МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Інститут інженерії та інформаційних технологій

Кафедра комп’ютерної інженерії та електромеханіки

**КВАЛІФІКАЦІЙНА РОБОТА**

на тему:

**\_\_\_Імплементація MVC патерну при розробці онлайн форуму на \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Python з використанням Django\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Рівень вищої освіти другий (магістерський)

Спеціальності 123 «Комп’ютерна інженерія»

Освітня програма «Комп’ютерна інженерія»

Виконав: студент групи МгКІ-23.

Гуденко Є.В. .

(прізвище та ініціали)

Керівник: к.т.н., доц. Стаценко Д.В.

Рецензент.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(науковий ступінь, вчене звання, прізвище та ініціали)

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Інститут інженерії та інформаційних технологій.

Кафедра комп’ютерної інженерії та електромеханіки

Спеціальність 123 «Комп’ютерна Інженерія».

Освітня програма «Комп’ютерна Інженерія»

**ЗАТВЕРДЖУЮ**

Завідувач кафедри КІЕМ

\_\_\_\_\_\_\_\_\_\_\_\_ Дмитро СТАЦЕНКО

“\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 року

**ЗАВДАННЯ**

**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

Гуденко Євгеній Валерійович .

(прізвище, ім’я, по батькові)

1. Тема кваліфікаційної роботи Імплементація MVC патерну при розробці онлайн форуму на Python з використанням Django

.

Науковий керівник роботи \_Стаценко Дмитро Володимирович, к.т.н., доцент . (прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

.

затверджені наказом вищого навчального закладу від 03.09.2024 № 188-уч.

2. Вихідні дані до кваліфікаційної роботи: *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

3. Зміст кваліфікаційної роботи (перелік питань, які потрібно опрацювати): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Дата видачі завдання \_\_\_\_\_\_\_\_\_\_\_\_2024

. **КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів кваліфікаційної роботи | Терміни виконання етапів | Примітка про виконання |
| 1 | Вступ | 09.09.2024 |  |
| 2 | РОЗДІЛ 1. ОНЛАЙН КОМУНІКАЦІЯ | 20.09.2024 |  |
| 3 | РОЗДІЛ2. ТЕОРЕТИЧНА ПІДГОТОВКА ДО РЕАЛІЗАЦІЇ ПРОЄКТУ | 05.10. 2024 |  |
| 4 | РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЄКТУ | 25.10.2024 |  |
| 5 | Висновки | 28.10.2024 |  |
| 6 | Оформлення (чистовий варіант) | 31.10.2024 |  |
| 7 | Подача кваліфікаційної роботи науковому керівнику для відгуку | 01.11.2024 |  |
| 8 | Подача кваліфікаційної роботи для рецензування (за 14 днів до захисту) | 09.11.2024 |  |
| 9 | Перевірка кваліфікаційної роботи на наявність ознак плагіату та текстових співпадінь (за 10 днів до захисту) | 11.11.2024 |  |
| 10 | Подання кваліфікаційної роботи на завідувачу кафедри (за 7днів до захисту) | 18.11.2024 |  |

З завданням ознайомлений:

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_**Євгеній ГУДЕНКО .

( підпис ) (Власне ім’я та ПРІЗВИЩЕ)

Науковий керівник \_\_\_\_\_\_\_\_\_\_\_\_ \_\_Дмитро СТАЦЕНКО.

( підпис ) (Власне ім’я та ПРІЗВИЩЕ)

**АНОТАЦІЯ**

**Гуденко Є. В. Імплементація MVC патерну при розробці онлайн форуму на Python з використанням Django. – Рукопис.**

Кваліфікаційна робота за спеціальністю 123 «Комп'ютерна інженерія», освітньою програмою «комп’ютерна інженерія». – Київський національний університет технологій та дизайну, Київ, 2024 рік.

Кваліфікаційну роботу присвячено створенню онлайн-форуму для студентів із використанням сучасних веб-технологій. У роботі досліджено основи веб-розробки, включаючи підходи до створення клієнтської та серверної складових, архітектурні патерни MVC та їх застосування у фреймворку Django. Проєкт реалізовано із застосуванням мов HTML, CSS і JavaScript для фронтенду, а також Python для бекенду.

Проведено локальне тестування додатку та розгортання на сервері PythonAnywhere. Проєкт підтримує створення тематичних дискусій, коментування, систему оцінювання постів, забезпечує автентифікацію користувачів і має гнучкий механізм управління контентом.

Результати роботи можуть бути використані для розробки подібних веб-додатків, орієнтованих на освітнє середовище та поліпшення комунікації між студентами.

*Ключові слова: форум, веб-розробка, Django, MVC, Python, HTML, CSS, JavaScript.*

**ABSTRACT**

**Hudenko Y. V. Implementation of the MVC Pattern in the Development of an Online Forum in Python Using Django – Manuscript.**

Qualification work in specialty 123 «Computer engineering», educational program «Computer engineering». – Kyiv national university of technology and design, Kyiv, 2020.

The qualification work is dedicated to the development of an online forum for students using modern web technologies. The work explores the fundamentals of web development, including approaches to creating client-side and server-side components, the MVC architectural pattern, and its implementation in the Django framework. The project was developed using HTML, CSS, and JavaScript for the frontend, as well as Python for the backend.

Local testing of the application was carried out, followed by deployment on the PythonAnywhere server. The project supports the creation of topic-based discussions, commenting, a post rating system, user authentication, and flexible content management.

The results of this work can be used for the development of similar web applications aimed at enhancing communication within educational environments.

*Keywords: forum, web development, Django, MVC, Python, HTML, CSS, JavaScript.*

**ЗМІСТ**

[ВСТУП 6](#_Toc182648696)

[РОЗДІЛ 1. ОНЛАЙН КОМУНІКАЦІЯ. 8](#_Toc182648697)

[1.1. Історія інтернет-спілкування. 8](#_Toc182648698)

[1.2. Важливість комунікації в навчальному процесі. 9](#_Toc182648699)

[1.3. Існуючі платформи зі схожими рішеннями. 10](#_Toc182648700)

[1.4. Вимоги до проєкту. 15](#_Toc182648701)

[ВИСНОВКИ ДО РОЗДІЛУ 1 16](#_Toc182648702)

[РОЗДІЛ 2. ТЕОРЕТИЧНА ПІДГОТОВКА ДО РЕАЛІЗАЦІЇ ПРОЄКТУ 17](#_Toc182648703)

[2.1. Підходи до розробки веб-додатків 17](#_Toc182648704)

[2.2. Дослідження MVC-патерну 18](#_Toc182648705)

[2.3. Вибір технологій розробки серверної складової проєкту 20](#_Toc182648706)

[2.4 Технології розробки клієнтської складової проєкту 22](#_Toc182648707)

[ВИСНОВКИ ДО РОЗДІЛУ 2 24](#_Toc182648708)

[РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЄКТУ 25](#_Toc182648709)

[3.1. Ініціалізація проєкту та опис його архітектури 25](#_Toc182648710)

[3.2. Реалізація серверної складової (Back end). 28](#_Toc182648711)

[3.3. Реалізація клієнтської складової (Front end). 39](#_Toc182648712)

[3.4. Локальне розгортання проєкту та тестування 48](#_Toc182648713)

[3.5. Розгортання проєкту на сервері 54](#_Toc182648714)

[ВИСНОВКИ ДО РОЗДІЛУ 3 57](#_Toc182648715)

[ВИСНОВКИ 58](#_Toc182648716)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59](#_Toc182648717)

[ДОДАТКИ 61](#_Toc182648718)

[ДОДАТОК А 62](#_Toc182648719)

[ДОДАТОК Б 63](#_Toc182648720)

[ДОДАТОК В 64](#_Toc182648721)

# ВСТУП

У сучасному світі, де швидкість передачі інформації та доступ до знань мають вирішальне значення, онлайн-комунікація стала невід’ємною частиною повсякденного життя. Особливо важливу роль вона відіграє у сфері освіти, де інтерактивні платформи забезпечують зручний доступ до навчальних матеріалів, обмін ідеями та обговорення складних тем. У зв’язку з цим, створення спеціалізованих онлайн-платформ для спілкування та співпраці студентів є актуальним завданням.

**Актуальність роботи.** Актуальність розробки студентського форуму зумовлена потребою в універсальному інструменті для взаємодії учасників навчального процесу. Сучасні студенти часто стикаються з викликами, що пов’язані з відсутністю доступних платформ для обговорення навчальних питань, пошуку колективних рішень і обміну досвідом. Більшість існуючих форумів не відповідають специфічним вимогам навчальних закладів, тому створення нового, орієнтованого саме на студентське середовище продукту, дозволяє забезпечити ефективну взаємодію в межах університетських груп і поза ними.

**Метою роботи** є розробка та впровадження студентського форуму з використанням сучасних веб-технологій, що забезпечить можливості для обміну знаннями, взаємодії між студентами, а також підтримки освітнього процесу в інтерактивному форматі.

**Об’єктом дослідження** є процеси онлайн-комунікації та їх застосування в освітньому середовищі..

**Предметом дослідження** є методи та технології створення інтерактивних веб-додатків, зокрема форумів для студентської аудиторії.

**Методи досліджень**. У роботі використано методи аналізу та синтезу інформації, методи моделювання програмного забезпечення, а також емпіричні методи, що включають тестування розробленого продукту.

**Наукова новизна** роботи полягає у створенні інноваційного підходу до інтеграції освітніх онлайн-комунікаційних платформ із використанням сучасного MVC-патерну та динамічного завантаження даних, що забезпечує оптимізацію процесу обміну інформацією.

**Практичне значення** роботи полягає у можливості використання створеного форуму в навчальному процесі як ефективного інструменту для обговорення навчальних матеріалів, взаємодії між студентами та викладачами, а також для підвищення якості освітнього середовища. Розроблений продукт може бути впроваджений у навчальних закладах різних рівнів, сприяючи покращенню комунікації та колективного навчання.

**Структура та обсяг роботи.** Кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаних джерел та додатків. Основний текст роботи викладений на 50 сторінках, містить 36 рисунків, список джерел з 18 найменувань. Загальний обсяг роботи, враховуючи 3 додатки, складає 90 аркушів.

# РОЗДІЛ 1. ОНЛАЙН КОМУНІКАЦІЯ.

1.1. Історія інтернет-спілкування.

Інтернет-спілкування виникло практично одночасно із появою глобальної мережі. На початкових етапах розвитку інтернету основною метою його використання була передача інформації між науковцями, дослідниками та військовими. Але з часом, коли доступ до мережі отримало ширше коло користувачів, інтернет почав активно використовуватись для особистого та групового спілкування.

Першими формами інтернет-спілкування стали електронна пошта (E-mail) і групи новин (Usenet). Вони дозволяли обмінюватися текстовими повідомленнями, створювати дискусійні теми та брати участь у групових обговореннях [1].

Згодом, у 1980-х роках, з’явилися чат-сервіси та форуми, які стали більш інтерактивним способом спілкування. Наприклад, такі системи, як IRC (Internet Relay Chat), забезпечували можливість миттєвого обміну повідомленнями між користувачами, що заклало основи сучасних месенджерів.

Форуми як окремий тип інтернет-спілкування набули популярності в 1990-х роках. Вони були організовані у вигляді тематичних розділів, де користувачі могли створювати теми (треди) та залишати коментарі до них. Одними з перших платформ такого типу були phpBB, vBulletin і Invision Power Board.

Форуми стали важливим інструментом для обміну досвідом, пошуку інформації та створення спільнот за інтересами. Вони дозволяли користувачам долучатися до дискусій, ділитися знаннями, вирішувати проблеми та спілкуватися з однодумцями.

З розвитком технологій виникли нові платформи для спілкування, такі як соціальні мережі (Facebook, Twitter), месенджери (Telegram, WhatsApp) та інші інтерактивні сервіси (Reddit, Discord). Проте форуми продовжують займати важливе місце серед інструментів комунікації, особливо для обговорення вузькоспеціалізованих тем.

Форуми стали основою для таких сучасних рішень, як Reddit, де збереглася структура тематичних розділів та функціонал голосування за контент. Вони залишаються популярними завдяки своїй простоті, структурованості та можливості вести архів дискусій.

1.2. Важливість комунікації в навчальному процесі.

Комунікація є одним із ключових елементів ефективного навчального процесу. У сучасному світі, де інформаційні технології стали невід’ємною частиною життя, студентам і викладачам важливо мати доступ до засобів обміну знаннями та співпраці. Форумні платформи, як частина таких засобів, є надзвичайно корисними в цьому контексті.

Взаємодія між студентами та викладачами, а також між самими студентами, сприяє кращому розумінню навчального матеріалу. Активне обговорення тем, обмін ідеями та вирішення проблем разом дозволяють більш глибоко засвоїти знання. Форумні платформи забезпечують такі можливості навіть у дистанційному форматі навчання.

Крім того, форуми створюють можливість для позакласного спілкування, де студенти можуть ставити запитання, які залишилися незрозумілими під час лекцій, отримувати консультації та ділитися власним досвідом. Це особливо важливо для дисциплін, які потребують глибокого обговорення, таких як математика, програмування чи інженерні науки.

У групових проектах важливою є можливість обговорення та координації дій між учасниками. Форумні платформи забезпечують зручний спосіб для створення окремих гілок обговорень, де можна ділитися прогресом, ставити питання, коментувати ідеї та планувати подальші дії.

У період пандемії COVID-19 використання онлайн-платформ для навчання стало особливо актуальним. Форумні платформи відіграли ключову роль у підтримці навчального процесу, забезпечуючи можливість комунікації та співпраці між студентами та викладачами, навіть у віддаленому режимі. Завдяки цьому навчання не лише зберегло свою ефективність, але й стало більш гнучким і адаптованим до сучасних умов.

1.3. Існуючі платформи зі схожими рішеннями.

На ринку існує багато популярних платформ, які забезпечують функціонал для інтернет-спілкування. Серед них можна виділити декілька, на які я спирався,

Reddit — це одна з найбільш популярних платформ для обміну інформацією та створення онлайн-спільнот. Вона поєднує функції форуму, новинного порталу та соціальної мережі, дозволяючи користувачам обговорювати різноманітні теми, ділитися контентом і голосувати за пости та коментарі.



Рис. 2.1 Логотип Reddit

Reddit був заснований у червні 2005 року двома студентами Університету Вірджинії — Стівом Хаффманом та Алексісом Оганяном. Первісна ідея полягала у створенні "головної сторінки Інтернету", де користувачі могли б отримувати новини та цікавий контент, організований за темами.

Історія Reddit розпочалася з невеликого стартапу, але вже через рік після запуску, у жовтні 2006 року, проект був придбаний видавничою компанією Condé Nast. Це стало переломним моментом для платформи, оскільки вона отримала фінансову підтримку та змогла розширити свою аудиторію[2].

Популярність Reddit почала стрімко зростати у 2010-х роках. Основними факторами цього зростання стали декілька факторів: Reddit організований у вигляді сабреддитів (підфорумів), які охоплюють усі можливі теми — від технологій і політики до мемів та хобі. Кожен користувач може знайти щось цікаве саме для себе або створити власний сабреддит; Також платформа використовує систему голосування, яка дозволяє користувачам впливати на те, який контент буде в топі. Цей механізм робить платформу динамічною та інтерактивною; Reddit надає користувачам можливість залишатися анонімними, що стимулює вільне висловлювання думок; Сабреддити стали важливими платформами для обговорення культурних, політичних і соціальних питань. Наприклад, сабреддити на кшталт r/politics і r/worldnews залучають користувачів до дискусій про актуальні події.

Основні віхи розвитку:

* 2008 рік: Відкрита можливість створення сабреддитів, що значно розширило функціонал та індивідуалізувало користувацький досвід.
* 2012 рік: Щоденна аудиторія Reddit перевищила 1 мільйон користувачів, а сайт став одним із найпопулярніших у США.
* 2015 рік: Вперше проведено AMA (Ask Me Anything) з Бараком Обамою, що стало важливою подією для платформи та показало її значення у політичних дискусіях.
* 2020 рік: Пандемія COVID-19 сприяла ще більшому зростанню популярності платформи. Люди з усього світу почали активно використовувати Reddit для обміну інформацією, порад і підтримки.

Серед проблем, які часто приписують платформі можна виділити проблеми з модерацією та поширенням дезінформації.

Discourse — інша сучасна платформа для створення форумів. Вона вирізняється зручним інтерфейсом, інтеграцією з іншими сервісами та багатим функціоналом.



Рис. 2.2. Логотип Discourse

Discourse був запущений у 2013 році Джеффом Аткінсом, одним із творців Stack Overflow, разом із співзасновниками Семом Саффроном і Робіном Ворду. Їхня місія полягала у створенні форуму нового покоління, який би вирішував багато проблем старих платформ, таких як складний інтерфейс, низька залученість користувачів та відсутність інновацій у модерації [3].

Важливою особливістю Discourse є його відкритий вихідний код, що дозволяє будь-якій організації чи спільноті завантажити, налаштувати та розгорнути власний форум. Це зробило платформу популярною серед різних організацій, як комерційних, так і некомерційних.

Серед особливостей платформи можна відмітити систему довіри. Discourse має багаторівневу систему довіри, яка мотивує користувачів до позитивної взаємодії. Нові користувачі мають обмежені права, а активні учасники спільноти з часом отримують більше можливостей, наприклад, редагування чужих повідомлень або модерацію.

Також Discourse оснащений розумними інструментами для боротьби зі спамом і некоректною поведінкою. Система автоматично приховує повідомлення, які отримали багато скарг, і сповіщає модераторів про потенційні проблеми.

Discourse легко інтегрується з іншими інструментами, такими як Slack, GitHub, та WordPress. Це дозволяє користувачам ефективніше працювати та об'єднувати різні платформи для спільної роботи.

Для залучення користувачів Discourse використовує бейджі, які видаються за певні досягнення, наприклад, за перший пост, відповідь чи активну участь у дискусіях.

Discourse став популярним завдяки кільком ключовим перевагам:

* Простота у використанні та налаштуванні. Власники форумів можуть легко змінювати вигляд і функціональність платформи відповідно до потреб спільноти.
* Мобільна підтримка. Інтерфейс Discourse повністю адаптивний, що робить його зручним для користувачів смартфонів і планшетів.
* Постійні оновлення. Розробники Discourse регулярно випускають оновлення, покращуючи безпеку та додаючи нові функції.
* Фокус на якісному спілкуванні. Платформа створена для тривалих і структурованих дискусій, на відміну від коротких і хаотичних повідомлень у соціальних мережах.

З моменту запуску Discourse став основою для тисяч форумів різних організацій. Серед них є технічні спільноти, освітні платформи та комерційні компанії.

Серед недоліків можна виділити: складність інтерфейсу для нових користувачів, велике використання серверних ресурсів та потреба технічних знань для розгортання та підтримки власного форуму.

Stack Overflow — спеціалізована платформа для програмістів, яка функціонує як форум питань і відповідей. На відміну від вищезгаданих платформ, Stack Overflow можна вважати вузьконаправленою. Через це платформі вдалося закріпитися у своїй ніші та розвити багату базу користувачів та репутацію сайту, на якому можна знайти інформацію з будь-якого питання, пов’язаного з індустрією програмування.



Рис. 2.3. Логотип Stack Overflow

Stack Overflow був заснований у 2008 році Джеффом Аткінсом і Джоелом Спольскі. Їхня мета полягала в тому, щоб створити відкриту, безкоштовну платформу, де розробники могли б обмінюватися досвідом, ставити запитання та знаходити відповіді на свої проблеми. До створення Stack Overflow більшість розробників покладалися на застарілі форуми, блоги або платні ресурси, які часто мали обмежену функціональність або були неструктурованими [4].

Stack Overflow став невіддільною частиною роботи програмістів, дозволяючи розробникам економити час і отримувати допомогу від спільноти.

Платформа побудований навколо чіткої структури запитань і відповідей. Користувачі можуть ставити конкретні питання, пов'язані з програмуванням, і отримувати відповіді від інших учасників. Учасники можуть голосувати за питання та відповіді. Це дозволяє найкориснішим матеріалам підніматися вище у списках, а менш корисним — опускатися. Користувачі отримують очки репутації за якісні відповіді або вдалі питання. Чим вища репутація, тим більше прав має користувач, наприклад, можливість редагувати чужі питання. Кожне питання може мати до п'яти тегів, що дозволяє розділяти контент за темами. Це спрощує пошук відповідей у конкретній галузі.

Stack Overflow має величезний вплив на розробку програмного забезпечення. Платформа стала головним ресурсом для навчання новачків та вирішення проблем досвідченими програмістами. Сайт:

phpBB — одна з найстаріших платформ для створення форумів. Вона є безкоштовною і пропонує базовий функціонал для обговорень:

* Створення тем і підтем.
* Система прав доступу для користувачів.
* Простий і зрозумілий інтерфейс.



