МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: **«Інформаційне та програмне забезпечення системи обміну даними між студентами»**

здобувача групи ІНз-91с Грищенка Микити Сергійовича

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

Микита ГРИЩЕНКО

(підпис)

Керівник, доцент,

кандидат фізико-математичних наук Сергій ШАПОВАЛОВ

(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувача групи ІНз-91с Грищенка Микити Сергійовича

1. Тема роботи: «Інформаційне та програмне забезпечення системи обміну даними між студентами»

затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для вирішення завдань дослідження. 3) Розробка інформаційного та програмного забезпечення. 4) Тестування та аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для вирішення завдань дослідження*		
3	*Розробка інформаційного та програмного забезпечення*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти 

(підпис)

Керівник 

(підпис)

АНОТАЦІЯ

Записка: 44 стр., 19 рис., 2 додаток, 11 використаних джерел.

Обґрунтування актуальності теми роботи – Ця дипломна робота присвячена розробці веб-сайту на основі фреймворку Django для обміну даними між студентами.

У роботі розглядається вибір мови програмування, фреймворку, методу рішення та інструментів для розробки. Детально аналізуються переваги та недоліки розробки на Django, вибір IDE та бази даних. Описано план розробки сайту та його реалізацію, включаючи реалізацію системи реєстрації та авторизації, механізмів контролю доступу до публікацій, пагінації та форматування тексту. Також описано тестування сайту та його виведення в експлуатацію.

Об’єкт дослідження — фреймворк Django

Мета роботи — метою даної дипломної роботи є розробка інформаційної та програмної системи обміну даними між студентами.

Методи дослідження — база даних, програмування, фремворки

Результати — розроблено з використанням мови програмування Python веб-сайт на основі фреймворку Django для обміну даними між студентами.

DJANGO, ВЕБ-САЙТ, ОБМІН ДАНИМИ, РЕЄСТРАЦІЯ, АВТОРИЗАЦІЯ, КОНТРОЛЬ ДОСТУПУ, ПАГІНАЦІЯ, ФОРМАТУВАННЯ ТЕКСТУ, ТЕСТУВАННЯ

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 6
ВСТУП 7
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1 Огляд предметної області 9
1.2 Огляд аналогів 10
1.3 Постановка задачі 12
2.ВИБІР МЕТОДОВ РІШЕННЯ ЗАДАЧІ 14
2.1 Вибір мови програмування для реалізації проекту 14
2.2 Вибір фреймворку 16
2.3 Переваги та недоліки розробки на Django 17
2.4 Моделі для Django 18
2.5 Вибір IDE для програмування 19
2.6 Вибір бази даних 21
3. ПРАКТИЧНА РЕАЛІЗАЦІЯ 23
3.1 План розробки сайту 23
3.2 Розробка сайту 28
3.3 Тестування сайту 32
ВИСНОВКИ 43
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 44
ДОДАТОК А 45
ДОДАТОК Б 48

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

1. Django - відкрите програмне забезпечення для веб-фреймворку на мові програмування Python.
2. Python - інтерпретована мова програмування високого рівня.
3. IDE - середовище розробки, яке надає розробникам зручний інтерфейс для розробки програм.
4. База даних - електронна система зберігання та організації даних.
5. Реляційна база даних - тип бази даних, що зберігає дані в таблицях зі зв'язками між ними.
6. MVC - патерн проектування програмного забезпечення, який розділяє програму на моделі, види та контролери.
7. URL - адреса ресурсу в інтернеті.
8. HTTP - протокол передачі гіпертексту.
9. HTML - мова розмітки гіпертексту.
10. CSS - каскадні таблиці стилів для оформлення веб-сторінок.
11. JavaScript - мова програмування, що використовується для динамічної зміни веб-сторінок.
12. ORM - об'єктно-реляційне відображення, що дозволяє використовувати об'єктно-орієнтовану парадигму для роботи з базами даних.
13. Автентифікація - процес перевірки, що користувач є дійсним користувачем системи.
14. Авторизація - процес контролю доступу до ресурсів системи для користувачів.
15. Пагінація - розділення великої кількості даних на сторінки для зручності користувачів.

ВСТУП

Актуальність теми дослідження: Інформаційні технології стали невід'ємною частиною нашого життя. Із зростанням кількості даних, що генеруються кожною секундою, потреба у зручних та ефективних інструментах обробки та обміну цими даними стає невідкладною. Серед таких інструментів особливе місце займають веб-сайти, які забезпечують доступ до інформації та можливість її обміну.

Задача, яку ставить дипломний проект, - розробити інформаційну систему, яка буде допомагати студентам обмінюватися інформацією між собою. Система має надавати можливість студентам публікувати, переглядати, редагувати і видаляти публікації, а також залишати коментарі до них. Крім того, вона повинна мати систему реєстрації та авторизації користувачів з механізмами контролю доступу до публікацій.

Для розробки цієї інформаційної системи було вибрано фреймворк Django, написаний на мові програмування Python. Django є дуже популярним фреймворком для розробки веб-додатків, який дозволяє швидко створювати високоякісні веб-додатки з базою даних, забезпечуючи безпеку, масштабованість та зручність у роботі з веб-сторінками.

Розвиток інформаційних технологій та зростання кількості даних ставлять перед нами виклик створення зручних та ефективних інструментів обробки та обміну цими даними. Дипломний проект спрямований на розробку інформаційної системи для обміну даними студентами, що відповідає актуальним потребам учасників освітнього процесу.

Об'єкт дослідження: Інформаційна система для обміну даними студентами.

Предмет дослідження: Розробка повнофункціонального веб-додатку на базі Django для публікації, перегляду, редагування та видалення публікацій студентами, з механізмами реєстрації, авторизації та контролю доступу.

Метою дипломної роботи: є розробка інформаційної системи, яка допомагатиме студентам обмінюватися інформацією між собою, публікувати, переглядати, редагувати і видаляти публікації, а також залишати коментарі до них. Система повинна мати механізми реєстрації та авторизації користувачів з контролем доступу до публікацій.

Основні завдання:

- Розробка веб-додатку на базі Django.
- Реалізація механізмів реєстрації та авторизації користувачів.
- Створення можливості публікації, перегляду, редагування та видалення публікацій.
- Розробка системи коментування публікацій.
- Реалізація механізмів контролю доступу до публікацій.

Практичне значення дослідження: Розроблена інформаційна система забезпечить студентам зручний та ефективний обмін інформацією та досвідом в навчанні. Вона може бути використана студентами для спілкування, обговорення навчальних питань та спільної роботи над проектами. Додаток може стати важливим інструментом для студентів, які навчаються дистанційно або мають обмежені можливості для спілкування зі своїми одногрупниками.

Наукова новизна: Дипломна робота використовує фреймворк Django для розробки інформаційної системи, що дозволяє ефективно створювати веб-додатки з базою даних, забезпечуючи безпеку, масштабованість та зручність у роботі з веб-сторінками. Розробка інформаційної системи для обміну даними студентами на базі Django є оригінальним внеском у галузь розробки веб-додатків.

Кваліфікаційна робота складається з: Розробки повнофункціонального веб-додатку на базі Django для обміну даними студентами. Реалізовані механізми реєстрації та авторизації користувачів з контролем доступу до публікацій, можливість публікації, перегляду.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

Предметною областю є розробка веб-сайту на основі фреймворку Django для обміну даними між студентами. Цей веб-сайт може бути призначений для студентів різних вузів, які можуть обмінюватися навчальним матеріалом.

На сайті можуть бути різні функції, які допоможуть студентам взаємодіяти між собою. Наприклад, можуть бути форуми для обговорення конкретних тем, чати для спілкування, блоги для написання дописів та публікації своїх думок. Також на сайті можуть бути розділи для завантаження документів, або фото, які допоможуть студентам вивчити певну тему.

Окрім цього, на сайті можуть бути різні інструменти, які допоможуть студентам здійснювати взаємодію та покращувати свої знання. Наприклад, можуть бути онлайн-тести для перевірки знань, відеоуроки для вивчення конкретної теми та інтерактивні вправи для закріплення знань.

У цілому, розробка веб-сайту на основі фреймворку Django для обміну даними між студентами може стати важливим інструментом для навчання та спілкування студентів. Вона дозволить викладачам та студентам ділитися навчальними матеріалами.

Для розробки веб-сайту на основі фреймворку Django для обміну даними між студентами можуть бути використані різні технології та інструменти. Наприклад, можуть бути використані бази даних для зберігання інформації про користувачів та їх дії на сайті, веб-сокети для забезпечення миттєвого спілкування та оновлення сторінок, інтерфейси програмування додатків (API) для підключення до зовнішніх сервісів та бібліотеки для роботи зі зображеннями та відео.

Крім того, для успішної розробки веб-сайту на основі фреймворку Django для обміну даними між студентами необхідно враховувати різні

аспекти, такі як безпека даних, швидкість роботи сайту, можливість масштабування та інші.

Розробка веб-сайту на основі Django - це важлива технологія, яка може допомогти поліпшити якість навчання та збереження навчальних матеріалів для студентів, викладачів та університетів.

1.2 Огляд аналогів

Огляд аналогів та подібних варіантів є важливою складовою розробки будь-якого програмного продукту, в тому числі і веб-сайту на основі програмної платформи Django для обміну даними між студентами. Нижче наведено огляд декількох аналогів та подібних варіантів таких веб-сайтів:

Moodle - це відкрите програмне забезпечення для навчання, яке містить в собі різноманітні функції для організації навчального процесу, включаючи обмін даними між викладачами та студентами.

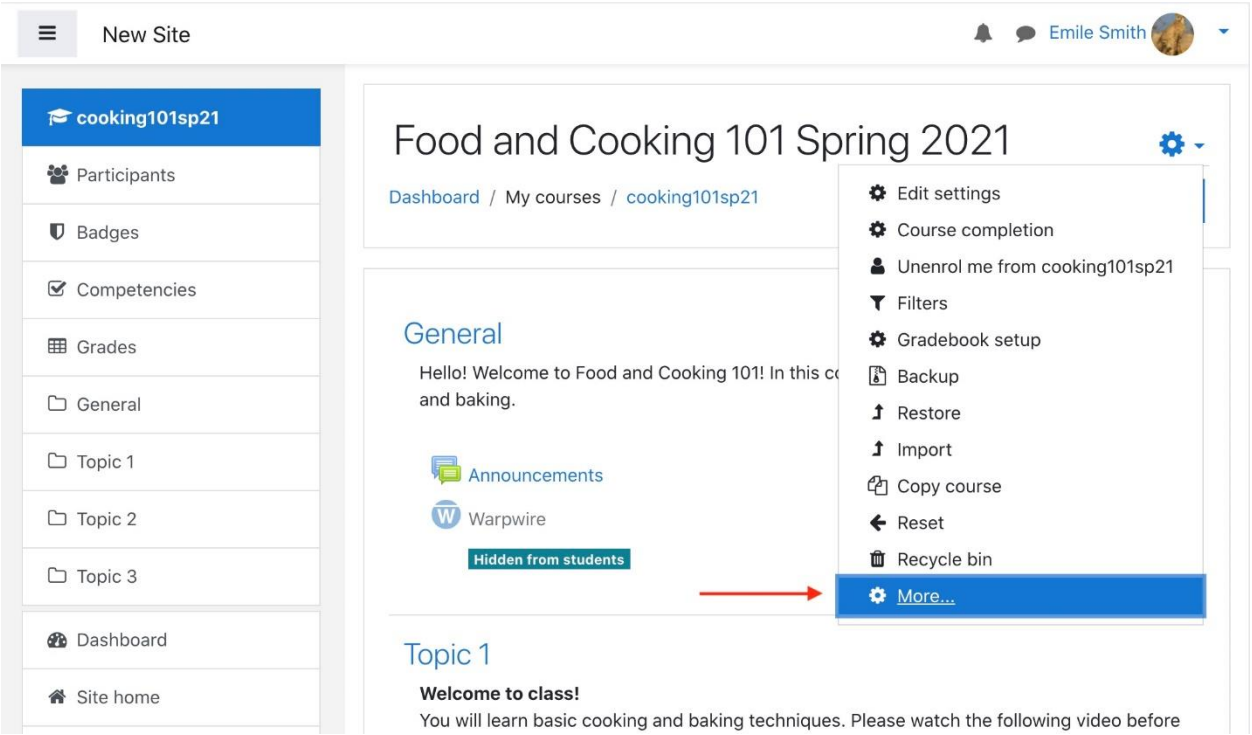

Рисунок 1.2.1 - Moodle

Google Classroom - це інструмент для навчання, який дозволяє викладачам створювати та організовувати класи, спілкуватися зі студентами та обмінюватися матеріалами.

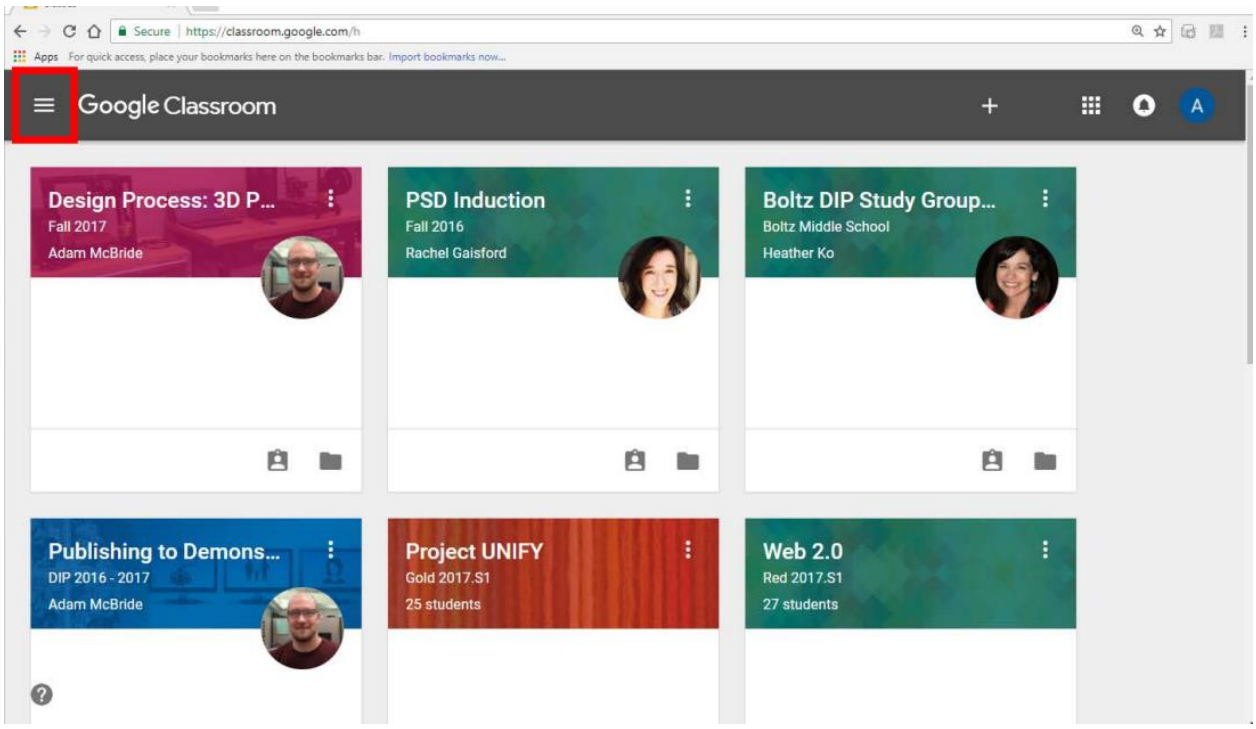

Рисунок 1.2.2 - Google Classroom

Schoology - це платформа для навчання, яка дозволяє викладачам та студентам спілкуватися, створювати та організовувати матеріали для навчання та здійснювати інтерактивне навчання. Крім того, вона містить функції для обміну даними та інтеграції з іншими системами навчання.

