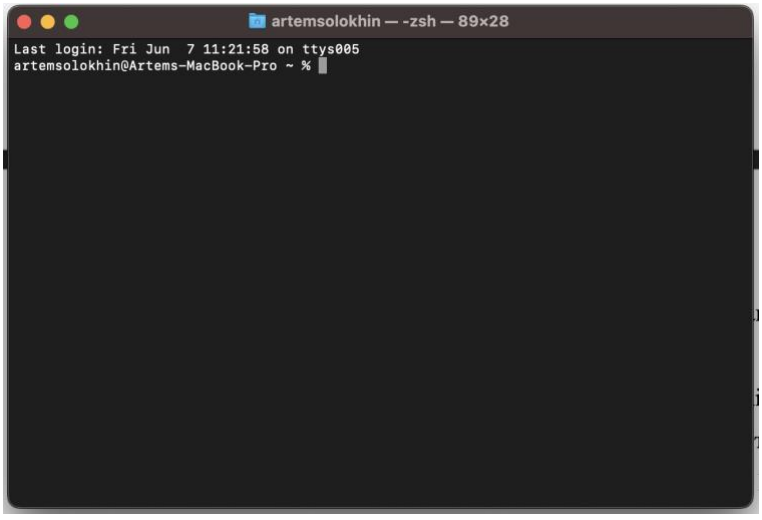


Рисунок 3.4 – Консоль

Ця команда встановить Django разом з усіма необхідними залежностями. Після успішного встановлення Django, можна створити новий проект за допомогою команди: django-admin startproject myproject. Ця команда створить нову директорію з базовою структурою проекту Django, включаючи конфігураційні файли та налаштування (рис 3.5).

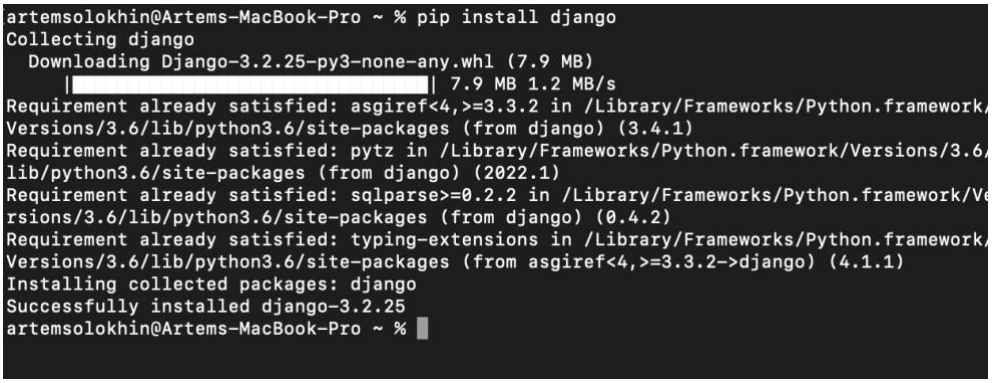


Рисунок 3.5 – Процес установки Django

Тепер, для створення нового проекту у обраній папці, необхідно виконати команду django-admin startproject <назва_проекту>. Ця команда автоматично створить папку з вказаною назвою за заданим шляхом та додасть до неї всі необхідні файли для подальшого розроблення сайту. Під час створення проекту будуть згенеровані конфігураційні файли, файли для налаштування бази даних, а також базова структура директорій (рис 3.6).

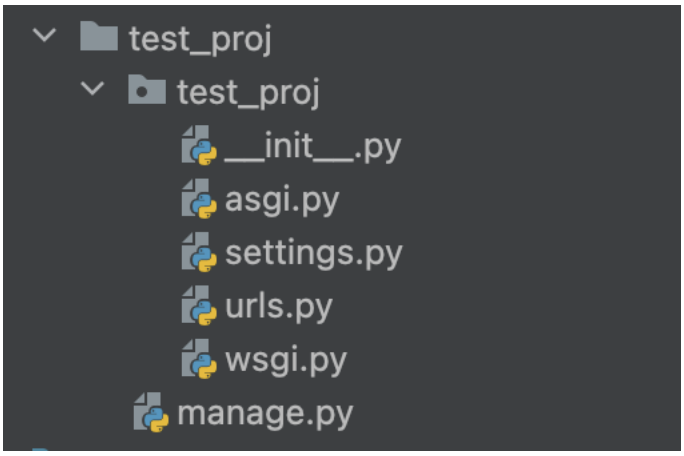


Рисунок 3.6 – Структура створених папок

Наступним етапом потрібно перейти до новоствореної папки у терміналі, щоб запустити локальний сервер. Перейти до директорії можна виконати за

допомогою команди cd <назва_проекту>. Для запуску локального сервера виконайте команду python manage.py runserver. Після виконання цієї команди, локальний сервер буде запущено, і він буде доступний за вказаною адресою у вашому браузері (рис. 3.7 - 3.8). На цій адресі ви зможете переглядати ваш проект у режимі розробки (рис. 3.9).

```
artemsolokhin@Artems-MacBook-Pro diplom % cd test_proj
artemsolokhin@Artems-MacBook-Pro test_proj % python manage.py runserver
```

Рисунок 3.7 – Команда запуску локального серверу

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 07, 2024 - 21:22:42
Django version 3.2.5, using settings 'test_proj.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Рисунок 3.8 – Результат

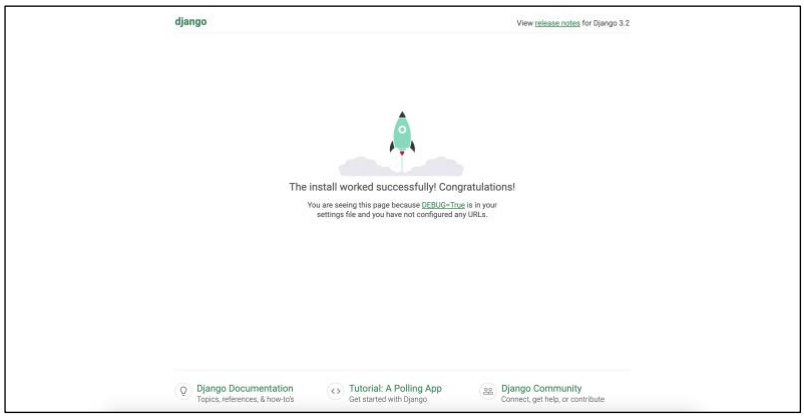


Рисунок 3.9 – Сторінка в браузері розташована в локальному сервері

3.3 Спосіб підключення фреймворку Flask

Для включення фреймворку Flask до структури проекту слід реалізувати послідовні дії, аналогічні тим, які застосовуються при інтеграції фреймворку Django, до етапу відкриття консолі. У випадку з Flask, необхідно виконати наступні кроки.

У першу чергу, нам треба інсталювати фреймворк Flask за допомогою команди pip install Flask.

Після того, як фреймворк інстальовано, створимо новий проект, перейшовши до необхідної директорії та створивши основний файл із кодом програми (рис. 3.10).

```
artemsolokhin@Artems-MacBook-Pro diplom % mkdir flask
artemsolokhin@Artems-MacBook-Pro diplom % cd flask
artemsolokhin@Artems-MacBook-Pro flask % touch app.py
```

Рисунок 3.10 – Команди для створення проекту на Flask

Далі, нам слід відкрити файл `app.py` та впровадити базовий код для ініціалізації сервера (рис. 3.11).

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def hello_world():
    return 'Hello, World!'


if __name__ == '__main__':
    app.run()
```

Рисунок 3.11 – Приклад коду на Flask

Даний код формує невеликий веб-сервер, який обробляє запити на адресу `/` повідомленням "Hello, world!". Це є лише вихідним кроком у роботі з Flask, і додаткові функціональні можливості будуть включені згідно з вимогами проекту.

3.4 Спосіб підключення фреймворку FastAPI

Підключення фреймворку FastAPI до структури проекту включає в себе кілька кроків, які дозволяють належним чином інтегрувати цей фреймворк у розроблювану систему.

Першим кроком є інсталяція фреймворку FastAPI за допомогою інструменту управління пакетами Python, такого як `pip`. Це можна зробити за допомогою наступної команди в терміналі pip install fastapi.

FastAPI використовує ASGI (Asynchronous Server Gateway Interface) сервер для обробки запитів. Для запуску FastAPI потрібно встановити ASGI сервер, такий як Uvicorn. Це можна зробити таким чином pip install uvicorn [13].

Uvicorn є сучасним, високопродуктивним ASGI сервером, призначеним для запуску асинхронних веб-додатків у фреймворках, таких як FastAPI. Він забезпечує швидкий та ефективний спосіб обробки HTTP запитів, підтримуючи як синхронні, так і асинхронні операції. Цей розділ описує основні характеристики та переваги Uvicorn, а також його використання з FastAPI.

ASGI (Asynchronous Server Gateway Interface) сервери є невід'ємною частиною сучасної веб-розробки на Python, особливо для створення асинхронних веб-додатків. ASGI сервери розширюють можливості традиційних WSGI серверів, дозволяючи обробляти не лише синхронні, але й асинхронні запити. У цьому розділі ми розглянемо, що таке ASGI сервер, його переваги та основні принципи роботи.

Після успішної інсталяції FastAPI та ASGI сервера, необхідно створити файл, в якому буде знаходитися основний код нашого веб-додатку. Створимо новий файл за допомогою команди touch main.py. Відкриймо файл `main.py` у текстовому редакторі та додамо код (рис. 3.12).

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def read_root():
    return {"message": "Hello, world!"}
```

Рисунок 3.12 – Код сторіни на FastAPI

У цьому коді створюється FastAPI додаток, який відповідає на GET запити за адресою `/` повідомленням "Hello, world!".

Для запуску FastAPI сервера використовуйте наступну команду uvicorn main:app –reload. Ця команда запускає ASGI сервер Uvicorn і відслідковує зміни у вашому коді за допомогою параметру `--reload`.

3.5 Відмінності процесу розробки в Django

Однією з головних відмінностей Django є його архітектура, яка слідує принципу "все в одному" (batteries-included). Це означає, що Django поставляється з великою кількістю вбудованих інструментів та бібліотек, необхідних для створення повноцінного веб-додатку, включаючи систему автентифікації, ORM (Object-Relational Mapping), адміністративну панель, формування шаблонів та маршрутизацію.

Django надає готову до використання систему автентифікації, яка включає моделі користувачів, форми для реєстрації та входу, а також механізми управління сесіями. Для додавання автентифікації достатньо налаштувати файл settings.py і включити вбудовані URL-маршрути (рис 3.13).

```python
# urls.py
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('django.contrib.auth.urls')),
]
```

Рисунок 3.13 – Приклад використання системи автентифікації

Також Django включає потужний ORM, що дозволяє взаємодіяти з базою даних за допомогою Python-класів. Це зменшує потребу у написанні SQL-запитів вручну. Django ORM (Object-Relational Mapping) є одним із найпотужніших та найзручніших інструментів, які пропонує фреймворк Django для роботи з базами даних. ORM дозволяє розробникам взаємодіяти з базою даних за допомогою Python-класів і об'єктів, абстрагуючись від написання SQL-запитів вручну. Це сприяє більш швидкому і безпечному розвитку додатків.

ORM не тільки надає можливість підключитись до бази даних, а й створити моделі. Моделі у Django представляють структуру таблиць у базі даних. Кожен клас у файлі models.py відповідає окремій таблиці, а кожне поле класу – колонці

таблиці. Визначення моделей здійснюється за допомогою класів, що наслідують від django.db.models.Model (рис 3.14).

```python
from django.db import models


class Article(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    published_date = models.DateTimeField(auto_now_add=True)

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

Рисунок 3.14 – Приклад моделі в Django

Поля моделі визначають тип даних, які зберігаються в таблиці бази даних. Django ORM підтримує різноманітні типи полів, включаючи CharField, TextField, IntegerField, BooleanField, DateTimeField та інші.

Для того щоб застосувати зміни до бази даних треба використати міграції. Міграції є механізмом для створення і модифікації таблиць бази даних відповідно до визначень моделей. Django автоматично генерує файли міграцій на основі змін у моделях, що дозволяє легко керувати схемою бази даних (рис 3.15).