Рис. 2.4. Логотип phpBB

## 1.4. Вимоги до проєкту.

Проаналізувавши існуючі платформи, було виведено наступні вимоги для форуму, що буде створено в процесі виконання проєкту:

1. Зручна і швидка система реєстрації та авторизації користувачів.
2. Система оцінювання постів для заохочування високоякісного користувацького контенту та відсіювання недоброякісних публікацій.
3. Система модерації контенту для видалення публікацій, що порушують правила платформи, переносу постів, опублікованих не на ту тематичну дошку, блокування та видалення користувачів тощо.
4. Система пошуку публікацій всередині сайту.
5. Направленість платформи на цільову аудиторію студентів.
6. Розбиття контенту на тематичні розділи.
7. Система коментарів до публікацій.
8. Зручний інтерфейс.

Платформа, створена в рамках цього проекту, матиме багато спільного з вищезгаданими рішеннями. Вона використовуватиме сучасні підходи до розробки, інтегрує можливість створення тем, коментування, голосування та організації обговорень за категоріями. Аналіз вищезгаданих платформ допоміг обрати напрямок для розвитку власної.

ВИСНОВКИ ДО РОЗДІЛУ 1

Розділ 1 розглядає еволюцію онлайн-комунікації, її роль у навчальному процесі та аналізує існуючі платформи, які пропонують схожі функціональні рішення. Проведене дослідження дозволяє зробити кілька важливих висновків.

По-перше, історія інтернет-спілкування показує, що від часу появи перших електронних листів та форумів до сучасних соціальних платформ, засоби онлайн-комунікації пройшли значний шлях. Це демонструє постійну потребу людей у швидкому та зручному обміні інформацією, що сприяло створенню все більш функціональних платформ.

По-друге, комунікація є ключовим елементом у навчальному процесі. Вона дозволяє студентам та викладачам обмінюватися знаннями, досвідом і матеріалами, що робить навчання інтерактивнішим і доступнішим. Особливо це стало важливим у період зростання популярності дистанційної освіти, коли онлайн-платформи забезпечили безперервність навчання.

По-третє, аналіз платформ на кшталт Reddit, Discourse та Stack Overflow показав, що, незважаючи на різні підходи до організації контенту та взаємодії користувачів, їхня спільна мета — ефективна комунікація та обмін знаннями. Ці платформи задають високий стандарт якості та зручності для нових проектів.

Останнім важливим аспектом є визначення вимог до створюваного проекту. Успішна реалізація залежить від точного розуміння потреб користувачів, функціональних і технічних вимог. Ці вимоги включають забезпечення інтуїтивно зрозумілого інтерфейсу, наявність базових функцій для обговорень, організацію контенту за темами, а також інтеграцію системи рейтингу.

Таким чином, проведений аналіз створив міцну теоретичну основу для подальшої розробки проєкту, дозволивши врахувати найкращі практики існуючих платформ та специфічні вимоги користувачів.

# РОЗДІЛ 2. ТЕОРЕТИЧНА ПІДГОТОВКА ДО РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1. Підходи до розробки веб-додатків

Розробка веб-додатків є комплексним процесом, що включає вибір архітектурного підходу, інструментів і технологій залежно від цілей проєкту, обсягу роботи та вимог до функціональності.

Загалом, розробку веб-сайтів поділяють на дві складові – користувацьку та серверну (front end та back end відповідно). Взаємодія цих двох складових залежить від підходу до розробки [8].

Традиційний серверно-клієнтський підхід (Server-Side Rendering, SSR) передбачає, що більшість логіки обробляється на сервері. Клієнт (браузер) запитує сторінку, і сервер формує її цілком, після чого відправляє клієнту готовий HTML. Сервер відповідає за всю обробку даних та рендеринг сторінок. Цей підхід використовується для додатків, що не потребують високої інтерактивності та популярний у проектах, де важлива SEO-оптимізація [5].

При роботі з клієнтсько-серверною архітектурою з використанням SPA (Single Page Application) основне завантаження та рендеринг відбуваються на клієнтській стороні. Після початкового завантаження додаток отримує дані з сервера через API-запити без перезавантаження сторінок. Такий підхід має високу інтерактивність і швидкість роботи та складнішу структуру коду та розподілення логіки між клієнтом і сервером [6].

Також популярною є архітектура на основі мікросервісів. Мікросервіси передбачають розбиття додатка на незалежні сервіси, кожен з яких відповідає за конкретну функціональність. Цей підхід добре підходить для великих, масштабованих систем, оскільки кожен сервіс можна розробляти, тестувати та розгортати окремо, і дозволяється використання API для взаємодії між сервісами. Але такий підхід ускладнює загальну інфраструктуру системи [7].

Архітектурний підхід, який був обраний для реалізації даного проекту – це патерн MVC (Model-View-Controller).

2.2. Дослідження MVC-патерну

Одним із найбільш поширених і перевірених патернів проектування є MVC (Model-View-Controller). Даний патерн вперше був запропонований у 1970-х роках Трюгве Реенскоугом, але своєї популярності він набув у 2000-х, коли активно почав використовуватися у веб-розробці.[9]

MVC-патерн базується на поділі додатку на три окремі компоненти: Model (Модель), View (Подання) та Controller (Контролер). Кожен з цих компонентів має свою зону відповідальності, що сприяє модульності, полегшує тестування і спрощує подальше обслуговування системи.

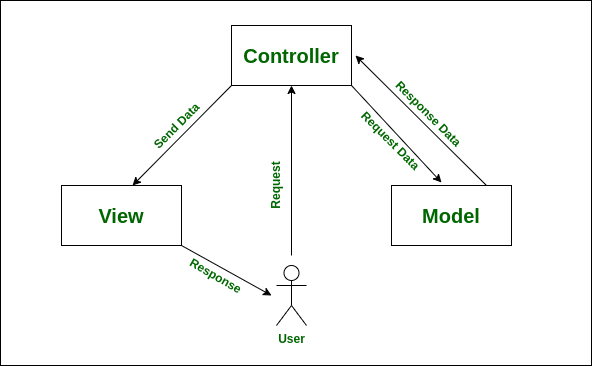


Рис.2.1. Візуалізація MVC-патерну [10]

Модель відповідає за управління даними та бізнес-логікою додатку. Вона безпосередньо взаємодіє з базою даних, забезпечуючи збереження, обробку і отримання даних. У контексті Django модель реалізується через класи, які визначають структуру таблиць у базі даних. Важливою перевагою Django є ORM (Object-Relational Mapping), яка дозволяє працювати з базою даних за допомогою Python-коду, що підвищує ефективність розробки.

Подання відповідає за відображення даних користувачеві. Воно формує HTML-сторінки або API-відповіді на основі даних, отриманих із моделі. Подання не містить бізнес-логіки, а лише реалізує її відображення, забезпечуючи розділення логіки та інтерфейсу. У Django подання реалізуються за допомогою функцій або класів, які отримують дані від контролерів і передають їх у шаблони.

Контролер є посередником між моделлю та поданням. Він обробляє вхідні запити користувачів, викликає відповідні методи моделі для отримання або оновлення даних і передає їх поданню. У Django функції-подання (views) фактично виконують роль контролера, оскільки вони обробляють запити та відповідають за бізнес-логіку, передаючи дані в шаблони.

Кожен компонент MVC має свою роль, що дозволяє розробникам працювати над різними частинами системи незалежно. Наприклад, фронтенд-розробники можуть зосередитися на створенні шаблонів, тоді як бекенд-розробники працюють з моделями та логікою. Завдяки модульності кожен компонент можна тестувати окремо, що дозволяє швидко знаходити і виправляти помилки, а зміни в одному компоненті не впливають на інші, якщо дотримуватися чітких правил взаємодії між ними. Наприклад, зміна структури бази даних потребує лише оновлення моделей, а не всієї системи.

Серед інших переваг і той факт, що компоненти можна повторно використовувати в інших проектах, що знижує витрати на розробку. Також MVC спрощує розширення системи шляхом додавання нових функцій.

У розробленому форумі для студентів архітектура MVC стала основою для побудови системи. Моделі відповідають за структуру даних, такі як дошки, треди, коментарі та оцінки. Контролери у вигляді функцій-подань забезпечують обробку запитів, таких як створення нового поста або обробка голосів. Нарешті, шаблони використовуються для генерації веб-сторінок із динамічним вмістом, які відображаються користувачам.

Таким чином, застосування MVC-патерну дозволило створити проєкт, який легко масштабувати, підтримувати та розширювати. Це підтверджує доцільність використання даного підходу в розробці веб-додатків.

2.3. Вибір технологій розробки серверної складової проєкту

Перед вибором Django було розглянуто кілька інших платформ і фреймворків, які могли б бути використані для реалізації проекту.

Ruby on Rails – популярний веб-фреймворк, що використовує аналогічний MVC-патерн. Він легкий у використанні завдяки філософії "конвенція понад конфігурацію", та має велику спільноту розробників. Проте, основним недоліком є слабша інтеграція з інструментами Python, які використовуються в аналітиці та машинному навчанні [11].

Flask – мікрофреймворк на Python, який забезпечує більшу гнучкість і мінімалізм у порівнянні з Django. Проте, він вимагає більше зусиль при реалізації складних функцій, таких як аутентифікація або управління адміністративною панеллю [12].

Laravel — це фреймворк на PHP з підтримкою MVC-патерну. Він має розширений набір інструментів для швидкої розробки веб-додатків, вбудовані засоби для аутентифікації, роботи з БД та маршрутизацією [13].

ASP.NET Core – фреймворк від Microsoft, що підтримує різноманітні архітектурні підходи. Він добре інтегрується з іншими інструментами Microsoft, але сильно залежить від їх екосистеми [14].

Django — це один із найпопулярніших фреймворків для веб-розробки на мові Python. Django був обраний як основний фреймворк для розробки проєкту оскільки він забезпечує все необхідне для створення функціонального веб-додатка: вбудовану систему маршрутизації, ORM для роботи з базами даних, засоби аутентифікації та авторизації, а також адміністративну панель.



Рис. 2.2 Логотип Django.

Django дозволяє швидко створювати прототипи завдяки широкому набору готових компонентів і зручній структурі проекту, а також забезпечує високий рівень безпеки завдяки вбудованим механізмам захисту від XSS, CSRF-атак, SQL-ін'єкцій та інших типових вразливостей [15].

Його історія почалася у 2003 році, коли розробники газети The Lawrence Journal-World в Канзасі (США) працювали над створенням нових веб-додатків для своїх новинних сайтів. Вони потребували інструменту, який би дозволяв швидко і ефективно створювати веб-додатки, враховуючи необхідність частих оновлень контенту. У пошуках рішення вони почали розробляти власний фреймворк [16].

Спочатку цей інструмент використовувався виключно всередині редакції, але згодом розробники зрозуміли, що їхній фреймворк може бути корисним для інших. У липні 2005 року вони вирішили відкрити вихідний код проєкту, назвавши його Django — на честь джазового гітариста Джанґо Райнхарда, відомого своєю швидкістю та майстерністю гри.

У вересні 2006 року була випущена перша офіційна версія Django 0.90. Фреймворк одразу привернув увагу розробників завдяки своїй простоті, потужності та надійності. Його філософія "The web framework for perfectionists with deadlines" підкреслювала, що Django створений для швидкого розгортання проєктів без втрати якості.

Django активно розвивався, регулярно виходили нові версії, які додавали функціональність, розширювали можливості та покращували продуктивність. З часом його почали використовувати для створення веб-додатків різного рівня складності: від невеликих сайтів до великих корпоративних систем.

У вересні 2008 року була випущена перша стабільна версія Django 1.0. Цей реліз став важливою віхою в розвитку фреймворку, оскільки багато компаній і розробників очікували стабільної версії для використання в продакшн-середовищі.

Django постійно адаптується до змін у світі веб-розробки. У версії 1.7 було впроваджено систему міграцій, яка спростила роботу з базами даних. У версії 1.11 додали підтримку шаблонів типу QuerySet, що полегшило створення складних запитів.

З виходом Django 2.0 у грудні 2017 року було припинено підтримку Python 2, що дало змогу розробникам фокусуватися виключно на сучасній версії Python. Це також дозволило використовувати нові можливості Python 3, покращуючи продуктивність та зручність розробки.

На сьогоднішній день Django активно підтримується та оновлюється. У фреймворк інтегровані сучасні підходи до розробки, зокрема підтримка асинхронного програмування, що є важливим для масштабованих додатків. Крім того, фреймворк має потужну екосистему з багатьма сторонніми бібліотеками.

* 1. **Технології розробки клієнтської складової проєкту**

Клієнтська складова веб-сайтів складається з класичної комбінації HTML, CSS та JavaScript [17].

HTML (HyperText Markup Language) відповідає за структуру вебсторінки. Він надає базову розмітку, яка визначає, як контент, такий як текст, зображення, відео та інші елементи, буде організований на сторінці. HTML створює кістяк вебсайту.

CSS (Cascading Style Sheets) забезпечує стиль і зовнішній вигляд вебсторінки. За допомогою CSS можна налаштувати кольори, шрифти, відступи, розміри елементів, а також їхнє розташування на сторінці. CSS дозволяє зробити сайт адаптивним, щоб він коректно виглядав на різних пристроях, від смартфонів до великих екранів. За допомогою медіазапитів веброзробники можуть створювати стилі, які змінюються залежно від ширини екрану.

JavaScript додає інтерактивності вебсайтам. Елементи сторінки можуть реагувати на дії користувачів, такі як кліки, введення тексту, або прокручування за допомогою саме JavaScript. JavaScript також дозволяє створювати складні функціональні можливості і навіть односторінкові додатки (SPA) за допомогою таких бібліотек і фреймворків, як React чи Vue.js.

Взаємодія між HTML, CSS і JavaScript дозволяє створювати ефективні вебсайти.

Вибір Django в якості основної технології для серверної складової проєкту, привносить і зміни в роботі з клієнтською, оскільки при розробці інтерфейсу веб-платформи використання вбудованого шаблонізатора Django є одним із ключових аспектів для ефективної реалізації функціоналу та зручності подальшої підтримки проєкту. Цей шаблонізатор дає змогу охайно організувати код і полегшує підтримку та розширення платформи. Завдяки цьому підходу, HTML-код сторінок залишається зручним для розробників, а також легко адаптується під зміни в логіці.

Шаблонізатор у Django призначений для динамічного формування HTML-сторінок на основі даних, отриманих з веб-додатка. Важливим аспектом є те, що він дозволяє чітко відокремити відображення інформації (HTML-шаблони) від її обробки (логіка в Python-коді). Це дозволяє зменшити складність коду, робить його більш зрозумілим і зручним для командної роботи, де одні розробники можуть займатися логікою, а інші — дизайном та представленням даних.

Шаблонізатор працює через спеціальні файли, які містять HTML-код з інтегрованими позначеннями та конструкціями для вставки змінних, виконання циклів, умовних операторів та інших функцій. Такий підхід дозволяє створювати гнучкі й динамічні веб-сторінки, які автоматично адаптуються під різні дані та умови.

Шаблонізатор Django також підтримує концепцію наслідування шаблонів, що дозволяє створювати базові шаблони, а потім розширювати їх на конкретних сторінках. Це зменшує кількість повторень одного і того самого коду, полегшує підтримку єдиної структури та стилю на всіх сторінках проєкту.

Окрім цього шаблонізація надає змогу використовувати фільтри для модифікації значення змінних перед їх виведенням на сторінці [18].

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі були розглянуті теоретичні аспекти, необхідні для успішної реалізації проєкту. Під час дослідження ми ознайомилися з основними підходами до розробки веб-додатків, що дало змогу краще зрозуміти сучасні тенденції та найкращі практики у цій сфері. Зокрема, було проаналізовано класичні підходи, такі як монолітна архітектура, а також сучасні методи, включаючи мікросервіси та односторінкові додатки (SPA). Це допомогло обрати оптимальну стратегію для розробки нашого проєкту.

Окрему увагу було приділено дослідженню MVC-патерну, який є одним із найпоширеніших підходів до організації коду в веб-додатках. MVC (Model-View-Controller) забезпечує розподіл відповідальностей між різними компонентами системи, що полегшує розробку, тестування і підтримку коду. Це зробило його ідеальним вибором для нашого проєкту, оскільки він дозволяє розробляти серверну та клієнтську складові з чітким розмежуванням їхніх функцій.

На основі проведеного аналізу було обрано технології для розробки серверної частини. Ми вирішили використовувати Django — фреймворк, який поєднує в собі простоту, високу продуктивність та багатий набір інструментів. Django підтримує роботу з базами даних, обробку запитів і надає зручні засоби для реалізації MVC-патерну.

Для клієнтської частини було обрано стандартні веб-технології: HTML, CSS і JavaScript..

Таким чином, теоретична підготовка дозволила визначити ключові аспекти розробки, обрати відповідні інструменти та створити основу для подальшої реалізації проєкту.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1. Ініціалізація проєкту та опис його архітектури

Для початку роботи був встановлений Django, що відбувається за допомогою команди "pip install django" в консолі середовища. Після того, як пакет було встановлено, був створений і шаблон архітектури проєкту за допомогою команди "django-admin startproject StudForum", де "StudForum" – це назва проєкту. Після виконання цієї команди в кореневій папці проєкту з’явилися файл manage.py та директорія StudForum, в якій знаходяться наступні файли:

* init.py -порожній файл, який вказує, що папка myproject є Python-модулем. Завдяки йому Python розпізнає цей каталог як модуль, що дозволяє імпортувати конфігурації проєкту;
* settings.py - основний конфігураційний файл, в якому налаштовуються базові параметри проєкту (підключення бази даних, налаштування встановлених додатків, шляхи до статичних та медіа файлів тощо);
* urls.py - файл для визначення маршрутів (URL-адрес) у проєкті, у ньому прописуються правила, за якими запити користувача перенаправляються до відповідних обробників;
* asgi.py - конфігураційний файл для запуску асинхронного сервера, який працює через протокол ASGI (Asynchronous Server Gateway Interface). Використовується для роботи з WebSocket, інтеграції з реальними часом або іншими асинхронними задачами;
* wsgi.py -конфігураційний файл для запуску WSGI-сумісного веб-сервера, що використовується в більшості продакшн-середовищ для обробки синхронних запитів.

Наступним кроком було створення окремого застосунку – модулю, що групує код, пов’язаний з основною функціональністю форуму. Застосунок був названий "forum" та ініціалізований командою "python manage.py startapp forum". У результаті утворилася структура застосунку з такими файлами та папками:

* migrations/ - папка для зберігання файлів міграцій, які створюються для управління схемою бази даних.
* admin.py – Цей файл використовується для налаштування адміністративної панелі Django. Він дозволяє реєструвати моделі, щоб ними можна було керувати через вбудований інтерфейс адміністратора.
* apps.py – містить клас конфігурації застосунку. Це ім’я використовується Django для розпізнавання застосунку.
* models.py – основний файл для опису моделей, які представляють структуру даних застосунку. Моделі визначають таблиці в базі даних.
* tests.py – файл для написання тестів, які перевіряють функціональність застосунку. Тут створюються юніт-тести, що автоматично перевіряють коректність коду.
* views.py – вміщує функції або класи, які відповідають за обробку запитів і повернення відповідей. Тут описуються логіка для відображення сторінок, взаємодія з моделями та формування даних для користувача.

Окрім стандартної структури, створеної командою startapp, для застосунку "forum" були додані додаткові папки та файли, що забезпечують його повну функціональність і дотримання сучасних практик розробки:

* static/ - папка, містить статичні файли, що використовуються у
* templates/ - папка, що містить HTML-шаблони, що використовуються у застосунку для відображення веб-сторінок.
* templatetags/ - папка, в якій розміщуються власні теги та фільтри для шаблонів.
* forms.py – файл для створення форм користувацького вводу, таких як створення тем або коментарів.
* urls.py – як і файл в загальному каталозі визначає маршрути (URL), що відповідають за доступ до сторінок в межах застосунку форуму.

Після створення усіх потрібних файлів структура проєкту виглядає наступним чином:

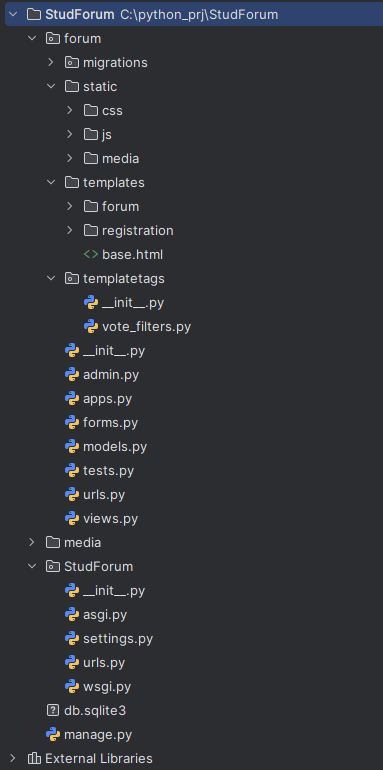


Рис. 3.1 Скріншот структури проєкту в середовищі PyCharm.