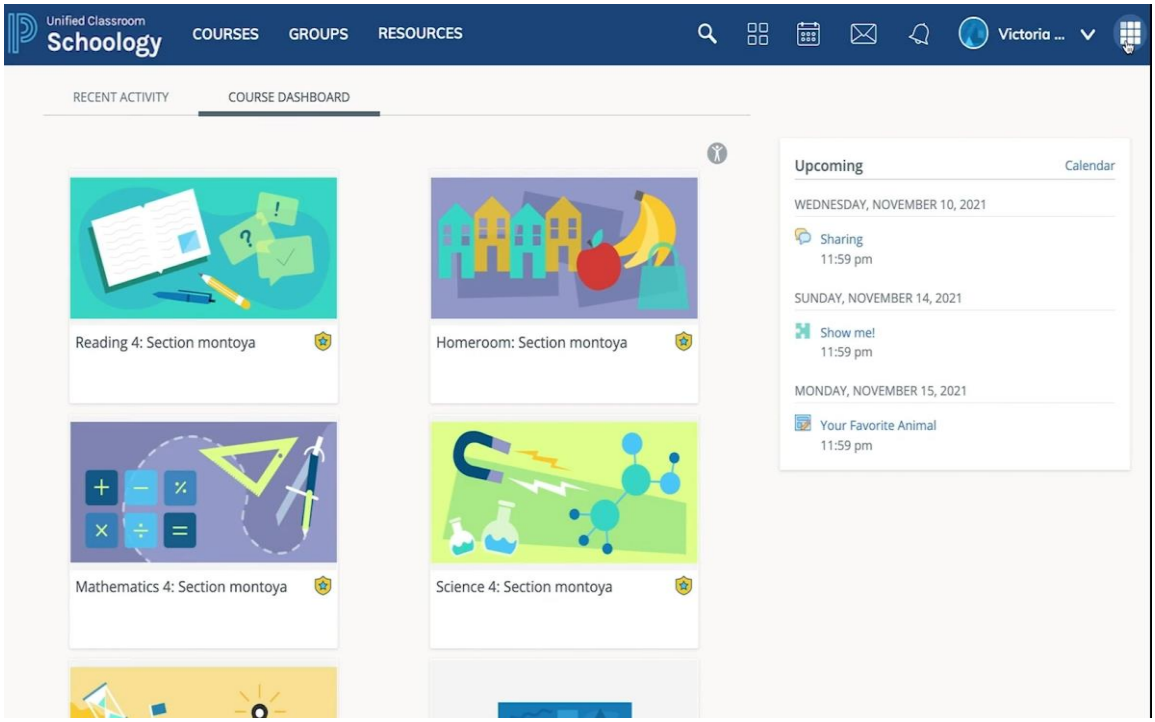

Рисунок 1.2.3 - Schoology

Sakai - це відкрите програмне забезпечення для навчання, яке містить в собі різноманітні інструменти для організації навчального процесу, включаючи обмін даними між викладачами та студентами.

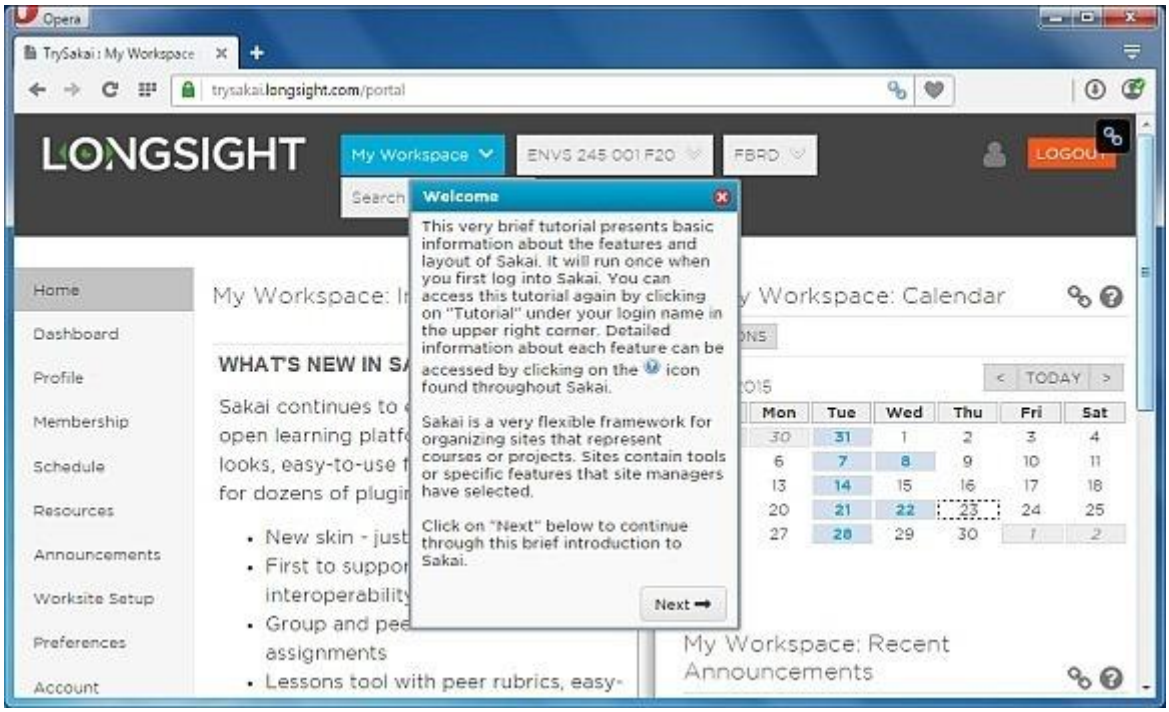

Рисунок 1.2.4 - Sakai

У порівнянні з цими аналогами, веб-сайт на базі Django для обміну даними між студентами може мати ряд переваг, таких як відкритість вихідного коду, можливість зміни та налаштування функціоналу залежно від потреб користувачів, широкий спектр інтеграцій з іншими системами та сервісами, підтримка мультиплатформеності та гнучкість в налаштуванні доступу користувачів до даних.

1.3 Постановка задачі

Метою даної дипломної роботи є розробка інформаційної та програмної системи обміну даними між студентами.

Система буде дозволяти користувачам публікувати, переглядати, редагувати та видаляти публікації, а також залишати коментарі до них. Буде забезпечено систему реєстрації та авторизації, а також механізми контролю доступу до публікацій. Пости будуть розподілені на категорії та буде можливість перегляду публікацій певної категорії або всіх категорій одразу.

Для форматування тексту публікацій будуть доступні різні опції, такі як вставка зображень, таблиць, списків та інші. Для збереження даних буде використана база даних, а розробка системи буде проводитися з використанням фреймворку Django та мови програмування Python.

У рамках дипломної роботи необхідно вирішити наступні задачі:

1. Вибрати мову програмування для реалізації проекту.
2. Вибрати фреймворк для розробки системи.
3. Вибрати метод розв'язання задачі та дослідити переваги та недоліки розробки на Django.
4. Розробити моделі для Django, вибрати IDE для програмування та вибрати базу даних.
5. Створити план розробки сайту та розробити його.
6. Протестувати систему та зробити відповідні корективи.
7. Дати висновки про розробку системи та її можливості.

2.ВИБІР МЕТОДОВ РІШЕННЯ ЗАДАЧІ

2.1 Вибір мови програмування для реалізації проекту

При виборі мови програмування для розробки веб-сайту для обміну даними між студентами, варто розглянути кілька альтернативних мов. Основні мови програмування для розробки веб-сайтів на даний момент - це Python, JavaScript, Ruby, PHP та Java.

Python - це інтерпретована мова програмування з відкритим кодом, яка широко використовується в програмуванні веб-додатків. Python має багато переваг, таких як легка читаємість, простота синтаксису та можливість використовувати різні фреймворки для розробки веб-сайтів. Python є однією з найпопулярніших мов програмування в світі, що забезпечує велику кількість розробників, які можуть допомогти у випадку проблем.

JavaScript - це мова програмування, яка використовується для створення динамічного контенту на веб-сторінках. JavaScript може взаємодіяти з HTML та CSS, що дозволяє створювати багатофункціональні веб-сайти. Однак, JavaScript має деякі недоліки, такі як поганий контроль типів даних та відсутність вбудованої підтримки для багатопоточності.

Ruby - це інтерпретована мова програмування, яка також використовується для розробки веб-додатків. Ruby має дуже чистий та простий синтаксис, що робить його легким для вивчення та розуміння. Однак, Ruby має деякі недоліки, такі як повільність в порівнянні з іншими мовами та відсутність вбудованих засобів масштабування.

PHP - це інтерпретована мова програмування, яка широко використовується для розробки веб-сайтів та додатків. Ця мова спроектована спеціально для веб-розробки і має безліч функцій та бібліотек, які допомагають розробникам швидко створювати різноманітні додатки.

Одним з головних переваг PHP є його простота в використанні та зрозумілість для початківців. Він має простий та легкий синтаксис, що дозволяє швидко створювати функції та скрипти. Крім того, PHP має велику спільноту розробників, яка забезпечує безперервний розвиток мови та

підтримку нових функцій та бібліотек. Також серед недоліків на сьогоднішній день, це проблеми з безпекою бо старі версії PHP мали багато вразливостей, а деякі з них залишаються й досі, що може призвести до вторгнень у веб-додатки, та відсутність типізації: в PHP змінні не потребують попередньої декларації та можуть змінювати свій тип в будь-який момент. Це може призвести до помилок в програмах, особливо великих проектах, де важко відслідковувати всі взаємодії між змінними.

Порівняно з Python, PHP може бути менш ефективним у деяких випадках. Наприклад, він не має такого рівня вбудованих функцій та бібліотек, як Python, що можуть значно полегшити роботу з даними, розробкою машинного навчання та обробкою природної мови. Крім того, PHP не має такої широкої підтримки для наукових та математичних обчислень, як Python.

У загальному, вибір між PHP та Python залежить від потреб конкретного проекту та вимог до функціональності. Коли в роботі потрібна швидка та проста мова для веб-розробки, PHP може бути хорошим вибором. Але якщо планується розробка складних додатків, наукових досліджень або машинного навчання, Python може бути більш підходящим варіантом.

Враховуючи всі переваги, Python був обраний мовою програмування для реалізації проекту. Проте ще варто нагадати що Python, є однією з найбільш популярних мов програмування в світі, що забезпечує багато ресурсів для вивчення та розвитку. Деякі з інших переваг Python включають:

1. Простота вивчення: Python має дуже читабельний синтаксис, що дозволяє новачкам швидко вивчати мову та починати програмувати.
2. Багата бібліотека: Python має велику бібліотеку, яка забезпечує підтримку різноманітних функцій та операцій. Це забезпечує швидкий розвиток та розширення функціональності додатку.
3. Висока продуктивність: Python забезпечує високу продуктивність та ефективність, що дозволяє швидко виконувати завдання.

4. Широкі можливості веб-розробки: Python має багато фреймворків для веб-розробки, таких як Django та Flask, які дозволяють створювати повнофункціональні веб-додатки за короткий час.

5. Підтримка спільноти: Python має велику та активну спільноту розробників, яка надає допомогу в розвитку та вирішенні проблем при програмуванні.

2.2 Вибір фреймворку

Фреймворки для розробки веб-додатків є набором інструментів, бібліотек та різноманітних компонентів, які допомагають програмістам в розробці веб-додатків швидше і ефективніше. Вони мають ряд готових модулів та рішень, які дозволяють зосередитись на розробці бізнес-логіки додатку, замість того, щоб тратити час на написання всього з нуля.

У світі Python є декілька популярних каркасних систем для розробки веб-додатків, таких як Flask, Pyramid, CherryPy, Bottle та Django. Кожен з цих програ має свої переваги та недоліки, але Django - це один з найбільш популярних основ Python, який має велику спільноту розробників та широкий набір функціональності. Django працює за MVT (Model View Template) архітектурою, тобто складається з трьох основних компонентів (рис. 2.2.1)[8].

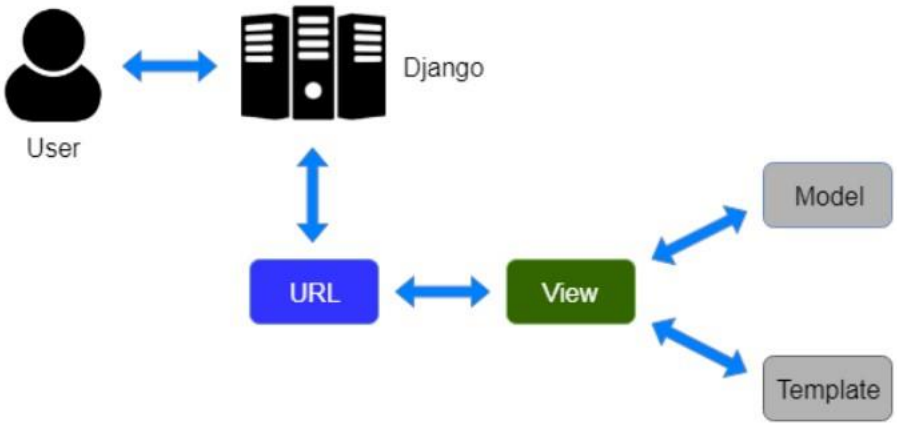

Рисунок 2.2.1 – Принцип роботи Django

Flask є досить легким та простою програмною платформою з більш гнучкою архітектурою, але він має менше вбудованої функціональності

порівняно з Django. Pyramid, з іншого боку, є більш гнучкою основою з більш розширеною архітектурою, що дозволяє розробникам вибирати тільки ті функції та модулі, які їм необхідні. CherryPy та Bottle є менш відомими фреймворками, але вони також досить прості та мають свої переваги.

З урахуванням того, що розробка веб-додатку для обміну даними між студентами вимагає великої функціональності, а також допоміжних інструментів, які забезпечують безпеку та ефективну роботу додатку, Django є кращим вибором для реалізації даного проекту.

2.3 Переваги та недоліки розробки на Django

Django - це високорівневий фреймворк, який дозволяє швидко та ефективно створювати повнофункціональні веб-додатки. Основні переваги Django наступні:

1. Велика і активна спільнота розробників. Django має велику кількість користувачів та активних розробників, що забезпечує постійний розвиток та підтримку фреймворку.
2. Концепція "все є об'єктом" (англ. "Everything is an object"). Django використовує об'єктно-орієнтований підхід до програмування, що сприяє створенню гнучких та легко розширюваних додатків.
3. Автоматична адміністративна панель. Django має вбудований механізм для створення адміністративної панелі, що дозволяє легко керувати базою даних вашого додатку.
4. Підтримка шаблонів. Django має вбудовану підтримку шаблонів, що дозволяє розділити логіку веб-додатку та представлення на різних рівнях.
5. Висока безпека. Django має вбудовані механізми безпеки, такі як захист від SQL-ін'єкцій, CSRF-атак та інші, що забезпечують високу безпеку вашого додатку.

Однак, Django також має деякі недоліки, такі як:

1. Великий розмір. Django має великий розмір та кількість функціональності, що може призвести до зменшення продуктивності та швидкості розробки.