```
python manage.py makemigrations
python manage.py migrate
```

Рисунок 3.15 – Команди для створення і застосування міграцій

Django ORM дозволяє виконувати запити до бази даних за допомогою об'єктів моделей. Основні методи включають save(), delete(), filter(), get(), all() та інші (рис. 3.16). Також ORM підтримує різні типи відносин між моделями, такі як один-до-одного (OneToOneField), один-до-багатьох (ForeignKey) та багато-до-багатьох (ManyToManyField).

```
article = Article(title="New Article", content="Content of the article")
article.save()

articles = Article.objects.all()

filtered_articles = Article.objects.filter(title__icontains="Article")

single_article = Article.objects.get(id=1)

article.title = "Updated Title"
article.save()

article.delete()
```

Рисунок 3.16 – Приклад використання запитів в Django

Ще, з вбудованих модулів, Django має Адміністративну панель, яка є одним із ключових компонентів фреймворку, яка надає зручний веб-інтерфейс для управління даними додатку. Вона автоматично генерується на основі моделей, визначених у вашому проекті, і дозволяє адміністраторам виконувати CRUD-операції (створення, читання, оновлення, видалення) з даними без написання коду для цих операцій.

Для додавання моделей до адміністративної панелі необхідно зареєструвати їх у файлі admin.py. Це можна зробити за допомогою функції admin.site.register (рис 3.17).

```
from django.contrib import admin
from models import Article

admin.site.register(Article)
```

Рисунок 3.17 – Приклад додавання моделі до адміністративної панелі

Також Django надає можливість кастомізації відображення моделей у адміністративній панелі за допомогою класів ModelAdmin. Це дозволяє налаштовувати різні аспекти інтерфейсу, такі як поля, що відображаються у списку, фільтри, пошукові поля та інші (рис 3.18).

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


1 usage
class ArticleAdmin(admin.ModelAdmin):
    list_display = ('title', 'author', 'published_date')
    list_filter = ('author', 'published_date')
    search_fields = ('title', 'content')


admin.site.register(Article, ArticleAdmin)
```

Рисунок 3.18 – Приклад кастомізації відображення моделей

Адміністративна панель Django дуже гнучка і може бути значно розширена за допомогою спеціальних плагінів, власних віджетів, кастомних дій та іншого. Це дозволяє створювати інтерфейси, що відповідають конкретним потребам бізнесу.

Наступним аспектом розробки сайтів є шаблони. Шаблони (templates) в Django є важливим компонентом для створення динамічних веб-сторінок. Вони дозволяють відокремити бізнес-логіку від презентаційного шару, що робить код більш чистим і зручним для підтримки. Шаблони написані з використанням HTML з вбудованими тегами та фільтрами шаблонів Django, які дозволяють динамічно відображати дані.

Шаблони зазвичай зберігаються у директорії templates всередині додатків або в загальній директорії проекту. Це дозволяє Django автоматично знаходити їх під час рендерингу. Для створення шаблону необхідно створити HTML-файл у відповідній директорії (рис 3.19).

```html
<!DOCTYPE html>
<html>
<head>
    <title>{{ article.title }}</title>
</head>
<body>
    <h1>{{ article.title }}</h1>
    <p>{{ article.content }}</p>
    <p><small>Published on: {{ article.published_date }}</small></p>
</body>
</html>
```

Рисунок 3.19 – Приклад html-файлу в Django

Шаблони Django використовують спеціальні теги та фільтри для відображення даних та виконання логічних операцій. Теги використовуються для виконання логічних операцій, циклів, включення інших шаблонів тощо (рис 3.20). Фільтри дозволяють змінювати вигляд відображуваних даних. Наприклад, форматування дати (рис 3.21). Для складних випадків можна використовувати кастомні теги та фільтри. Їх можна визначити у файлах Python.

Рисунок 3.20 – Приклад використання тега

Рисунок 3.21 – Приклад використання фільтра

Також Django підтримує наслідування шаблонів, що дозволяє створювати базові шаблони, які інші шаблони можуть наслідувати. Це сприяє повторному використанню коду та підтримці єдиного стилю. А іноді є необхідність включити один шаблон у інший для повторного використання частин коду, таких як навігаційні меню або форми.

Щоб передати дані до шаблону і відрендерити його, необхідно використовувати функції або класові представлення у файлі views.py.

Щоб відрендерити та відобразити шаблоні на сторінці в браузері нам потрібні предсставлення. Представлення в Django відповідають за обробку HTTP-запитів і повернення HTTP-відповідей. Вони грають роль зв'язуючого елемента між моделями і шаблонами, забезпечуючи логіку бізнес-процесів додатка. Django підтримує два основні підходи до створення представлень: функціональні представлення (FBV - Function-Based Views) і класові представлення (CBV - Class-Based Views).

Функціональні представлення представляють собою звичайні Python-функції, які приймають об'єкт запиту (HttpRequest) і повертають об'єкт відповіді

(HttpResponse) (рис. 3.22). Вони надають простий і прямий спосіб визначення поведінки додатка.

```python
from django.http import HttpResponse


def hello_world(request):
    return HttpResponse("Hello, World!")
```

Рисунок 3.22 – Приклад використання функціональних представлень

Функціональні представлення можуть використовувати шаблони для рендерингу HTML-сторінок. Використовується функція render, яка поєднує шаблон з даними контексту і повертає об'єкт HttpResponse (рис. 3.23).

```python
from django.shortcuts import render


def article_detail(request, article_id):
    article = Article.objects.get(id=article_id)
    return render(request, template_name: 'article_detail.html', context: {'article': article})
```

Рисунок 3.23 – Використання шаблона в функціональному представлені

Класові представлення забезпечують більш структурований підхід до визначення поведінки додатка, розділяючи логіку на методи класів. Вони надають зручні засоби для повторного використання коду і є більш гнучкими та масштабованими (рис 3.24).

```python
from django.http import HttpResponse
from django.views import View


class HelloWorldView(View):
    1 usage (1 dynamic)
    def get(self, request):
        return HttpResponse("Hello, World!")
```

Рисунок 3.24 – Приклад використання класового представлення

Для використання шаблонів у класових представленнях можна скористатися вбудованими класами, такими як TemplateView, або створити власні представлення на основі класів (рис. 3.25).

```python
from django.views.generic import TemplateView


class ArticleDetailView(TemplateView):
    template_name = 'article_detail.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['article'] = Article.objects.get(id=kwargs['article_id'])
        return context
```

Рисунок 3.25 – Приклад використання шаблона

Django надає набір вбудованих класових представлень, які дозволяють швидко створювати стандартні представлення, такі як детальні представлення, списки об'єктів, форми тощо. DetailView використовується для відображення деталей одного об'єкта. ListView використовується для відображення списку об'єктів. CreateView забезпечує форму для створення нового об'єкта. UpdateView забезпечує форму для оновлення існуючого об'єкта. DeleteView забезпечує інтерфейс для видалення об'єкта.

Для зв'язування представлень з URL-адресами використовується модуль urls.py. URL-паттерни визначають, яке представлення буде обробляти той чи інший запит (рис. 3.26).

```python
urlpatterns = [
    path('article/<int:pk>/', ArticleDetailView.as_view(), name='article_detail'),
    path('articles/', ArticleListView.as_view(), name='article_list'),
    path('article/new/', ArticleCreateView.as_view(), name='article_new'),
    path('article/<int:pk>/edit/', ArticleUpdateView.as_view(), name='article_edit'),
    path('article/<int:pk>/delete/', ArticleDeleteView.as_view(), name='article_delete'),
]
```

Рисунок 3.26 – Приклад URL-конфігурації

Роутинг в Django визначає, як URL-запити на веб-сервері зв'язуються з конкретними представленнями. Це важливий аспект, який дозволяє правильно обробляти запити та повертати відповідні відповіді. Параметри URL дозволяють передавати змінні до представлень. Вони визначаються у фігурних дужках.

Також в Django наявні вбудовані форми, які є потужним інструментом для обробки вводу користувачів. Вони використовуються для створення, редагування та валідації даних. Django надає зручний механізм для роботи з формами, який включає як визначення форм, так і їх рендеринг та обробку.

Форми в Django створюються як класи, що успадковують від django.forms.Form або django.forms.ModelForm для форм, пов'язаних з моделями. Звичайні форми створюються на основі класу Form і використовуються для збору даних, які не обов'язково пов'язані з моделями (рис. 3.27). Форми на основі моделей (ModelForm) автоматично генеруються на основі моделі Django, що дозволяє легко створювати та редагувати екземпляри моделей.

Форми можуть бути відображені в шаблонах за допомогою різних методів, включаючи ручне рендеринг полів та використання вбудованих тегів шаблонів Django. {{ form.as_p }} рендерить форму з кожним полем у абзаці (<p>). Можна також використовувати form.as_table або form.as_ul для рендерингу полів у вигляді таблиці або списку відповідно.

Після надсилання форми дані необхідно обробити у представленнях, перевірити їх валідність та вжити відповідних дій.

```python
from django import forms


class ContactForm(forms.Form):
    name = forms.CharField(max_length=100)
    email = forms.EmailField()
    message = forms.CharField(widget=forms.Textarea)
```

Рисунок 3.27 – Приклад форми в Django

Django автоматично забезпечує валідацію даних на основі типів полів. Можна також додавати власні методи валідації.

Django надає всеосяжний набір інструментів та бібліотек, що робить процес розробки більш швидким та ефективним. Його архітектура "все в одному" дозволяє швидко розгортати проекти з мінімальними налаштуваннями. У той час як інші фреймворки, такі як Flask та FastAPI, пропонують більшу гнучкість та можливості для налаштування, Django забезпечує комплексне рішення для швидкої розробки потужних веб-додатків.

3.6 Відмінності процесу розробки в Flask

Flask є мікрофреймворком для розробки веб-додатків на Python. На відміну від Django, який є повноцінним фреймворком з великою кількістю вбудованих функцій, Flask надає більш мінімалістичний підхід, залишаючи розробникам більшу гнучкість у виборі компонентів і архітектурних рішень [14].

У Django структура проекту суворо визначена: є каталоги для додатків, шаблонів, статичних файлів і конфігурацій. Flask, навпаки, не нав'язує жорсткої структури, дозволяючи розробникам організовувати проект відповідно до власних уподобань і потреб (3.28).

```
my_flask_app/
    app/
        __init__.py
        views.py
        models.py
        templates/
        static/
    config.py
    run.py
```

Рисунок 3.28 – Приклад структури проекту на Flask

Flask, будучи мікрофреймворком, не включає вбудованого ORM (Object-Relational Mapping) або інших інструментів для роботи з базами даних. Натомість, розробники можуть вибирати з широкого спектру сторонніх бібліотек і розширень для інтеграції з базами даних. Однією з найпопулярніших бібліотек для роботи з базами даних у Flask є SQLAlchemy.

SQLAlchemy є потужним ORM, який забезпечує гнучкість і ефективність роботи з реляційними базами даних. Flask-SQLAlchemy – це розширення для Flask, яке спрощує інтеграцію SQLAlchemy з Flask-додатками.

Після встановлення розширення необхідно налаштувати його у проекті (рис. 3.29).