3.2. Реалізація серверної складової (Back end).

Розробка серверної складової проєкту почалася з створення основних моделей: дошки, треду та коментаря, а також імпорту моделі користувача, що вже існує в Django. Розберемо код моделей дошки і треду:



Рис.3.2 Код задання моделей дошки і треду

Модель Board представляє дошку, де обговорюються певні теми. Вона зберігає назву, опис і дату створення дошки. Метод \_\_str\_\_ використовується для зручного відображення назви дошки у вигляді тексту, наприклад, в адміністративній панелі або в інших текстових представленнях.

Модель Thread представляє потік обговорення в межах конкретної дошки. Потік має прив'язку до дошки (board) та користувача (user), який його створив. У потоці зберігаються такі дані, як заголовок, основний текст, зображення (якщо додано), рейтинг, час створення і унікальний ідентифікатор.

Ключовий метод у Thread — це update\_rating. Він обчислює рейтинг потоку на основі кількості позитивних (upvotes) та негативних (downvotes) голосів, які зберігаються в окремій моделі Vote. Різниця між кількістю upvotes і downvotes визначає поточний рейтинг, який оновлюється в базі даних.

Метод time\_since\_posted показує, скільки часу минуло з моменту створення потоку. Це додає зручності користувачам, оскільки дозволяє бачити, як "свіжий" цей потік, наприклад, "5 хвилин тому" або "2 дні тому". Ця інформація генерується за допомогою функції timesince.

Далі розберемо моделі коментаря та голосу:



Рис. 3.3 Код задання моделей коментаря і голосу

Клас Comment представляє коментар до певної теми (посту) на форумі. Він містить посилання на модель Thread (тема, до якої належить коментар) через зовнішній ключ. Це дозволяє зручно отримувати всі коментарі, пов'язані з конкретною темою, через атрибут comments. Також у класі є поле user, яке визначає, хто залишив коментар. Вміст самого коментаря зберігається в полі content, а якщо користувач додав зображення, воно зберігається в полі image. Поле created\_at автоматично записує дату та час створення коментаря, що дозволяє відстежувати хронологію подій. Метод \_\_str\_\_ забезпечує зрозуміле текстове представлення коментаря, показуючи, хто і до якого посту його додав.

Клас Vote відповідає за систему голосування за теми. Він містить посилання на користувача (user) та тему (thread), за яку був відданий голос. Поле is\_upvote зберігає інформацію про тип голосу: True для позитивного голосу та False для негативного. Логіка, закладена в цьому класі, гарантує, що кожен користувач може проголосувати за конкретну тему лише один раз. Це досягається за допомогою мета-класу Meta та параметра unique\_together, який вказує, що пара user і thread повинна бути унікальною. Таким чином, користувач може додати або відняти темі одиницю рейтингу, але не більше одного разу.

Після написання коду для моделей, що використовуються на платформі було проведено міграції для налаштування структури бази даних. Наступним кроком було створення шаблонів сторінок, що відображаються у браузері та написання коду представлення у файлі views.py. Розберемо код представлення головної сторінки сайту:



Рис. 3.4. Код представлення головної сторінки

Цей фрагмент коду містить функції home та sorting, які відповідають за відображення головної сторінки форуму та логіку сортування тем.

Функція home обробляє запити на головну сторінку. Спочатку вона отримує параметр сортування з URL-адреси (sort\_by) і передає його у функцію sorting для отримання відсортованих тем. За замовчуванням сортування відбувається за останніми темами (recency).

Далі функція обробляє голоси. Якщо користувач автентифікований, для кожної теми перевіряється, чи існує голос від цього користувача. Якщо голос є, встановлюється статус голосу для теми: 'upvoted', 'downvoted' або 'not\_voted'. Якщо користувач не ввійшов у систему, всі теми позначаються як 'not\_voted'.

Темами управляє Paginator, який розбиває їх на сторінки по 10 елементів. Залежно від типу запиту, функція повертає або частковий HTML (для AJAX-запитів), або повний рендер сторінки.

Функція sorting відповідає за сортування тем відповідно до вибраного параметра. Спочатку створюється базовий запит для отримання всіх тем. Якщо передані параметри user або board, до запиту додаються відповідні фільтри, щоб відобразити теми певного користувача або з певної дошки. Сортування підтримує два варіанти – за рейтингом та за датою

Функція також додає кількість коментарів до кожної теми за допомогою annotate.

Ці функції дозволяють користувачам бачити різні версії відображення тем залежно від їхніх уподобань, забезпечуючи інтерактивну систему сортування та підтримуючи механізм безперервного завантаження нових тем через AJAX.

Далі розберемо код представлення сторінки профіля користувача. Принцип відображення сторінки користувача залишається той же, що і у головної сторінки. Серед особливостей перевірка на існування заданого користувача і виклик помилки 404 у разі його відсутності. Також підраховується сума рейтингів серед усіх постів користувача, яка виводиться як загальний рейтинг цього користувача:



Рис.3.5. Код представлення сторінки профіля користувача

Код представлення сторінки з дошками принципово не відрізняється від сторінки профілю. Розберемо код голосування за пост:



Рис.3.6. Код голосування за пост

Функція upvote\_thread забезпечує механізм для голосування за тему шляхом додавання або зміни позитивного голосу (upvote). Спочатку функція отримує об'єкт Thread за його унікальним ідентифікатором thread\_id. Якщо тема з таким ID не існує, повертається сторінка 404. Потім перевіряється, чи існує вже голос користувача за цю тему. Якщо для посту вже є голос upvote, то голос видаляється, а рейтинг теми зменшується на 1. Якщо голос є і це downvote, то голос оновлюється до upvote, і рейтинг теми збільшується на 2 (анулювання downvote та додавання upvote). Якщо голосу немає, то створюється новий позитивний голос, і рейтинг теми збільшується на 1. У будь-якому випадку, після зміни голосу, оновлюється рейтинг теми, і ці зміни зберігаються у базі даних.

Після виконання логіки голосування користувача перенаправляють на іншу сторінку. Параметр next в URL вказує, куди повертати користувача після голосування (наприклад, назад на сторінку деталей теми або список тем). Якщо в URL є параметр сортування sort, він також додається до перенаправлення для збереження порядку сортування. Завдяки логіці створення та оновлення об'єктів Vote, один користувач може мати лише один голос за одну тему, що запобігає дублюванню. Схожа функція написана і для реалізації негативного голосу.

Також реалізована представлення сторінки з деталями конкретного треду. Спочатку завантажується тред за його ідентифікатором, і якщо його немає, користувач отримує помилку 404. Потім вибираються всі коментарі до цього треду, відсортовані за часом створення у зворотному порядку, щоб спочатку показувати найновіші. Якщо користувач авторизований, визначається його стан голосування щодо треду: чи він проголосував "за", "проти", чи ще не голосував. Якщо це POST-запит, обробляється форма коментаря, яка створює новий запис і прив’язує його до поточного треду і користувача, після чого сторінка перезавантажується. Якщо це GET-запит, підготовлюється порожня форма. У кінці формується контекст, що включає тред, його коментарі, форму для нового коментаря та список усіх доступних дошок для навігації, після чого все це передається в шаблон для рендерингу. Нижче прикріплено код її представлення:



Рис.3.7. Код представлення сторінки треду.

Останніми з тих функцій в файлі view.py, що ми розглянемо, будуть функції створення нового треду та пошуку тредів на сайті. Їх код можна побачити нижче:



Рис.3.8. Код функцій створення треду та пошуку по сайту.

Функція create\_post відповідає за створення нового треду (теми) на форумі. Вона доступна лише для авторизованих користувачів завдяки декоратору @login\_required. Якщо користувач надіслав POST-запит, формується екземпляр форми ThreadForm, і якщо вона валідна, створюється новий тред, який прив’язується до поточного користувача та дошки. Якщо назва дошки передана в URL, використовується відповідна дошка, інакше вибирається перша доступна. Після збереження нового треду користувача перенаправляють на головну сторінку. Для GET-запитів відображається порожня форма створення треду. Також передаються всі доступні дошки для рендерингу навігації.

Функція search обробляє запити на пошук тредів. Вона отримує текст пошукового запиту через параметр q із GET-запиту. Якщо запит не порожній, виконується пошук у заголовках, змісті тредів та іменах користувачів, сортування результатів здійснюється за часом створення в зворотному порядку. Потім результати розбиваються на сторінки за допомогою Paginator, де на одній сторінці відображається до 10 записів. У результаті рендериться шаблон із переданими результатами пошуку, поточним запитом і списком доступних дошок для навігації.

Після написання коду подання, в файлі urls.py була налаштована маршрутизація шляхів для коректного переходу з однієї сторінки сайту на інші:



Рис.3.9. Реалізація маршрутизації на сайті

Список urlpatterns визначає відповідність між URL-шляхами і функціями представлень, які обробляють ці запити. Основна логіка така:

* Кореневий URL ('') перенаправляє на головну сторінку (views.home).
* register/ відповідає за реєстрацію нових користувачів через функцію register.
* Шляхи для голосування за треди (thread/<int:thread\_id>/upvote/ і thread/<int:thread\_id>/downvote/) дозволяють користувачам голосувати за або проти конкретного треду.
* login/ та logout/ використовують класи представлень для входу та виходу користувачів.
* profile/<str:username>/ відображає профіль користувача, який визначається за переданим іменем.
* thread/<int:thread\_id>/ відображає деталі конкретного треду через функцію thread\_detail.
* board/<str:board\_name>/ дозволяє переглядати всі треди, що належать певній дошці.
* create\_post/ забезпечує створення нового треду, з можливістю вказати дошку через додатковий параметр <str:board\_name>.
* search/ забезпечує пошук по тредах та відображення результатів.

Далі в директорії templatetags був створений файл vote\_filters.py для реалізації роботи двух власних фільтрів для використання в шаблонах.

Фільтр get\_vote\_status отримує голос користувача для конкретного треду і повертає стан його голосування: "upvoted" (якщо голос був позитивним), "downvoted" (якщо голос був негативним) або None, якщо користувач не голосував за цей тред. Використовується модель Vote, щоб перевірити, чи є голос цього користувача для вказаного треду. За допомогою цього фільтру на сторінках з тредами можна окремим стилем відображати кнопки голосування за теми, голос за які вже було віддано, що дає можливість зробити UX сайту набагато більш інтуїтивним.

Фільтр shorten\_rating скорочує великі числові значення до більш зручного вигляду з позначками "K" для тисяч і "M" для мільйонів. Наприклад, значення 1,200 стає "1.2K", а 1,000,000 стає "1.0M". Число округляється до одного знаку після коми для зручності читання, при цьому зайві нулі після десяткової точки видаляються. Це робить дизайн сторінки більш чистим і запобігає візуальним проблемам з відображенням великих значень рейтингів. Код цих фільтрів виглядає наступним чином:

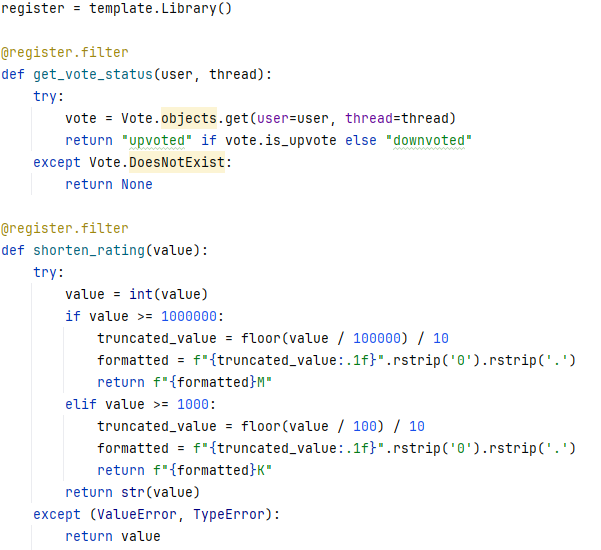


Рис. 3.10. Код реалізації власних фільтрів.

Далі в середині файлу forms.py був реалізований код, що визначає форми для створення коментарів і тредів на форумі з підтримкою стиснення зображень:

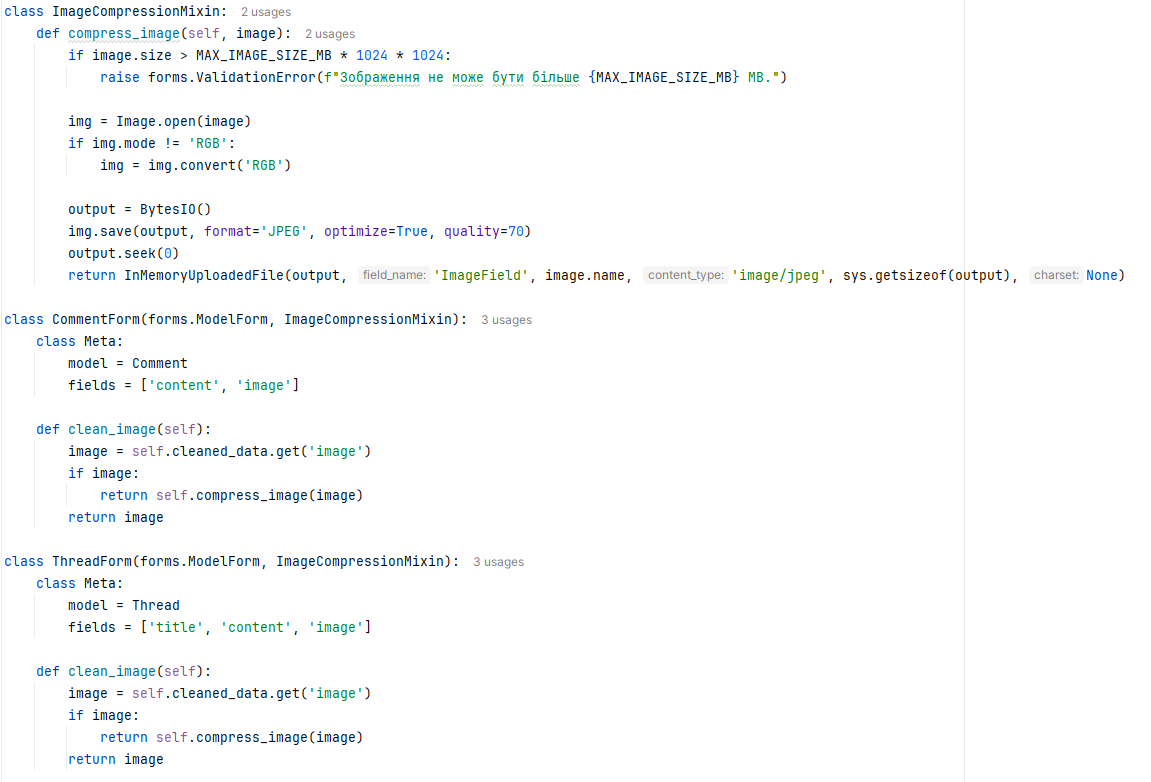


Рис. 3.11. Код форм створення тредів і коментарів

Клас ImageCompressionMixin надає метод compress\_image, який перевіряє розмір зображення. Якщо розмір перевищує максимальний ліміт (2 MB), виникає помилка валідації. Зображення відкривається за допомогою бібліотеки PIL (Python Imaging Library) і конвертується в формат RGB, якщо це необхідно. Потім зображення стискається і зберігається у форматі JPEG з якістю 70%, що дозволяє зменшити розмір файлу без значних втрат у якості. Після стиснення зображення повертається як об'єкт InMemoryUploadedFile, який можна використовувати для збереження в базі даних.

Форма CommentForm призначена для створення коментарів. Вона включає поля для введення тексту і для додавання зображення. Метод clean\_image перевіряє зображення після його завантаження і застосовує метод стиснення зображення, якщо файл існує.

Форма ThreadForm використовується для створення або редагування тредів і також включає поля для заголовка, контенту і зображення. Зображення також проходить через метод стиснення перед збереженням.

3.3. Реалізація клієнтської складової (Front end).

Для реалізація клієнтської складової проєкту використовувалась класична комбінація HTML, CSS та JavaScript з використанням шаблонізації Django. Був створений базовий шаблон base.html, який містить в собі основні елементи сторінки, що повторюються на кожній іншій сторінці сайту, та який доповнюють усі інші шаблони:



Рис.3.12. Фрагмент коду base.html

Цей HTML-шаблон використовує теги шаблонізатора Django для інтеграції динамічного контенту, умовної логіки та ресурсів статичних файлів у веб-додаток. Розглянемо ключові моменти детальніше:

* Тег {% load static %} завантажує теги для роботи зі статичними файлами (CSS, JavaScript, зображення тощо).
* Тег {% load vote\_filters %} підключає кастомні теги або фільтри вказані в файлі vote\_filters, які було розглянуто раніше.
* Тег {% static %} використовується для отримання URL-адреси статичного файлу. Наприклад, у <link rel="stylesheet" href="{% static 'css/styles.css' %}"> додається посилання на файл стилів, що зберігається в папці static/css.
* Тег {% url %} формує URL-адресу для певного представлення. Наприклад, {% url 'home' %} генерує посилання на домашню сторінку, а {% url 'profile' user.username %} створює URL до сторінки профілю користувача.
* Умовна логіка {% if %} перевіряє, чи користувач авторизований ({% if user.is\_authenticated %}), і в залежності від цього показує відповідний набір елементів. Авторизованим користувачам відображається ім'я та кнопка виходу, тоді як гості бачать посилання на вхід і реєстрацію.
* Тег {{ user.username }} вставляє ім'я поточного користувача.
* Тег {{ request.GET.q|default\_if\_none:'' }} використовується для відображення значення пошукового запиту в полі введення. Якщо запит порожній, замість цього вставляється порожній рядок.
* У формі виходу тег {% csrf\_token %} додає токен для захисту від CSRF-атак, обов’язковий для POST-запитів.
* Теги {% block content %} та {% endblock %} визначають місце для вмісту дочірніх шаблонів, які розширюють цей базовий шаблон.

Цей шаблон організовано так, щоб забезпечити зручний доступ з усіх сторінок сайту до функціоналу веб-додатка: навігація, пошук, автентифікація та інші елементи інтерфейсу.

Після основного шаблону були створені шаблони кожної потрібної сторінки: головної, сторінки користувача, сторінки дошки, сторінки інформації про тред, сторінки результату пошуку, сторінки входу в аккаунт та реєстрації, а також часткові шаблони екрану створення треду та розбиття тредів на сторінки, що використовуються в декількох інших шаблонах. Також були створені статичні файли стилів, скриптів та іконок. Таким чином структура файлів, що використовуються для задання клієнтської складової сайту, виглядає так:



Рис. 3.13 Структура шаблонів та статичних файлів.

Розглянемо доповнення базового шаблону на прикладі файлу головної сторінки, де відображаються треди з усіх дошок разом:



Рис.3.14. Код main.html

Цей шаблон є дочірнім і розширює базовий шаблон base.html. Його основне завдання — відобразити головну сторінку із можливістю сортування та створення постів. Нижче детально пояснюється логіка використаних тегів та елементів:

* Тег {% extends "base.html" %} дозволяє використовувати спільну структуру (шапку, футер тощо) з базового шаблону, зосереджуючись лише на вмісті блоку {% block content %}.
* Тег {% load vote\_filters %} підключає спеціальні фільтри для обробки голосів.
* Тег {% load static %} дозволяє працювати зі статичними файлами, такими як CSS чи JavaScript.
* В сайдбарі відображається кнопка створення посту та список доступних дошок ({% for board in boards %} ітеративно проходить через список дошок, додаючи посилання на кожну дошку за допомогою {% url 'board\_view' board.name %}).
* Варіанти сортування (Нові і Популярні) дозволяють сортувати треди. У кожному посиланні до параметра запиту sort додається відповідне значення (recency для нових тредів або rating для популярних). {% if sort\_by == 'recency' %}active{% endif %} додає клас active для активної опції сортування, що забезпечує її візуальне виділення.
* Форма створення посту включається через тег {% include 'forum/partials/post\_creation.html' with boards=boards %}. Це дозволяє повторно використовувати шаблон форми для створення посту, передаючи доступні дошки у вигляді контексту.
* Список тредів відображається за допомогою тегу {% include 'forum/partials/thread\_list.html' with threads=threads %}. Передаються всі треди для рендерингу за допомогою окремого шаблону, що відповідає за структуру відображення списку тредів, які, за допомогою підключених скриптів, непомітно з’являються на сторінці по 10 постів за раз в процесі прокручування сторінки вниз.