2. Складність. Django може бути складним для новачків, оскільки він має велику кількість функціональності та складну структуру.

3. Обмеженість у виборі бази даних. Django підтримує тільки певні типи баз даних, включаючи PostgreSQL, MySQL, SQLite та Oracle. Це означає, що, якщо вам потрібно використовувати іншу базу даних, вам доведеться самостійно створювати драйвер для цієї бази даних.

Також слід зазначити, що Django має деякі обмеження щодо масштабування. Хоча він може обробляти велику кількість запитів до бази даних, на дуже великих проектах може знадобитися розподілення додатку на декілька серверів, щоб забезпечити швидкість та надійність роботи.

Незважаючи на ці обмеження, Django залишається одним з найпопулярніших програмних платформ для веб-розробки, завдяки своїм перевагам, таким як швидкість розробки, висока якість коду, велика кількість готових компонентів та різноманітних плагінів, які значно спрощують процес розробки.

2.4 Моделі для Django

У каркасній системі Django модель є основним елементом для опису даних та взаємодії з ними в базі даних. В основному моделі визначаються поля, які потім створюються як таблиці в базі даних, тому їх правильний вибір та визначення дуже важливі.

Моделі в Django можуть бути описані за допомогою спеціального класу Model, який містить поля та методи для взаємодії з ними. Наприклад, поле CharField визначає текстове поле, а ForeignKey - зв'язок з іншою моделлю.

Основні типи полів моделей Django:

- BooleanField

- CharField
- DateField
- DateTimeField
- DecimalField
- EmailField
- FileField
- FloatField
- ForeignKey
- ImageField
- IntegerField
- ManyToManyField
- TextField
- TimeField

Кожен тип поля має свої особливості та може бути налаштований за допомогою параметрів.

Також в Django є можливість використовувати унікальність, індексацію та валідацію даних за допомогою параметрів полів.

2.5 Вибір IDE для програмування

Під час розробки програмного забезпечення важливим фактором є вибір зручного та ефективного інструменту для програмування. Сьогодні на ринку представлено багато IDE (інтегрованих середовищ розробки), які пропонують свої функціональні можливості та переваги. У даній дипломній роботі було зроблено вибір PyCharm для розробки веб-сайту на Django, і цей вибір був зумовлений кількома факторами.

Огляд існуючих аналогів:

Серед альтернативних інструментів для розробки веб-сайту на Django можна виділити такі IDE, як Visual Studio Code, Sublime Text, Atom, Eclipse та інші. Кожен з цих інструментів має свої переваги та недоліки.

Наприклад, Visual Studio Code є досить популярним серед програмістів та має велику кількість розширень та плагінів для роботи з різними мовами програмування. Він також має вбудовану підтримку Git, що дозволяє легко працювати з репозиторіями коду. Однак, для повноцінної підтримки Django він потребує додаткових налаштувань та встановлення плагінів.

Sublime Text відрізняється від Visual Studio Code відсутністю вбудованої підтримки Git, але має дуже швидкий та легкий інтерфейс. Atom, як і Visual Studio Code, має велику кількість розширень та плагінів для роботи з різними мовами програмування, але може працювати повільніше на менш потужних комп'ютерах.

Eclipse є однією з найстаріших та найбільш поширених IDE, але його інтерфейс може бути менш інтуїтивним та зручним для роботи з Django.

Порівнявши різні інтегральні середовища розробки, можна зробити висновок, що PyCharm є одним з найбільш зручних та потужних IDE для розробки на Python з відкритими інструментами, такими як Django. Основні переваги PyCharm для роботи з Django включають:

1. Підтримка Django: PyCharm має вбудовану підтримку для Django, що робить розробку на цьому фреймворку простішою та швидшою. Завдяки цьому можна зосередитися на програмуванні, а не на конфігуруванні середовища розробки.
2. Розширені можливості рефакторингу: PyCharm надає широкий набір інструментів для автоматичного переписування коду, що забезпечує його структурованість та покращує його читабельність.
3. Зручний інтерфейс: PyCharm має інтуїтивно зрозумілий інтерфейс користувача, що дозволяє зосередитися на роботі з кодом, а не на навчанні користувача роботі з IDE.
4. Вбудована система контролю версій: PyCharm підтримує популярні системи контролю версій, такі як Git, і дозволяє легко використовувати їх прямо з IDE.

5. Розширені можливості дебагінгу: PyCharm має вбудований дебагер, що дозволяє зупиняти виконання програми в будь-якому місці та аналізувати її стан.

Загалом, PyCharm є потужним та зручним середовищем розробки, що дозволяє розробникам швидко та ефективно працювати з Python та Django. Його функціональність та зручний інтерфейс роблять його найкращим вибором для розробки проекту на Django.

2.6 Вибір бази даних

База даних db.sqlite3, що використовується в Django, є вбудованою базою даних SQLite. Вона має кілька переваг, але також має свої недоліки порівняно з MySQL та PostgreSQL.

Достатки бази даних db.sqlite3:

- Вона є вбудованою і не потребує додаткової установки;
- Вона має малі вимоги до ресурсів і підходить для розробки невеликих проектів;
- Вона підтримує транзакції, що дозволяє зберігати цілісність даних;
- Вона має досить швидкий доступ до даних.

Недоліки бази даних db.sqlite3:

- Вона не підходить для великих проектів, оскільки має обмеження на розмір бази даних і кількість одночасних з'єднань;
- Вона не має підтримки для деяких типів даних, таких як JSON;
- Вона не підтримує віддалений доступ до бази даних.

MySQL і PostgreSQL є більш потужними базами даних з більш широкими можливостями і можуть використовуватися для проектів будь-якого розміру. Вони підтримують віддалений доступ до баз даних, мають багаті можливості резервного копіювання та відновлення даних, а також підтримують більшість типів даних, включаючи JSON.

MySQL і PostgreSQL також мають свої переваги і недоліки. Наприклад, MySQL має швидкий доступ до даних, а PostgreSQL має більш продуману систему консистентних даних.

Отже, вибір бази даних залежить від потреб проекту. Якщо проект невеликий і не потребує великої кількості ресурсів, база даних db.sqlite3 може бути гарним вибором. Однак, якщо проект великий і потребує більш потужної бази даних, MySQL або PostgreSQL можуть бути кращим вибором.

На основі вказаних факторів, можна зробити висновок, що для даного проекту використання бази даних db.sqlite3 є вдалим варіантом.

Переваги db.sqlite3 включають:

- Легка встановлення і налаштування: db.sqlite3 є легкою для встановлення та налаштування, що зменшує час і зусилля, потрібні для підготовки бази даних.
- Мале використання ресурсів: db.sqlite3 використовує мало ресурсів системи, що є важливим для малих проектів, як у вашому випадку.
- Простий синтаксис: синтаксис для роботи з db.sqlite3 є простим та зрозумілим для більшості розробників, що сприяє швидкому розробленню та тестуванню коду.
- Підтримка Django: Django має добру підтримку для db.sqlite3, що забезпечує гладку роботу між веб-сайтом та базою даних.

Треба зазначити, що db.sqlite3 має обмеження щодо обсягу даних, що можуть бути збережені в базі даних. Однак це легка, ефективна та зручна база даних для малого проекту.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 План розробки сайту

У пункті план розробки сайту - міститься опис послідовності етапів розробки веб-сайту з метою досягнення максимальної ефективності та мінімізації ризиків. Обравши основними етапами розробки:

Аналіз та планування:

- Визначення цілей та задач сайту
- Аналіз цільової аудиторії
- Визначення функціональних вимог та можливостей сайту
- Визначення технічних вимог до сайту
- Планування структури сайту

1. Визначення цілей та задач сайту:

Головною метою сайту є створення системи обміну даними між студентами, що дозволяє користувачам публікувати, переглядати, редагувати та видаляти публікації, а також коментувати їх.

Додатковою метою є забезпечення безпеки використання сайту, зокрема забезпечення безпеки входу користувачів за допомогою системи реєстрації та авторизації, а також контролю доступу до публікацій.

2. Аналіз цільової аудиторії:

Цільовою аудиторією є студенти та викладачі, які зацікавлені в обміні даними та інформацією на певні теми.

До аудиторії також можуть належати інші користувачі, які бажають отримувати корисну інформацію та спілкуватись зі студентами та викладачами.

3. Визначення функціональних вимог та можливостей сайту:

- Користувачі можуть публікувати, переглядати, редагувати та видаляти публікації на сайті.
- Система реєстрації та авторизації забезпечує безпеку використання сайту.
- Контроль доступу до публікацій забезпечує безпеку та захист приватної інформації користувачів.
- Публікації розділені на категорії для зручності навігації користувачів.
- Публікації та коментарі підтримують пагінацію для зручного перегляду великої кількості інформації.
- Форматування тексту публікацій дозволяє користувачам використовувати різноманітні функції для зручного виводу та обробки інформації.

4. Визначення технічних вимог до сайту

Мінімальні вимоги до програмно-апаратної частини для розгортання веб-додатку на основі Django та Python зазвичай визначаються рекомендаціями розробників.

Зазвичай ці вимоги пов'язані з версією Python, Django та операційної системи, що використовується. Загалом, мінімальні вимоги для розгортання веб-додатку на Django можуть бути наступними:

Мінімальні вимоги до програмного забезпечення:

- Python версії 3.10 або вище
- Django версії 4.1.7

Мінімальні вимоги до апаратного забезпечення:

- Процесор з частотою не менше 1 ГГц
- Оперативна пам'ять не менше 512 МБ
- Мінімум 100 МБ вільного місця на жорсткому диску

Щодо сервера для розташування веб-додатку, то він також повинен відповідати певним вимогам, зокрема:

- Можливість встановлення та запуску веб-сервера Nginx

- Наявність підтримки віртуалізації, якщо використовується віртуальний сервер
- Мінімум 1 ГБ оперативної пам'яті для стабільної роботи веб-додатку
- Доступ до бази даних, яка використовується в веб-додатку.

Загалом, вимоги до програмно-апаратної частини можуть змінюватись в залежності від конкретної реалізації веб-додатку, його розміру та специфіки використовуваних функцій.

5. Планування структури сайту

Маючи усі необхідні вхідні дані потрібно розробити план структури сайту, щоб по ньому робити усі необхідні сторінки, та функціонал. Почати треба с розробки діаграм. Виходячи з основних поставлених задач та принципів побудови веб-сайтів, створимо діаграму для відображення структури сайту (рис.3.1.1)

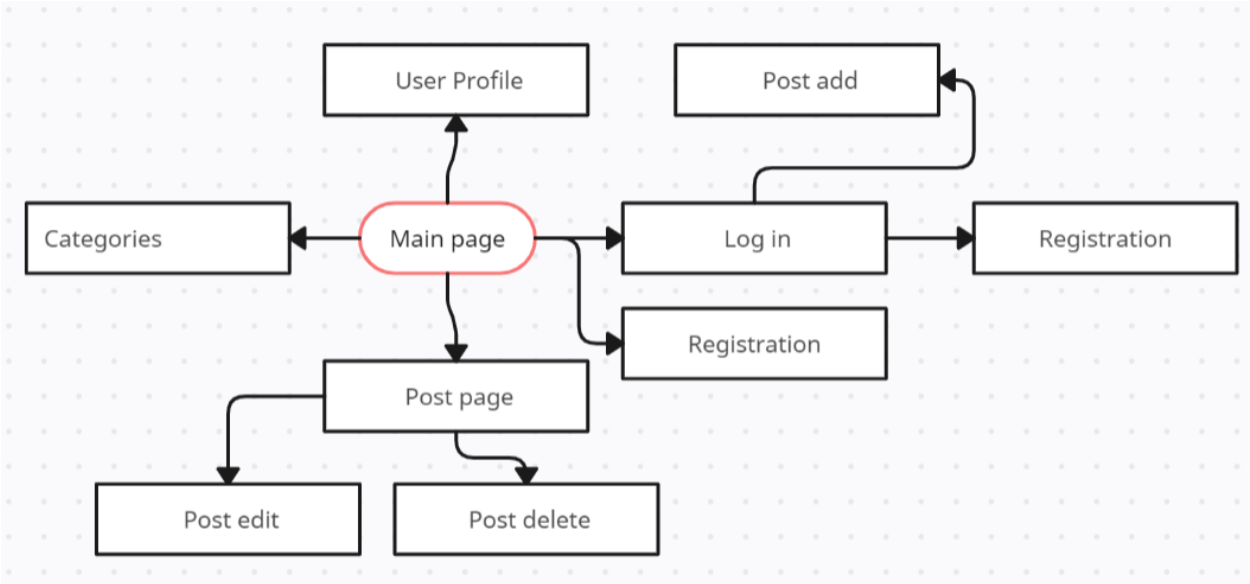

Рисунок 3.1.1 – UML diagram

Далі потрібно створити UML діаграму, в які зазначу куди мають доступи авторизований та не авторизований користувачі рис(3.1.2).

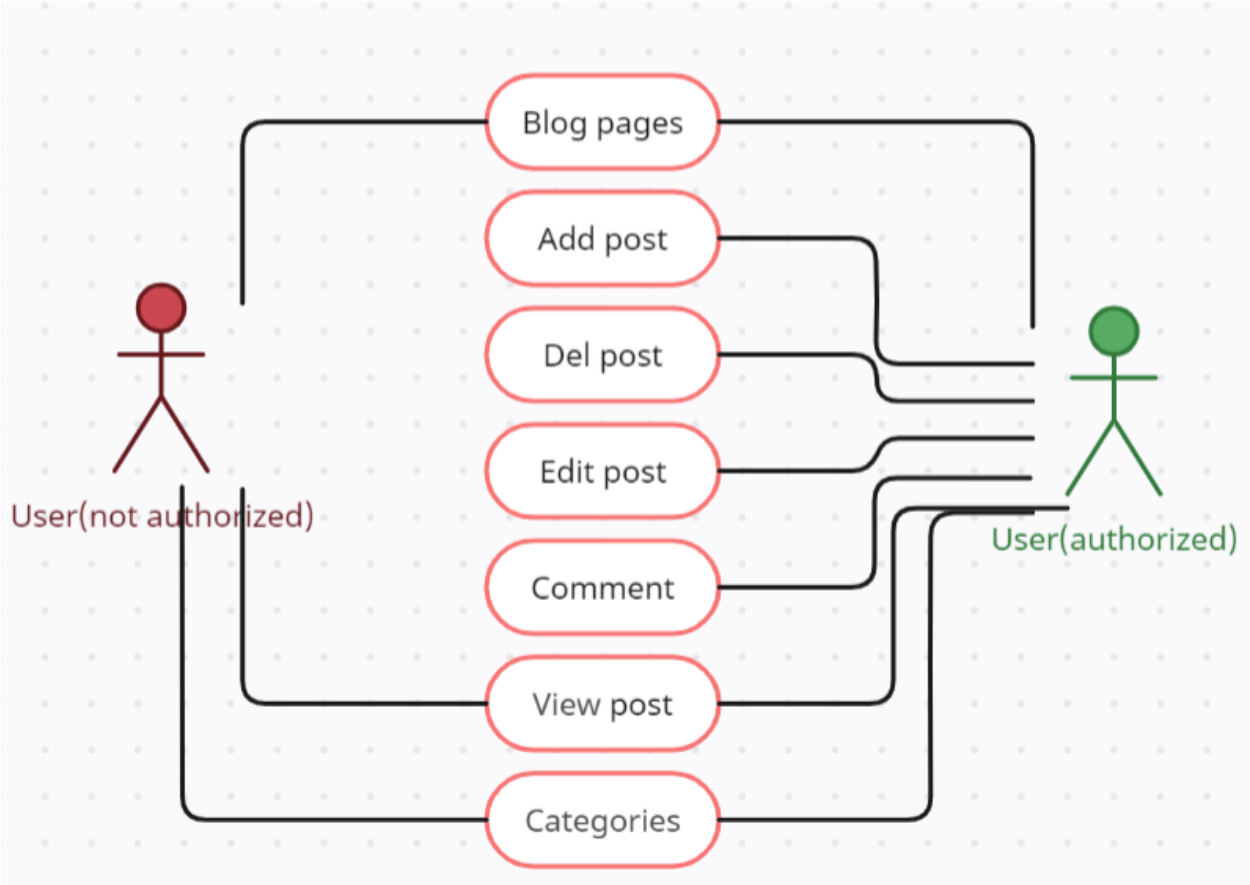

Рисунок 3.1.2 – UML diagram

На головних сторінках буде пагінація, максимум на сторінці повинно бути 10 блогів.