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
db = SQLAlchemy(app)


if __name__ == '__main__':
    app.run(debug=True)
```

Рисунок 3.29 – Приклад налаштування БД

Моделі визначають структуру таблиць у базі даних. Вони створюються як класи, які успадковують від db.Model. Кожна модель містить атрибути, що представляють колонки в таблиці бази даних. Атрибути мають типи даних, визначені SQLAlchemy, такі як db.String, db.Integer, db.DateTime тощо (рис. 3.30).

Після визначення моделей необхідно створити відповідні таблиці в базі даних. Це робиться за допомогою методу create_all().

Після налаштування моделей і ініціалізації бази даних можна почати використовувати ці моделі для створення, читання, оновлення та видалення записів у базі даних.

```python
from datetime import datetime
from app import db


class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
    password = db.Column(db.String(60), nullable=False)
    posts = db.relationship('Post', backref='author', lazy=True)

    def __repr__(self):
        return f"User('{self.username}', '{self.email}', '{self.image_file}')"


class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    content = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

    def __repr__(self):
        return f"Post('{self.title}', '{self.date_posted}')"
```

Рисунок 3.30 – Приклад визначення моделей

Щодо шаблонізатора, то Flask, так само як і Django, використовує Jinja2 як основний шаблонізатор, що дозволяє розробникам легко створювати HTML-шаблони з динамічним вмістом. Хоча набір вбудованих тегів і фільтрів у Flask не такий розширений, як у Django, Flask надає гнучкість у розширенні функціональності шаблонів. Jinja2, що використовується у Flask, містить базовий набір тегів і фільтрів, схожих на ті, що є у Django.

Також, одним з основних аспектів є робота з формами. Для роботи з формами допоможе використання розширень, таких як Flask-WTF (Flask-Form), дозволяє зручно створювати та обробляти форми у веб-додатках на Flask. Flask-WTF надає інструменти для створення форм, включаючи поля вводу, вибору, чекбокси тощо, а також механізми для валідації введених даних та захисту від атак типу Cross-Site Request Forgery (CSRF)

Flask-WTF підтримує різні типи полів, включаючи текстові поля, поля для введення паролів, поля вибору, чекбокси тощо. Кожне поле може бути налаштоване за допомогою валідаторів та інших атрибутів (рис 3.31).

```
from wtforms import PasswordField, BooleanField


class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember_me = BooleanField('Remember Me')
    submit = SubmitField('Login')
```

Рисунок 3.31 – Приклад форми

Валідація введених даних є важливою частиною роботи з формами. Flask-WTF надає вбудовані валідатори, такі як DataRequired, Email, Length та інші. Також можна створювати власні валідатори.

Flask-WTF автоматично забезпечує захист від CSRF-атак за допомогою генерації та перевірки CSRF-токенів. Це робиться шляхом додавання прихованого поля з токеном у форму та його перевірки при обробці запиту.

Обробляти форми нам допоможуть представлення. На відміну від Django, у Flask представлення це функції, які обробляють запити до конкретних URL-адрес і повертають відповідь клієнту. Представлення визначають, як обробляються запити та що відображається користувачу.

Представлення створюється як функція Python, яка прив'язана до URL-адреси за допомогою декоратора @app.route (рис. 3.32). Цей механізм дозволяє структурувати та організовувати маршрутизацію веб-додатку. За замовчуванням представлення обробляє тільки GET-запити. Щоб обробляти інші методи (наприклад, POST), потрібно явно вказати це у декораторі @app.route (рис. 3.33).

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def index():
    return 'Hello, World!'
```

Рисунок 3.32 – Приклад роботи з URL-правилами

```
from flask import request


@app.route('/submit', methods=['GET', 'POST'])
def submit():
    if request.method == 'POST':
        data = request.form['data']
        return f"Отримано дані: {data}"
    return '''
        <form method="post">
            Дані: <input type="text" name="data">
            <input type="submit" value="Відправити">
        </form>
    '''
```

Рисунок 3.33 – Приклад обробки POST-запиту

Таким чином, розробка на Flask має свої унікальні особливості та переваги. Усі складові розробки сайтів, які були описані, Flask має на досить високому рівні, проте більшість з цих складових надається за допомогою сторонніх бібліотек, на відміну від Django, де більшість вже є вбудовано. Синтаксис в Flask також відрізняється, проте залишається досить простим, що спрощує рівень входження для новачків.

3.7 Відмінності процесу розробки в FastAPI

FastAPI базується на асинхронній архітектурі, що дозволяє ефективніше обробляти запити та підвищує продуктивність додатків. Це особливо корисно для систем з високим навантаженням, оскільки асинхронна модель забезпечує ефективне управління ресурсами і зменшує затримки. Основними компонентами архітектури FastAPI є [15]:

– асинхронні функції: завдяки використанню async та await, FastAPI дозволяє виконувати багато операцій паралельно;
– pydantic: використовується для валідації даних та створення моделей, що забезпечує простоту і надійність;
– starlette: використовується як основа для роутінгу та обробки запитів.

FastAPI не має вбудованого механізму для роботи з шаблонами, як, наприклад, у Flask чи Django. Замість цього рекомендується використовувати Jinja2 або інші бібліотеки для створення HTML-шаблонів. Це дозволяє зберегти гнучкість у виборі інструментів для рендерингу шаблонів (рис 3.34).

```
from fastapi import FastAPI
from fastapi.templating import Jinja2Templates
from starlette.requests import Request

app = FastAPI()
templates = Jinja2Templates(directory="templates")


@app.get("/")
async def read_root(request: Request):
    return templates.TemplateResponse( name: "index.html",  context: {"request": request})
```

Рисунок 3.34 – Приклад використання шаблона в FastAPI

Як ми бачимо на рисунку 3.34, для визначення маршрутів FastAPI використовує декоратори, що забезпечує простоту і зрозумілість коду. Основні HTTP методи (GET, POST, PUT, DELETE) легко визначаються через відповідні декоратори. Декоратори дозволяють швидко додавати нові маршрути, а типізація параметрів покращує зчитуваність і надійність коду.

Як у всіх попередніх фреймворках, FastAPI не може працювати з невизначеними даними. Серіалізація даних є основою роботи в FastAPI. В цьому фреймворку зазвичай використовується Pydantic моделі для валідації та серіалізації даних (рис. 3.35-3.36). Це дозволяє створювати чітко визначені моделі даних, які автоматично перевіряються при передачі через API.

Pydantic – це потужна бібліотека для Python, що забезпечує валідацію даних та управління конфігурацією на основі типів. Вона активно використовується у FastAPI для валідації та серіалізації вхідних і вихідних даних. Основна ідея Pydantic полягає у використанні аннотацій типів Python для автоматичної валідації даних, що робить її простою у використанні та ефективною.

```python
from pydantic import BaseModel


class Item(BaseModel):
    name: str
    description: str = None
    price: float
    tax: float = None
```

Рисунок 3.35 – Приклад моделі в Pydantic

```python
from fastapi import FastAPI
from typing import List

app = FastAPI()


@app.post( path: "/items/", response_model=List[Item])
async def create_item(item: Item):
    items_db.append(item)
    return item
```

Рисунок 3.36 – Використання представлення з моделлю Pydantic

FastAPI дозволяє легко обробляти HTML-форми, використовуючи залежності і типізацію. Завдяки підтримці стандартних форм, ви можете створювати і обробляти дані, що надходять з веб-форм, так само легко, як і JSON-запити. FastAPI дозволяє легко обробляти HTML-форми, використовуючи залежності і типізацію. Завдяки підтримці стандартних форм, ви можете створювати і обробляти дані, що надходять з веб-форм, так само легко, як і JSON-запити (рис. 3.37).

```python
from fastapi import FastAPI, Form
from fastapi.responses import HTMLResponse
from starlette.requests import Request

app = FastAPI()


@app.get( path: "/login/", response_class=HTMLResponse)
async def get_login_form(request: Request):
    return templates.TemplateResponse("login.html", {"request": request})


@app.post("/login/")
async def login(username: str = Form(...), password: str = Form(...)):
    return {"username": username, "password": password}
```

Рисунок 3.37 – Приклад форми в FastAPI

Останнім аспектом, котрий слід розглянути є підключення до БД. FastAPI не має вбудованого ORM, але його можна легко інтегрувати з будь-яким ORM або бібліотекою для роботи з базами даних, наприклад, SQLAlchemy або Tortoise-ORM. З SQLAlchemy ми вже знайомі в Flask, тому розглянемо Tortoise-ORM.

Tortoise-ORM – це асинхронний Object-Relational Mapping (ORM) інструмент для Python, розроблений спеціально для інтеграції з асинхронними фреймворками, такими як FastAPI. Він забезпечує зручний і зрозумілий інтерфейс для взаємодії з базами даних, використовуючи сучасні можливості асинхронного програмування.

Основні можливості Tortoise-ORM:

– асинхронність: підтримка асинхронних операцій дозволяє ефективно працювати з базами даних у високонавантажених асинхронних додатках;
– моделі: моделі даних визначаються як класи Python, що забезпечує зручність і зрозумілість коду;
– валідація і типізація: використання типів даних і валідаторів забезпечує точність і безпеку даних;
– міграції: підтримка міграцій баз даних дозволяє легко змінювати структуру бази даних без втрати даних.

Моделі даних визначаються як класи, що наслідуються від класу Model. Кожне поле моделі визначається як атрибут класу (рис. 3.38).

```python
from tortoise import Tortoise, fields
from tortoise.models import Model


class Item(Model):
    id = fields.IntField(pk=True)
    name = fields.CharField(max_length=255)
    description = fields.TextField(null=True)
    price = fields.FloatField()
    tax = fields.FloatField(null=True)

    class Meta:
        table = "items"
```

Рисунок 3.38 – Приклад моделі в Tortoise-ORM

Tortoise-ORM забезпечує ефективну роботу з базою даних у FastAPI, включаючи створення, отримання, оновлення та видалення записів. Для створення нового запису використовується метод create, наприклад: await Item.create(name="Laptop", description="A powerful laptop", price=1500.0,

tax=100.0). Отримати всі записи можна за допомогою await Item.all(), а конкретний запис за await Item.get(id=1). Для фільтрації використовується метод filter, наприклад: await Item.filter(price__gt=1000) для отримання товарів з ціною понад 1000. Оновлення запису здійснюється методом update: await Item.filter(id=1).update(description="An updated powerful laptop"), а видалення – методом delete: await Item.filter(id=1).delete().

Tortoise-ORM підтримує складні запити, такі як агрегація: await Item.all().count() для підрахунку кількості товарів або await Item.all().aggregate(avg_price=Avg("price")) для обчислення середньої ціни. Він також дозволяє працювати зі зв’язками між таблицями, наприклад, створення товару з категорією: category = await Category.create(name="Electronics") та await Item.create(name="Laptop", description="A powerful laptop", price=1500.0, tax=100.0, category=category), що дозволяє отримувати товари певної категорії: await Item.filter(category__name="Electronics").

Як бачимо, в FastAPI наявні усі складові, проте, як і випадку з Flask, для багатьох складових треба використовувати сторонні бібліотеки. Синтаксично FastAPI більше подібний до Flask, але деякі особливості Django він також перейняв. FastAPI відзначається високою швидкодією та підтримкою асинхронності, що робить його ідеальним вибором для реалізації високонавантажених та ефективних API. Вбудована підтримка Pydantic для валідації та серіалізації даних дозволяє швидко створювати безпечні та документовані API.

4 ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

4.1 Створення сайтів

Експериментальну частину роботи описано як практичну реалізацію запропонованих математичних моделей, методів і алгоритмів. Для цього розробляється план експериментальних досліджень, описується процес виконання та наводяться результати. Тому, у моєму випадку, експериментальну частину потрібно починати зі створення сайтів.

Для створення веб-сайтів насамперед слід визначитися з послідовністю завдань. У моєму випадку було обрано таку послідовність: визначення цілей і завдань проєкту, планування архітектури проєкту, вибір інструментальних засобів, проєктування графічного інтерфейсу, верстання та програмування, наповнення контентом, тестування та запуск.