Оскільки структура списку тредів залишається незмінною як на головний сторінці, так і на сторінці профіля користувача, сторінці дошки, та сторінці результатів пошуку, винесення її в окремий файл часткового шаблону є вдалим рішенням. Це дає можливість зробити шаблони чистішими та не повторювати однакові фрагменти коду багато разів. Розглянемо такий частковий шаблон thread\_list.html:

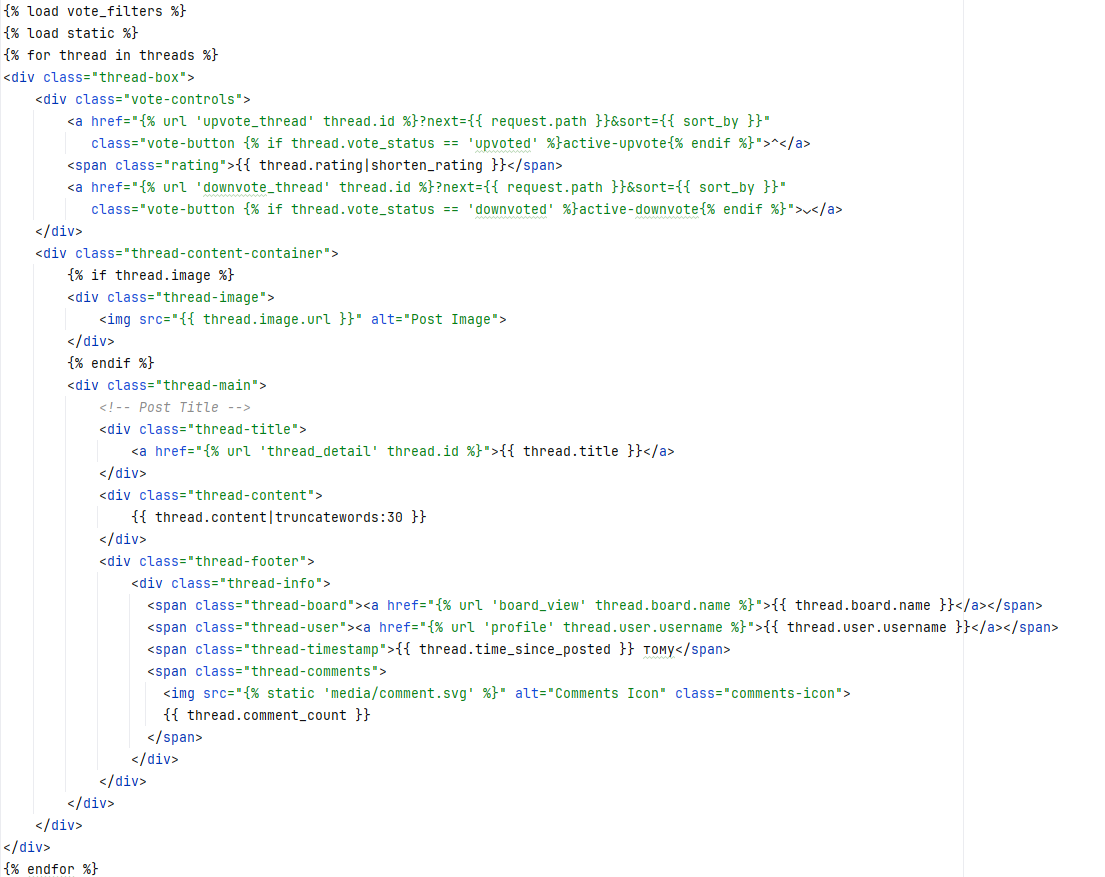


Рис.3.15 Код файлу thread\_list.html

Шаблон перебирає всі треди, передані через контекст, за допомогою циклу {% for thread in threads %}. Для кожного треду створюється окремий блок з його елементами.

* Блок контролів голосуванняскладається з кнопок для підняття та зниження рейтингу. Кожна кнопка посилається на відповідні URL-адреси для голосування: {% url 'upvote\_thread' thread.id %} і {% url 'downvote\_thread' thread.id %}. Додаються GET-параметри next (щоб після дії перенаправити назад на поточну сторінку) та sort (для збереження сортування). Класи active-upvote або active-downvote додаються динамічно залежно від статусу голосу користувача (thread.vote\_status), щоб візуально відобразити активний стан.
* Якщо тред містить зображення, воно відображається в окремому контейнері.
* Назва треду є посиланням на сторінку деталей треду ({% url 'thread\_detail' thread.id %}).
* Зміст треду обрізається до 30 слів за допомогою фільтра truncatewords.
* Назва дошки, де розміщений тред, з посиланням на відповідну дошку.
* Ім'я користувача, який створив тред, з посиланням на його профіль.
* Час, що минув з моменту створення треду, форматований для відображення в стилі "N годин тому".
* Кількість коментарів з відповідною іконкою, що завантажується через {% static %}.

Серед статичних даних є два файли JavaScript: main.js та infinite\_scroll.js. Розглянемо логіку їх роботи починаючи з файлу main.js:



Рис. 3.16 Код main.js

В цьому файлі реалізована робота кнопки «повернутися вгору», яка з’являється в правому нижньому екрані при гортанні сторінки вниз та піднімає користувача на початок сторінки при натисканні на неї, та функція виводу на екран форми для створення нового треду при натисканні на кнопку «створити пост».

При натисканні на кнопку з ідентифікатором create-post-btn, зникає клас hidden у модальному вікні з ідентифікатором post-modal, що робить його видимим на сторінці. Це дозволяє користувачеві побачити форму для створення посту.

Кнопка з ідентифікатором cancel-post-btn додає клас hidden назад до модального вікна, що ховає його. Крім того, якщо користувач натискає на будь-яку частину екрану поза модальним вікном, воно також ховається.

Кнопка для прокручування до верху (scroll-to-top) спочатку прихована. Вона стає видимою, коли користувач прокручує сторінку більше ніж на 200 пікселів.

При натисканні на кнопку прокрутки виконується плавний скролл до верхньої частини сторінки за допомогою window.scrollTo з параметром behavior: 'smooth', що додає анімацію прокрутки.

В файлі infinite\_scroll.js реалізований скрипт «нескінченного скролу», який подає нові пости при прокручуванні сторінки вниз, розглянемо логіку його роботи. Вводиться змінна page для відстеження поточної сторінки (початково встановлено на 2, що означає, що перша сторінка вже завантажена) та змінна loading для перевірки, чи триває запит.

Функція loadMore відповідає за завантаження наступної порції даних. Перевірка if (loading) return; гарантує, що новий запит не буде відправлений, поки не завершиться поточний.

Використовується URLSearchParams для додавання або оновлення параметра page у URL-адресі, щоб вказати серверу, яку сторінку потрібно завантажити. Код можна побачити нижче:

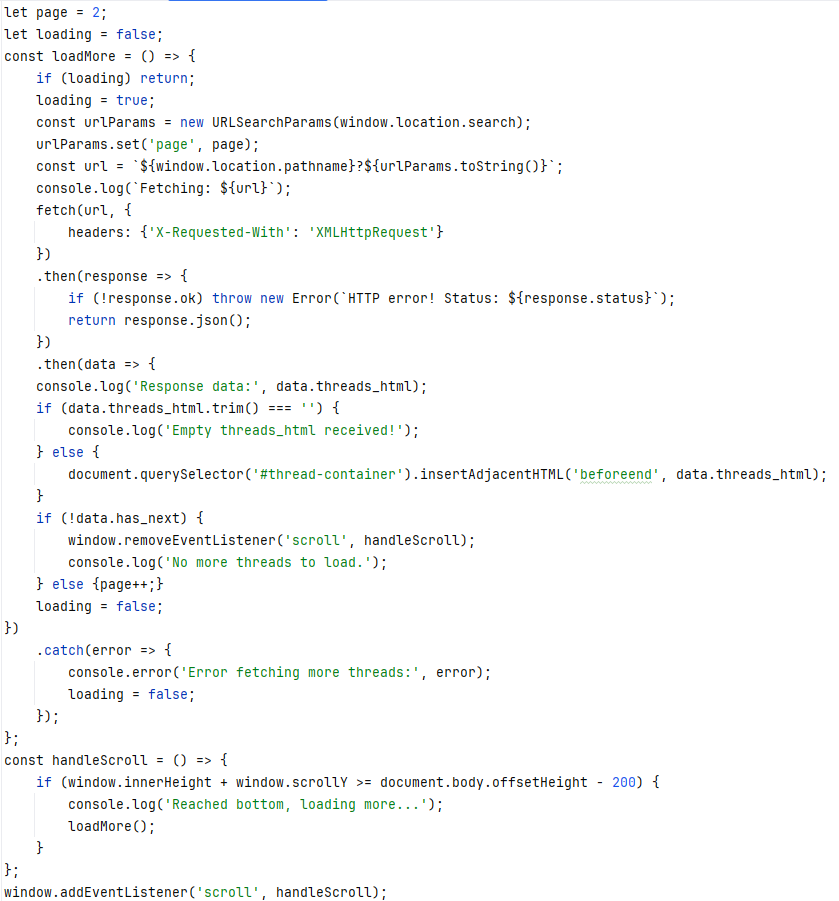


Рис.3.17. Код infinite\_scroll.js

Потім запит надсилається через fetch з параметром X-Requested-With: XMLHttpRequest, щоб сервер розпізнав запит як AJAX-запит.

Якщо сервер повертає коректний JSON, у якому міститься HTML нових тредів (через ключ threads\_html), вони вставляються у контейнер з ідентифікатором #thread-container.

Якщо більше немає тредів для завантаження (через data.has\_next), слухач подій прокрутки видаляється за допомогою window.removeEventListener('scroll', handleScroll).

Функція handleScroll визначає, чи дійшов користувач до низу сторінки (в межах 200 пікселів). Якщо так, викликається функція loadMore для завантаження нових тредів.

Рядок window.addEventListener('scroll', handleScroll); додає слухача події на прокрутку, який викликає handleScroll при кожному русі прокрутки сторінки.

3.4. Локальне розгортання проєкту та тестування

Для запуску проєкту Django достатньо ввести в термінал команду "python manage.py runserver". Після цього в URL-поле браузера треба ввести адресу //127.0.0.1:8000/, що виведе на екран головну сторінку сайту. В цей час усі зміни, внесені до файлів проєкту та збережені, будуть з’являтися на сайті після оновлення сторінки в браузері. А якщо в файлі settings.py залишити змінну DEBUG = True, то при виникненні помилок, вся інформація про них буде виводитись у вікно браузера. Це робить процес налагодження роботи коду та усунення проблем в Django дуже простим.

Почнемо тестувати функціонал сайту з головної сторінки:

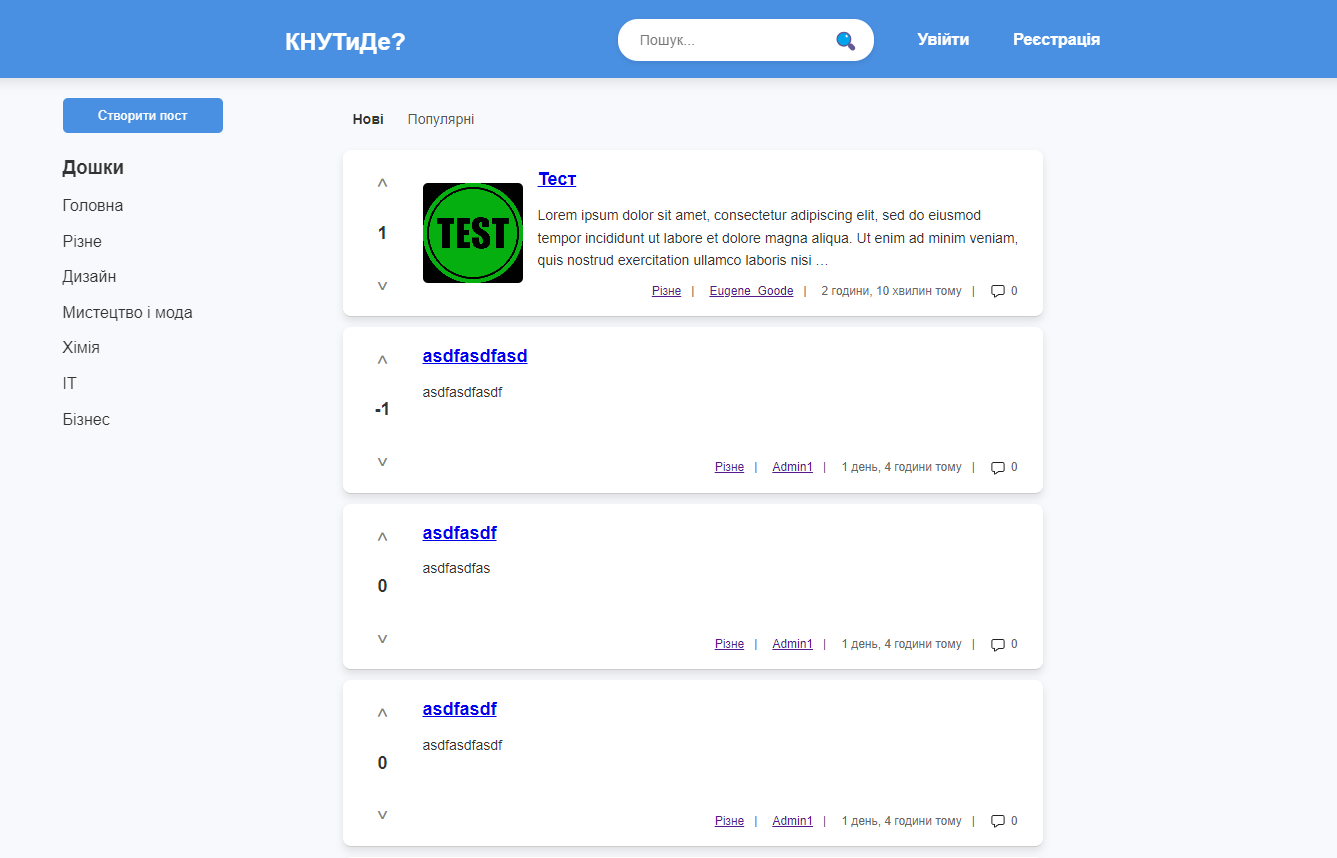


Рис.3.18. Головна сторінка сайту без входу в аккаунт.

Спроби проголосувати за будь-який пост не увійшовши в аккаунт користувача перенаправлять на сторінку входу в аккаунт.

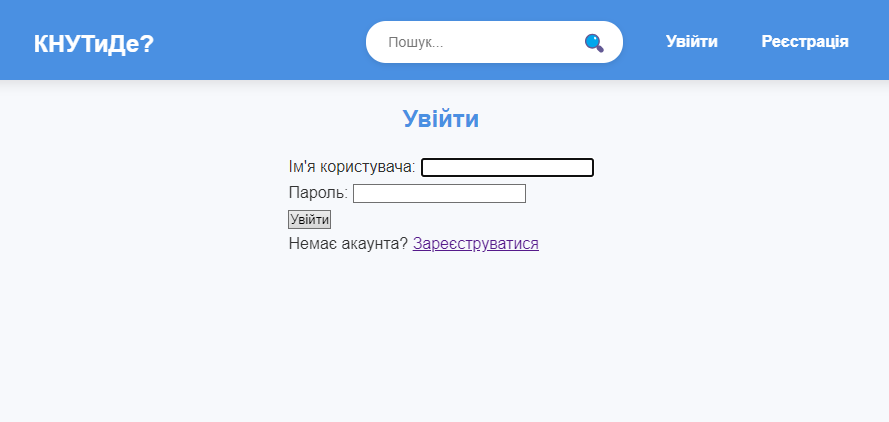


Рис.3.19. Сторінка входу в аккаунт.

Після входу в аккаунт користувача, треди, за які був відданий голос, мають або позначену зеленим кнопку позитивного голосу, або позначену червоним кнопку негативного голосу, що слугує індикацією того, яка операція була здійснена користувачем.

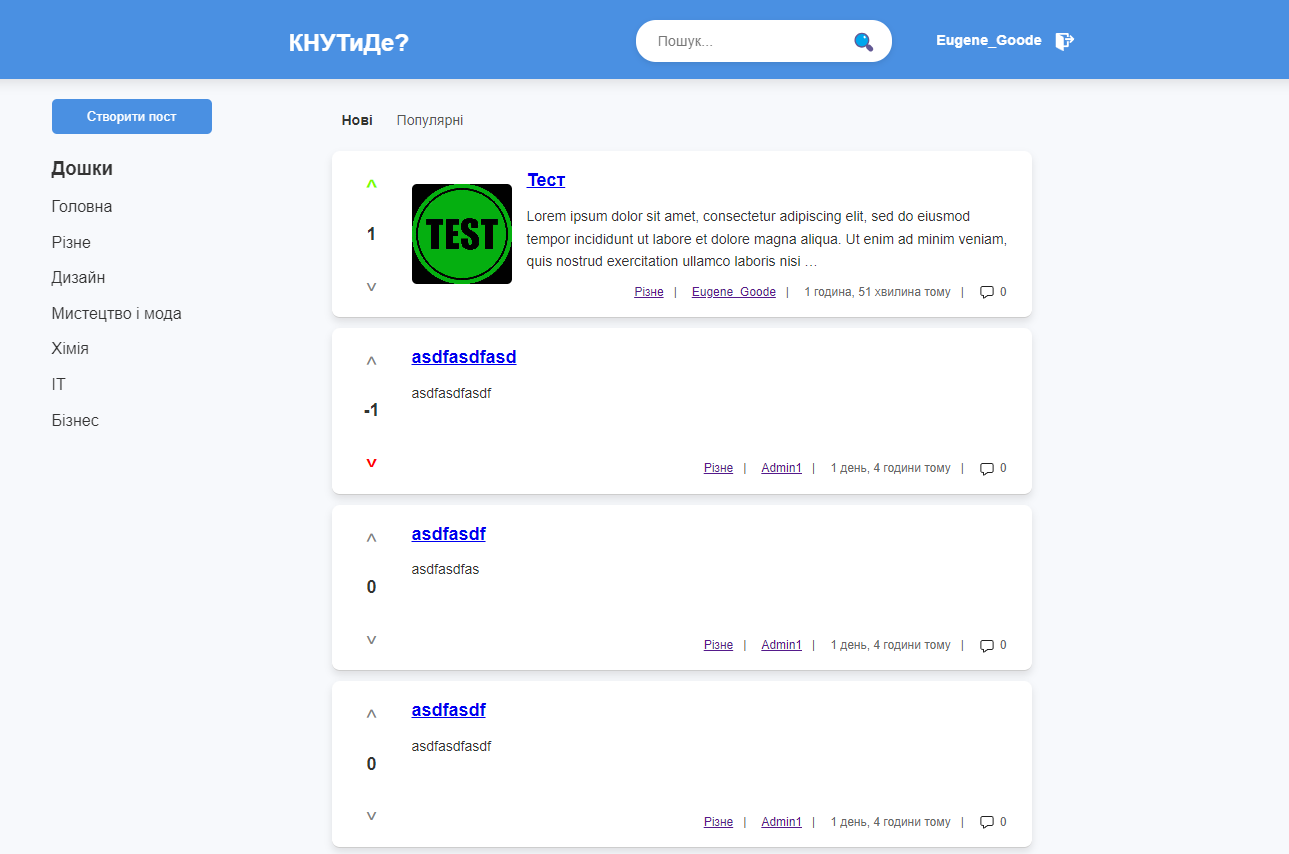


Рис.3.20. Головна сторінка з відзначеними тредами.

Перевіримо роботу кнопки «Створити пост».