- Розробка візуального дизайну та користувацького інтерфейсу

Наверстаємо html сторінки, css буду використовувати bootstrap.min.css

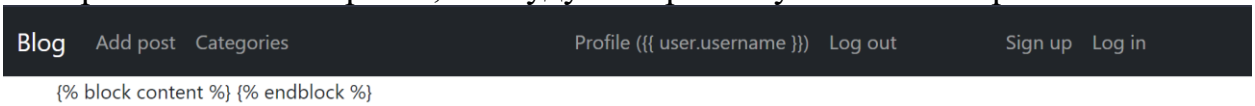

Рисунок 3.1.3 – Головна сторінка сайтів.

- Проектування бази даних

У даному проекті було зроблено прості бази даних під django. (рис. 3.1.3). Розписаний код створення моделей бд буде у додатку (А).

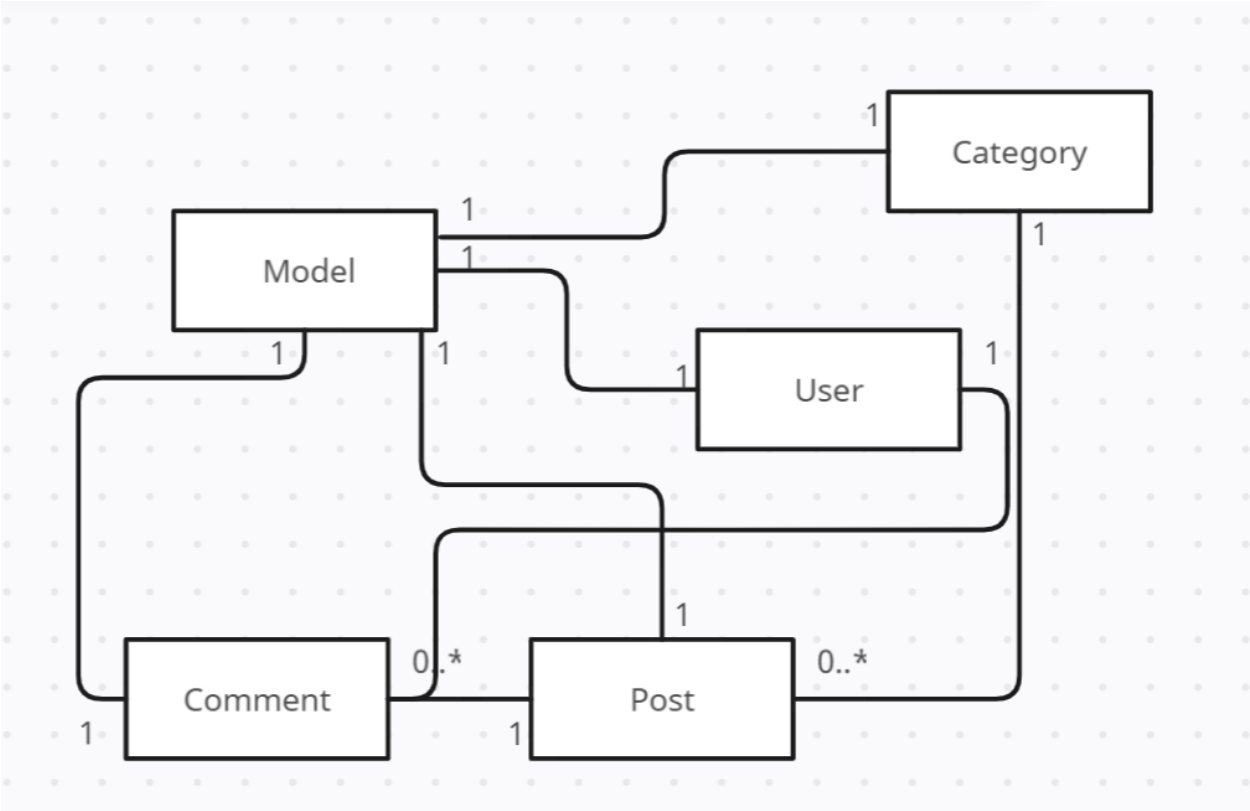

Рисунок 3.1.3 – ER діаграма

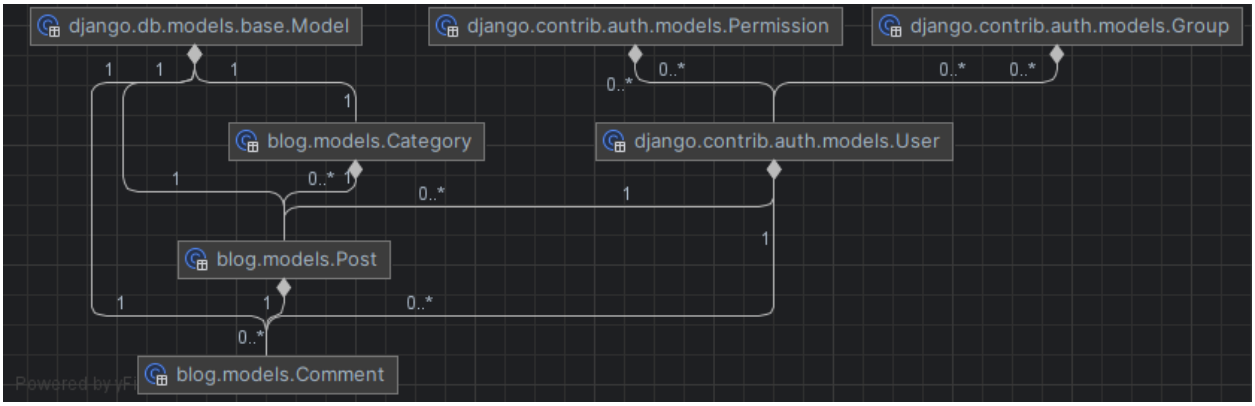

Рисунок 3.1.4 – Таблиці бд у PyCharm

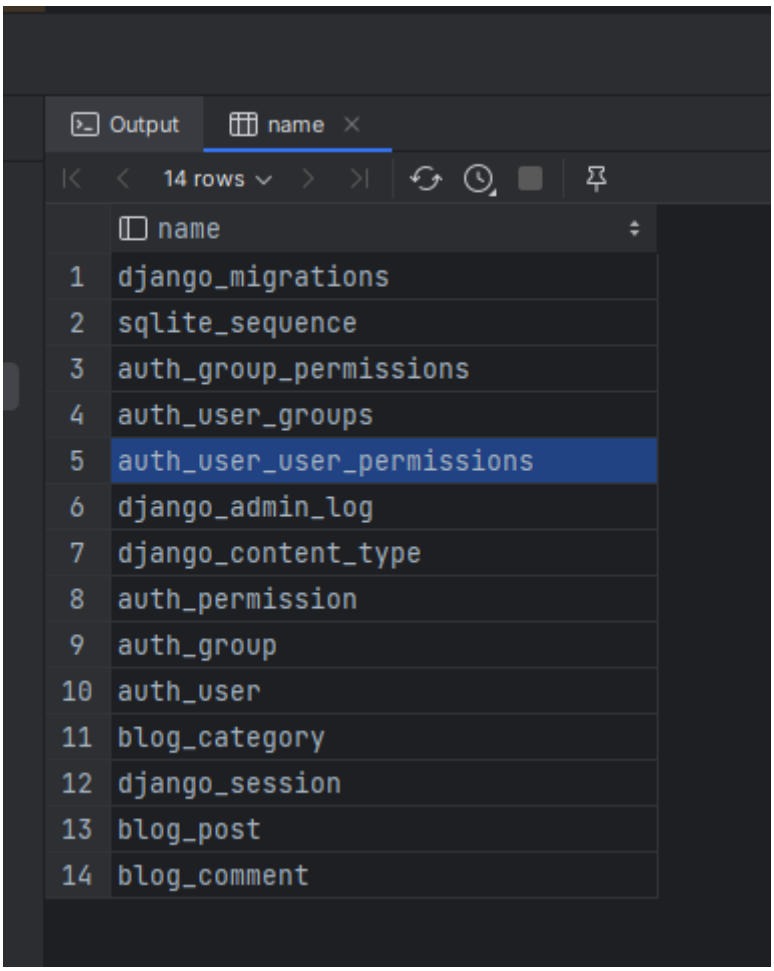

Рисунок 3.1.5 – Таблиці бд

3.2 Розробка сайту

Для початку розробки сайту, потрібно запустити IDE PyCharm у якому потрібно створити проект Django. Після створення проекту потрібно створити додаток. Зробити це можна через консоль PyCharm , в якій потрібно ввести:

python manage.py startapp blog

Після цього потрібно налаштувати базу даних. Для цього відкривши файл settings.py, що знаходиться в папці з проектом, та встановіть налаштування для бази даних.

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

```
}
```

Далі створюю моделі для записів у блозі. Для цього в файлі models.py, що знаходиться у папці з додатком створюю клас моделі запису. Нижче наведу лише одну моделі, ознайомитись з кодом інших моделей, можна у додатках(А,Б) :

```
# створюємо модель категорії для блогу
class Category(models.Model):
    """
    Model that represents a category of blog posts.
    Attributes:
        name: name of the category
    """
    # встановлюємо атрибут name
    name = models.CharField(max_length=255, unique=True)
    # визначаємо метод __str__ для відображення об'єктів моделі
    def __str__(self):
        return f"{self.name}"
    # встановлюємо додаткові параметри моделі
    class Meta:
        verbose_name_plural = "Categories"  # Встановлюємо назву моделі у множині
```

У цьому коді створюється клас моделі Category, який має поле "name". Після цього необхідно створити міграції для моделі. Для цього відкривши консоль PyCharm та виконую наступну команду:

```
python manage.py makemigrations
```

Ця команда створить файл міграції для моделі. Потім потрібно застосувати міграції до бази даних. Для цього застосовую команду:

```
python manage.py migrate
```

Ця команда створить таблицю для моделі у базі даних.

Після цього можна створити адміністративний інтерфейс для записів у блозі де зареєструю моделі в адмінці.

```
# реєструємо моделі в адмінці
admin.site.register(Category)
```

Тепер потрібно створити супер користувача Django, який може управляти блогом. Для цього відкривши командний рядок у директорії проекту та виконаю наступну команду:

```
python manage.py createsuperuser
```

Тепер запустивши локальний сервер Django за допомогою команди:

```
python manage.py runserver
```

Після чого можна відкрити браузер і введіть адресу http://127.0.0.1:8000/admin/. Увійти є змога за допомогою імені користувача та пароля, який створив раніше, коли авторизація успішна буде ридеркт до сторінки адміністративного інтерфейсу Django. Де можно буде додати вже першу категорію до блогу.

Продовживши програмування потрібно додати у файл проекту forms.py, форми для інших моделей:

```
    # створюємо форму для моделі Post
class PostForm(forms.ModelForm):
  class Meta:
    # вказуємо модель, для якої створюється форма
    model = Post
    # вказуємо поля, які будуть включені в форму
    fields = ["category", "title", "snippet", "body"]

    # налаштовуємо віджети для полів форми
    widgets = {
      "category": forms.Select(attrs={"class": "form-control"}),
      "title": forms.TextInput(attrs={"class": "form-control"}),
      "snippet": forms.TextInput(attrs={"class": "form-control"}),
      "body": forms.Textarea(attrs={"class": "form-control"}),
    }
# створюємо форму для моделі Comment
class CommentForm(forms.ModelForm):
  class Meta:
    # вказуємо модель, для якої створюється форма
```

```
        model = Comment
        # вказуємо поля, які будуть включені в форму
        fields = ["text"]
        # налаштовуємо віджети для полів форми
        widgets = {"text": forms.Textarea(attrs={"class": "form-control"})}
```

Також необхідно попердньо наверстані сторінки bootstrap підключити до проекту, аби все не обхідне ми бачили та робилось оброблення необхідних даних

```
        # визначаємо клас для відображення деталей категорії
def CategoryView(request, pk):
    # Отримуємо категорію за pk або повертаємо 404 помилку
    category = get_object_or_404(Category, pk=pk)
    # Отримуємо всі пости з цієї категорії та сортуємо їх за id
    posts = Post.objects.filter(category=category).order_by("-id")
    # Створюємо пагінацію для постів
    paginator = Paginator(posts, 10)
    page = request.GET.get("page")
    try:
        posts = paginator.page(page)
    except PageNotAnInteger:
        # Якщо сторінка не є цілим числом, повертаємо першу сторінку
        posts = paginator.page(1)
    except EmptyPage:
        # Якщо сторінка поза діапазоном, повертаємо останню сторінку
        posts = paginator.page(paginator.num_pages)
    # Створюємо контекст для шаблону
    context = {
        "category": category,
        "posts": posts,
    }
    # Рендеримо шаблон з контекстом
    return render(request, "blog/category.html", context)
```

Далі у файлі urls.py потрібно визначити URL-адресу до шаблону

```
        # імпортуємо необхідні модулі та функції
from django.urls import path
from .views import (
```

```
    CategoriesView,
)
# визначаємо шаблони URL-адрес
urlpatterns = [
    # сторінка зі списком категорій
    path("posts/categories", CategoriesView.as_view(), name="categories")
]
```

Далі потрібно продовжувати програмування, робивши сторінки згідно створеного тз, після чого потрібно провести тестування розробленого веб-додатку. Ознайомитись з усім кодом та коментарями можна у додатках.

3.3 Тестування сайту

Тестування сайту є необхідним етапом розробки, оскільки дозволяє виявляти та виправляти помилки та недоліки в роботі сайту. Незважаючи на те, що ваш проект може працювати стабільно на вашому комп'ютері, тестування дозволить переконатися, що все працює правильно на різних пристроях, в різних браузерах та операційних системах.

Під час тестування необхідно переконатися, що всі функції та функціонал сайту працюють належним чином, в тому числі:

1. Перевірити правильність реєстрації та авторизації користувачів.
2. Перевірити, чи можуть користувачі додавати, редагувати та видаляти публікації та коментарі.
3. Перевірити, чи можуть користувачі переглядати публікації з різних категорій та застосовувати пагінацію.
4. Перевірити, чи правильно працюють функції форматування тексту, вставки зображень, таблиць та списків.
5. Перевірити, чи дійсно функції контролю доступу працюють належним чином та користувачі не можуть видаляти чужі публікації.

Крім того, під час тестування необхідно перевірити, чи працює сайт правильно під час навантаження та чи відповідає він стандартам безпеки.