Оскільки мета цих сайтів полягає у їх функціональному порівнянні, потрібно зосередити увагу саме на функціональних можливостях бекенд-фреймворків та реалізувати їх. Для правильного функціонування сайтів буде використовуватись додавання контролерів, моделей, роутів та шаблонів. Усе це дозволить повністю використати можливості фреймворків і досягти максимальної оптимізації.

Для експериментальної розробки було обрано сторінку зі списком елементів та API, на якій відображено багато елементів, кожен з яких являє собою рядок таблиці. Пагінація відсутня, щоб перевірити, як система справляється з обробкою складних запитів у реальному часі. Схема бази даних навмисно ускладнена багатьма зв’язками, щоб оцінити працездатність ORM фреймворка при роботі зі складними запитами та перевірити його ефективність у таких умовах. У веб-частині та API використовується єдиний формат виводу даних, що забезпечує узгодженість і спрощує процес обробки інформації на всіх рівнях системи.

Написання коду буде здійснюватися в програмному забезпеченні PyCharm. Ця програма є платною, але вона виправдовує витрати завдяки своєму зручному інтерфейсу, можливості швидкого підключення до Git (середовища для викладання проєктів в інтернет), а також здатності легко додавати безліч

бібліотек і фреймворків за допомогою вбудованої консолі. У PyCharm можна писати основний код, який пов'язується між собою. Наприклад, натиснувши на клас елемента з затиснутою кнопкою Ctrl, програма покаже, де ще використовується цей клас (тобто, де визначений цей клас чи де він використовується у файлах Python). Якщо перейменувати файл, PyCharm автоматично змінить усі посилання на нього у всьому коді, попередньо запитавши у користувача про необхідність цього. Це забезпечує значну економію часу, тому PyCharm було обрано для цієї роботи.

PyCharm є інтелектуальним редактором для Python, HTML і JavaScript з можливостями аналізу коду на льоту, запобігання помилок у сирцевому коді і автоматизованими засобами рефакторингу для Python і JavaScript. Програмне забезпечення можна налаштувати під необхідні бекенд-фреймворки за допомогою системи плагінів, що дозволяє програмі автоматично попереджати про помилки у коді обраного фреймворка, включаючи не тільки синтаксичні помилки, але й помилки, такі як неправильна передача типу змінної до методу. Крім того, PyCharm пропонує потужні засоби для відладки і тестування, інтеграцію з базами даних і системами управління версіями, що робить його незамінним інструментом для розробників, які прагнуть забезпечити високу якість коду і ефективність роботи.

Для встановлення допоміжних бібліотек було обрано сервіс pip, який за допомогою команд може завантажити бібліотеки як пакетний менеджер до будь-якого проєкту, що дозволяє використовувати їх у розробці. Pip – це менеджер пакетів, який входить до складу Python і значно спрощує процес управління залежностями. Установка і використання pip здійснюються через команди, які вводяться в консоль, що робить процес швидким і зручним. Завдяки pip можна легко встановлювати, оновлювати та видаляти бібліотеки, а також створювати і керувати віртуальними середовищами, що забезпечує ізоляцію проєктів і запобігає конфліктам між залежностями.

Графічний інтерфейс буде максимально простим, оскільки головна мета сайтів – продемонструвати функціональну складову фреймворків. Тому основний

акцент буде зроблено саме на цьому. Було створено дизайн-макети для головної сторінки зі списком елементів та API ендпоінт, які можна побачити на рисунках 4.1-4.2.

ID	Title	University name	University country	University city	Professors
7202	Brianna Turner	Kent Inc	Barbados	West Lisa	Sophia Sanchez - Sango, Belarusian, Danish, Thai, Belarusian, Steven Davis - Mongolian, Maltese, Avaric, Sango, Nepali, Lindsay Armstrong - Sundanese, Tsonga, Inupiaq, Persian, Luba-Katanga, Blake Norris - Belarusian, Swahili, Persian, Bihari languages, Sotho, Southern, Charles Estrada - Walloon, Tagalog, Kirghiz, Hungarian, Portuguese,
7203	Daniel Lewis	Fleming LLC	Guam	West Teresafurt	Kevin Glover - Hindi, Nauru, Macedonian, Kalaallisut, Chinese, Jeremy Burton - Russian, Japanese, Inuktitut, Tahitian, Zulu, April Lee DVM - Portuguese, Ewe, Sindhi, Estonian, Nepali, Amy Marquez - Dutch, Tigrinya, Sinhala, Latin, Galician, Amy Aguilar - German, Guarani, Akan, Bislama, Basque,
7204	Susan Thompson	Humphrey-Coleman	Turkmenistan	Malloryville	Janice Manning - Danish, Persian, Thai, Kannada, Wolof, Stanley Roth - Abkhazian, Luba-Katanga, Ganda, Uzbek, Catalan, Robert Roberson - Malagasy, Walloon, English, Serbian, Czech, Dr. David Taylor Jr. - Guarani, German, Luba-Katanga, Luxembourgish, Basque, Bradley Hudson - Nauru, Maori, Nepali, Kongo, Northern Sami,
7205	Jeff Smith	Fischer, Bartlett and Townsend	Anguilla	Port Johnchester	Charles Davis - Luba-Katanga, Mongolian, Latvian, Italian, Vietnamese, Bailey Scott - Swahili, Greek, Nauru, Ukrainian, Czech, Benjamin Bryant DVM - Belarusian, North Ndebele, Czech, Italian, Persian, Joseph Armstrong - Malay, Czech, Malay, Bislama, Aymara, Steven Barry - Latvian, Irish, Telugu, Tagalog, Sotho, Southern,
7206	Mary Rivas	Snyder, Weeks and Mcgee	Isle of Man	Spencerview	Cristina Austin - Serbian, Hungarian, Chichewa, Icelandic, Central Khmer, Ms. Meagan Obrien - Marathi, Gaelic, Chamorro, Pali, Latvian, Brian Smith - Church Slavic, Pushto, Twi, Chichewa, Sundanese, Shane Smith - Limburgan, Uzbek, English, Nepali, Assamese, Kelsey Obrien - Zulu, Tajik, Swati, Uzbek, Tamil,
7207	Jordan Morrison	Winters and Sons	Guam	Serranofurt	Cassandra Porter - Venda, Ido, Uzbek, Romanian, Polish, Douglas Smith - Belarusian, Kalaallisut, Portuguese, Zhuang, Chuvash, Sarah Hughes - Kirghiz, Telugu, Korean, Chinese, Thai, Cassandra Pennington - Sardinian, Luxembourgish, Portuguese, Finnish, Norwegian, Andrew Reynolds Jr. - Walloon, Spanish, Somali, Kannada, Kikuyu,
7208	Cheryl Sanders	Miller-Travis	Kuwait	Lake Kimberly	Kimberly Delgado - Telugu, Kinyarwanda, Basque, Guarani, Tamil, Angela Perez - Bihari languages, Central Khmer, Azerbaijani, Maori, Haitian, Joel Ford - Tahitian, Urdu, North Ndebele, Italian, Interlingue, Samuel Gutierrez - Sardinian, Sanskrit, Interlingua, Nepali, Malagasy, Marissa Valdez - Arabic, Maori, Nepali, Inupiaq, Hungarian,
7209	Stephen Robles	Martin-Moody	Pitcairn Islands	Rayport	Jessica Sanders - Faroese, Occitan, Sango, Latin, Pali, Elizabeth Coleman - Norwegian Nynorsk, Church Slavic, Azerbaijani, Inuktitut, Somali, Matthew Pierce - Panjabi, Divehi, Uighur, Chuvash, Komi, Paul Morris - Sardinian, Slovak, Tigrinya, Basque, Turkish, Elizabeth Schmitt - Bambara, Sichuan Yi, Chamorro, Latvian, Tibetan,
7210	Lisa Whitehead	Smith PLC	French Polynesia	Port Matthewshire	Kimberly Terry - Kurdish, Burmese, Divehi, Limburgan, Telugu, Kelly Perez - Inuktitut, Breton, Persian, Kurdish, Javanese, Donald Anderson - Bulgarian, Estonian, Cornish, Oromo, Kurdish, Michael Rodriguez - Esperanto, Uzbek, Dzongkha, Norwegian Nynorsk, Lao, Roger Russo - Bashkir, Danish, Uzbek, Inuktitut, Malagasy,
7211	Daniel Donovan	Bell-Bryant	Burundi	Matthewview	Whitney Marshall - Cornish, Polish, Basque, Chichewa, Kannada, Mark Blanchard II - Northern Sami, Italian, Komi, Lao, Portuguese, Ashley Lara - Italian, Faroese, Oriya, Shona, Turkish, Brian Smith - Hindi, Welsh, Romansh, Inuktitut, French, Timothy Smith - Tibetan, Walloon, Slovenian, Oriya, Interlingue,
7212	Eugene Lam	Smith Ltd	Saint Pierre and Miquelon	West Michaelville	Veronica Beck - Malagasy, Fijian, Bosnian, Rundi, Northern Sami, Tamara Everett - Kalaallisut, Cree, Occitan, Kongo, Kongo, Angela Hill - Ojibwa, Chamorro, Tahitian, Serbian, Akan, Melinda Moore - Navajo, Sardinian, Romanian, Guarani, Danish, Nicole Burton - Czech, Tahitian, Assamese, Kalaallisut, Persian,
7213	Kimberly Craig	Jones-Gibson	Argentina	West Austin	Eric Campbell - Bihari languages, Tajik, Bislama, Serbian, Thai, Mrs. Lisa Parker - Hungarian, Dutch, Interlingue, Romansh, Kanuri, Stacey Alexander - Samoan, Sundanese, Uzbek, South Ndebele, Armenian, Martin Velez - Ndonga, Russian, South Ndebele, Irish, Dzongkha, Jonathan Holt - Inuktitut, Kurdish, Sundanese, Hindi, Corsican,
7214	Timothy Gardner MD	Decker-Warren	Hungary	Jakestad	Savannah Mckenzie - Thai, Lithuanian, Thai, Twi, Vietnamese, Daniel Conner - Walloon, Rundi, Gaelic, Portuguese, Kalaallisut, Donald Anderson - Bulgarian, Estonian, Cornish, Oromo, Kurdish, Angela Vega - Slovak, Slovenian, Sindhi, Kuanyama, Uzbek, Joann Gonzalez - Turkmen, Luba-Katanga, Hiri Motu, Marathi, Belarusian,
7215	Tricia Adams	Clark-Wright	Aruba	Robertshire	Lorraine Rodriguez - Marathi, Slovenian, Igbo, Aragonese, Welsh, Elizabeth Weaver - Sindhi, Latin, Norwegian Nynorsk, Igbo, Central Khmer, Roy Davis - Inuktitut, Basque, Swedish, Georgian, Chechen, Mary Mccoy - Slovenian, Gaelic, Kongo, Gaelic, Interlingua, William Nelson - Navajo, Irish, Icelandic, Georgian, Greek,