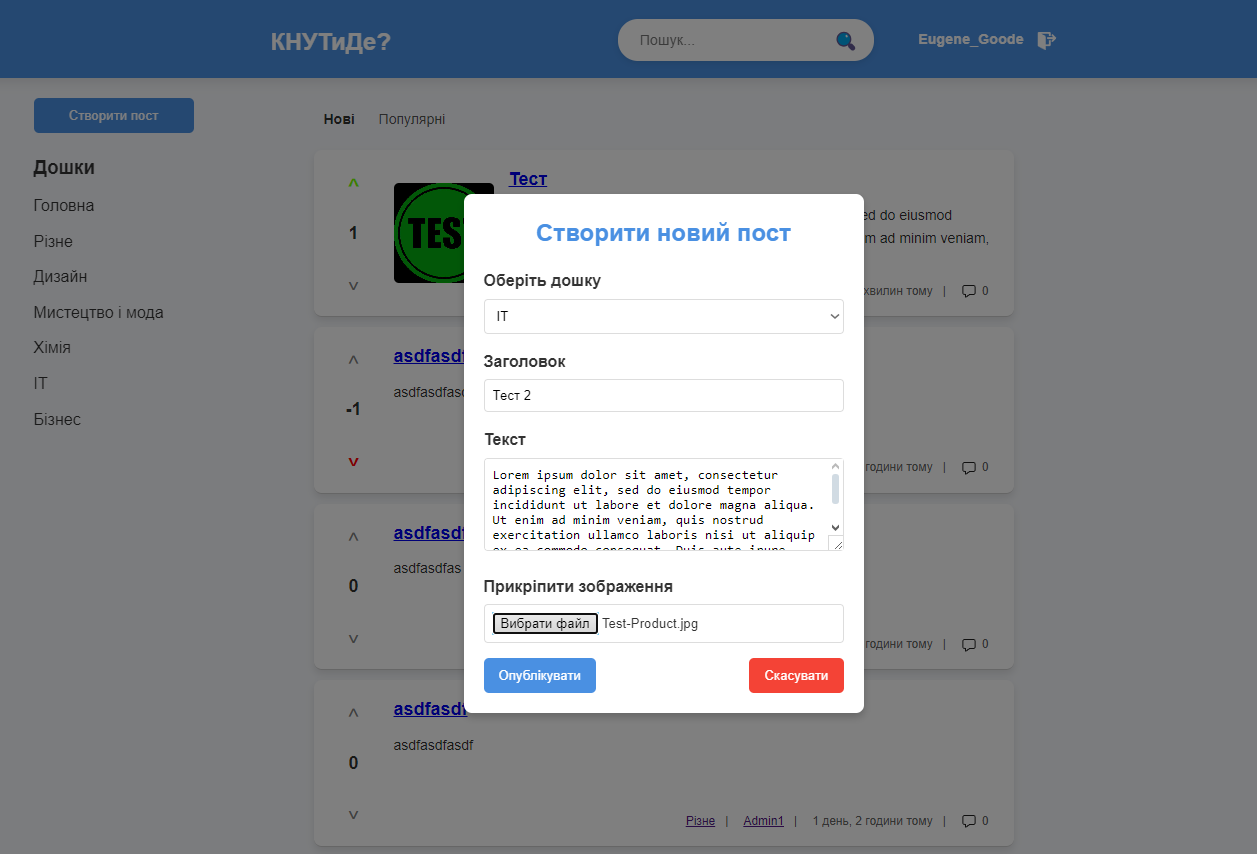


Рис. 3.21. Форма створення нового треду.

Натискання кнопки відкрило форму для заповнення інформації для нового треду. Після натискання на кнопку «Опублікувати» в дошці «IT» було створено новий тред.

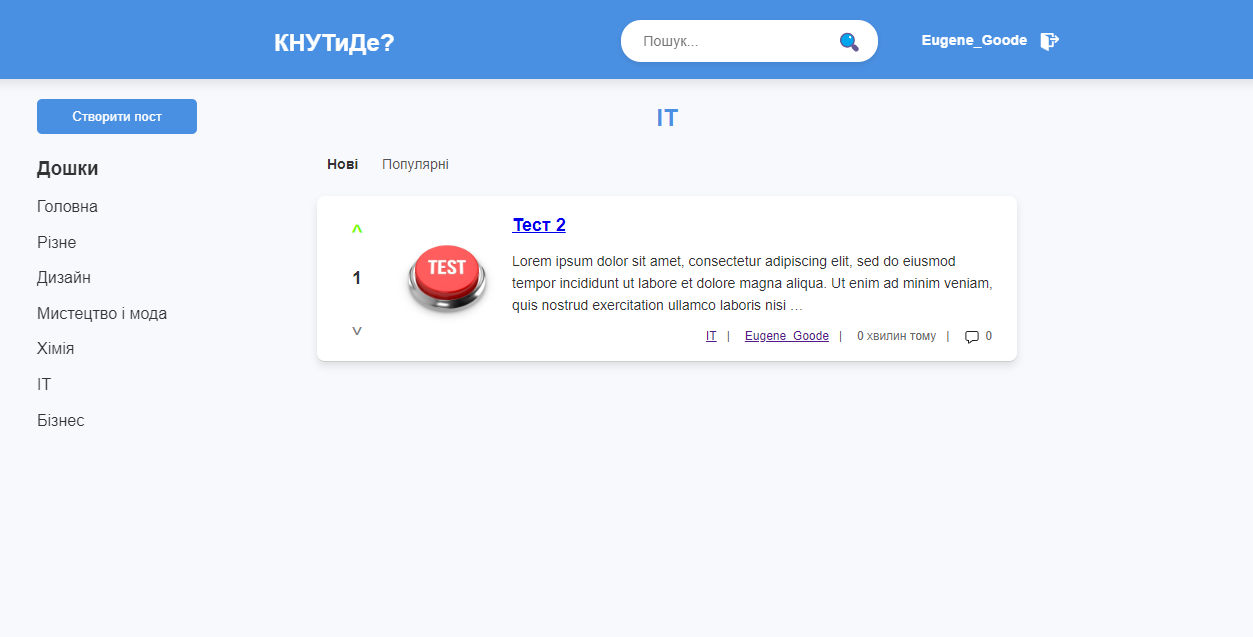


Рис. 3.22. Дошка «IT».

Перевіримо сторінку деталей треду:

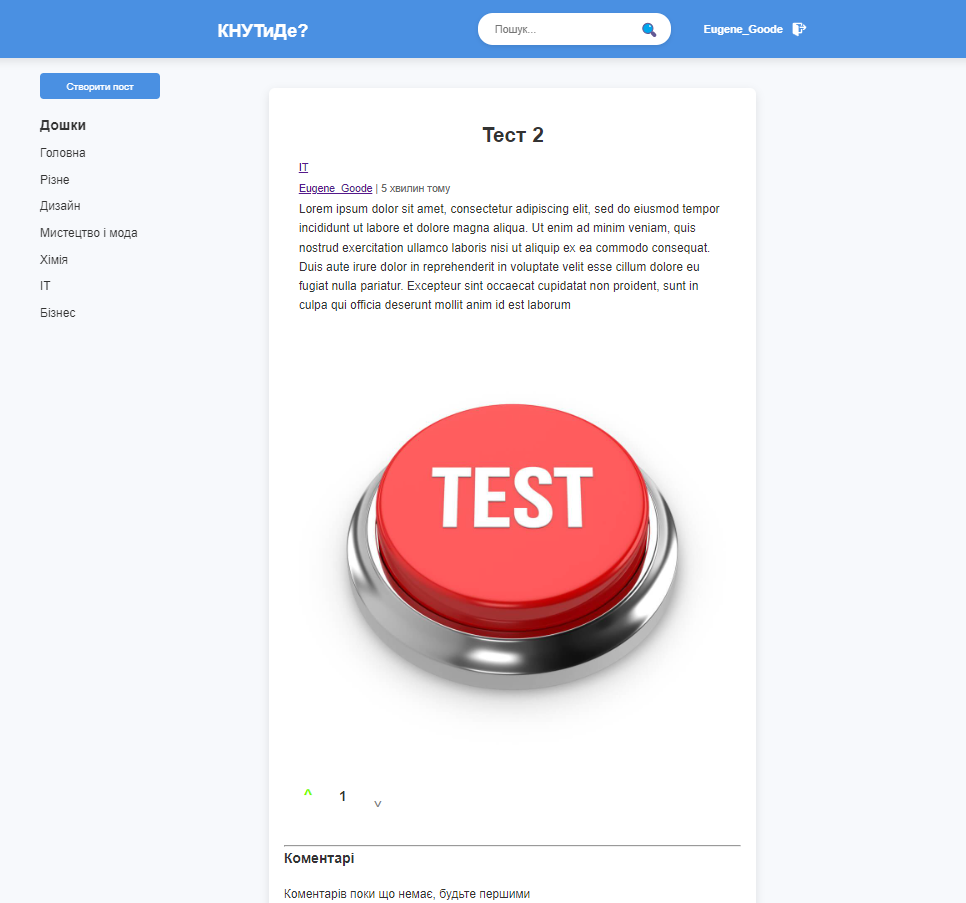


Рис. 3.23. Сторінка деталей треду.

Тепер спробуємо залишити коментар:

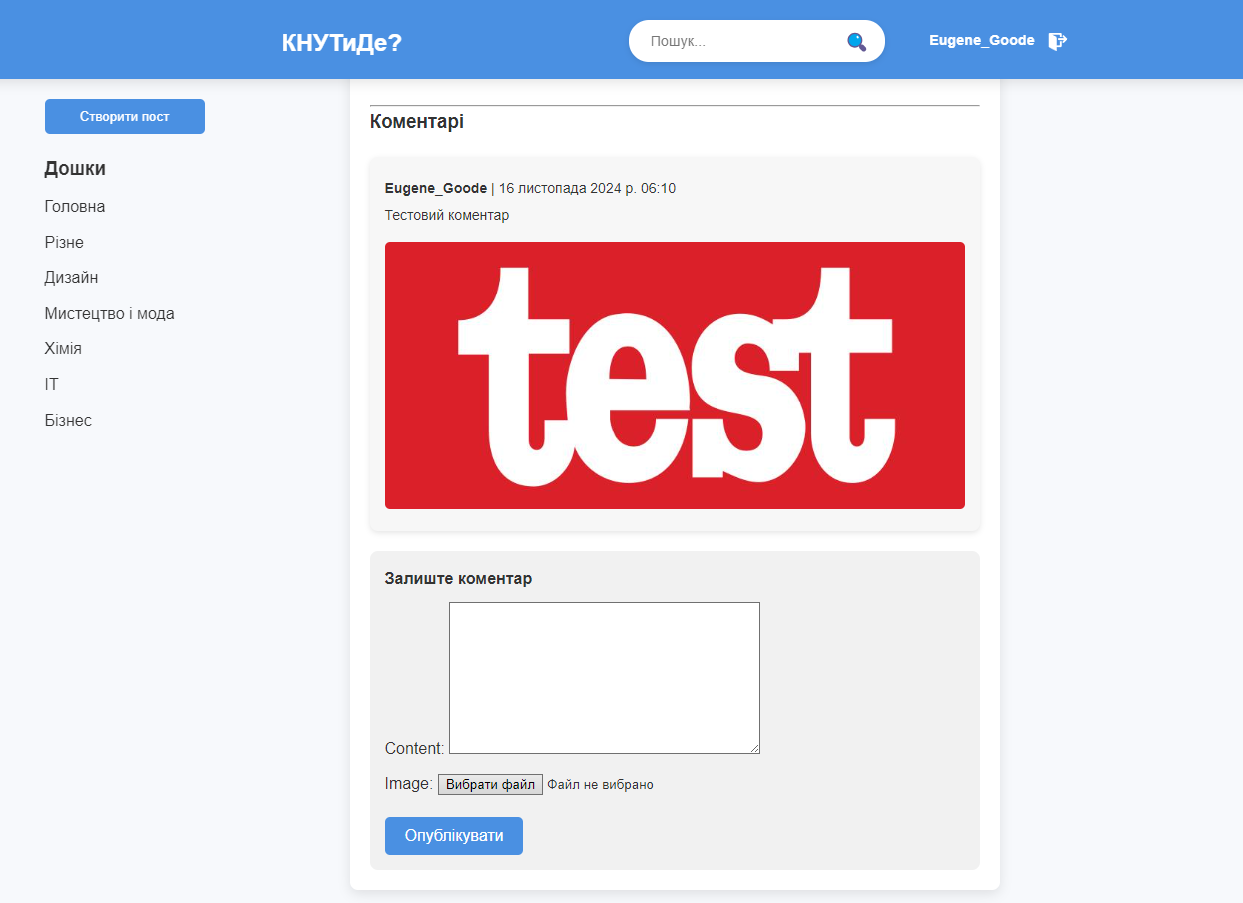


Рис. 3.24 Секція коментарів на сторінці треду.

Сторінка деталей треду коректно відображає всю інформацію, а заповнення форми нового коментаря і натискання на кнопку «Опублікувати» вдало створило новий коментар. Перевіримо роботу пошуку по сайту.

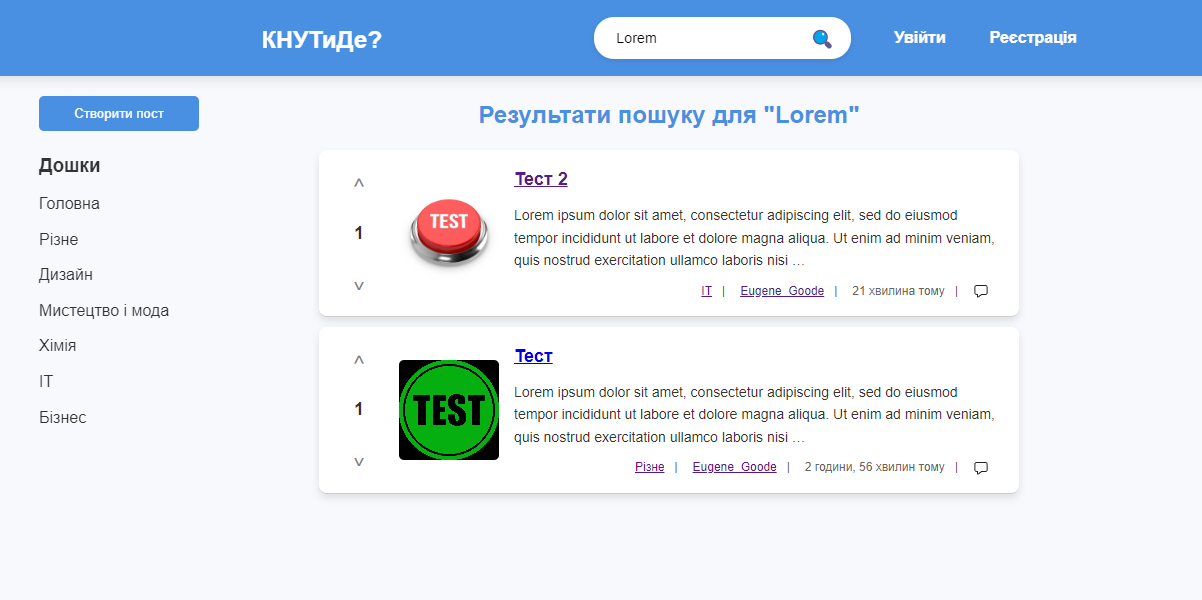


Рис.3.25 Сторінка результатів пошуку

При введенні запиту «Lorem» в поле пошуку та натисканні на кнопку пошуку відбувся перехід на сторінку результатів, яка коректно вивела усі треди, що мають в собі слово з запиту. Перевіримо сторінку профіля користувача.

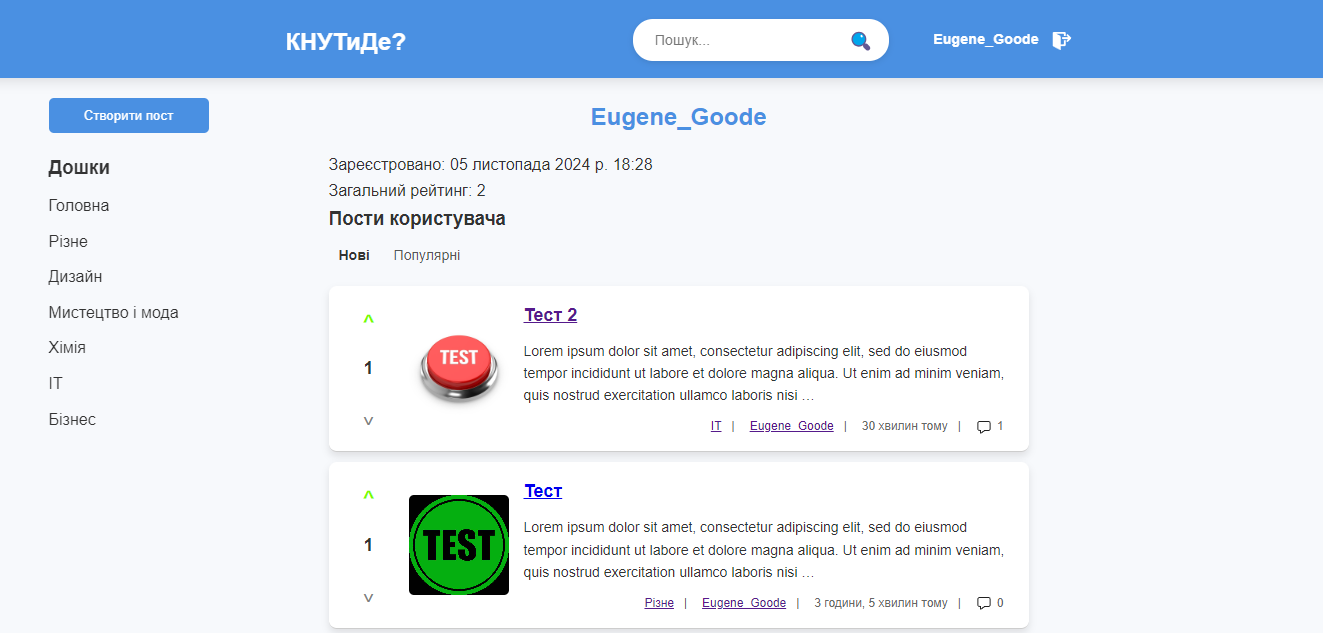


Рис.3.26. Сторінка профіля користувача

При переході по гіперпосиланню в імені автора треду відображається сторінка його профілю в ній виводяться його ім’я, дата реєстрації, усі його треди та сума рейтингів усіх його тредів. Перевіримо роботу сторінки адміністратора, додавши "/admin" до адреси головної сторінки.

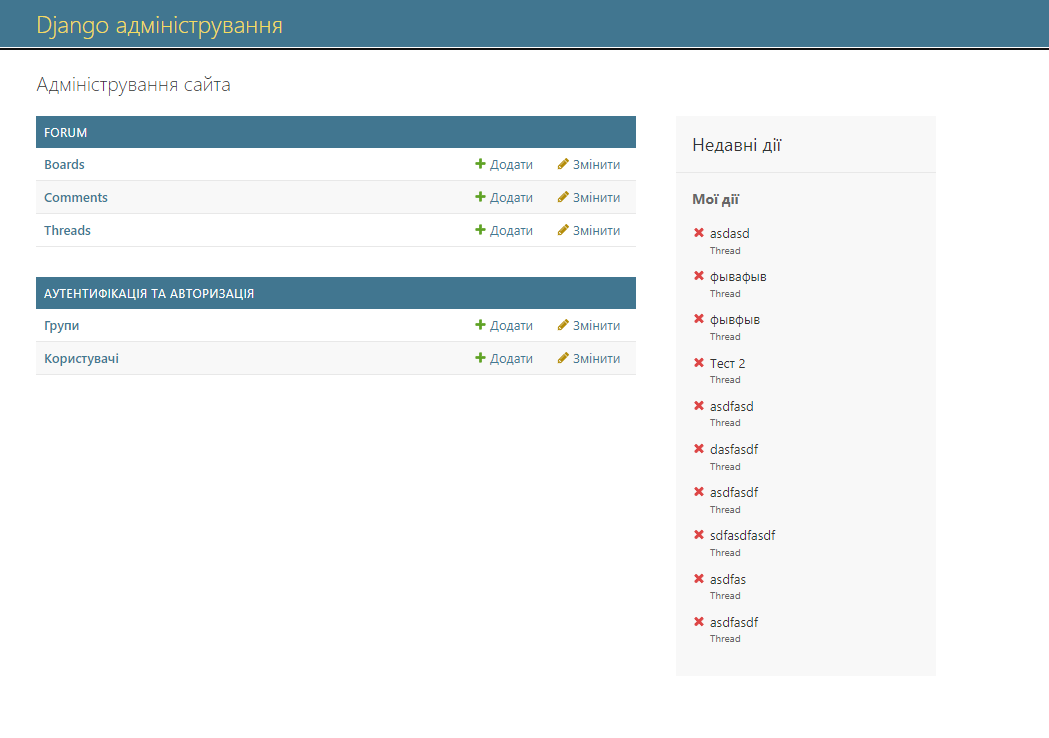


Рис. 3.27. Сторінка адміністратора сайту

Ввівши дані користувача-адміністратора можна побачити інтерфейс сторінки адміністрування. З цієї сторінки можна переглядати, редагувати та видаляти об’єкти моделей, що підключені до платформи. Це буде використовуватись модерацією платформи.