Після успішного завершення тестування та виправлення всіх помилок та недоліків, можна стверджувати, що сайт працює належним чином та відповідає всім вимогам та очікуванням.

Для початку почну с запуску PyCharm проект буду тестувати на локальному веб сервері, відкривши проект, та запустивши його, шляхи показані на рисунках.

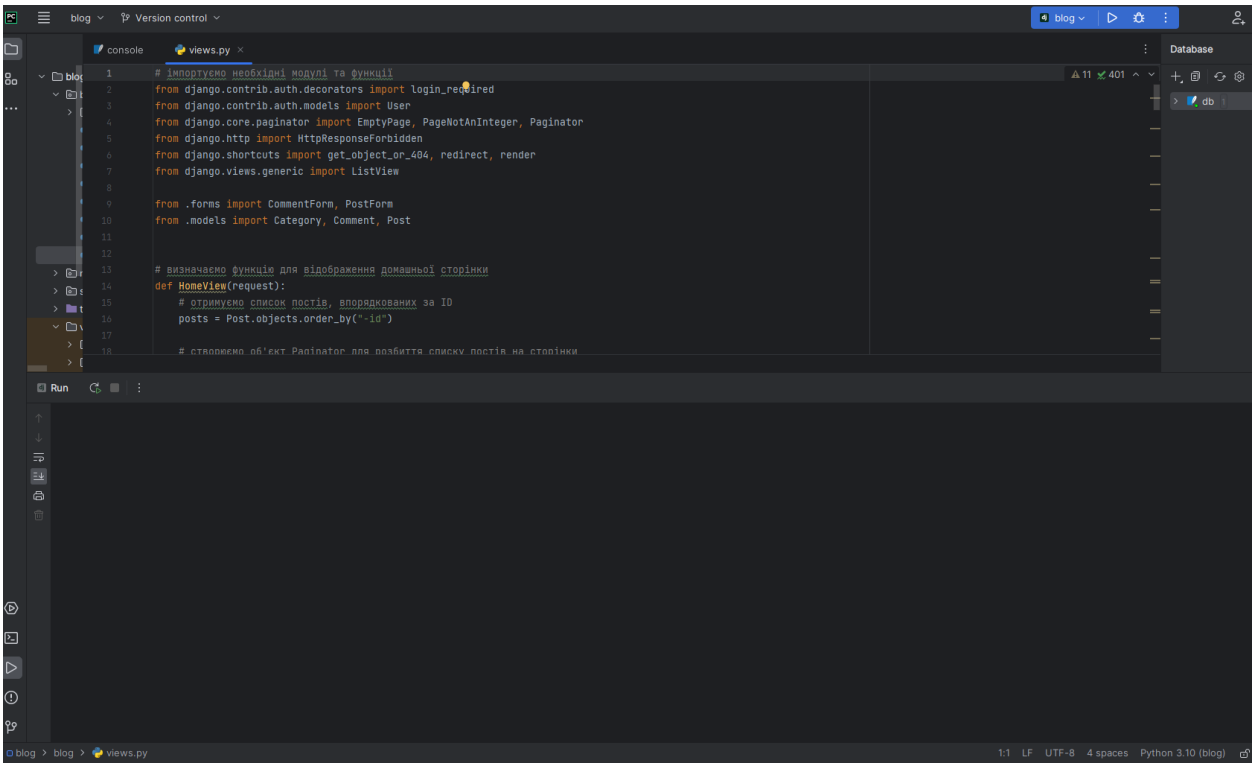

Рисунок 3.3.1 – Проект PyCharm

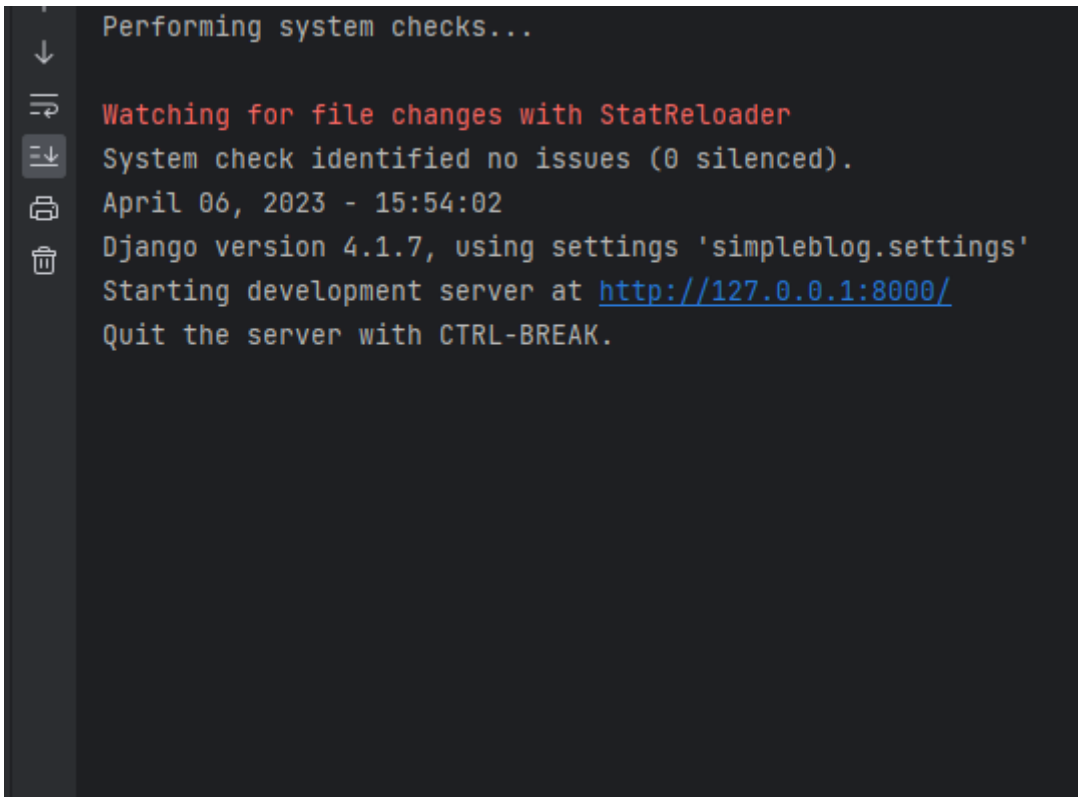

Рисунок 3.3.2 – Запущений проект

Далі відкриваю браузер та ввожу в адресну строку. http://127.0.0.1:8000/posts/ та потраплю на головну сторінку, сайту.

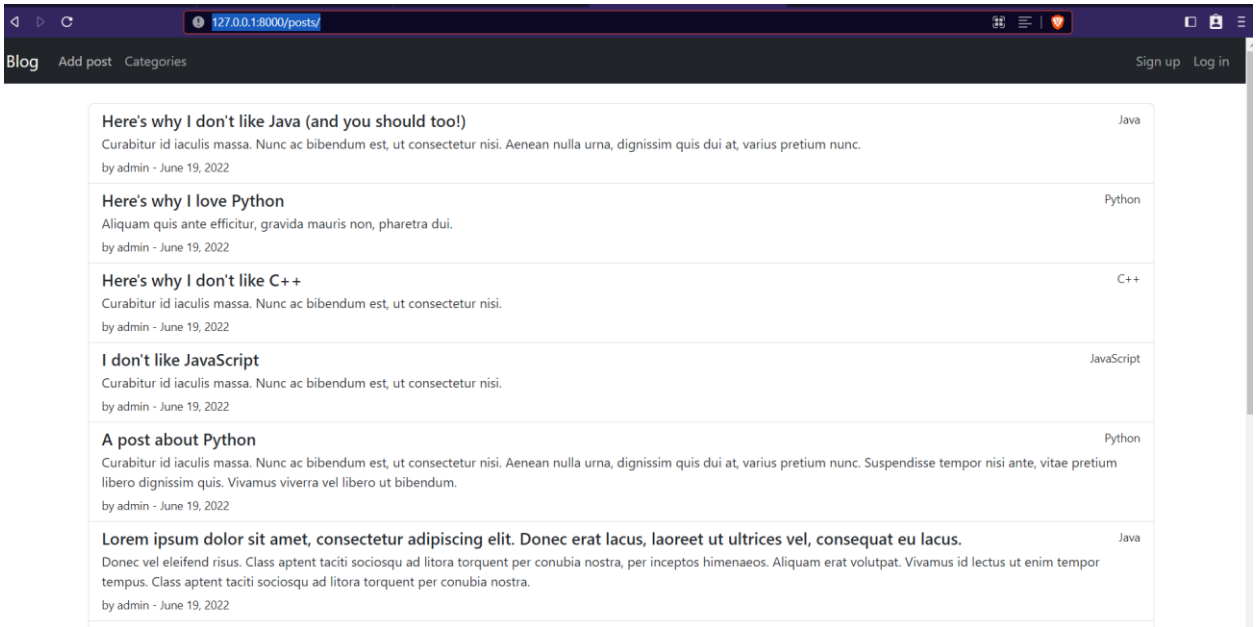

Рисунок. 3.3.3 – Відкрити сторінка постів

Перевіряю пагінацію, при натисканні на Next повинна відкритись нова сторінка, при натисканні на » повинна відкритись остання сторінка

Рисунок 3.3.4 – Перевірка пагінації

Перевірка пагінації пройшла успішно, усе працює як повинно було бути. Додавати нові категорії можна через адмінку. Для цього потрібно перейти на /admin та зайти як адміністратор . Використовуючи відповідний логін та пароль, відокремлений від основного сайту.

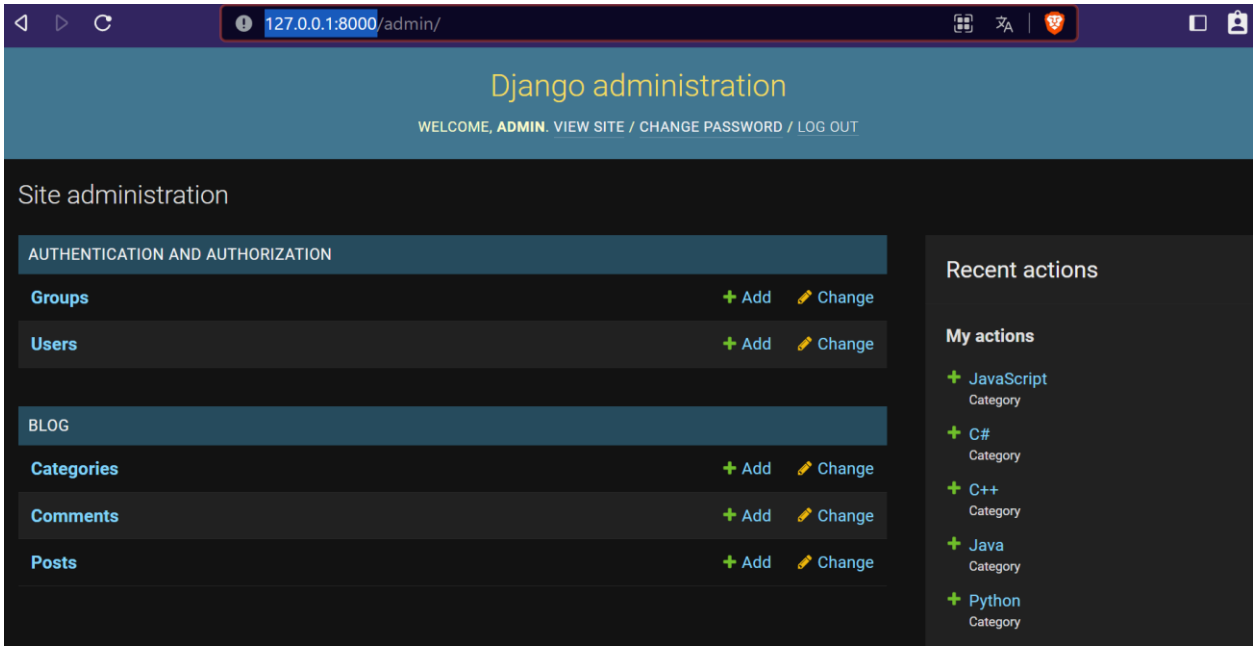

Рисунок 3.3.5 – Адмінка Django

Потрібно протестувати сторінку реєстрації натискаю на «Sign up»

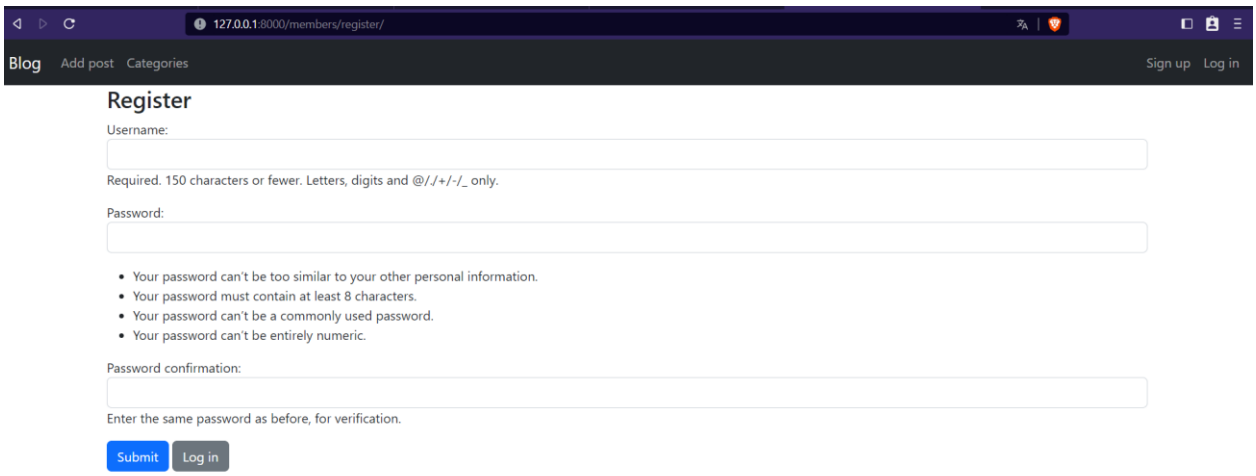

Рисунок 3.3.6 - Реєстрація

Заповнюємо поля тестовими даними, та натискаємо Submit. При успішному вводі повинно відразу перекинути на сторінку авторизації.