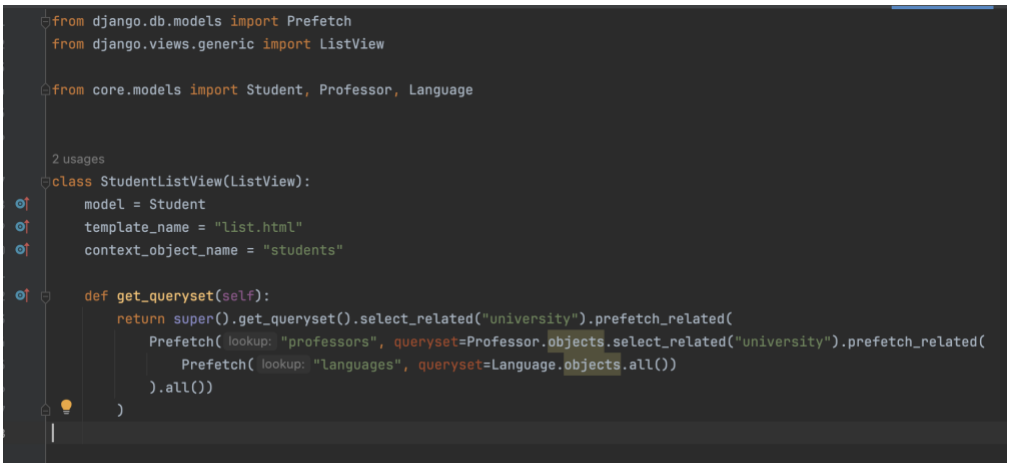
Рисунок 4.1 – Головна сторінка

Ці сторінки включають таблицю з елементами та API ендпоінт для взаємодії з даними. Простий дизайн дозволяє зосередитися на функціональності, тестуванні продуктивності та оцінці можливостей різних фреймворків


```
[{"id":7202,"name":"Brianna Turner","university":{"id":9430,"name":"Kent Inc","country":{"id":9430,"name":"Barbados"},"city":{"id":9430,"name":"West Lisa","country_id":9430}},"professors":[{"id":9409,"name":"Sophia
Sanchez","university":{"id":9538,"name":"Parker-Atkins","country":{"id":9538,"name":"Gibraltar"},"city":{"id":9538,"name":"Rivasside","country_id":9538}},"languages":[{"id":9918,"name":"Sango"},
{"id":9233,"name":"Belarusian"},{"id":9134,"name":"Danish"},{"id":9355,"name":"Thai"},{"id":9602,"name":"Belarusian"}]},{"id":9186,"name":"Steven Davis","university":{"id":9452,"name":"Garcia Group","country":
{"id":9452,"name":"Turks and Caicos Islands"},"city":{"id":9452,"name":"Brendaville","country_id":9452}},"languages":[{"id":9503,"name":"Mongolian"},{"id":9879,"name":"Maltese"},{"id":9357,"name":"Avaric"},
{"id":9227,"name":"Sango"},{"id":10048,"name":"Nepali"}]},{"id":8678,"name":"Lindsay Armstrong","university":{"id":9373,"name":"Roman, Townsend and Cox","country":{"id":9373,"name":"Belize"},"city":
{"id":9373,"name":"South Emilyton","country_id":9373}},"languages":[{"id":10007,"name":"Sundanese"},{"id":9493,"name":"Tsonga"},{"id":9896,"name":"Inupiaq"},{"id":9317,"name":"Persian"},{"id":9252,"name":"Luba-
Katanga"}]},{"id":8622,"name":"Blake Norris","university":{"id":9797,"name":"Morrow-Buck","country":{"id":9797,"name":"Germany"},"city":{"id":9797,"name":"North Nancy","country_id":9797}},"languages":
[{"id":9406,"name":"Belarusian"},{"id":9693,"name":"Swahili"},{"id":9458,"name":"Persian"},{"id":9226,"name":"Bihari languages"},{"id":9188,"name":"Sotho, Southern"}]},{"id":8639,"name":"Charles
Estrada","university":{"id":9666,"name":"Rodriguez-Roberts","country":{"id":9666,"name":"Djibouti"},"city":{"id":9666,"name":"North Erik","country_id":9666}},"languages":[{"id":9139,"name":"Walloon"},
{"id":9897,"name":"Tagalog"},{"id":9545,"name":"Kirghiz"},{"id":9478,"name":"Hungarian"},{"id":10053,"name":"Portuguese"}]}},{"id":7203,"name":"Daniel Lewis","university":{"id":9437,"name":"Fleming LLC","country":
{"id":9437,"name":"Guam"},"city":{"id":9437,"name":"West Teresafurt","country_id":9437}},"professors":[{"id":9197,"name":"Kevin Glover","university":{"id":9559,"name":"Barnett, Avila and Lewis","country":
{"id":9559,"name":"Bulgaria"},"city":{"id":9559,"name":"Lake Melanie","country_id":9559}},"languages":[{"id":9534,"name":"Hindi"},{"id":9429,"name":"Nauru"},{"id":9427,"name":"Macedonian"},
{"id":9584,"name":"Kalaallisut"},{"id":9615,"name":"Chinese"}]},{"id":8626,"name":"Jeremy Burton","university":{"id":10024,"name":"Jordan-Clay","country":{"id":10024,"name":"Fiji"},"city":{"id":10024,"name":"New
Karenside","country_id":10024}},"languages":[{"id":9501,"name":"Russian"},{"id":9980,"name":"Japanese"},{"id":9871,"name":"Inuktitut"},{"id":9290,"name":"Tahitian"},{"id":9608,"name":"Zulu"}]},
{"id":9274,"name":"April Lee DVM","university":{"id":10052,"name":"Wise-Moore","country":{"id":10052,"name":"Saudi Arabia"},"city":{"id":10052,"name":"Johnview","country_id":10052}},"languages":
[{"id":9275,"name":"Portuguese"},{"id":9461,"name":"Sindhi"},{"id":9619,"name":"Czech"},{"id":9437,"name":"Estonian"},{"id":10048,"name":"Nepali"}]},{"id":9299,"name":"Amy Marquez","university":
{"id":9656,"name":"Williams-Harris","country":{"id":9656,"name":"South Africa"},"city":{"id":9656,"name":"Aprilberg","country_id":9656}},"languages":[{"id":9176,"name":"Dutch"},{"id":9787,"name":"Tigrinya"},
{"id":9464,"name":"Sinhala"},{"id":9870,"name":"Latin"},{"id":9142,"name":"Galician"}]},{"id":9372,"name":"Amy Aguilar","university":{"id":9839,"name":"Graham-Davis","country":{"id":9839,"name":"Kyrgyz
Republic"},"city":{"id":9839,"name":"Lake Brittneystad","country_id":9839}},"languages":[{"id":9338,"name":"German"},{"id":10064,"name":"Guarani"},{"id":9262,"name":"Akan"},{"id":10029,"name":"Bislama"},
{"id":9543,"name":"Basque"}]}],{"id":7204,"name":"Susan Thompson","university":{"id":9752,"name":"Humphrey-Coleman","country":{"id":9752,"name":"Turkmenistan"},"city":
{"id":9752,"name":"Malloryville","country_id":9752}},"professors":[{"id":9351,"name":"Janice Manning","university":{"id":9433,"name":"Miller PLC","country":{"id":9433,"name":"San Marino"},"city":
{"id":9433,"name":"New Lisa","country_id":9433}},"languages":[{"id":9343,"name":"Danish"},{"id":9272,"name":"Persian"},{"id":9230,"name":"Thai"},{"id":9189,"name":"Kannada"},{"id":9700,"name":"Wolof"}]},
{"id":9035,"name":"Stanley Roth","university":{"id":9750,"name":"Baker LLC","country":{"id":9750,"name":"Turkmenistan"},"city":{"id":9750,"name":"East Brandonchester","country_id":9750}},"languages":
[{"id":9439,"name":"Abkhazian"},{"id":9562,"name":"Luba-Katanga"},{"id":9554,"name":"Ganda"},{"id":10065,"name":"Uzbek"},{"id":9631,"name":"Catalan"}]},{"id":9198,"name":"Robert Roberson","university":
{"id":9700,"name":"Rivera, Ray and Malone","country":{"id":9700,"name":"Jamaica"},"city":{"id":9700,"name":"North Jason","country_id":9700}},"languages":[{"id":9247,"name":"Malagasy"},{"id":9392,"name":"Walloon"},
{"id":9964,"name":"English"},{"id":9954,"name":"Serbian"},{"id":9377,"name":"Czech"}]},{"id":8827,"name":"Dr. David Taylor Jr.","university":{"id":9802,"name":"Allen-Howard","country":
{"id":9802,"name":"Fiji"},"city":{"id":9802,"name":"Lake Dana","country_id":9802}},"languages":[{"id":10014,"name":"Guarani"},{"id":9338,"name":"German"},{"id":9562,"name":"Luba-Katanga"},
{"id":9641,"name":"Luxembourgish"},{"id":9476,"name":"Basque"}]},{"id":9085,"name":"Bradley Hudson","university":{"id":9452,"name":"Garcia Group","country":{"id":9452,"name":"Turks and Caicos Islands"},"city":
{"id":9452,"name":"Brendaville","country_id":9452}},"languages":[{"id":9814,"name":"Nauru"},{"id":9237,"name":"Maori"},{"id":9456,"name":"Nepali"},{"id":9859,"name":"Kongo"},{"id":9601,"name":"Northern Sami"}]}},
{"id":7205,"name":"Jeff Smith","university":{"id":9270,"name":"Fischer, Bartlett and Townsend","country":{"id":9270,"name":"Anguilla"},"city":{"id":9270,"name":"Port Johnchester","country_id":9270}},"professors":
[{"id":9449,"name":"Charles Davis","university":{"id":9726,"name":"Gordon-Bernard","country":{"id":9726,"name":"French Polynesia"},"city":{"id":9726,"name":"Michelleland","country_id":9726}},"languages":
[{"id":9759,"name":"Luba-Katanga"},{"id":9844,"name":"Mongolian"},{"id":9295,"name":"Latvian"},{"id":9157,"name":"Italian"},{"id":9825,"name":"Vietnamese"}]},{"id":9073,"name":"Bailey Scott","university":
{"id":10011,"name":"Hutchinson Ltd","country":{"id":10011,"name":"Bulgaria"},"city":{"id":10011,"name":"Morrowberg","country_id":10011}},"languages":[{"id":9178,"name":"Swahili"},{"id":9201,"name":"Greek"},
{"id":9356,"name":"Nauru"},{"id":9991,"name":"Ukrainian"},{"id":9377,"name":"Czech"}]},{"id":8818,"name":"Benjamin Bryant DVM","university":{"id":10040,"name":"Moreno-Dickson","country":
{"id":10040,"name":"Serbia"},"city":{"id":10040,"name":"West Valerie","country_id":10040}},"languages":[{"id":9486,"name":"Belarusian"},{"id":9872,"name":"North Ndebele"},{"id":9926,"name":"Czech"},
{"id":9925,"name":"Italian"},{"id":9282,"name":"Persian"}]},{"id":9463,"name":"Joseph Armstrong","university":{"id":9624,"name":"Marks and Sons","country":{"id":9624,"name":"Trinidad and Tobago"},"city":
{"id":9624,"name":"Ericbury","country_id":9624}},"languages":[{"id":9791,"name":"Malay"},{"id":9563,"name":"Czech"},{"id":10005,"name":"Malay"},{"id":10029,"name":"Bislama"},{"id":9767,"name":"Aymara"}]},
{"id":8570,"name":"Steven Barry","university":{"id":9925,"name":"Mcdonald-Logan","country":{"id":9925,"name":"Equatorial Guinea"},"city":{"id":9925,"name":"South Tylerchester","country_id":9925}},"languages":
[{"id":10037,"name":"Latvian"},{"id":9172,"name":"Irish"},{"id":9606,"name":"Telugu"},{"id":9701,"name":"Tagalog"},{"id":9188,"name":"Sotho, Southern"}]}]},{"id":7206,"name":"Mary Rivas","university":
{"id":9802,"name":"Snyder, Weeks and Mcgee","country":{"id":9802,"name":"Isle of Man"},"city":{"id":9802,"name":"Spencerview","country_id":9802}},"professors":[{"id":8972,"name":"Cristina Austin","university":
{"id":9437,"name":"Fleming LLC","country":{"id":9437,"name":"Guam"},"city":{"id":9437,"name":"West Teresafurt","country_id":9437}},"languages":[{"id":9817,"name":"Serbian"},{"id":9784,"name":"Hungarian"},
{"id":9488,"name":"Chichewa"},{"id":9443,"name":"Icelandic"},{"id":9792,"name":"Central Khmer"}]},{"id":9133,"name":"Ms. Meagan Obrien","university":{"id":9988,"name":"Morris, Hodge and Flores","country":
{"id":9988,"name":"Mauritius"},"city":{"id":9988,"name":"Harrisonville","country_id":9988}},"languages":[{"id":9816,"name":"Marathi"},{"id":9687,"name":"Gaelic"},{"id":9803,"name":"Chamorro"},
{"id":9417,"name":"Pali"},{"id":9124,"name":"Latvian"}]},{"id":9301,"name":"Brian Smith","university":{"id":9862,"name":"Olson Group","country":{"id":9862,"name":"Algeria"},"city":{"id":9862,"name":"New
Stevenshire","country_id":9862}},"languages":{"id":10076,"name":"Church Slavic"},{"id":9463,"name":"Pushto"},{"id":9558,"name":"Twi"},{"id":9944,"name":"Chichewa"},{"id":9637,"name":"Sundanese"}]},
{"id":9429,"name":"Shane Smith","university":{"id":9508,"name":"Orozco, Garcia and Murillo","country":{"id":9508,"name":"Martinique"},"city":{"id":9508,"name":"Kellyburn","country_id":9508}},"languages":
[{"id":9149,"name":"Limburgan"},{"id":9198,"name":"Uzbek"},{"id":9732,"name":"English"},{"id":9920,"name":"Nepali"},{"id":9120,"name":"Assamese"}]},{"id":8606,"name":"Kelsey Obrien","university":
{"id":9956,"name":"Sanchez Ltd","country":{"id":9956,"name":"Jordan"},"city":{"id":9956,"name":"Wilsonhaven","country_id":9956}},"languages":[{"id":9212,"name":"Zulu"},{"id":9369,"name":"Tajik"},
{"id":9876,"name":"Swati"},{"id":9198,"name":"Uzbek"},{"id":9633,"name":"Tamil"}]}]},{"id":7207,"name":"Jordan Morrison","university":{"id":9728,"name":"Winters and Sons","country":{"id":9728,"name":"Guam"},"city":
{"id":9728,"name":"Serranofurt","country_id":9728}},"professors":[{"id":9284,"name":"Cassandra Porter","university":{"id":9395,"name":"Silva-Morris","country":{"id":9395,"name":"Equatorial Guinea"},"city":
{"id":9395,"name":"Toddburgh","country_id":9395}},"languages":[{"id":9658,"name":"Venda"},{"id":9173,"name":"Ido"},{"id":9781,"name":"Uzbek"},{"id":9362,"name":"Romanian"},{"id":9519,"name":"Polish"}]},
{"id":9029,"name":"Douglas Smith","university":{"id":9583,"name":"Miller, Waters and Colon","country":{"id":9583,"name":"Luxembourg"},"city":{"id":9583,"name":"Lake Victoria","country_id":9583}},"languages":
[{"id":9337,"name":"Belarusian"},{"id":9584,"name":"Kalaallisut"},{"id":10060,"name":"Portuguese"},{"id":9386,"name":"Zhuang"},{"id":9665,"name":"Chuvash"}]},{"id":8940,"name":"Sarah Hughes","university":
{"id":9591,"name":"Davis and Sons","country":{"id":9591,"name":"Serbia"},"city":{"id":9591,"name":"New Emily","country_id":9591}},"languages":[{"id":9267,"name":"Kirghiz"},{"id":10003,"name":"Telugu"},
{"id":9903,"name":"Korean"},{"id":9615,"name":"Chinese"},{"id":9355,"name":"Thai"}]},{"id":9398,"name":"Cassandra Pennington","university":{"id":9444,"name":"Mata-Sharp","country":
```