Розглянемо процес видалення об’єктів на прикладі тредів. Перейшовши на сторінку керування тредами, можна побачити таблицю, в якій є вся інформація про усі треди на сайті. Обравши деякі з них за допомогою чек-боксів, можна обрати операцію «видалити обрані треди» та підтвердити, після чого ці треди будуть видалені з сайту. Таким же чином відбувається видалення будь-яких об’єктів. Всі операції з об’єктами потім з’являються у вікні «недавні діі». Саме вікно вибору дії виглядає наступним чином:

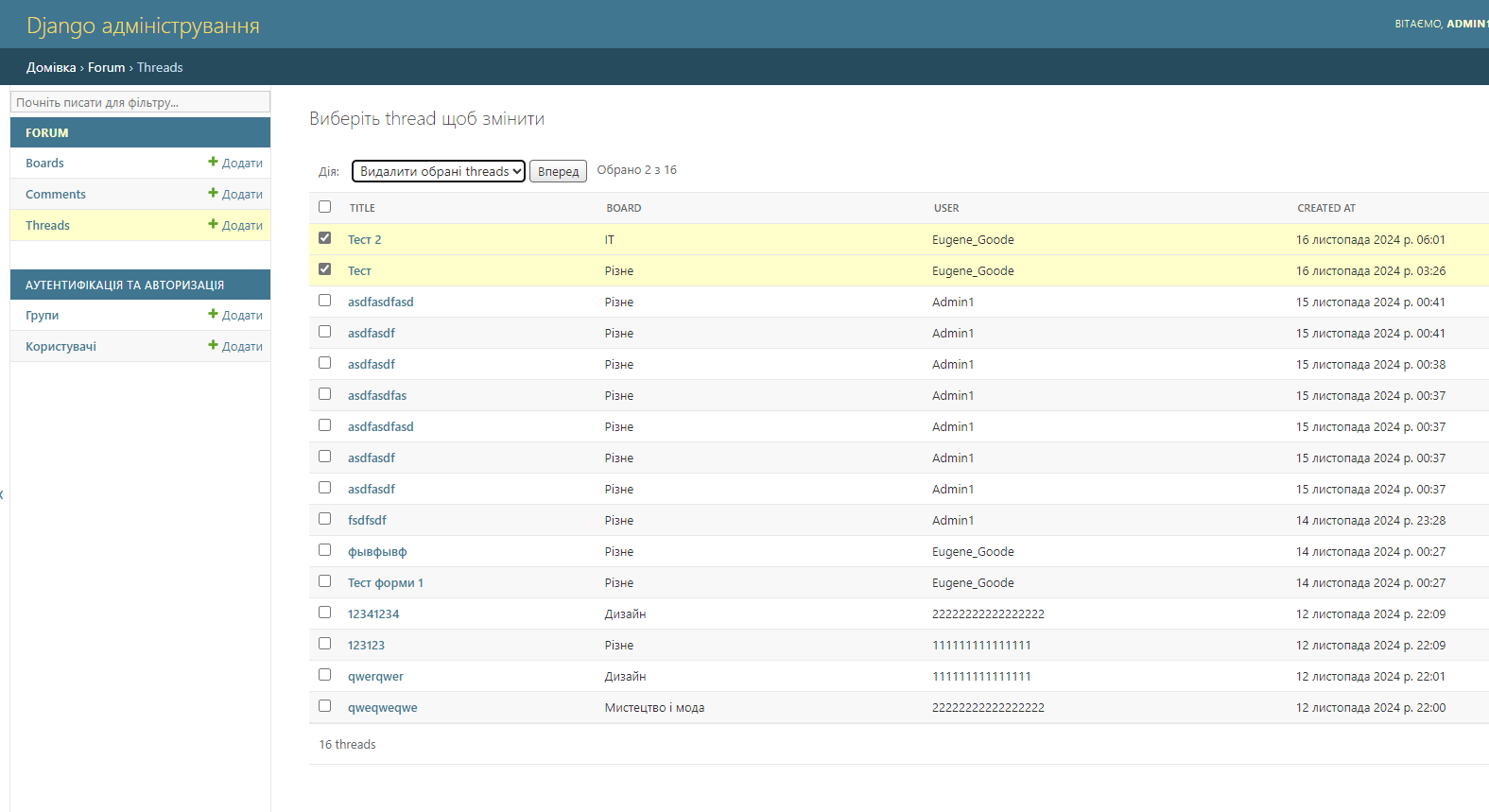


Рис. 3.28. Сторінка вибору операції над об’єктами.

Проєкт готовий до розгортання на сервері.

3.5. Розгортання проєкту на сервері

Для розгортання проєкту було обрано платформу PythonAnywhere, що є одним із найпопулярніших сервісів для хостингу Django-застосунків. Основними факторами вибору цього сервісу стали його простота у використанні, доступність безкоштовного тарифного плану, що включає всі необхідні функції для невеликих проєктів, а також вбудоване середовище з підтримкою Python, що значно спрощує процес налаштування.



Рис.3.28 Логотип PythonAnywhere

Першим кроком було створення облікового запису на PythonAnywhere. Після цього було створено новий Web App із підтримкою Django. Платформа автоматично створила базову структуру проєкту, яку згодом було замінено на власний код.

Для завантаження файлів проєкту було використано вбудований файловий менеджер PythonAnywhere. Було підготовлено архів із проєктом, який завантажено безпосередньо через веб-інтерфейс. Після розпакування архіву всі файли зайняли відповідні місця в каталозі, призначеному для веб-додатка.

Після завантаження файлів необхідно було налаштувати середовище виконання. На сервері було створено віртуальне середовище, і до нього були встановлені всі залежності з файлу requirements.txt за допомогою команди pip install -r requirements.txt.

У налаштуваннях Django було оновлено значення змінної ALLOWED\_HOSTS, щоб додати домен PythonAnywhere. Також було налаштовано статичні файли, які розміщуються на сервері окремо.

Для створення структури бази даних виконувались команди python manage.py makemigrations та python manage.py migrate. Це дозволило перенести всі моделі з проєкту до бази даних, яку пропонує PythonAnywhere.

Для коректного відображення CSS і JavaScript файлів було виконано команду python manage.py collectstatic, яка зібрала всі статичні файли в одну папку, після чого вони стали доступними для сервера.

Після завершення налаштувань веб-додаток було перезапущено через панель керування PythonAnywhere. Було проведено тестування, щоб переконатися, що всі функції працюють коректно: реєстрація користувачів, створення постів, коментування та голосування.

PythonAnywhere також надає доступ до журналів помилок і запитів, що значно полегшило процес налагодження в разі виникнення проблем.

Завдяки PythonAnywhere процес розгортання проєкту був значно спрощений, що дозволило зосередитися на доопрацюванні функціональності та дизайну форуму. Сервіс забезпечує зручне середовище для хостингу Django-додатків, включаючи можливість швидко оновлювати проєкт у разі внесення змін.

Після розгортання Django проєкту на безкоштовному аккаунті PythonAnywhere, на вкладці Web можна побачити обмежену інформацію про трафік на сайті:

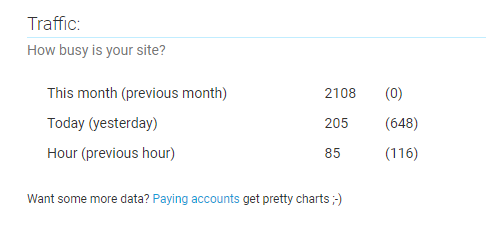


Рис.3.29. Інформація про трафік

Також варто не забувати заходити в свій обліковий запис хоча б раз на три місяці, інакше веб-додаток буде відключено.

ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі було детально розглянуто процес розробки та впровадження програмного продукту. Розпочавши з ініціалізації проєкту, було описано його архітектуру та базові компоненти, що забезпечують структурованість і гнучкість у подальшій розробці. Зокрема, було визначено модельну частину, основні маршрути й взаємодію між різними модулями системи.

На етапі реалізації серверної складової (Back end) розроблено функціонал, який забезпечує роботу з базою даних, обробку користувацьких запитів та виконання основної бізнес-логіки. Особливу увагу приділено функціям авторизації, створення й управління постами, а також системі голосування, що гарантує належний користувацький досвід.

Локальне розгортання проєкту дало змогу провести ретельне тестування системи. Це включало перевірку функціональності основних модулів, виправлення помилок і забезпечення відповідності програмного забезпечення поставленим вимогам. Тестування підтвердило стабільність роботи та коректність виконання всіх ключових операцій.

Останнім етапом стало розгортання проєкту на сервері, що дозволило зробити його доступним для кінцевих користувачів. Було обрано платформу PythonAnywhere, яка забезпечила простоту налаштування, швидке завантаження файлів і зручність у подальшому управлінні проєктом. Успішне розгортання підтвердило готовність системи до використання в реальному середовищі.

Таким чином, виконання всіх зазначених етапів дозволило створити функціональний і стабільний програмний продукт, який відповідає початковим вимогам і готовий до розширення в майбутньому.

ВИСНОВКИ

У ході виконання даного проєкту було успішно досягнуто поставлених цілей та виконано всі завдання, визначені у вступній частині.

Перший розділ звіту був присвячений аналізу важливості онлайн-комунікації та її ролі у сучасному навчальному процесі. Було проведено дослідження існуючих платформ для обміну знаннями та обговорення ідей. Це дозволило визначити основні функціональні вимоги до створюваного проєкту. У результаті було розроблено концепцію форуму, який відповідає потребам студентської аудиторії.

Другий розділ зосередився на теоретичній підготовці до реалізації проєкту. Було досліджено основні підходи до розробки веб-додатків, із яких ключовим для проєкту став MVC-патерн. Це забезпечило розподіл логіки додатка на серверну, клієнтську частини та модель даних. На основі аналізу було обрано оптимальні технології: Django для серверної складової та HTML, CSS, JavaScript для клієнтської частини.

Третій розділ описував процес практичної реалізації та розгортання проєкту. Після створення основних функціональних модулів проєкт було протестовано локально, що дозволило виявити та усунути помилки. Подальше розгортання на сервері PythonAnywhere зробило форум доступним для користувачів. Було описано етапи налаштування середовища, завантаження файлів та конфігурації бази даних. У результаті вдалося створити стабільний і функціональний веб-додаток.

У підсумку, розроблений форум є ефективним інструментом для студентів, сприяє обміну знаннями та підвищує рівень взаємодії в навчальному процесі. Виконана робота дозволила не лише поглибити знання у сфері веб-розробки, але й отримати цінний практичний досвід із використанням сучасних технологій та підходів.

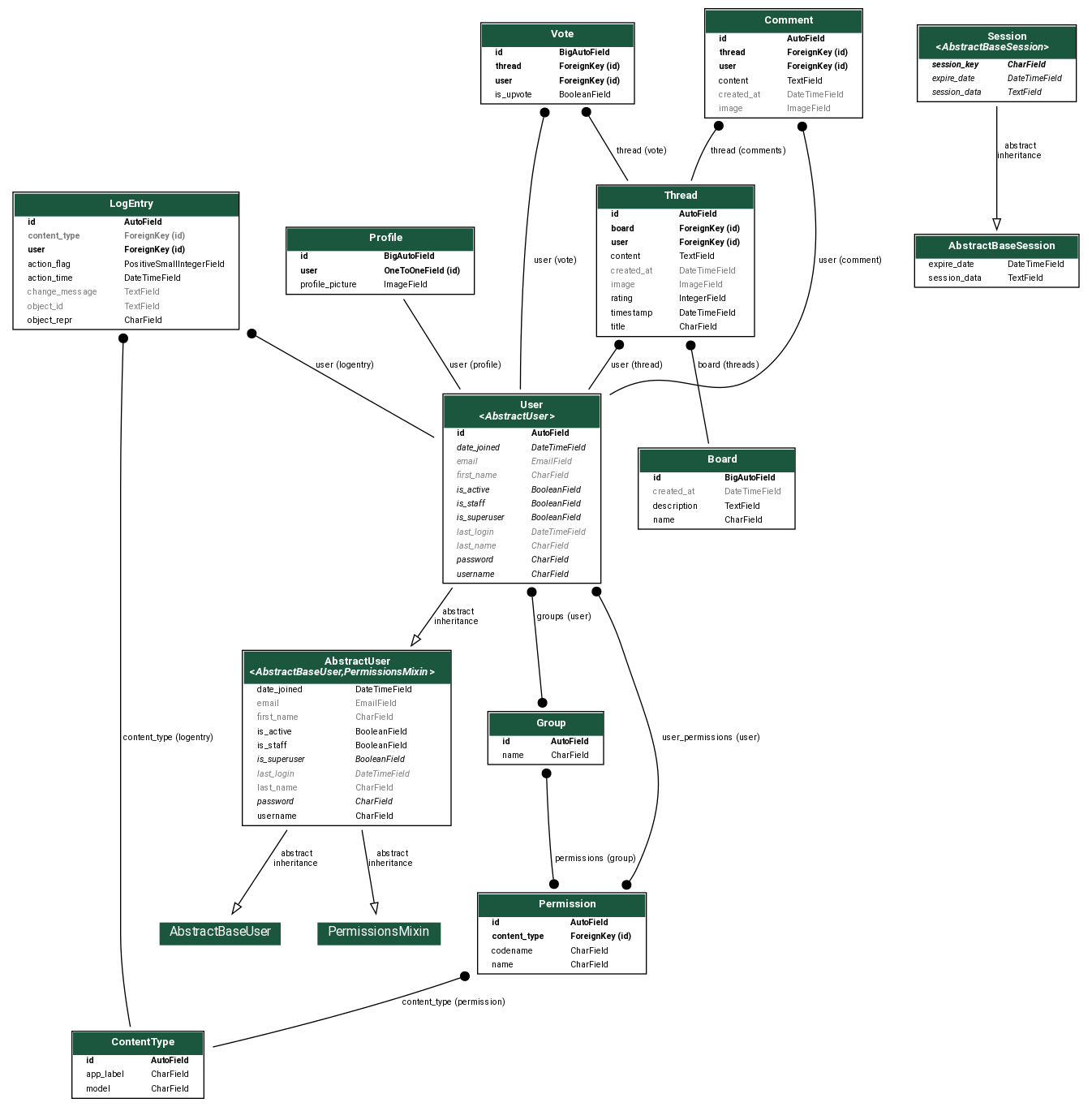
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. History of online communication [Електронний ресурс] // Geekboots. – 2018. – Режим доступу до ресурсу: <https://www.geekboots.com/story/history-of-online-communication>.
2. Moradian M. The History of Reddit [Електронний ресурс] / Mike Moradian // Honor Society. – 2020. – Режим доступу до ресурсу: <https://www.honorsociety.org/articles/history-reddit>.
3. An Introduction to Discourse [Електронний ресурс] // bitnami. – 2014. – Режим доступу до ресурсу: <https://blog.bitnami.com/2014/01/an-introduction-to-discourse.html>.
4. The Story Behind The Success Of StackOverflow [Електронний ресурс] // techwithmaddy. – 2022. – Режим доступу до ресурсу: <https://techwithmaddy.com/the-story-behind-the-success-of-stackoverflow>.
5. What is server-side rendering: definition, benefits and risks [Електронний ресурс] // solutionshub. – 2022. – Режим доступу до ресурсу: <https://solutionshub.epam.com/blog/post/what-is-server-side-rendering>.
6. Natasha Ferguson. Single Page Applications (SPA) [Електронний ресурс] / Natasha Ferguson. – 2023. – Режим доступу до ресурсу: <https://medium.com/@teamtechsis/single-page-applications-spa-48b1b845b446>.
7. Mehmet Ozkaya. Microservices Architecture [Електронний ресурс] / Mehmet Ozkaya. – 2021. – Режим доступу до ресурсу: <https://medium.com/design-microservices-architecture-with-patterns/microservices-architecture-2bec9da7d42a>.
8. Front end vs. Back end: What's The Difference? [Електронний ресурс] // Pluralsight. – 2022. – Режим доступу до ресурсу: <https://www.pluralsight.com/blog/software-development/front-end-vs-back-end>
9. Introduction to MVC (Model-View-Controller) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.inmotionhosting.com/support/edu/laravel/model-view-controller-intro/>.
10. Kerim Kara. Understanding MVC (Model-View-Controller) with C# and .NET [Електронний ресурс] / Kerim Kara – Режим доступу до ресурсу: <https://medium.com/@kerimkkara/understanding-mvc-model-view-controller-with-c-and-net-5197bbbc7ed5>.
11. Krešimir B. What are the Benefits of Ruby on Rails? After Two Decades of Programming, I Use Rails [Електронний ресурс] / Bojči Krešimir // Toptal – Режим доступу до ресурсу: <https://www.toptal.com/ruby-on-rails/after-two-decades-of-programming-i-use-rails>
12. An introduction to the Flask Python web app framework [Електронний ресурс] // opensource.com. – 2018. – Режим доступу до ресурсу: <https://opensource.com/article/18/4/flask>.
13. Artturi Jalli. What Is Laravel? [Електронний ресурс] / Artturi Jalli. – 2022. – Режим доступу до ресурсу: <https://builtin.com/software-engineering-perspectives/laravel>.
14. Assis Zang. ASP.NET Core Basics: ASP.NET Core Overview [Електронний ресурс] / Assis Zang. – 2023. – Режим доступу до ресурсу: <https://www.telerik.com/blogs/aspnet-core-basics-aspnet-core-overview>.
15. An introduction to the Django Python web app framework [Електронний ресурс] // opensource.com. – 2018. – Режим доступу до ресурсу: <https://opensource.com/article/18/8/django-framework>.
16. Complete Django History [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/complete-django-history-python/>
17. Kolowich L. C. Coding for Web Design 101: How HTML, CSS, and JavaScript Work [Електронний ресурс] / Lindsay Cox Kolowich // hubspot.com. – 2020. – Режим доступу до ресурсу: <https://blog.hubspot.com/marketing/web-design-html-css-javascript>.
18. Fundamentals of Front-End Django [Електронний ресурс] // mkdev – Режим доступу до ресурсу: <https://mkdev.me/posts/fundamentals-of-front-end-django>.

# ДОДАТКИ

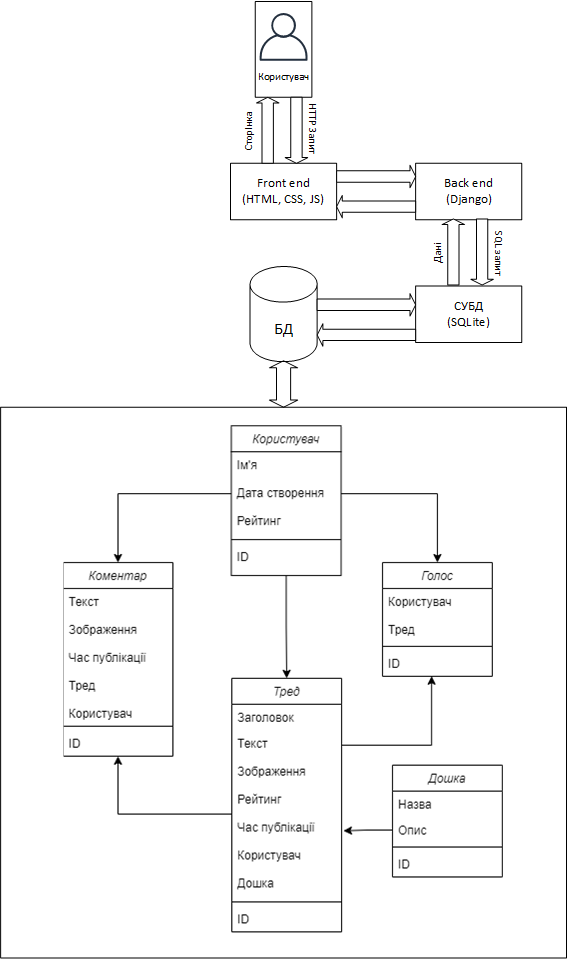
ДОДАТОК А

Діаграма класів



ДОДАТОК Б

Структурна схема платформи



## ДОДАТОК В

**Текст програмного коду**

**models.py**

from django.db import models  
from django.contrib.auth.models import User  
from django.utils import timezone  
from django.utils.timesince import timesince  
  
  
class Board(models.Model):  
 name = models.CharField(max\_length=255, unique=True)  
 description = models.TextField()  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class Thread(models.Model):  
 board = models.ForeignKey(Board, on\_delete=models.CASCADE, related\_name='threads')  
 user = models.ForeignKey(User, on\_delete=models.CASCADE)  
 title = models.CharField(max\_length=255)  
 content = models.TextField()  
 timestamp = models.DateTimeField(default=timezone.now)  
 image = models.ImageField(upload\_to='thread\_images/', blank=True, null=True)  
 rating = models.IntegerField(default=0)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 id = models.AutoField(primary\_key=True)  
  
 def update\_rating(self):  
 upvotes = Vote.objects.filter(thread=self, is\_upvote=True).count()  
 downvotes = Vote.objects.filter(thread=self, is\_upvote=False).count()  
 self.rating = upvotes - downvotes  
 self.save()  
  
 def \_\_str\_\_(self):  
 return self.title  
  
 def time\_since\_posted(self):  
 return timesince(self.timestamp)  
  
  
class Comment(models.Model):  
 thread = models.ForeignKey(Thread, on\_delete=models.CASCADE, related\_name='comments')  
 user = models.ForeignKey(User, on\_delete=models.CASCADE)  
 content = models.TextField()  
 image = models.ImageField(upload\_to='comment\_images/', blank=True, null=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 id = models.AutoField(primary\_key=True)  
  
 def \_\_str\_\_(self):  
 return f'Comment by {self.user} on {self.thread}'  
  
  
class Vote(models.Model):  
 user = models.ForeignKey(User, on\_delete=models.CASCADE)  
 thread = models.ForeignKey(Thread, on\_delete=models.CASCADE)  
 is\_upvote = models.BooleanField()  
  
 class Meta:  
 unique\_together = ('user', 'thread')  
  