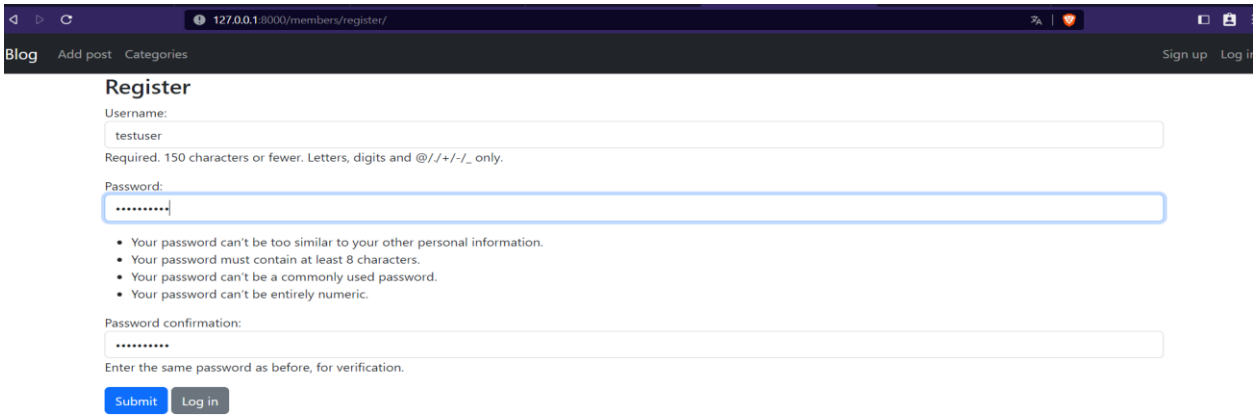

Рисунок 3.3.7 – Введення даних

Якщо поля не будуть заповнені , про це буде сповіщено

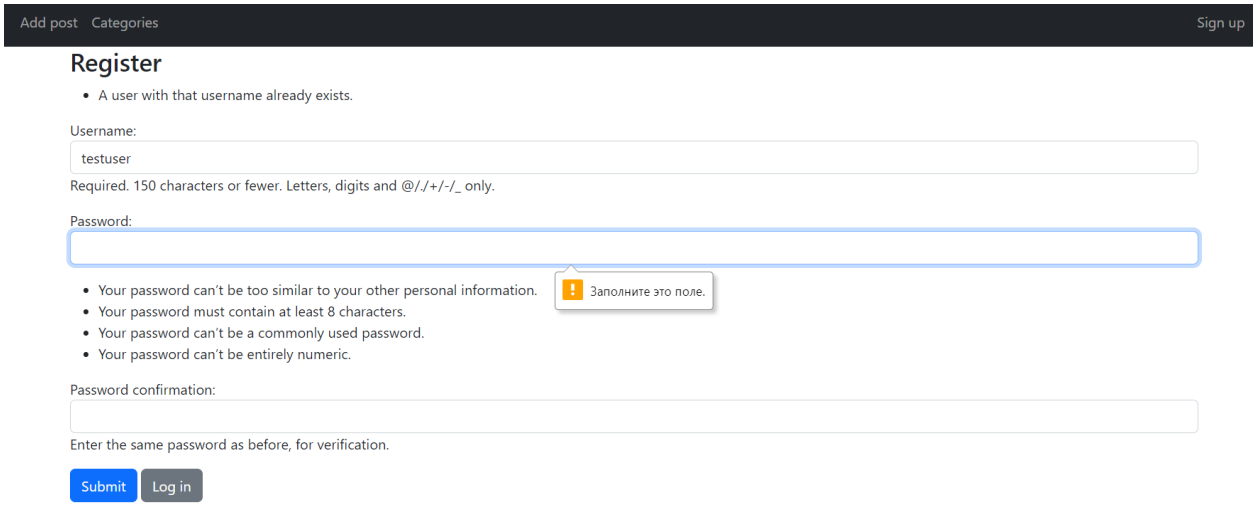

Рисунок 3.3.8 – Введення не вірних даних

Інші сповіщення будуть з’являтись у нижчій часті сайту. Місце розташування сповіщень показано на рисунку нижче.

Рисунок 3.3.9 – Відображення помилок

При відкритті сторінки блогу, не авторизованим юзером, буде можливість лише переглянути текст, залишити коментар будучі не авторизованим не можливо. Потрібно це перевірити, заходимо на сторінку не авторизованим користувачем, потім пишу test у коментарях та натискаємо «Comment». Усе працює як потрібно. Юзер не авторизовано і повинно перекинути до сторінки авторизації.

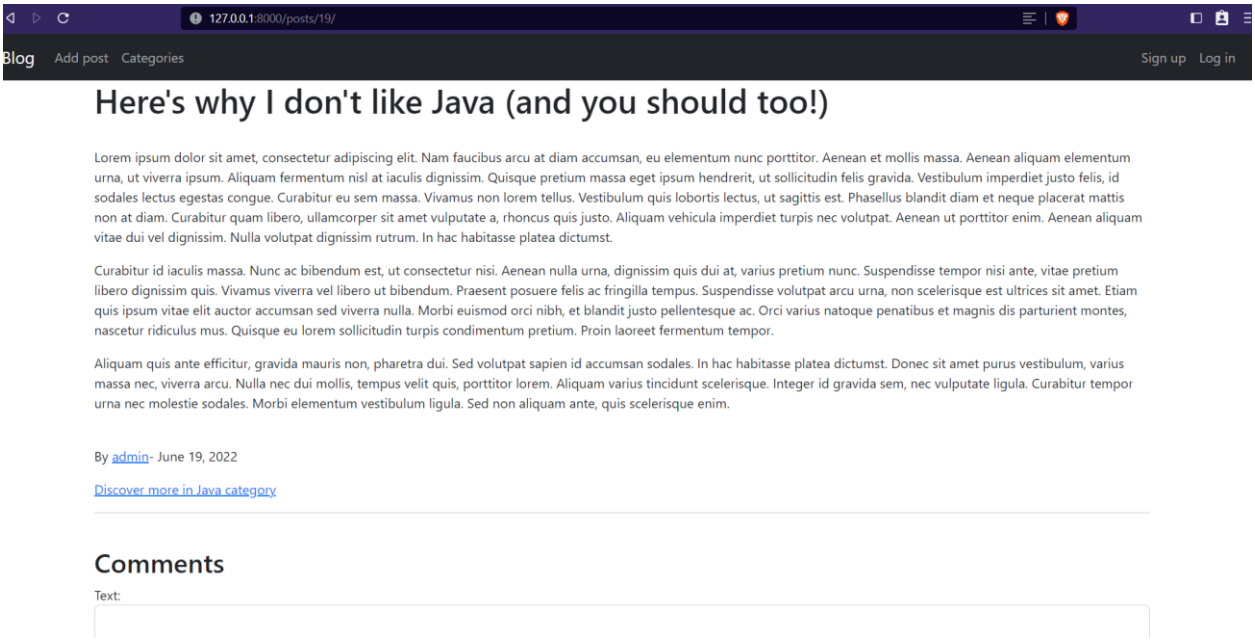

Рисунок 3.3.10 – Перегляд блогу

Наступним буде тестування з авторизацією, переходим за кнопкою “Log in”, після чого можна побачити наступну сторінку:

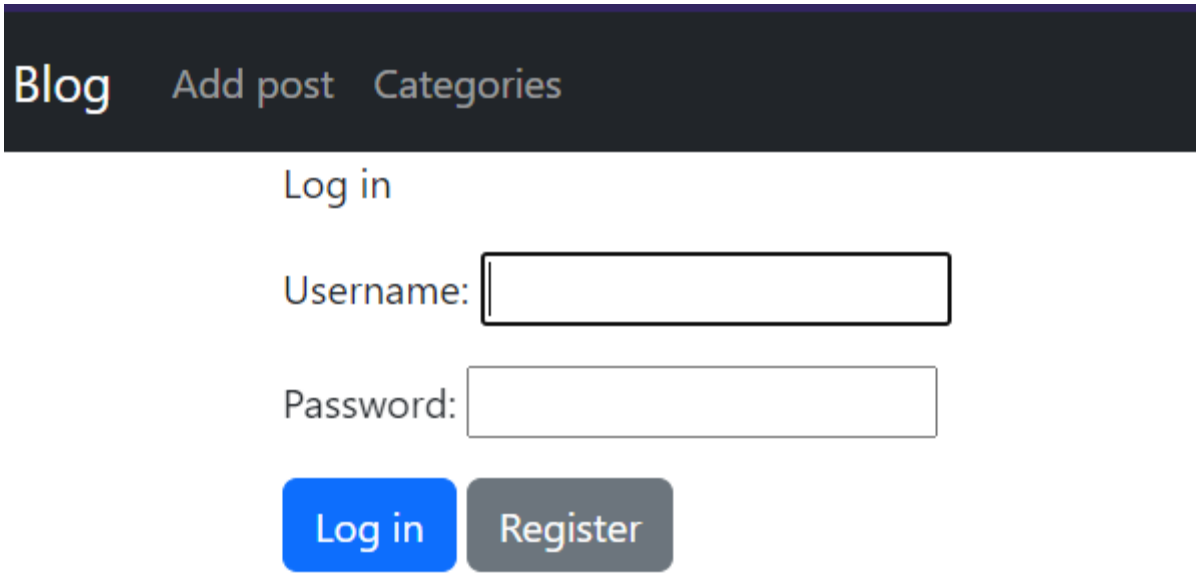

Рисунок 3.3.11 - Авторизація

При невірному вводі даних, буде відображено відповідне сповіщення, та буде змога повторно ввести логін та пароль.

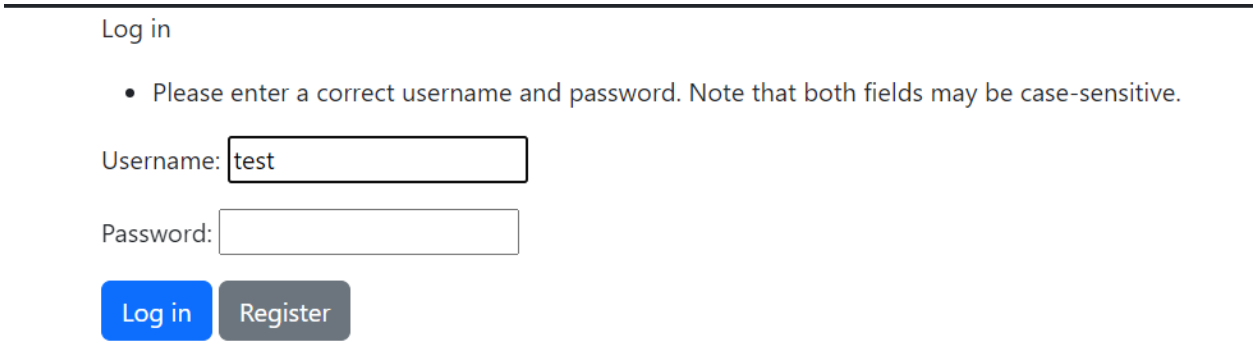

Рисунок 3.3.12 – Помилка при авторизації

При валідному вводі даних до сторінки авторизації, автоматично відбудеться перехід до головної сторінки, вже з статусом авторизованого юзера.

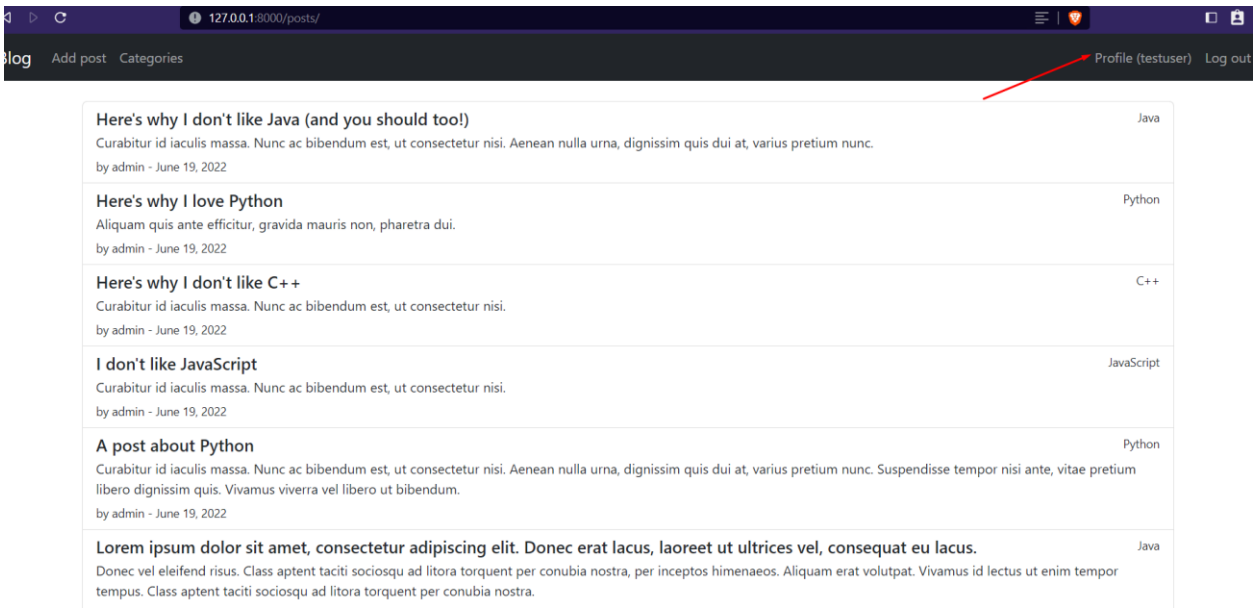

Рисунок 3.3.13 – Профіль юзера

При авторизованому юзері потрібно протестувати коментарі, авторизуюсь, та тестую знов, пишу тест та натискаю «Comment», у нижній частині сторінки повинен появитись коментар, як на рисунку нижче.

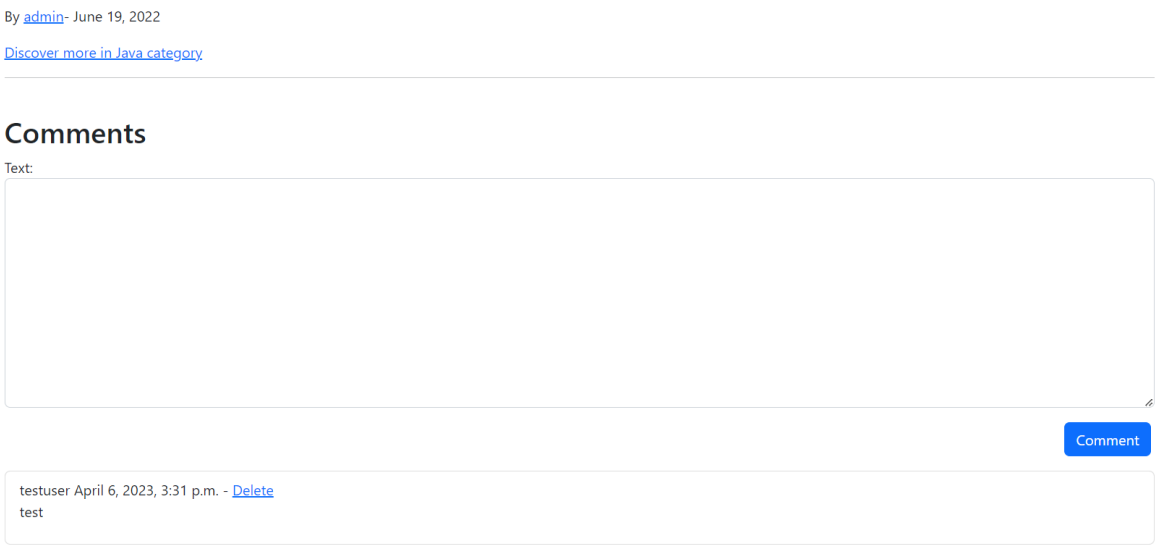

Рисунок 3.3.14 – Додання коментаря

Наступним потрібно перевірити додання статей натискаючи «Add post» відобразиться наступна сторінка, її потрібно заповнити, та натиснути «Post»