Рисунок 4.2 – API ендпоінт

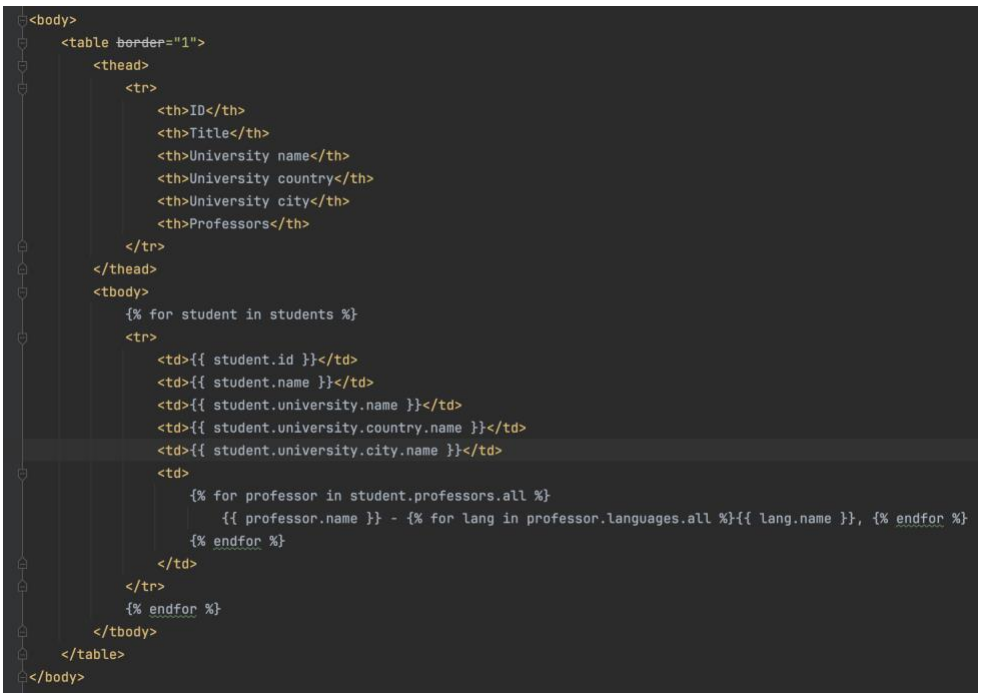
Наступним етапом є верстання, програмування та наповнення контентом. Використовуючи актуальну документацію для кожного фреймворка (Django, Flask, FastAPI), були створені відповідні елементи, враховуючи специфіку кожного з них. Приклади реалізації можна побачити на рисунках 4.3-4.12. Усі частини проєкту були правильно пов'язані, оскільки це важливо для коректної роботи застосунку.



```python
from django.db.models import Prefetch
from django.views.generic import ListView

from core.models import Student, Professor, Language


2 usages
class StudentListView(ListView):
    model = Student
    template_name = "list.html"
    context_object_name = "students"

    def get_queryset(self):
        return super().get_queryset().select_related("university").prefetch_related(
            Prefetch( lookup: "professors", queryset=Professor.objects.select_related("university").prefetch_related(
                Prefetch( lookup: "languages", queryset=Language.objects.all())
            ).all())
        )
```

Рисунок 4.3 – Уявлення сторінки на Django

Уявлення StudentListView наслідує вбудований клас ListView, що дозволяє автоматично відображати список об'єктів. Цей клас надає зручний спосіб для представлення даних у вигляді списку з мінімальною кількістю налаштувань.



```html
<body>
    <table ~~border~~="1">
        <thead>
            <tr>
                <th>ID</th>
                <th>Title</th>
                <th>University name</th>
                <th>University country</th>
                <th>University city</th>
                <th>Professors</th>
            </tr>
        </thead>
        <tbody>
            {% for student in students %}
            <tr>
                <td>{{ student.id }}</td>
                <td>{{ student.name }}</td>
                <td>{{ student.university.name }}</td>
                <td>{{ student.university.country.name }}</td>
                <td>{{ student.university.city.name }}</td>
                <td>
                    {% for professor in student.professors.all %}
                        {{ professor.name }} - {% for lang in professor.languages.all %}{{ lang.name }}, {% endfor %}
                    {% endfor %}
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
```

Рисунок 4.4 – Шаблон сторінки на Django

Для візуального відображення сторінки в браузері, уявленню треба використати шаблон який визначає, як саме дані будуть представлені користувачеві. У шаблоні можна використовувати HTML та шаблонні теги Django для динамічного відображення інформації.

```
4 usages
class City(models.Model):
    name = models.CharField(max_length=100)
    country = models.ForeignKey(Country, on_delete=models.CASCADE)

    def __str__(self):
        return self.name


9 usages
class Language(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name


7 usages
class University(models.Model):
    name = models.CharField(max_length=200)
    country = models.ForeignKey(Country, on_delete=models.CASCADE)
    city = models.ForeignKey(City, on_delete=models.CASCADE)

    def __str__(self):
        return self.name


9 usages
class Professor(models.Model):
    name = models.CharField(max_length=100)
    university = models.ForeignKey(University, on_delete=models.CASCADE)
    languages = models.ManyToManyField(Language)

    def __str__(self):
        return self.name


7 usages
class Student(models.Model):
    name = models.CharField(max_length=100)
    university = models.ForeignKey(University, on_delete=models.CASCADE)
    professors = models.ManyToManyField(Professor)

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

Рисунок 4.5 – Моделі бази даних на Django

Також для коректної роботи застосунку, та проводження операцій з даними, що зберігаються в базі даних, треба використати моделі.

```python
@api.get("/api", response=List[StudentSchema])
def get_students(request):
    students = Student.objects.select_related("university").prefetch_related(
        Prefetch( lookup: "professors", queryset=Professor.objects.select_related("university").prefetch_related(
            Prefetch( lookup: "languages", queryset=Language.objects.all())
        ).all())
    )

    student_list = []
    for student in students:
        student_list.append({
            "id": student.id,
            "name": student.name,
            "university": {
                "name": student.university.name,
                "country": student.university.country.name,
                "city": student.university.city.name,
            },
            "professors": [
                {
                    "name": professor.name,
                    "languages": [{"name": lang.name} for lang in professor.languages.all()]
                } for professor in student.professors.all()
            ]
        })
    return student_list
```

Рисунок 4.6 – API на Django

На останок залишається лише написати API ендпоінт, який буде віддавати дані, що потрібні для експерименту у форматі JSON.

```python
professors_languages = db.Table(
    "core_professor_languages", db.Model.metadata,
    db.Column("professor_id", db.ForeignKey("core_professor.id"), primary_key=True),
    db.Column("language_id", db.ForeignKey("core_language.id"), primary_key=True)
)

student_professors = db.Table(
    "core_student_professors", db.Model.metadata,
    db.Column("professor_id", db.ForeignKey("core_professor.id"), primary_key=True),
    db.Column("student_id", db.ForeignKey("core_student.id"), primary_key=True)
)


6 usages
class Professor(db.Model):
    __tablename__ = 'core_professor'
    id = db.Column(db.Integer, primary_key=True, index=True)
    name = db.Column(db.String(100))
    university_id = db.Column(db.Integer, db.ForeignKey('core_university.id'))
    university = db.relationship('University')
    languages = db.relationship("Language", secondary=professors_languages)
    students = db.relationship("Student", secondary=student_professors)


class Language(db.Model):
    __tablename__ = 'core_language'
    id = db.Column(db.Integer, primary_key=True, index=True)
    name = db.Column(db.String(100))
    professors = db.relationship("Professor", secondary=professors_languages)


8 usages
class Student(db.Model):
    __tablename__ = 'core_student'
    id = db.Column(db.Integer, primary_key=True, index=True)
    name = db.Column(db.String(100))
    university_id = db.Column(db.Integer, db.ForeignKey('core_university.id'))
    university = db.relationship('University')
    professors = db.relationship("Professor", secondary=student_professors)
```

Рисунок 4.7 – Моделі на Flask

Ті ж самі кроки слід проробити і в Flask, починаючи з моделів, котрі виглядатимуть по-іншому, тому що використовуються інший пакет в якості ORM.