  
  
class Profile(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
 profile\_picture = models.ImageField(upload\_to='profile\_pics/', default='media/Default\_pfp.svg')  
  
 def \_\_str\_\_(self):  
 return f'{self.user.username} Profile'

**views.py**

from django.contrib.auth.decorators import login\_required  
from django.contrib.auth.models import User  
from django.shortcuts import render, redirect, get\_object\_or\_404  
from django.contrib.auth import login, authenticate  
from django.contrib.auth.forms import UserCreationForm  
from django.contrib.auth.views import LoginView, LogoutView  
from django.template.loader import render\_to\_string  
from django.urls import reverse, reverse\_lazy  
from django.core.paginator import Paginator  
from django.http import JsonResponse  
from django.db.models import Count, Sum  
from .models import Profile, Thread, Vote, Comment, Board  
from .forms import ThreadForm, CommentForm  
from django.db.models import Q  
  
  
def home(request):  
 sort\_by = request.GET.get('sort', 'recency')  
 threads = sorting(sort\_by)  
 page\_number = request.GET.get('page', 1)  
  
 if request.user.is\_authenticated:  
 for thread in threads:  
 vote = Vote.objects.filter(user=request.user, thread=thread).first()  
 thread.vote\_status = 'upvoted' if vote and vote.is\_upvote else 'downvoted' if vote else 'not\_voted'  
 else:  
 for thread in threads:  
 thread.vote\_status = 'not\_voted'  
  
 paginator = Paginator(threads, 10)  
 threads\_page = paginator.get\_page(page\_number)  
  
 if request.headers.get('x-requested-with') == 'XMLHttpRequest':  
 threads\_html = render\_to\_string('forum/partials/thread\_list.html', {'threads': threads\_page}, request=request)  
 return JsonResponse({  
 'threads\_html': threads\_html,  
 'has\_next': threads\_page.has\_next(),  
 })  
  
 return render(request, 'forum/main.html', {'threads': threads\_page, 'sort\_by': sort\_by, 'boards': Board.objects.all()})  
  
  
def register(request):  
 if request.method == 'POST':  
 form = UserCreationForm(request.POST)  
 if form.is\_valid():  
 user = form.save()  
 Profile.objects.create(user=user)  
 login(request, user)   
 return redirect(reverse('home'))  
 else:  
 form = UserCreationForm()  
 return render(request, 'registration/register.html', {'form': form, 'boards': Board.objects.all()})  
  
  
  
def rules(request):  
 return render(request, 'forum/rules.html', {'boards': Board.objects.all()})  
  
  
class CustomLoginView(LoginView):  
 template\_name = 'registration/login.html'  
  
  
  
class CustomLogoutView(LogoutView):  
 template\_name = 'registration/logged\_out.html'  
 next\_page = reverse\_lazy('home')  
  
  
  
@login\_required  
def profile(request, username):  
 profile\_user = get\_object\_or\_404(User, username=username)  
 sort\_by = request.GET.get('sort', 'recency')  
 threads = sorting(sort\_by, user=profile\_user)  
  
   
 if request.user.is\_authenticated:  
 for thread in threads:  
 vote = Vote.objects.filter(user=request.user, thread=thread).first()  
 thread.vote\_status = 'upvoted' if vote and vote.is\_upvote else 'downvoted' if vote else 'not\_voted'  
 else:  
 for thread in threads:  
 thread.vote\_status = 'not\_voted'  
  
 total\_rating = threads.aggregate(total\_rating=Sum('rating'))['total\_rating'] or 0  
 paginator = Paginator(threads, 10)  
 page\_number = request.GET.get('page', 1)  
 threads\_page = paginator.get\_page(page\_number)  
  
 if request.headers.get('x-requested-with') == 'XMLHttpRequest':  
 threads\_html = render\_to\_string('forum/partials/thread\_list.html', {'threads': threads\_page})  
 return JsonResponse({'threads\_html': threads\_html, 'has\_next': threads\_page.has\_next()})  
  
 return render(request, 'forum/profile.html', {  
 'profile\_user': profile\_user,  
 'total\_rating': total\_rating,  
 'threads\_page': threads\_page,  
 'boards': Board.objects.all(),  
 'sort\_by': sort\_by  
 })  
  
  
def board\_view(request, board\_name):  
 board = get\_object\_or\_404(Board, name=board\_name)  
 sort\_by = request.GET.get('sort', 'recency')  
 threads = sorting(sort\_by, board=board)  
  
   
 if request.user.is\_authenticated:  
 for thread in threads:  
 vote = Vote.objects.filter(user=request.user, thread=thread).first()  
 thread.vote\_status = 'upvoted' if vote and vote.is\_upvote else 'downvoted' if vote else 'not\_voted'  
 else:  
 for thread in threads:  
 thread.vote\_status = 'not\_voted'  
  
 paginator = Paginator(threads, 10)  
 page\_number = request.GET.get('page', 1)  
 threads\_page = paginator.get\_page(page\_number)  
  
 if request.headers.get('x-requested-with') == 'XMLHttpRequest':  
 threads\_html = render\_to\_string('forum/partials/thread\_list.html', {'threads': threads\_page})  
 return JsonResponse({'threads\_html': threads\_html, 'has\_next': threads\_page.has\_next()})  
  
 return render(request, 'forum/board.html', {  
 'board': board,  
 'threads\_page': threads\_page,  
 'boards': Board.objects.all(),  
 'sort\_by': sort\_by  
 })  
  
@login\_required  
def upvote\_thread(request, thread\_id):  
 thread = get\_object\_or\_404(Thread, id=thread\_id)  
 vote = Vote.objects.filter(user=request.user, thread=thread).first()  
  
 if vote:  
 if vote.is\_upvote:  
 vote.delete()  
 thread.rating -= 1  
 else:  
 vote.is\_upvote = True  
 vote.save()  
 thread.rating += 2  
 else:  
 Vote.objects.create(user=request.user, thread=thread, is\_upvote=True)  
 thread.rating += 1  
  
 thread.save()  
  
 next\_url = request.GET.get('next', reverse('thread\_detail', args=[thread\_id]))  
 sort\_option = request.GET.get('sort')  
  
 if sort\_option:  
 next\_url += f'?sort={sort\_option}'  
  
 return redirect(next\_url)  
  
  
@login\_required  
def downvote\_thread(request, thread\_id):  
 thread = get\_object\_or\_404(Thread, id=thread\_id)  
 vote = Vote.objects.filter(user=request.user, thread=thread).first()  
  
 if vote:  
 if not vote.is\_upvote:  
 vote.delete()  
 thread.rating += 1  
 else:  
 vote.is\_upvote = False  
 vote.save()  
 thread.rating -= 2  
 else:  
 Vote.objects.create(user=request.user, thread=thread, is\_upvote=False)  
 thread.rating -= 1  
  
 thread.save()  
  
 next\_url = request.GET.get('next', reverse('thread\_detail', args=[thread\_id]))  
 sort\_option = request.GET.get('sort')  
  
 if sort\_option:  
 next\_url += f'?sort={sort\_option}'  
  
 return redirect(next\_url)  
  
def thread\_detail(request, thread\_id):  
 thread = get\_object\_or\_404(Thread, id=thread\_id)  
 comments = thread.comments.all().order\_by('-created\_at')  
  
 if request.user.is\_authenticated:  
 vote = Vote.objects.filter(user=request.user, thread=thread).first()  
 if vote:  
 thread.vote\_status = 'upvoted' if vote.is\_upvote else 'downvoted'  
 else:  
 thread.vote\_status = 'not\_voted'  
 else:  
 thread.vote\_status = 'not\_voted'  
  
 if request.method == 'POST':  
 form = CommentForm(request.POST, request.FILES)  
 if form.is\_valid():  
 comment = form.save(commit=False)  
 comment.thread = thread  
 comment.user = request.user  
 comment.save()  
 return redirect('thread\_detail', thread\_id=thread\_id)  
 else:  
 form = CommentForm()  
  
 context = {  
 'thread': thread,  
 'comments': comments,  
 'form': form,  
 'boards': Board.objects.all()  
 }  
 return render(request, 'forum/thread\_detail.html', context)  
  
  
@login\_required  
def create\_post(request, board\_name=None):  
 if request.method == 'POST':  
 form = ThreadForm(request.POST, request.FILES)  
 if form.is\_valid():  
 thread = form.save(commit=False)  
 thread.user = request.user  
 board\_id = request.POST.get('board')  
 thread.board = Board.objects.get(id=board\_id)  
  
 thread.save()  
 return redirect('home')  
 else:  
 form = ThreadForm()  
  
 boards = Board.objects.all()  
 return render(request, 'forum/create\_post.html', {'form': form, 'boards': boards})  
  
def search(request):  
 query = request.GET.get('q', '').strip()  
 results = []  
  
 if query:  
 results = Thread.objects.filter(  
 Q(title\_\_icontains=query) | Q(content\_\_icontains=query) | Q(user\_\_username\_\_icontains=query)  
 ).order\_by('-timestamp')  
  
   
 paginator = Paginator(results, 10)  
 page\_number = request.GET.get('page', 1)  
 results\_page = paginator.get\_page(page\_number)  
  
 return render(request, 'forum/search\_results.html', {  
 'query': query,  
 'results': results\_page,  
 'boards': Board.objects.all(),  
 })  
  
  
def sorting(sort\_by, user=None, board=None):  
 queryset = Thread.objects.all()  
 if user:  
 queryset = queryset.filter(user=user)  
 if board:  
 queryset = queryset.filter(board=board)  
  
 if sort\_by == 'rating':  
 return queryset.annotate(comment\_count=Count('comments')).order\_by('-rating', '-timestamp')  
 return queryset.annotate(comment\_count=Count('comments')).order\_by('-timestamp')

**forms.py**

from django import forms  
from .models import Comment, Thread  
from PIL import Image  
from io import BytesIO  
from django.core.files.uploadedfile import InMemoryUploadedFile  
import sys  
  
MAX\_IMAGE\_SIZE\_MB = 2  
  
class ImageCompressionMixin:  
 def compress\_image(self, image):  
 if image.size > MAX\_IMAGE\_SIZE\_MB \* 1024 \* 1024:  
 raise forms.ValidationError(f"Зображення не може бути більше {MAX\_IMAGE\_SIZE\_MB} MB.")  
  
 img = Image.open(image)  
 if img.mode != 'RGB':  
 img = img.convert('RGB')  
  
 output = BytesIO()  
 img.save(output, format='JPEG', optimize=True, quality=70)  
 output.seek(0)  
 return InMemoryUploadedFile(output, 'ImageField', image.name, 'image/jpeg', sys.getsizeof(output), None)  
  
class CommentForm(forms.ModelForm, ImageCompressionMixin):  
 class Meta:  
 model = Comment  
 fields = ['content', 'image']  
  
 def clean\_image(self):  
 image = self.cleaned\_data.get('image')  
 if image:  
 return self.compress\_image(image)  
 return image  
  
class ThreadForm(forms.ModelForm, ImageCompressionMixin):  
 class Meta:  
 model = Thread  
 fields = ['title', 'content', 'image']  
  
 def clean\_image(self):  
 image = self.cleaned\_data.get('image')  
 if image:  
 return self.compress\_image(image)  
 return image

**urls.py**

from django.conf import settings  
from django.conf.urls.static import static  
from . import views  
from django.urls import path  
from .views import register, CustomLoginView, CustomLogoutView  
from django.contrib.auth import views as auth\_views  
  
urlpatterns = [  
 path('', views.home, name='home'),  
 path('register/', register, name='register'),  
 path('thread/<int:thread\_id>/upvote/', views.upvote\_thread, name='upvote\_thread'),  
 path('thread/<int:thread\_id>/downvote/', views.downvote\_thread, name='downvote\_thread'),  
 path('login/', CustomLoginView.as\_view(), name='login'),  
 path('logout/', CustomLogoutView.as\_view(), name='logout'),  
 path('profile/<str:username>/', views.profile, name='profile'),  
 path('thread/<int:thread\_id>/', views.thread\_detail, name='thread\_detail'),  
 path('board/<str:board\_name>/', views.board\_view, name='board\_view'),  
 path('create\_post/', views.create\_post, name='create\_post'),  
 path('create\_post/<str:board\_name>/', views.create\_post, name='create\_post'),  
 path('search/', views.search, name='search')  
]  
  
if settings.DEBUG:  
 urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**apps.py**

from django.apps import AppConfig  
  
  
class ForumConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'forum'

**admin.py**

from django.contrib import admin  
from .models import Board, Thread, Comment  
  
@admin.register(Board)  
class BoardAdmin(admin.ModelAdmin):  
 list\_display = ('name', 'description', 'created\_at')  
  
@admin.register(Thread)  
class ThreadAdmin(admin.ModelAdmin):  
 list\_display = ('title', 'board', 'user', 'created\_at')  
 readonly\_fields = ('rating',)  
  
@admin.register(Comment)  
class CommentAdmin(admin.ModelAdmin):  
 list\_display = ('thread', 'user', 'created\_at')

**base.html**

<!DOCTYPE html>  
{% load static %}  
{% load vote\_filters %}  
<html lang="uk">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>КНУТиДе?</title>  
 <link rel="stylesheet" href="{% static 'css/styles.css' %}">  
</head>  
<body>  
 <header class="header">  
 <div class="container">  
 <h1><a href="{% url 'home' %}">КНУТиДе?</a></h1>  
 <nav>  
 <ul class="nav-links">  
 <li>  
 <form method="GET" action="{% url 'search' %}" class="search-form">  
 <input type="text" name="q" placeholder="Пошук..." value="{{ request.GET.q|default\_if\_none:'' }}">  
 <button type="submit">🔍</button>  
 </form>  
 </li>  
 {% if user.is\_authenticated %}  
 <li>  
 <div class="profile-container">  
 <a href="{% url 'profile' user.username %}" class="profile-container">  
 <span class="username">{{ user.username }}</span> *<!-- Display the username -->* </a>  
 <form action="{% url 'logout' %}" method="post" class="logout-form">  
 {% csrf\_token %}  
 <button type="submit" class="logout-button">  
 <img src="{% static 'media/logout.svg' %}" alt="Logout Icon" class="logout-icon">  
 </button>  
 </form>  
 </div>  
 </li>  
 {% else %}  
 <li><a href="{% url 'login' %}">Увійти</a></li>  
 <li><a href="{% url 'register' %}">Реєстрація</a></li>  
 {% endif %}  
 </ul>  
 </nav>  
 </div>  
 </header>  
 <div class="spacer"> </div>  
 <main>  
 {% block content %}  
 {% endblock %}  
 </main>  
 </div>  
 <button id="scroll-to-top" title="До гори">  
 <img src="{% static 'media/up-arrow-button.svg' %}" alt="Scroll to Top">  
 </button>  
 <script src="{% static 'js/main.js' %}"></script>  
</body>  
</html>

**main.html**

{% extends "base.html" %}  
{% load vote\_filters %}  
{% load static %}  
{% block content %}  
  
<div class="main-grid">  
 <div class="sidebar">  
 <button id="create-post-btn" class="create-post-btn">Створити пост</button>  
 <h3>Дошки</h3>  
 <ul class="board-list">  
 <li>  
 <a href="{% url 'home' %}">Головна</a>  
 </li>  
 {% for board in boards %}  
 <li>  
 <a href="{% url 'board\_view' board.name %}">{{ board.name }}</a>  
 </li>  
 {% endfor %}  
 </ul>  
 </div>  
 <div class="main-container">  
 <div class="sorting-options">  
 <div class="left-options">  
 <a href="?sort=recency" class="{% if sort\_by == 'recency' %}active{% endif %}">Нові</a>  
 <a href="?sort=rating" class="{% if sort\_by == 'rating' %}active{% endif %}">Популярні</a>  
 </div>  
 </div>  
 {% include 'forum/partials/post\_creation.html' with boards=boards %}  
 <div id="thread-container">  
 {% include 'forum/partials/thread\_list.html' with threads=threads %}  
 </div>  
 </div>  
</div>  
  
<script src="{% static 'js/infinite\_scroll.js' %}"></script>  
<script src="{% static 'js/main.js' %}"></script>  
  
{% endblock %}

**thread\_list.html**

{% load vote\_filters %}  
{% load static %}  
{% for thread in threads %}  
<div class="thread-box">  
 <div class="vote-controls">  
 <a href="{% url 'upvote\_thread' thread.id %}?next={{ request.path }}&sort={{ sort\_by }}"  
 class="vote-button {% if thread.vote\_status == 'upvoted' %}active-upvote{% endif %}">⌃</a>  
 <span class="rating">{{ thread.rating|shorten\_rating }}</span>  
 <a href="{% url 'downvote\_thread' thread.id %}?next={{ request.path }}&sort={{ sort\_by }}"  
 class="vote-button {% if thread.vote\_status == 'downvoted' %}active-downvote{% endif %}">⌄</a>  
 </div>  
 <div class="thread-content-container">  
 {% if thread.image %}  
 <div class="thread-image">  
 <img src="{{ thread.image.url }}" alt="Post Image">  
 </div>  
 {% endif %}  
 <div class="thread-main">  
 *<!-- Post Title -->* <div class="thread-title">  
 <a href="{% url 'thread\_detail' thread.id %}">{{ thread.title }}</a>  
 </div>  
 <div class="thread-content">  
 {{ thread.content|truncatewords:30 }}  
 </div>  
 <div class="thread-footer">  
 <div class="thread-info">  
 <span class="thread-board"><a href="{% url 'board\_view' thread.board.name %}">{{ thread.board.name }}</a></span>  
 <span class="thread-user"><a href="{% url 'profile' thread.user.username %}">{{ thread.user.username }}</a></span>  
 <span class="thread-timestamp">{{ thread.time\_since\_posted }} тому</span>  
 <span class="thread-comments">  
 <img src="{% static 'media/comment.svg' %}" alt="Comments Icon" class="comments-icon">  
 {{ thread.comment\_count }}  
 </span>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
{% endfor %}

**post\_creation.html**

*<!-- Post Creation Modal -->*<div id="post-modal" class="post-modal hidden">  
 <div class="post-modal-content">  
 <h2>Створити новий пост</h2>  
 <form action="{% url 'create\_post' %}" method="post" enctype="multipart/form-data">  
 {% csrf\_token %}  
 <div class="form-group">  
 <label for="post-board">Оберіть дошку</label>  
 <select id="post-board" name="board" required>  
 {% for board in boards %}  
 <option value="{{ board.id }}">{{ board.name }}</option>  
 {% endfor %}  
 </select>  
 </div>  
 <div class="form-group">  
 <label for="post-title">Заголовок</label>  
 <input type="text" id="post-title" name="title" required>  
 </div>  
 <div class="form-group">  
 <label for="post-content">Текст</label>  
 <textarea id="post-content" name="content" rows="5" required></textarea>  
 </div>  
 <div class="form-group">  
 <label for="post-image">Прикріпити зображення</label>  
 <input type="file" id="post-image" name="image">  
 </div>  
 <div class="form-actions">  
 <button type="submit" class="submit-btn">Опублікувати</button>  
 <button type="button" id="cancel-post-btn" class="cancel-btn">Скасувати</button>  
 </div>  
 </form>  
 </div>  
</div>

**board.html**

{% extends "base.html" %}  
{% load vote\_filters %}  
{% load static %}  
{% block content %}  
  
<div class="main-grid">  
 <div class="sidebar">  
 <button id="create-post-btn" class="create-post-btn">Створити пост</button>  
 {% include 'forum/partials/post\_creation.html' with boards=boards %}  
 <h3>Дошки</h3>  
 <ul class="board-list">  
 <li>  
 <a href="{% url 'home' %}">Головна</a>  
 </li>  
 {% for board in boards %}  
 <li>  
 <a href="{% url 'board\_view' board.name %}">{{ board.name }}</a>  
 </li>  
 {% endfor %}  
 </ul>  
 </div>  
 <div class="main-container">  
  
  
 <div class="board-header">  
 <h2>{{ board.name }}</h2>  
 </div>  
  
<div class="sorting-options">  
 <div class="left-options">  
 <a href="?sort=recency" class="{% if sort\_by == 'recency' %}active{% endif %}">Нові</a>  
 <a href="?sort=rating" class="{% if sort\_by == 'rating' %}active{% endif %}">Популярні</a>  
 </div>  
</div>  
  
 <div id="thread-container">  
 {% include 'forum/partials/thread\_list.html' with threads=threads\_page %}  
 </div>  
 </div>  
</div>  
  
  
<script src="{% static 'js/infinite\_scroll.js' %}"></script>  
<script src="{% static 'js/main.js' %}"></script>  
  
{% endblock %}

**profile.html**

{% extends "base.html" %}  
{% load vote\_filters %}  
{% load static %}  
{% block content %}  
  