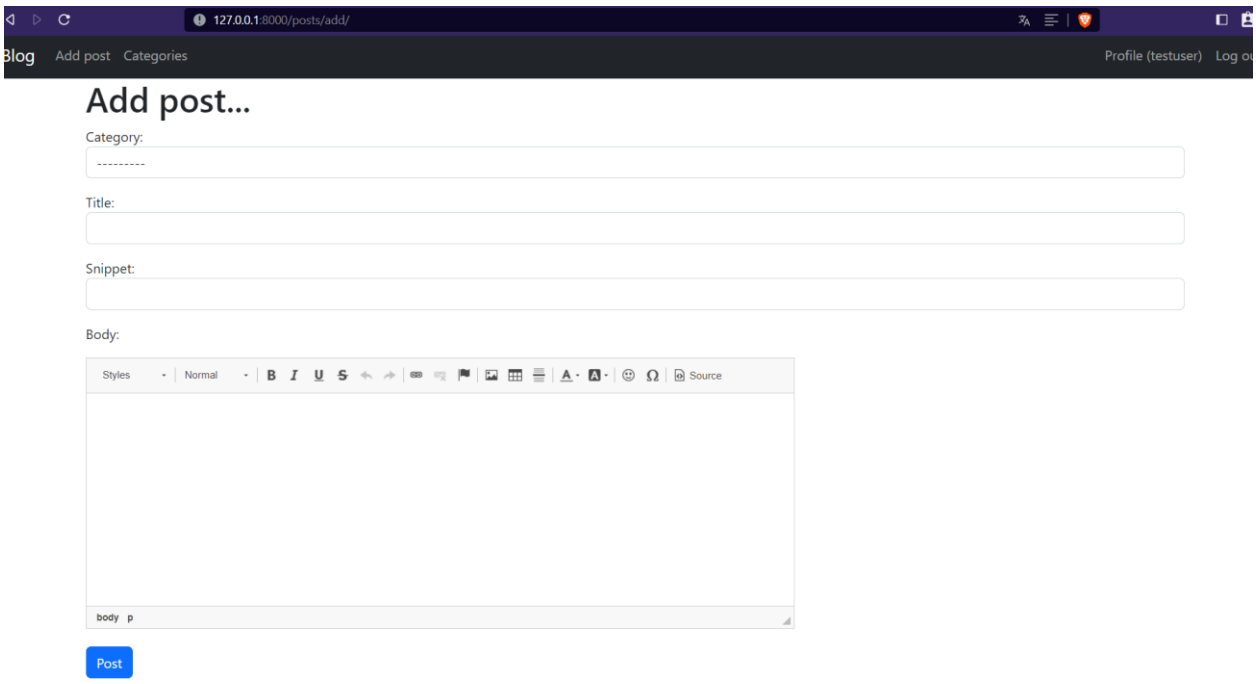

Рисунок 3.3.15 – Сторінка додавання посту

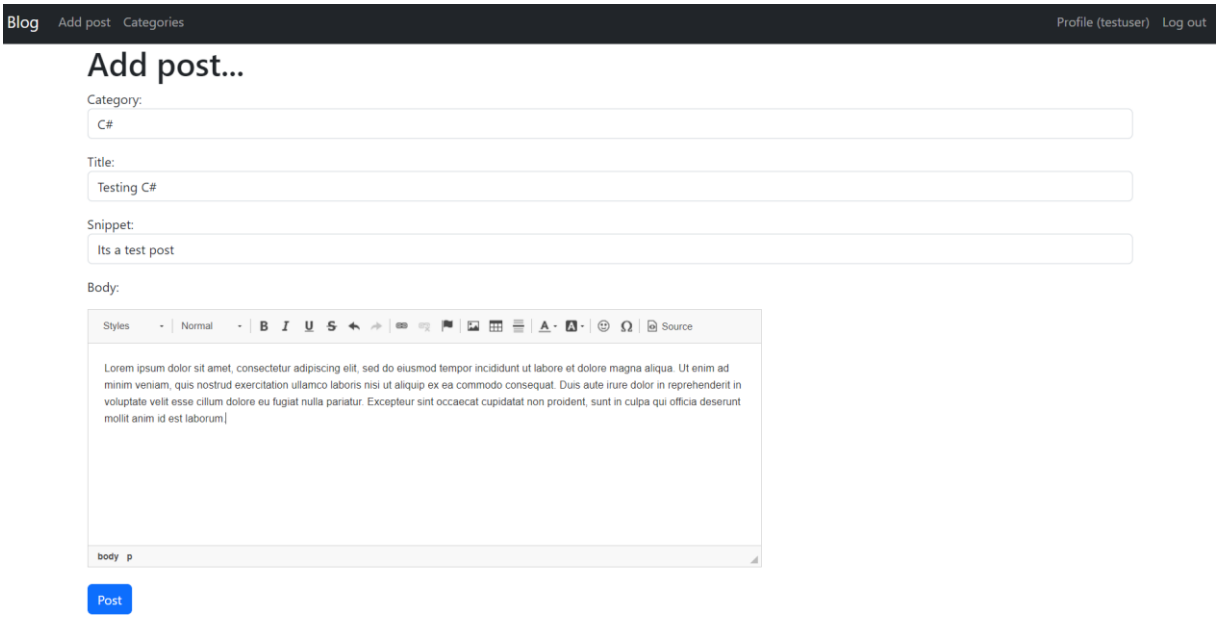

Рисунок 3.3.16 – Заповнення даними

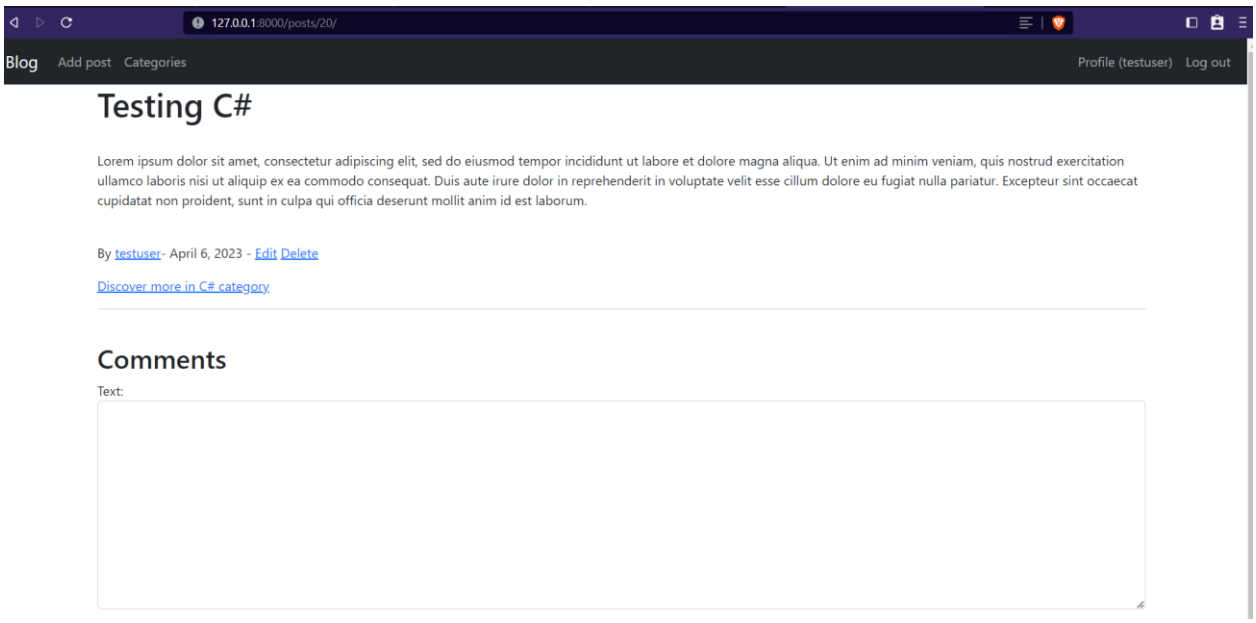

Рисунок 3.3.17 – Відображення на сторінці поста

На пості можна видалити пост чи щось змінити у пості, також даний пост з’являться на головній сторінці.

Натиснувши на Profile(testuser) є можливість переглянути створені пости.

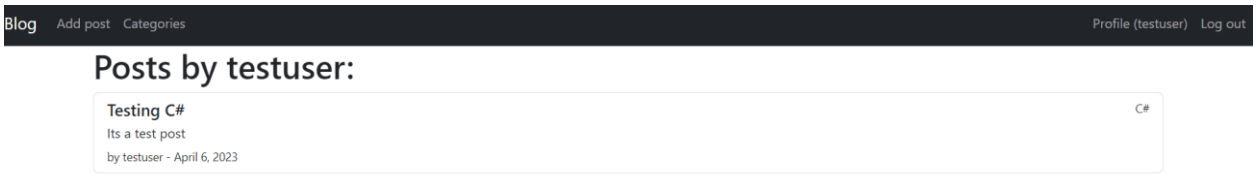

Рисунок 3.3.18 - Відображення у профілі користувача

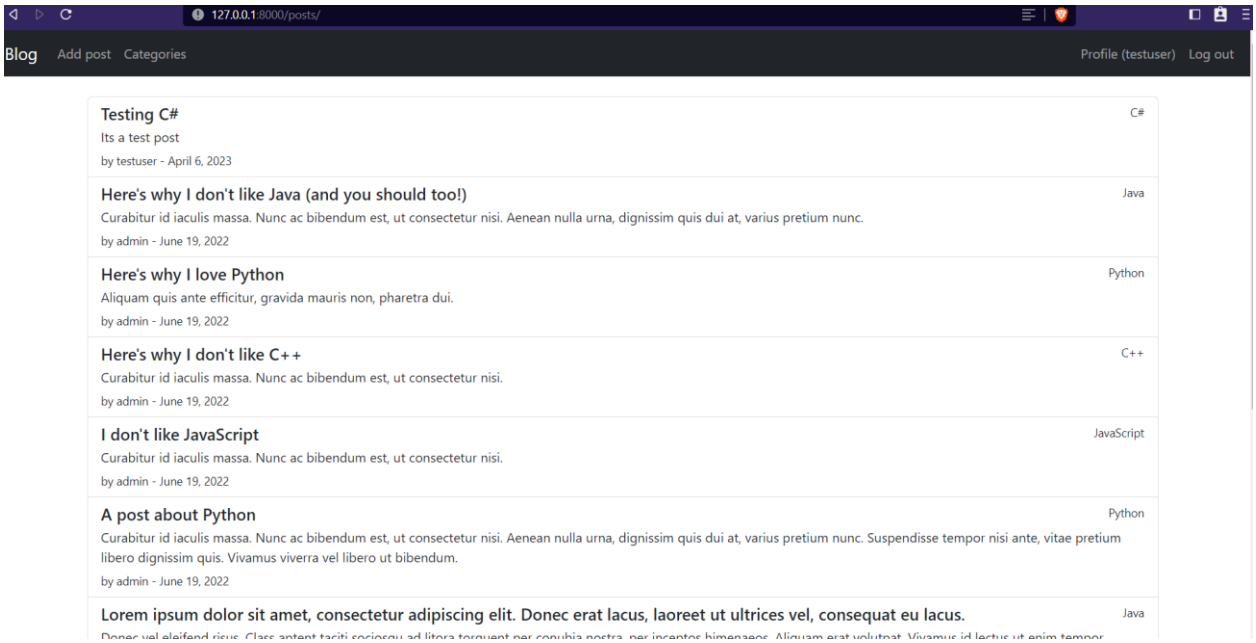

Рисунок 3.3.19 - Відображення на головній сторінці

Усі попередні тестування пройшли успішно, я їх проводив, щоб було розуміння, що новий користувач зможе додавати пости, та робити коментарі.

Під час тестування сайту було виявлено, що він функціонує відповідно до заданих вимог та плану розробки. Всі функціональні можливості, такі як створення, редагування, видалення публікацій, категорій, коментарів, а також механізм контролю доступу та форматування тексту працюють належним чином.

Тестування допомогло виявити декілька помилок та недоліків, які були виправлені під час наступних ітерацій розробки.

Після завершення тестування та виправлення помилок, можна стверджувати, що веб-сайт забезпечує зручний та безпечний обмін даними між студентами. Для реалізації проекту було обрано мову програмування Python та фреймворк Django. Інтеграція з PyCharm дозволила ефективно розробляти проект та проводити тестування.

ВИСНОВКИ

В ході виконання дипломної роботи було розроблено повнофункціональний веб-додаток на базі Django, який дозволяє користувачам публікувати, переглядати, редагувати та видаляти публікації, а також залишати коментарі. Було створено систему реєстрації та авторизації користувачів з механізмами контролю доступу до публікацій.

Під час розробки було використано мову програмування Python та фреймворк Django, що забезпечило швидку та ефективну розробку. Було використано базу даних SQLite для зберігання даних.

В результаті тестування додаток продемонстрував стабільну роботу та відповідність вимогам, що були поставлені в постановці задачі. Користувачі можуть публікувати та переглядати публікації, залишати коментарі та редагувати свої публікації. Механізми контролю доступу до публікацій забезпечують безпеку даних та унеможливлюють несанкціонований доступ.

Отже, можна зробити висновок, що дипломна робота успішно виконана, а розроблений додаток є функціональним та стабільним. Результати роботи можуть бути використані для подальшого розвитку та вдосконалення веб-додатків на базі Django.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Біла, Н. М. Використання Python та Django для розробки веб-додатків / Н. М. Біла // Матеріали VI Всеукраїнської науково-практичної конференції «Сучасні технології в комп'ютерній науці», 2018 р.

2. Котляров, В. Створення веб-додатків на Django: огляд фреймворка та практичні рекомендації / В. Котляров // Технології та інженерія сучасного виробництва: матеріали III Всеукраїнської науково-технічної конференції (м. Херсон, 23-25 листопада 2017 р.). – Херсон: ХНТУ, 2017. – С. 141-143.

3. Базовий курс програмування на мові Python [Електронний ресурс] / IT Education Academy. - Режим доступу: https://itea.ua/ua/courses/python-base-course/

4. Основи програмування на Python [Електронний ресурс] / Python.org. - Режим доступу: https://www.python.org/about/gettingstarted/

5. Віртуальна машина Python [Електронний ресурс] / Python.org. - Режим доступу: https://www.python.org/downloads/

6. Django Documentation [Електронний ресурс] / Django. - Режим доступу: https://docs.djangoproject.com/en/3.2/

7. Офіційний сайт Django [Електронний ресурс] / Django. - Режим доступу: https://www.djangoproject.com/

8. Learning Django Web Development [Електронний ресурс] / Ayman Hourieh. - Режим доступу: https://www.packtpub.com/product/learning-django-web-development/9781783984400

9. Лутц, М. Програмування на Python. Т. 1 / М. Лутц. - М.: Символ, 2016. - 992 с

10. Лутц М. Програмування на Python. Т. 2 / М. Лутц. - М.: Символ, 2016. - 876 с.