```python
@app.route('/')
def index():
    from models import Student, Professor
    students = Student.query.options(
        joinedload(Student.university),
        subqueryload(Student.professors).options(
            joinedload(Professor.university),
            subqueryload(Professor.languages)
        )
    ).all()
    return render_template( template_name_or_list: 'list.html', students=students)
```

Рисунок 4.8 – Уявлення на Flask

Уявлення на Flask також мають інший вигляд, вони представленні у якості функцій, а не класів, як це було в Django.

```python
@app.route( rule: "/api", methods=["GET"])
def get_students():
    from models import Student, Professor
    students = Student.query.options(
        joinedload(Student.university),
        subqueryload(Student.professors).options(
            joinedload(Professor.university),
            subqueryload(Professor.languages)
        )
    ).all()

    from schemas import StudentSchema
    student_schema = StudentSchema(many=True)
    result = student_schema.dump(students)
    return jsonify(result)
```

Рисунок 4.9 – API на Flask

API ендпоінт представлено у якості функції та використовує схему для серіалізації даних у потрібний формат.

```python
@app.get( path: "/", response_model=List[schemas.Student])
def get_students(request: Request, db: Session = Depends(get_db)):
    students = db.query(models.Student).options(
         *args: joinedload(models.Student.university),
        subqueryload(models.Student.professors).options(
            joinedload(models.Professor.university),
            subqueryload(models.Professor.languages)
        )
    ).all()
    return templates.TemplateResponse(
        name="list.html", context={"request": request, "students": students}
    )
```

Рисунок 4.10 – Уявлення на FastAPI

Щодо FastAPI, то більшість функціоналу, такого як моделі, є ідентичною до Flask, проте є і різниці.

```python
@app.get( path: "/api", response_model=List[schemas.Student])
def read_students(db: Session = Depends(get_db)):
    students = db.query(models.Student).options(
        *args: joinedload(models.Student.university),
        subqueryload(models.Student.professors).options(
            joinedload(models.Professor.university),
            subqueryload(models.Professor.languages)
        )
    ).all()
    return students
```

Рисунок 4.11 – API на FastAPI

Наприклад, для написання API ендпоінта також використовується серіалізатор, проте визначається він у декораторі і дані серіалізуються автоматично, коли на ендпоінт приходить запит. Проте схеми прописуються окремо.

```python
2 usages
class University(BaseModel):
    id: int
    name: str
    country: Country
    city: City

    class Config:
        orm_mode = True


1 usage
class Professor(BaseModel):
    id: int
    name: str
    university: University
    languages: List[Language]

    class Config:
        orm_mode = True


2 usages
class Student(BaseModel):
    id: int
    name: str
    university: University
    professors: List[Professor]