<div class="main-grid">  
 <div class="sidebar">  
 <button id="create-post-btn" class="create-post-btn">Створити пост</button>  
 <h3>Дошки</h3>  
 <ul class="board-list">  
 <li>  
 <a href="{% url 'home' %}">Головна</a>  
 </li>  
 {% for board in boards %}  
 <li>  
 <a href="{% url 'board\_view' board.name %}">{{ board.name }}</a>  
 </li>  
 {% endfor %}  
 </ul>  
 </div>  
 <div class="main-container">  
 {% include 'forum/partials/post\_creation.html' with boards=boards %}  
 <div class="profile-header">  
 <h2>{{ profile\_user.username }}</h2>  
 <p>Зареєстровано: {{ profile\_user.date\_joined }}</p>  
 <p>Загальний рейтинг: {{ total\_rating }}</p>  
 </div>  
  
 <h3>Пости користувача</h3>  
 <div class="sorting-options">  
 <div class="left-options">  
 <a href="?sort=recency" class="{% if sort\_by == 'recency' %}active{% endif %}">Нові</a>  
 <a href="?sort=rating" class="{% if sort\_by == 'rating' %}active{% endif %}">Популярні</a>  
 </div>  
 </div>  
 <div id="thread-container">  
 {% include 'forum/partials/thread\_list.html' with threads=threads\_page %}  
 </div>  
 </div>  
</div>  
<script src="{% static 'js/infinite\_scroll.js' %}"></script>  
{% endblock %}

**search\_result.html**

{% extends 'base.html' %}  
  
{% block content %}  
<div class="main-grid">  
 <div class="sidebar">  
 <button id="create-post-btn" class="create-post-btn">Створити пост</button>  
 <h3>Дошки</h3>  
 <ul class="board-list">  
 <li>  
 <a href="{% url 'home' %}">Головна</a>  
 </li>  
 {% for board in boards %}  
 <li>  
 <a href="{% url 'board\_view' board.name %}">{{ board.name }}</a>  
 </li>  
 {% endfor %}  
 </ul>  
 </div>  
 <div class="main-container">  
 <h2>Результати пошуку для "{{ query }}"</h2>  
  
 {% include 'forum/partials/post\_creation.html' with boards=boards %}  
 {% if results %}  
 <div id="thread-container">  
 {% include 'forum/partials/thread\_list.html' with threads=results %}  
 </div>  
 {% else %}  
 <p>Немає результатів за запитом "{{ query }}".</p>  
 {% endif %}  
 </div>  
</div>  
{% endblock %}

**thread\_detail.html**

{% extends "base.html" %}  
  
{% block content %}  
<div class="main-grid">  
 <div class="sidebar">  
 <button id="create-post-btn" class="create-post-btn">Створити пост</button>  
 <h3>Дошки</h3>  
 <ul class="board-list">  
 <li>  
 <a href="{% url 'home' %}">Головна</a>  
 </li>  
 {% for board in boards %}  
 <li>  
 <a href="{% url 'board\_view' board.name %}">{{ board.name }}</a>  
 </li>  
 {% endfor %}  
 </ul>  
 </div>  
 <div class="main-container">  
 {% include 'forum/partials/post\_creation.html' with boards=boards %}  
 <div class="thread-detail-wrapper">  
 *<!-- Thread Detail Section -->* <div class="thread-detail-box">  
 <div class="thread-detail-header">  
 <h2>{{ thread.title }}</h2>  
 <p> <a href="{% url 'board\_view' thread.board.name %}">{{ thread.board.name }}</a></p>  
 <p> <a href="{% url 'profile' thread.user.username %}">{{ thread.user.username }}</a> | {{ thread.time\_since\_posted }} тому</p>  
 </div>  
  
 <div class="thread-detail-content">  
 <p>{{ thread.content }}</p>  
 {% if thread.image %}  
 <div class="thread-detail-image">  
 <img src="{{ thread.image.url }}" alt="Thread Image">  
 </div>  
 {% endif %}  
 </div>  
  
 *<!-- Rating Controls -->* <div class="thread-detail-rating">  
 <a href="{% url 'upvote\_thread' thread.id %}?next={{ request.path }}" class="vote-button {% if thread.vote\_status == 'upvoted' %}active-upvote{% endif %}">⌃</a>  
 <span class="rating">{{ thread.rating }}</span>  
 <a href="{% url 'downvote\_thread' thread.id %}?next={{ request.path }}" class="vote-button {% if thread.vote\_status == 'downvoted' %}active-downvote{% endif %}">⌄</a>  
 </div>  
 </div>  
  
 <hr>  
  
 *<!-- Comments Section -->* <h3>Коментарі</h3>  
 <div class="comments-section">  
 {% for comment in comments %}  
 <div class="comment-box">  
 <p><strong>{{ comment.user.username }}</strong> | {{ comment.created\_at }}</p>  
 <p>{{ comment.content }}</p>  
 {% if comment.image %}  
 <div class="comment-image">  
 <img src="{{ comment.image.url }}" alt="Comment Image">  
 </div>  
 {% endif %}  
 </div>  
 {% empty %}  
 <p>Коментарів поки що немає, будьте першими</p>  
 {% endfor %}  
 </div>  
  
 *<!-- Comment Form -->* <div class="comment-form">  
 <h4>Залиште коментар</h4>  
 <form method="post" enctype="multipart/form-data">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit" class="button">Опублікувати</button>  
 </form>  
 </div>  
 </div>  
 </div>  
</div>  
{% endblock %}

**login.html**

{% extends "base.html" %}  
  
{% block content %}  
 <h2>Увійти</h2>  
 <form method="post">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Увійти</button>  
 </form>  
 <p>Немає акаунта? <a href="{% url 'register' %}">Зареєструватися</a></p>  
  
{% endblock %}

**register.html**

{% extends "base.html" %}  
  
{% block content %}  
 <h2>Зареєструватися</h2>  
 <form method="post">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Зареєструватися</button>  
 </form>  
 <p>Вже маєте акаунт? <a href="{% url 'login' %}">Увійти</a></p>  
{% endblock %}

**infinite\_scroll.js**

let page = 2;  
let loading = false;  
  
const loadMore = () => {  
 if (loading) return;  
 loading = true;  
 const urlParams = new URLSearchParams(window.location.search);  
 urlParams.set('page', page);  
 const url = `${window.location.pathname}?${urlParams.toString()}`;  
 console.log(`Fetching: ${url}`);  
 fetch(url, {  
 headers: {  
 'X-Requested-With': 'XMLHttpRequest'  
 }  
 })  
 .then(response => {  
 if (!response.ok) throw new Error(`HTTP error! Status: ${response.status}`);  
 return response.json();  
 })  
 .then(data => {  
 console.log('Response data:', data.threads\_html);  
  
 if (data.threads\_html.trim() === '') {  
 console.log('Empty threads\_html received!');  
 } else {  
 document.querySelector('#thread-container').insertAdjacentHTML('beforeend', data.threads\_html);  
 }  
  
 if (!data.has\_next) {  
 window.removeEventListener('scroll', handleScroll);  
 console.log('No more threads to load.');  
 } else {  
 page++;  
 }  
 loading = false;  
})  
 .catch(error => {  
 console.error('Error fetching more threads:', error);  
 loading = false;  
 });  
};  
  
const handleScroll = () => {  
 if (window.innerHeight + window.scrollY >= document.body.offsetHeight - 200) {  
 console.log('Reached bottom, loading more...');  
 loadMore();  
 }  
};  
  
window.addEventListener('scroll', handleScroll);

**main.js**

document.getElementById('create-post-btn').addEventListener('click', function() {  
 document.getElementById('post-modal').classList.remove('hidden');  
});  
  
document.getElementById('cancel-post-btn').addEventListener('click', function() {  
 document.getElementById('post-modal').classList.add('hidden');  
});  
  
window.addEventListener('click', function(event) {  
 const modal = document.getElementById('post-modal');  
 if (event.target === modal) {  
 modal.classList.add('hidden');  
 }  
});  
  
document.addEventListener('DOMContentLoaded', function () {  
 const scrollToTopBtn = document.getElementById('scroll-to-top');  
  
 window.addEventListener('scroll', () => {  
 if (window.scrollY > 200) {  
 scrollToTopBtn.style.display = 'block';  
 } else {  
 scrollToTopBtn.style.display = 'none';  
 }  
 });  
  
 scrollToTopBtn.addEventListener('click', () => {  
 window.scrollTo({  
 top: 0,  
 behavior: 'smooth'  
 });  
 });  
});

**styles.css**

\* {  
 margin: 0;  
 padding: 0;  
 box-sizing: border-box;  
}  
  
body {  
 font-family: Arial, sans-serif;  
 background-color: #f7f9fc;  
 color: #333;  
 line-height: 1.6;  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
  
.header {  
 position: fixed;  
 background-color: #4a90e2;  
 color: #fff;  
 width: 100%;  
 height: 80px;  
 padding: 1rem 0;  
 box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);  
 z-index: 1000; }  
  
.header .container {  
 display: flex;  
 align-items: center;  
 justify-content: space-evenly;  
 margin: 0 auto;  
 max-width: 1200px;  
 width: 100%;  
}  
  
.header h1 a {  
 color: #fff;  
 text-decoration: none;  
 font-size: 1.5rem;  
 font-weight: bold;  
}  
  
.nav-links {  
 list-style: none;  
 display: flex;  
 gap: 1.5rem;  
 margin: 0;  
 padding: 0;  
}  
  
.nav-links li {  
 display: flex;  
 align-items: center;  
}  
  
.search-form {  
 display: flex;  
 align-items: center;  
 background-color: #fff;  
 border-radius: 20px;  
 padding: 5px 10px;  
 box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);  
}  
  
.search-form input[type="text"] {  
 border: none;  
 padding: 0.5rem 0.75rem;  
 width: 200px;  
 font-size: 0.9rem;  
 border-radius: 20px;  
 outline: none;  
}  
  
.search-form button {  
 background: none;  
 border: none;  
 font-size: 1.2rem;  
 cursor: pointer;  
 padding: 0 5px;  
 color: #4a90e2;  
}  
  
.profile-container {  
 display: flex;  
 align-items: center;  
 gap: 0.5rem;  
}  
  
.profile-pic {  
 width: 35px;  
 height: 35px;  
 border-radius: 50%;  
 background-color: #fff;  
}  
  
.username {  
 color: #fff;  
 font-size: 0.9rem;  
 font-weight: bold;  
}  
  
.logout-form {  
 margin-left: 0.5rem;  
}  
  
.logout-button {  
 background: none;  
 border: none;  
 cursor: pointer;  
}  
  
.logout-icon {  
 width: 20px;  
 height: 20px;  
}  
  
@media (max-width: 768px) {  
 .header .container {  
 flex-direction: column;  
 align-items: flex-start;  
 height: auto;  
 }  
  
 .nav-links {  
 flex-direction: column;  
 gap: 1rem;  
 margin-top: 0.5rem;  
 }  
  
 .search-form input[type="text"] {  
 width: 150px;  
 }  
}  
.spacer {  
 margin-top: 100px;  
}  
  
.sorting-options {  
 display: flex;  
 flex-direction: row;  
 margin-top: 10px;  
 margin-bottom: 20px;  
 font-size: 14px;  
 align-items: center;  
 justify-content: space-between;  
}  
  
.sorting-options a {  
 text-decoration: none;  
 margin: 0 10px;  
 color: #555;  
}  
  
.sorting-options a.active {  
 font-weight: bold;  
 color: #333;  
}  
  
.nav-links {  
 list-style: none;  
 display: flex;  
 justify-content: flex-end;  
}  
  
.nav-links li {  
 margin-left: 20px;  
}  
  
.nav-links a {  
 color: #fff;  
 text-decoration: none;  
 font-weight: bold;  
}  
  
.nav-links a:hover {  
 text-decoration: underline;  
}  
  
.main-grid {  
 display: grid;  
 grid-template-columns: 1fr 2fr 1fr;  
 width: 1300px;  
 justify-self: center;  
}  
  
.sidebar {  
 position: fixed;  
 width: 200px;  
 height: 100vh;  
 padding: 0 20px 20px 20px;  
 overflow-y: auto;  
 display: flex;  
 flex-direction: column;  
 justify-items:center;  
 grid-column-start: 1;  
}  
  
.main-container{  
 grid-column-start: 2;  
 grid-column-end: 2;  
}  
  
.create-post-btn {  
 padding: 10px 20px;  
 background-color: #4a90e2;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 margin-bottom: 20px;  
}  
  
.create-post-btn:hover {  
 background-color: #2b5a92;  
}  
  
.sidebar h3 {  
 margin-top: 0;  
}  
  
.board-list {  
 list-style: none;  
 padding: 0;  
}  
  
.board-list li {  
 margin: 10px 0;  
}  
  
.board-list a {  
 text-decoration: none;  
 color: #333;  
}  
  
.board-list a:hover {  
 color: #007bff;  
}  
  
.main-content {  
 max-width: 800px;  
 width: 100%;  
 background: #fff;  
 border-radius: 8px;  
 box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);  
 padding: 2rem;  
 margin-top: 1rem;  
}  
  
h2 {  
 color: #4a90e2;  
 margin-bottom: 1rem;  
 text-align: center;  
}  
  
.board, .post {  
 border-bottom: 1px solid #e0e0e0;  
 padding: 1rem 0;  
}  
  
.board-title, .post-title {  
 font-size: 1.2rem;  
 font-weight: bold;  
}  
  
.post-content {  
 font-size: 1rem;  
 color: #666;  
 margin-top: 0.5rem;  
}  
  
.button {  
 background-color: #4a90e2;  
 color: #fff;  
 padding: 10px 20px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 text-decoration: none;  
 display: inline-block;  
 font-size: 1rem;  
 margin-top: 1rem;  
 text-align: center;  
}  
  
.button:hover {  
 background-color: #357ABD;  
}  
  
.profile-container {  
 display: flex;  
 align-items: top  
}  
  
.profile-pic {  
 width: 1.5vw;  
 height: auto;  
 border-radius: 50%;  
 object-fit: cover;  
 margin-right: 5px;  
}  
  
.logout-form {  
 display: inline-block;  
 margin: 0;  
 padding-left: 5px;  
 border: none;  
 background: none;  
}  
  
.logout-button {  
 border: none;  
 background: none;  
 padding-top: 2px;  
 cursor: pointer;  
 display: flex;  
 align-self: center;  
 align-items: center;  
 justify-content: center;  
}  
  
.logout-icon {  
 width: 20px;  
 height: 20px;  
 cursor: pointer;  
 transition: opacity 0.3s ease;  
 fill: #ff0000;  
}  
  
.logout-icon:hover {  
 opacity: 0.7;  
}  
  
.thread-box {  
 background: #fff;  
 border-radius: 8px;  
 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
 display: flex;  
 align-items: stretch;  
 padding: 15px;  
 border-bottom: 1px solid #ccc;  
 margin: 10px 0;  
 width: 700px;  
 min-height: 150px;  
}  
  
.vote-controls {  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 justify-content: center;  
 margin-right: 15px;  
}  
  
.vote-button.active-upvote {  
 color: LawnGreen;  
 font-weight: bold;  
}  
  
.vote-button.active-downvote {  
 color: red;  
 font-weight: bold;  
}  
  
.rating {  
 display: inline-block;  
 width: 50px;  
 text-align: center;  
}  
  
.thread-content-container {  
 display: flex;  
 flex: 1;  
 align-items: stretch;  
}  
  
.thread-image {  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 margin-right: 15px;  
}  
  
.thread-image img {  
 width: 100px;  
 height: 100px;  
 object-fit: cover;  
 border-radius: 5px;  
}  
  
.thread-main {  
 flex: 1;  
 display: flex;  
 flex-direction: column;  
 justify-content: space-between;  
}  
  
.thread-title {  
 font-size: 18px;  
 font-weight: bold;  
 margin-bottom: 10px;  
}  
  
.thread-content {  
 flex-grow: 1;  
 font-size: 14px;  
}  
  
.thread-footer {  
 display: flex;  
 justify-content: flex-end;  
 margin-top: 10px;  
}  
  
.thread-info {  
 display: flex;  
 justify-content: space-between;  
 gap: 15px;  
 font-size: 12px;  
 color: #666;  
 flex-wrap: wrap;  
}  
  
.thread-info span {  
 display: inline-flex;  
 align-items: center;  
}  
  
.thread-info span::after {  
 content: '|';  
 margin-left: 10px;  
}  
  
.thread-info span:last-child::after {  
 content: '';  
}  
  
.thread-comments {  
 display: flex;  
 align-items: center;  
}  
  
.comments-icon {  
 width: 16px;  
 height: 16px;  
 margin-right: 5px;  
}  
  
.post-modal {  
 position: fixed;  
 top: 0;  
 left: 0;  
 width: 100%;  
 height: 100%;  
 background-color: rgba(0, 0, 0, 0.5);  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 z-index: 1000;  
}  
  
.post-modal.hidden {  
 display: none;  
}  
  
.post-modal-content {  
 background-color: white;  
 padding: 20px;  
 border-radius: 8px;  
 width: 400px;  
 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);  
 text-align: center;  
}  
  
.form-group {  
 margin-bottom: 15px;  
 text-align: left;  
}  
  
.form-group label {  
 display: block;  
 font-weight: bold;  
 margin-bottom: 5px;  
}  
  
.form-group input,  
.form-group textarea,  
.form-group select {  
 width: 100%;  
 padding: 8px;  
 border: 1px solid #ddd;  
 border-radius: 4px;  
}  
  
.form-actions {  
 display: flex;  
 justify-content: space-between;  
}  
  
.submit-btn {  
 background-color: #4a90e2;  
 color: white;  
 padding: 10px 15px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
}  
  
.cancel-btn {  
 background-color: #f44336;  
 color: white;  
 padding: 10px 15px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
}  
  
.thread-detail-wrapper {  
 max-width: 900px;  
 margin: 20px auto;  
 background: #fff;  
 padding: 20px;  
 border-radius: 8px;  
 box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);  
}  
  
.thread-detail-box {  
 padding: 20px;  
 border-radius: 8px;  
 margin-bottom: 20px;  
}  
  
.thread-detail-header h2 {  
 font-size: 28px;  
 margin-bottom: 10px;  
 color: #333;  
}  
  
.thread-detail-header p {  
 font-size: 14px;  
 color: #666;  
 margin-bottom: 5px;  
}  
  
.thread-detail-content p {  
 font-size: 16px;  
 margin-bottom: 15px;  
}  
  
.thread-detail-image img {  
 max-width: 100%;  
 height: auto;  
 border-radius: 5px;  
 margin-top: 10px;  
}  
  
.thread-detail-rating {  
 margin-top: 15px;  
 display: flex;  
 justify-content: flex-start;  
 align-items: center;  
}  
  
.vote-controls {  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 gap: 5px;  
}  
  
.comments-section {  
 margin-top: 20px;  
}  
  
.comment-box {  
 background: #f7f7f7;  
 padding: 15px;  
 margin-bottom: 15px;  
 border-radius: 8px;  
 box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);  
}  
  
.comment-box p {  
 margin: 5px 0;  
 font-size: 14px;  
}  
  
.comment-image img {  
 max-width: 100%;  
 height: auto;  
 border-radius: 5px;  
 margin-top: 10px;  
}  
  
*/\* Comment Form \*/*.comment-form {  
 background: #f1f1f1;  
 padding: 15px;  
 border-radius: 8px;  
 margin-top: 20px;  
}  
  
.comment-form form {  
 display: flex;  
 flex-direction: column;  
}  
  
.comment-form h4 {  
 margin-bottom: 10px;  
}  
  
.comment-form button {  
 align-self: flex-start;  
 margin-top: 10px;  
 padding: 10px 20px;  
 background-color: #4a90e2;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
}  
  
.comment-form button:hover {  
 background-color: #357abd;  
}  
  
.thread-detail-rating {  
 display: flex;  
 align-items: center;  
 gap: 10px;  
 margin-top: 15px;  
}  
  
.vote-button {  
 font-size: 24px;  
 color: gray;  
 text-decoration: none;  
 padding: 5px;  
 cursor: pointer;  
}  
  
.rating {  
 font-size: 18px;  
 font-weight: bold;  
}  
  
.comment-form {  
 background: #f1f1f1;  
 padding: 15px;  
 border-radius: 8px;  
 margin-top: 20px;  
}  
  
.comment-form form {  
 display: flex;  
 flex-direction: column;  
 gap: 10px;  
}  
  
.comment-form h4 {  
 margin-bottom: 10px;  
}  
  
.comment-form button {  
 align-self: flex-start;  
 margin-top: 10px;  
 padding: 10px 20px;  
 background-color: #4a90e2;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
}  
  
.comment-form button:hover {  
 background-color: #357abd;  
}  
  
#scroll-to-top {  
 position: fixed;  
 bottom: 20px;  
 right: 20px;  
 background-color: #007bff;  
 border: none;  
 border-radius: 50%;  
 padding: 10px;  
 cursor: pointer;  
 display: none; */\* Hidden by default \*/* box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);  
 z-index: 1000;  
}  
  
#scroll-to-top img {  
 width: 24px;  
 height: 24px;  
}  
  
#scroll-to-top:hover {  
 background-color: #0056b3;  
}