11. Форс Д. Django: Розробка веб-додатків на Python / Д. Форс, П. Біссекс, У. Чан. – Символ-плюс, 2010.– 267 с

Додаток А(Створення моделей до БД)

0001_initial.py

```
class Migration(migrations.Migration):
    initial = True

    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.CreateModel(
            name="Category",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("name", models.CharField(max_length=255, unique=True)),
            ],
            options={
                "verbose_name_plural": "Categories",
            },
        ),
        migrations.CreateModel(
            name="Post",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("title", models.CharField(max_length=255)),
                ("body", models.TextField()),
                ("post_date", models.DateField(auto_now_add=True)),
                (
                    "author",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        to=settings.AUTH_USER_MODEL,
                    ),
                ),
                (
                    "category",
                    models.ForeignKey(
                        default=None,
                        on_delete=django.db.models.deletion.CASCADE,
                        to="blog.category",
                    ),
                ),
            ],
```

```
        ),
        migrations.CreateModel(
            name="Comment",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("text", models.TextField(max_length=1000)),
                (
                    "post",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE, to="blog.post"
                    ),
                ),
                (
                    "user",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        to=settings.AUTH_USER_MODEL,
                    ),
                ),
            ],
        ),
    ]
```

0002_alter_post_body.py

```
class Migration(migrations.Migration):
    dependencies = [
        ("blog", "0001_initial"),
    ]

    operations = [
        migrations.AlterField(
            model_name="post",
            name="body",
            field=ckeditor.fields.RichTextField(blank=True, null=True),
        ),
    ]
```

0003_post_snippet.py

```
class Migration(migrations.Migration):
    dependencies = [
        ("blog", "0002_alter_post_body"),
    ]

    operations = [
        migrations.AddField(
            model_name="post",
            name="snippet",
            field=models.CharField(default="hey", max_length=255),
            preserve_default=False,
        ),
    ]
```

0004_comment_date_added_alter_comment_post.py

```
class Migration(migrations.Migration):
    dependencies = [
        ("blog", "0003_post_snippet"),
    ]

    operations = [
        migrations.AddField(
            model_name="comment",
            name="date_added",
            field=models.DateTimeField(
                auto_now_add=True, default=django.utils.timezone.now
            ),
            preserve_default=False,
        ),
        migrations.AlterField(
            model_name="comment",
            name="post",
            field=models.ForeignKey(
                on_delete=django.db.models.deletion.CASCADE,
                related_name="comments",
                to="blog.post",
            ),
        ),
    ]
```

0005_rename_user_comment_author.py

```
class Migration(migrations.Migration):
    dependencies = [
        ("blog", "0004_comment_date_added_alter_comment_post"),
    ]

    operations = [
        migrations.RenameField(
            model_name="comment",
            old_name="user",
            new_name="author",
        ),
    ]
```

Додаток Б (Створення необхідних форм та моделей)

admin.py

```
# імпортуємо необхідні модулі
from django.contrib import admin

from .models import Category, Comment, Post

# реєструємо моделі в адмінці
admin.site.register(Post)
admin.site.register(Category)
admin.site.register(Comment)
```

apps.py

```
# імпортуємо необхідний модуль
from django.apps import AppConfig


# створюємо клас налаштувань додатку
class BlogConfig(AppConfig):
    # встановлюємо тип поля первинного ключа за замовчуванням для моделей у додатку
    default_auto_field = "django.db.models.BigAutoField"
    # вказуємо ім'я додатку
    name = "blog"
```

forms.py

```
# імпортуємо необхідні модулі
from django import forms

from .models import Comment, Post


# створюємо форму для моделі Post
class PostForm(forms.ModelForm):
    class Meta:
        # вказуємо модель, для якої створюється форма
        model = Post
        # вказуємо поля, які будуть включені в форму
        fields = ["category", "title", "snippet", "body"]

        # налаштовуємо віджети для полів форми
        widgets = {
            "category": forms.Select(attrs={"class": "form-control"}),
            "title": forms.TextInput(attrs={"class": "form-control"}),
            "snippet": forms.TextInput(attrs={"class": "form-control"}),
            "body": forms.Textarea(attrs={"class": "form-control"}),
        }


# створюємо форму для моделі Comment
class CommentForm(forms.ModelForm):
    class Meta:
        # вказуємо модель, для якої створюється форма
        model = Comment
        # вказуємо поля, які будуть включені в форму
        fields = ["text"]

        # налаштовуємо віджети для полів форми
        widgets = {"text": forms.Textarea(attrs={"class": "form-control"})}
```

models.py

```
# імпортуємо необхідні модулі
from ckeditor.fields import RichTextField
from django.contrib.auth.models import User
from django.db import models
from django.urls import reverse

# створюємо модель категорії для блогу
class Category(models.Model):
    """
    Model that represents a category of blog posts.

    Attributes:
        name: name of the category
    """

    # встановлюємо атрибут name
    name = models.CharField(max_length=255, unique=True)

    # визначаємо метод __str__ для відображення об'єктів моделі
    def __str__(self):
        return f"{self.name}"

    # встановлюємо додаткові параметри моделі
    class Meta:
        verbose_name_plural = "Categories"  # Встановлюємо назву моделі у множині

# створюємо модель поста для блогу
class Post(models.Model):
    """
    Model that represents a blog post.

    Attributes:
        category: category of the post
        title: title of the post
        snippet: short description of the post
        author: author of the post
        body: main content of the post
        post_date: date the post was created
    """

    # встановлюємо атрибути моделі
    category = models.ForeignKey(Category, on_delete=models.CASCADE, default=None)
    title = models.CharField(max_length=255)
    snippet = models.CharField(max_length=255)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    body = RichTextField(blank=True, null=True)
    post_date = models.DateField(auto_now_add=True)

    # визначаємо метод __str__ для відображення об'єктів моделі
    def __str__(self):
        return f"{self.title} | {self.author}"
    # визначаємо метод get_absolute_url для отримання посилання на деталі поста
    def get_absolute_url(self):
        return reverse("post_details", args=(str(self.id)))

# створюємо модель коментаря для блогу
class Comment(models.Model):
    """
    Model that represents a comment on a blog post.
```

```
    Attributes:
        post: post the comment belongs to
        author: author of the comment
        text: text of the comment
        date_added: date and time the comment was added
    """
    # встановлюємо атрибути моделі
    post = models.ForeignKey(Post, related_name="comments", on_delete=models.CASCADE)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    text = models.TextField(max_length=1000)
    date_added = models.DateTimeField(auto_now_add=True)

    # визначаємо метод __str__ для відображення об'єктів моделі
    def __str__(self):
        return f"{self.post.title} - {self.author}"
```

urls.py

```
# імпортуємо необхідні модулі та функції
from django.urls import path

from .views import (
    CategoriesListView,
    CategoryView,
    CreatePostView,
    DeleteCommentView,
    DeletePostView,
    HomeView,
    PostDetailView,
    UpdatePostView,
    UserProfileView,
)

# визначаємо шаблони URL-адрес
urlpatterns = [
    # домашня сторінка
    path("posts/", HomeView, name="home"),
    # сторінка додавання поста
    path("posts/add/", CreatePostView, name="add_post"),
    # сторінка деталей поста
    path("posts/<int:pk>/", PostDetailView, name="post_details"),
    # сторінка редагування поста
    path("posts/<int:pk>/edit", UpdatePostView, name="edit_post"),
    # сторінка видалення поста
    path("posts/<int:pk>/delete", DeletePostView, name="delete_post"),
    # сторінка профілю користувача
    path("profile/<int:pk>", UserProfileView, name="profile"),
    # сторінка зі списком категорій
    path("posts/categories", CategoriesListView.as_view(), name="categories"),
    # сторінка зі списком постів у категорії
    path("posts/categories/<int:pk>", CategoryView, name="category"),
    # сторінка видалення коментаря
    path("comments/<int:pk>/delete", DeleteCommentView, name="delete_comment"),
]
```

views.py

```
# імпортуємо необхідні модулі та функції
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.core.paginator import EmptyPage, PageNotAnInteger, Paginator
```

```
from django.http import HttpResponseForbidden
from django.shortcuts import get_object_or_404, redirect, render
from django.views.generic import ListView

from .forms import CommentForm, PostForm
from .models import Category, Comment, Post


# визначаємо функцію для відображення домашньої сторінки
def HomeView(request):
    # отримуємо список постів, впорядкованих за ID
    posts = Post.objects.order_by("-id")

    # створюємо об'єкт Paginator для розбиття списку постів на сторінки
    paginator = Paginator(posts, 10)
    # отримуємо номер поточної сторінки з GET-параметрів запиту
    page = request.GET.get("page")

    # отримуємо список постів для поточної сторінки
    try:
        posts = paginator.page(page)
    except PageNotAnInteger:
        # якщо номер сторінки не є цілим числом, показуємо першу сторінку
        posts = paginator.page(1)
    except EmptyPage:
        # якщо номер сторінки більший за кількість сторінок, показуємо останню сторінку
        posts = paginator.page(paginator.num_pages)

    # формуємо контекст для шаблону
    context = {
        "posts": posts,
    }

    # відображаємо шаблон з заданим контекстом
    return render(request, "home.html", context)


# визначаємо функцію для відображення деталей поста
def PostDetailView(request, pk):
    # отримуємо об'єкт Post з бази даних або повертаємо 404-помилку, якщо такого об'єкта немає
    post = get_object_or_404(Post, pk=pk)
    # перевіряємо, чи запит був відправлений методом POST
    if request.method == "POST":
        # створюємо об'єкт форми з даними з POST-запиту
        comment_form = CommentForm(request.POST)
        # перевіряємо, чи користувач автентифікований
        if request.user.is_authenticated:
            # перевіряємо, чи дані форми валідні
            if comment_form.is_valid():
                # зберігаємо коментар у базі даних
                comment = comment_form.save(commit=False)
                comment.author = request.user
                comment.post = post
                comment.save()
                # перенаправляємо користувача на сторінку деталей поста
                return redirect("post_details", pk=pk)
        else:
            # якщо користувач не автентифікований, перенаправляємо його на сторінку входу
            return redirect("login")
    else:
        # якщо запит був відправлений не методом POST, створюємо порожню форму
        comment_form = CommentForm()
```

```
    # отримуємо список коментарів для даного поста, впорядкованих за ID
    comments = Comment.objects.filter(post=post).order_by("-id")

    # створюємо об'єкт Paginator для розбиття списку коментарів на сторінки
    paginator = Paginator(comments, 10)
    # отримуємо номер поточної сторінки з GET-параметрів запиту
    page = request.GET.get("page")

    # отримуємо список коментарів для поточної сторінки
    try:
        comments = paginator.page(page)
    except PageNotAnInteger:
        # якщо номер сторінки не є цілим числом, показуємо першу сторінку
        comments = paginator.page(1)
    except EmptyPage:
        # якщо номер сторінки більший за кількість сторінок, показуємо останню сторінку
        comments = paginator.page(paginator.num_pages)

    # формуємо контекст для шаблону
    context = {"post": post, "comments": comments, "comment_form": comment_form}

    # відображаємо шаблон з заданим контекстом
    return render(request, "blog/post_details.html", context)


# декоратор login_required перевіряє, чи користувач автентифікований
@login_required
# визначаємо функцію для створення нового поста
def CreatePostView(request):
    # перевіряємо, чи запит був відправлений методом POST
    if request.method == "POST":
        # створюємо об'єкт форми з даними з POST-запиту
        form = PostForm(request.POST)
        # перевіряємо, чи дані форми валідні
        if form.is_valid():
            # зберігаємо пост у базі даних
            post = form.save(commit=False)
            post.author = request.user
            post.save()
            # перенаправляємо користувача на сторінку деталей поста
            return redirect("post_details", pk=post.pk)
    else:
        # якщо запит був відправлений не методом POST, створюємо порожню форму
        form = PostForm()

    # відображаємо шаблон з заданим контекстом
    return render(request, "blog/add_post.html", context={"form": form})


# декоратор login_required перевіряє, чи користувач автентифікований
@login_required
# визначаємо функцію для редагування поста
def UpdatePostView(request, pk):
    # отримуємо об'єкт Post з бази даних або повертаємо 404-помилку, якщо такого об'єкта немає
    post = get_object_or_404(Post, id=pk)

    # перевіряємо, чи користувач є автором поста
    if request.user == post.author:
        # перевіряємо, чи запит був відправлений методом POST
        if request.method == "POST":
            # створюємо об'єкт форми з даними з POST-запиту
            post_form = PostForm(request.POST)
            # перевіряємо, чи дані форми валідні
```

```
            if post_form.is_valid():
                # оновлюємо дані поста у базі даних
                post.title = post_form.cleaned_data["title"]
                post.body = post_form.cleaned_data["body"]
                post.snippet = post_form.cleaned_data["snippet"]
                post.save()

                # перенаправляємо користувача на сторінку деталей поста
                return redirect("post_details", pk=pk)
            else:
                # якщо дані форми не валідні, відображаємо шаблон з заданим контекстом
                return render(request, "blog/edit_post.html", context={"post": post})
        else:
            # якщо запит був відправлений не методом POST, створюємо форму з початковими даними
            post_form = PostForm(
                initial={
                    "title": post.title,
                    "body": post.body,
                    "category": post.category,
                    "snippet": post.snippet,
                }
            )

            # формуємо контекст для шаблону
        context = {"post": post, "form": post_form}
        # відображаємо шаблон з заданим контекстом
        return render(request, "blog/edit_post.html", context)
    else:
        # якщо користувач не є автором поста, повертаємо 403-помилку
        return HttpResponseForbidden


# декоратор login_required перевіряє, чи користувач автентифікований
@login_required
# визначаємо функцію для видалення поста
def DeletePostView(request, pk):
    # отримуємо об'єкт Post з бази даних або повертаємо 404-помилку, якщо такого об'єкта немає
    post = get_object_or_404(Post, id=pk)
    # перевіряємо, чи користувач є автором поста
    if request.user == post.author:
        # видаляємо пост з бази даних
        post.delete()

    # перенаправляємо користувача на головну сторінку
    return redirect("home")


# визначаємо клас для відображення списку категорій
class CategoriesListView(ListView):
    # вказуємо модель, з якої будуть отримуватися дані
    model = Category
    # вказуємо шаблон, який буде використовуватися для відображення списку
    template_name = "blog/categories_list.html"


# визначаємо клас для відображення деталей категорії
def CategoryView(request, pk):
    # Отримуємо категорію за pk або повертаємо 404 помилку
    category = get_object_or_404(Category, pk=pk)
    # Отримуємо всі пости з цієї категорії та сортуємо їх за id
    posts = Post.objects.filter(category=category).order_by("-id")

    # Створюємо пагінацію для постів
```

```
    paginator = Paginator(posts, 10)
    page = request.GET.get("page")

    try:
        posts = paginator.page(page)
    except PageNotAnInteger:
        # Якщо сторінка не є цілим числом, повертаємо першу сторінку
        posts = paginator.page(1)
    except EmptyPage:
        # Якщо сторінка поза діапазоном, повертаємо останню сторінку
        posts = paginator.page(paginator.num_pages)

    # Створюємо контекст для шаблону
    context = {
        "category": category,
        "posts": posts,
    }
    # Рендеримо шаблон з контекстом
    return render(request, "blog/category.html", context)


def UserProfileView(request, pk):
    # Отримуємо користувача за id або повертаємо 404 помилку
    profile_owner = get_object_or_404(User, id=pk)
    # Отримуємо всі пости цього користувача та сортуємо їх за id
    posts = Post.objects.filter(author=profile_owner).order_by("-id")

    # Створюємо пагінацію для постів
    paginator = Paginator(posts, 10)
    page = request.GET.get("page")

    try:
        posts = paginator.page(page)
    except PageNotAnInteger:
        # Якщо сторінка не є цілим числом, повертаємо першу сторінку
        posts = paginator.page(1)
    except EmptyPage:
        # Якщо сторінка поза діапазоном, повертаємо останню сторінку
        posts = paginator.page(paginator.num_pages)

    # Створюємо контекст для шаблону
    context = {"profile_owner": profile_owner, "posts": posts}

    # Рендеримо шаблон з контекстом
    return render(request, "blog/profile.html", context)

@login_required
def DeleteCommentView(request, pk):
    # Отримуємо коментар за id або повертаємо 404 помилку
    comment = get_object_or_404(Comment, id=pk)
    # Якщо користувач є автором коментаря, видаляємо його
    if request.user == comment.author:
        comment.delete()

    # Повертаємося на попередню сторінку
    return redirect(request.META.get("HTTP_REFERER"))
```