    class Config:
        orm_mode = True
```

Рисунок 4.12 – Схема API на FastAPI

У Django це включає створення моделей, форм, уявлень (views) та шаблонів. Маршрути налаштовуються у файлах `urls.py`. У Flask реалізовано маршрути (routes), шаблони та розширення. У FastAPI фокус був на налаштуванні маршрутів, залежностей та схем.

Правильна інтеграція цих елементів забезпечує безперебійну роботу застосунку та ефективне використання можливостей кожного фреймворка. Наприклад, головна сторінка містить список постів, а сторінка окремого поста є частиною цього списку. Зв'язки між елементами повинні відповідати конкретним частинам проєкту, на яких вони використовуються, а змінні повинні бути надійно захищені.

Змістом сайту було обрано інформацію про студентів, університети та професорів. Хоча цей контент не має вирішального значення для моєї роботи, він необхідний для наочного демонстрування функціональності сайту.

Тестування сайтів проводилося за кількома критеріями, зокрема на швидкодію та кросбраузерність.

Спочатку сайти перевіряли на кросбраузерність – це здатність сайту коректно відображатися та працювати в різних браузерах. Тестування проводилося у найпопулярніших браузерах: Google Chrome, Microsoft Edge, Opera та Mozilla Firefox. У всіх цих браузерах сайти відображалися коректно і функціонували без жодних проблем. Відображення і функціонал залишалися незмінними у всіх протестованих середовищах.

Наступним етапом було тестування на швидкодію. Це здійснювалося шляхом надсилання запитів до кожної зі сторінок, створених за допомогою трьох різних фреймворків. Результати тестування швидкодії дозволили оцінити ефективність обробки запитів кожним фреймворком і виявити можливі вузькі місця у продуктивності.

Така методика тестування забезпечила всебічну оцінку роботи сайтів у різних умовах, що є важливим для загальної оцінки обраних фреймворків.

4.2 Вибір методу

Експеримент служить для отримання нових наукових знань. Від звичайного пасивного спостереження експеримент відрізняється активним впливом дослідника на явище, що вивчається. Основна мета експерименту – перевірка теоретичних положень, підтвердження робочої гіпотези, а також ширше та глибше вивчення теми наукового дослідження.

Розрізняють експерименти природні та штучні. Перші експерименти притаманні соціальних явищ, наприклад, виробництва. Штучні експерименти широко застосовуються насамперед у технічних науках. Експериментальні дослідження своєю чергою поділяються на лабораторні та виробничі.

Оскільки тема кваліфікаційної роботи – «Дослідження ефективності використання фреймворків веб-розробки на продуктивність веб-додатку: порівняльний аналіз Django, Flask, та FastAPI», тому методи оцінки повинні бути пов’язані з якістю досвіду користувача, сприйняття користувача та досвіду взаємодії. Для формального розрахунку буде обрано метод експертного оцінювання.

4.3 Обґрунтування вибору методу та аналіз прогнозованих результатів

Для оцінки якості досвіду користувача було обрано метод експертного оцінювання, оскільки цей метод допомагає у ході роботи з фокус-групою оцінити обрані експертами критерії, відповідні до функціональності сайтів. Цей метод краще інших підійде для отримання правильних результатів кваліфікаційної роботи. Тепер можна отримати результати за допомогою математичних розрахунків та на їх основі зробити висновки.

4.4 Проведення експерименту

Метод експертних оцінок – один з основних класів методів науково-технічного прогнозування, який ґрунтується на припущенні, що на основі думок експертів можна збудувати адекватну модель майбутнього розвитку об'єкта прогнозування.

Першим етапом цього методу є визначення головних критеріїв оцінювання сайтів, на основі опитувань експертів. Таким чином було обрано такі критерії, як: оптимізація (швидкість завантаження сторінок сайту), масштабованість, архітектура, складність написання коду з боку програміста, безпека та розмір збірки.

Далі необхідно провести оцінювання експертами обраних критеріїв, щоб знайти їх значимість, та думки експертів щодо узгодженості V (табл. 4.1).

Таблиця 4.1 – Оцінка критеріїв

	Експерт														
Критерій	1	2	3	4	5	6	7	8	9	10	Сума	Вага	X	σ	V
Швидкодія	4	3	4	4	5	3	6	4	3	4	40	0,19	4,0	0,94	0,24
Масштабованість	5	6	5	2	2	6	3	5	5	5	44	0,21	4,4	1,51	0,34
Підтримка асинхронності	1	2	1	3	1	2	2	1	1	2	16	0,08	1,6	0,70	0,44
Безпека	6	5	6	6	4	4	5	6	6	6	54	0,26	5,4	0,84	0,16
Споживання пам'яті	3	4	3	5	6	5	4	2	4	3	39	0,19	3,9	1,20	0,31
Розробка API	2	1	2	1	3	1	1	3	2	1	17	0,08	1,7	0,82	0,48

Чим вище значення ваги критерія, тим він більш важливіший за інші. Як можна побачити з результату, найважливішим критерієм на думку експертів є саме безпека, далі йде масштабованість, швидкодія, споживання пам'яті та розробка API і підтримка асинхронності. За цими критеріями експерти оцінили кожний з трьох розроблених сайтів. Відносні ваги сайтів, думки експертів щодо узгодженості V для кожного сайту наведені в таблицях 4.2-4.7.

Таблиця 4.2 – Оцінка фреймворків за швидкодією

Фрейм	Експерт										Сума	Вага кри.	X	σ	V
	1	2	3	4	5	6	7	8	9	10					
Django	3	3	3	2	3	2	2	3	3	3	27	0,45	2,70	0,48	0,18
Flask	2	1	2	3	1	3	1	1	2	2	18	0,30	1,80	0,79	0,44
FastAPI	1	2	1	1	2	1	3	2	1	1	15	0,25	1,50	0,71	0,47
Швидкодія															

Таблиця 4.3 – Оцінка фреймворків за масштабованістю

Фрейм	Експерт										Сума	Вага кри.	X	σ	V
	1	2	3	4	5	6	7	8	9	10					
Django	1	1	1	1	2	3	2	2	1	1	15	0,25	1,50	0,71	0,47
Flask	2	3	2	2	1	1	1	1	2	3	18	0,30	1,80	0,79	0,44
FastAPI	3	2	3	3	3	2	3	3	3	2	27	0,45	2,70	0,48	0,18
Масштабованість															

Таблиця 4.4 – Оцінка фреймворків за підтримкою асинхроності

Фрейм	Експерт										Сума	Вага кри.	X	σ	V
	1	2	3	4	5	6	7	8	9	10					
Django	3	2	3	2	3	3	3	2	3	1	25	0,42	2,50	0,71	0,28
Flask	2	3	1	3	2	2	1	3	2	3	22	0,37	2,20	0,79	0,36
FastAPI	1	1	2	1	1	1	2	1	1	2	13	0,22	1,30	0,48	0,37
Підтримка асинхронності															

Таблиця 4.5 – Оцінка фреймворків за Безпекою

Фрейм	Експерт										Сума	Вага кри.	X	σ	V
	1	2	3	4	5	6	7	8	9	10					
Django	1	1	2	2	1	1	3	1	2	1	15	0,25	1,50	0,71	0,47
Flask	3	2	3	3	2	2	1	2	3	3	24	0,40	2,40	0,70	0,29
FastAPI	2	3	1	1	3	3	2	3	1	2	21	0,35	2,10	0,88	0,42
Безпека															

Таблиця 4.6 – Оцінка фреймворків за споживанням пам’яті

Фрейм	Експерт										Сума	Вага кри.	X	σ	V
	1	2	3	4	5	6	7	8	9	10					
Django	3	3	2	2	3	3	3	1	3	2	25	0,42	2,50	0,71	0,28
Flask	2	1	1	3	2	2	1	2	2	3	19	0,32	1,90	0,74	0,39
FastAPI	1	2	3	1	1	1	2	3	1	1	16	0,27	1,60	0,84	0,53
Споживання пам’яті															

Таблиця 4.7 – Оцінка фреймворків за розробкою АПІ

Фрейм	Експерт										Сума	Вага кри.	X	σ	V
	1	2	3	4	5	6	7	8	9	10					
Django	2	3	2	1	3	2	1	2	2	1	19	0,32	1,90	0,74	0,39
Flask	3	2	3	3	2	3	3	3	3	3	28	0,47	2,80	0,42	0,15
FastAPI	1	1	1	2	1	1	2	1	1	2	13	0,22	1,30	0,48	0,37
Розробка АПІ															

Для підтвердження результатів оцінювання експертів за показником швидкодії було використано розширення Lighthouse для браузера Google Chrome, яке аналізує сайт і визначає якість продуктивності для комп'ютера (рис. 4.13-4.15).

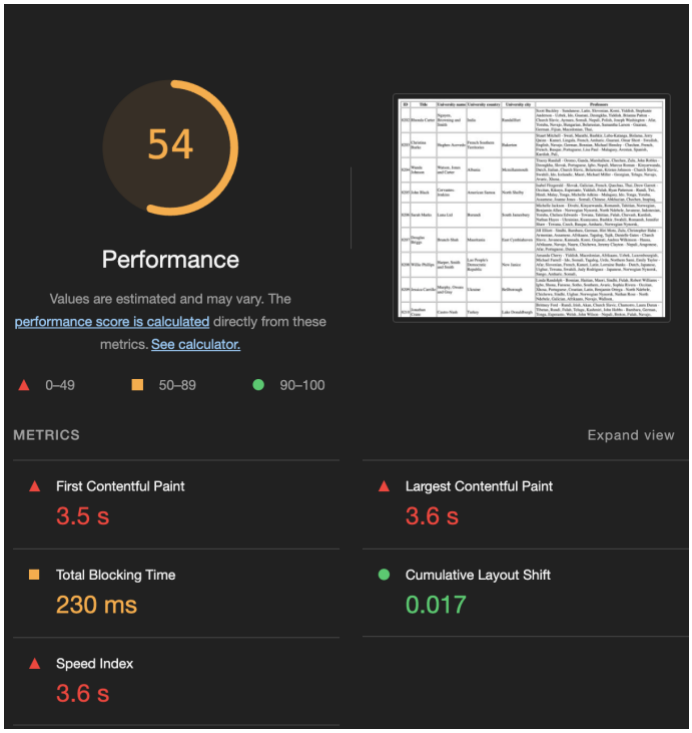


Рисунок 4.13 – Швидкодія FastAPI

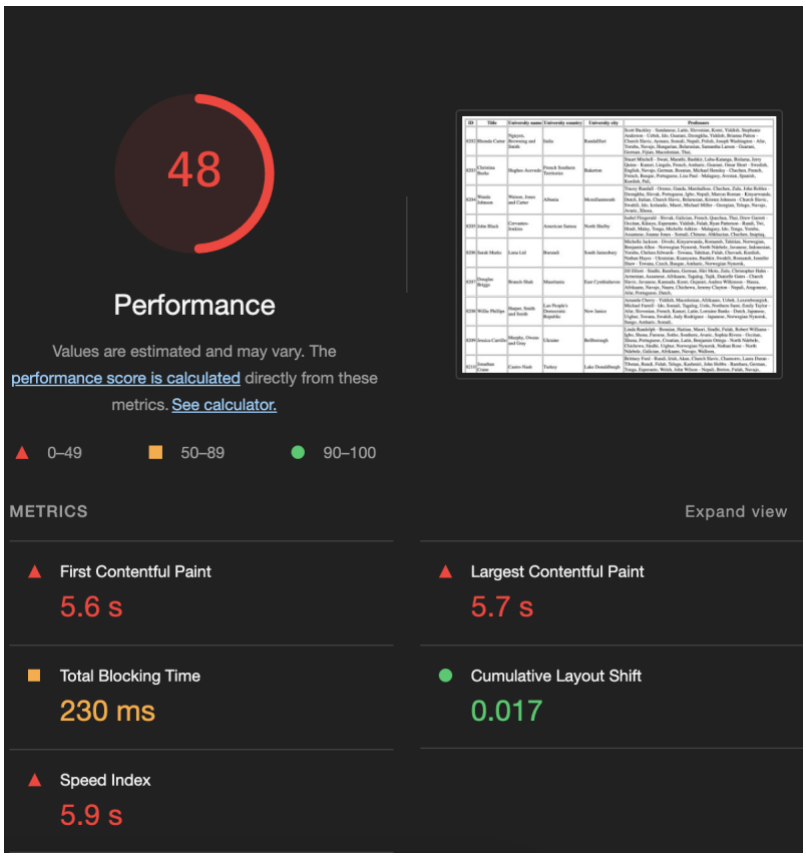


Рисунок 4.14 – Швидкодія Flask

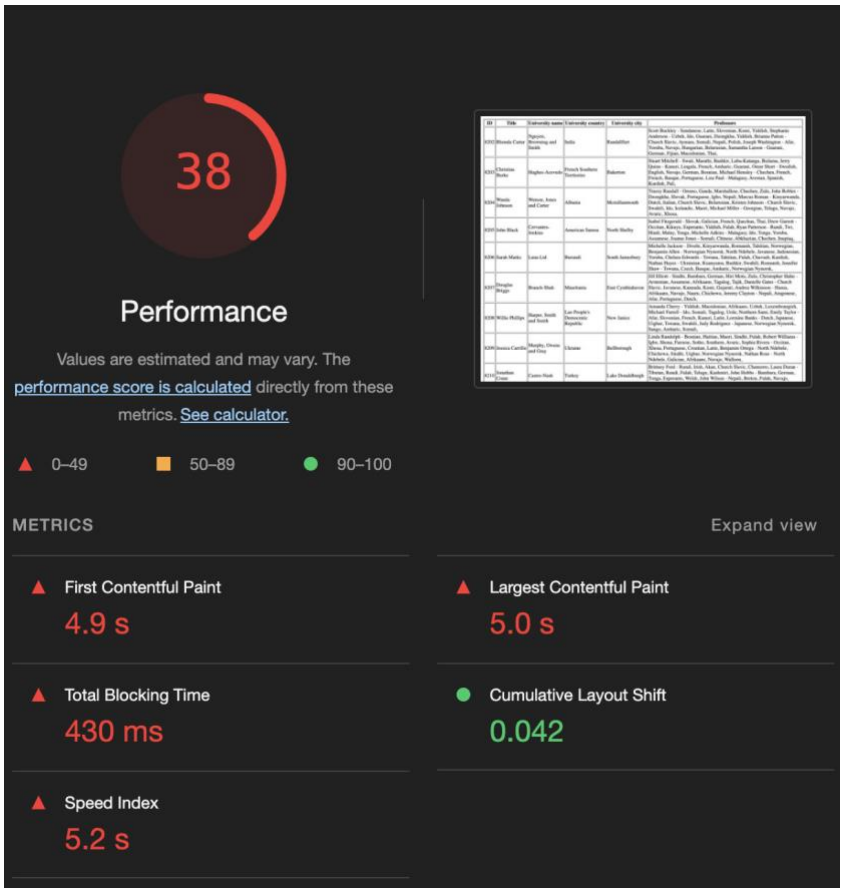


Рисунок 4.15 – Швидкодія Django

Результати розширення Lighthouse повністю підтверджують отримані оцінки експертів за даним показником.

За отриманим значенням ваги критерія можна сказати, що, на думку експертів, Django краще за Flask та FastAPi по критеріями швидкодії, підтримки асинхронності та споживання пам’яті, Flask кращий у безпеці та розробці АПІ та FastAPI кращий у масштабованості.

Після дослідження було порівняно фреймворки за узагальненими рейтингами:

$R_1 = 0,19 * 0,45 + 0,21 * 0,25 + 0,08 * 0,42 + 0,26 * 0,25 + 0,19 * 0,42 + 0,08 * 0,32 = 0,342$

$R_2 = 0,19 * 0,30 + 0,21 * 0,30 + 0,08 * 0,37 + 0,26 * 0,40 + 0,19 * 0,32 + 0,08 *0,47 = 0,352$

$R_3 = 0,19 * 0,25 + 0,21 * 0,45 + 0,08 * 0,22 + 0,26 * 0,35 + 0,19 * 0,27 + 0,08 *0,22 = 0,3195$

У результаті чого, можна сказати що Flask ($R_2 = 0,352$), за думкою експертів, буде кращім вибором для нашого проекту, ніж Django ($R_1 = 0,342$), хоча Django має більшість позитивних якостей, на відміну від Flask, та на останньому місці це FastAPI ($R_3 = 0,3195$).

Це дослідження підтверджує об'єктивні показники продуктивності, визначені за допомогою стороннього розширення Lighthouse. Включення етапу вибору фреймворку та застосування цього експертного методу оцінювання дозволяє обрати найбільш підходящий інструмент для створення конкретного типу сайту і забезпечити найвищу можливу якість.

ВИСНОВКИ

У результаті проведеного дослідження впливу фреймворків веб-розробки на продуктивність можна зробити кілька важливих висновків.

По-перше, порівняння Django, Flask та FastAPI показало, що кожен з них має свої унікальні переваги та особливості. Django, з орієнтацією на повністю готові рішення, вражає своєю високою продуктивністю при швидкій розробці. Flask, з іншого боку, стає відмінним вибором для менших проектів, де потрібна більша гнучкість та контроль. FastAPI, з фокусом на швидкість та автоматизацію, проявив себе як ефективний інструмент для створення API.

По-друге, важливо враховувати вимоги конкретного проекту при виборі фреймворку. Наприклад, якщо важлива велика швидкість відгуку для API, то FastAPI може бути кращим вибором, в той час як для швидкого розгортання повноцінного веб-застосунку Django може бути більш підходящим.

По-третє, результати дослідження слід використовувати як рекомендації для розробників та архітекторів при виборі фреймворку в залежності від конкретних вимог та завдань проекту. При цьому важливо враховувати не лише продуктивність фреймворку, але і його загальну сумісність із завданнями проекту, легкість вивчення та підтримку спільноти розробників.

Усе враховуючи, вибір веб-фреймворку – це компроміс між різними факторами, і розробники повинні ретельно враховувати всі аспекти, перш ніж прийняти остаточне рішення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Django introduction URL - https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction (дата звернення: 17.12.2023).

2. What is Flask? URL - https://www.educative.io/courses/flask-develop-web-applications-in-python/what-is-flask (дата звернення: 17.12.2023).

3. History, Design and Future URL - https://fastapi.tiangolo.com/history-design-future/ (дата звернення: 17.12.2023).

4. What is Django? Advantages and Disadvantages URL - https://hackr.io/blog/what-is-django-advantages-and-disadvantages-of-using-django (дата звернення: 17.12.2023).

5. What Is Flask and How Do Developers Use It? URL - https://careerfoundry.com/en/blog/web-development/what-is-flask/ (дата звернення: 17.12.2023).

6. Comparison of Flask, Django, and FastAPI: Advantages, Disadvantages, and Use Cases URL - https://medium.com/@tubelwj/comparison-of-flask-django-and-fastapi-advantages-disadvantages-and-use-cases-63e7c692382a (дата звернення: 18.12.2023).

7. How Django Works URL - https://masteringdjango.com/django-tutorials/beginner-lesson-2-how-django-works/ (дата звернення: 18.12.2023).

8. Django Architecture URL - https://data-flair.training/blogs/django-architecture/ (дата звернення: 18.12.2023).

9. Flask Architecture URL - https://flask-monitoringdashboard.readthedocs.io/en/latest/developing.html (дата звернення: 18.12.2023).

10. Davydova, I., Marina, O., Peleshchyshyn,, A., Serhii Marin. Webometric Analysis of National Libraries Websites 2nd International Workshop on Control, Optimisation and Analytical Processing of Social Networks (COAPSN-2020), Lviv, Ukraine, May 21, 2020. – Lviv, 2020. – P. 165-176

11. Creating a new Django Project in PyCharm URL: https://www.jetbrains.com/guide/django/tutorials/django-aws/setup-django/ (дата звернення: 01.06.2024).

12. Krakowczyk D.G., Reich D.R., Chwastek J., Jakobi D.N., Prasse P., Süss A., Turuta O., Kasprowski P., Jäger, Lena A. A. Pymovements: A Python Package for Eye Movement Data Processing. Eye Tracking Research and Applications Symposium (ETRA), 2023, 53.

13. Tutorial: Developing FastAPI Applications using K8s & AWS URL: https://blog.jetbrains.com/pycharm/2022/02/tutorial-fastapi-k8s-aws/ (дата звернення: 02.06.2024).

14. Django vs Flask: Which is the Best Python Web Framework URL: https://blog.jetbrains.com/pycharm/2023/11/django-vs-flask-which-is-the-best-python-web-framework/ (дата звернення: 03.06.2024).

15. Django vs FastAPI: Which is the Best Python Web Framework URL: https://blog.jetbrains.com/pycharm/2023/12/django-vs-fastapi-which-is-the-best-python-web-framework/ (дата звернення: 02.06.2024).