КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технологій програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра
За спеціальністю 122 Комп'ютерні науки

на тему:

ПОРІВНЯЛЬНИЙ АНАЛІЗ СУЧАСНОГО ІНСТРУМЕНТАРІЮ РЕАЛІЗАЦІЇ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

Виконав студент 4-го курсу
Максим ЮДКІН

_____
(підпис)

Науковий керівник:
Кандидат технічних
наук, доцент Олексій
ТКАЧЕНКО

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень
з праць інших авторів без відповідних
посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту
На засіданні кафедри теорії та технології програмування
«____»_____2023 р.,
протокол №
Завідувач кафедри
Микола
НІКІТЧЕНКО _____
(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи – 52 сторінок, 11 рисунок, 22 таблиць

DJANGO, FLASK, ФРЕЙМВОРК, КОРИСТУВАЧ, ORM, PYTHON

Об'єкт та предмет дослідження. Об’єктом дослідження є методи ітехнології розробки інформаційних систем. Предметом дослідження є технологіїствоерення веб орієнтованих систем на базі фреймворків Flask та Django.

Мета й завдання роботи. Метою є дослідження способів реалізації клієнт серверного підходу та інструментарію. Задачі для досягнення цієї мети складаються з дослідження архітектури клієнт-серверного підходу , розробка клієнтську частину на базі фреймворку Flask та окремо на базі фреймворку Django, розробка серверної частини на базі фреймворків Flask та Django та порівняти підходидо реалізації клієнт серверної архітектури у застосунку розробленому за допомогою фреймворку Django та за допомогою фреймворку Flask.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

CRUD - Create Read Update Delete

IDE - Integrated Drive Electronics

API - Application Programming Interface

REST - Representational State Transfer

SMTP - Simple Mail Transfer Protocol

IMAP - Internet Message Access Protocol

POP - Post Office Protocol

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

DTL – Django Template Language

ACID – Atomic Consistency Isolation Durability

ORM - Object-Relational Mapping

MVC – Model View Controller

ВСТУП

Оцінка сучасного стану об’єкта дослідження. Архітектура клієнт-сервер стала домінуючою моделлю для управління величезним ландшафтом сучасної комп’ютерної техніки. Вона продовжує відігравати досить важливу роль в управлінні та оптимізації використання спільних ресурсів у мережах. Модель «клієнт-сервер» являє собою розподілену структуру програми, яка розподіляє завдання або робочінавантаження між постачальниками ресурсів або послуг, які зазвичай називаються серверами, і запитувачами послуг, які називаються клієнтами. Ця модель виділяється своєю ефективністю, модульністю та гнучкістю. Це дозволяє спеціалізованим машинам виконувати завдання, для виконання яких вони найкраще підходять, тоді як інші пристрої можуть бути розроблені з урахуванням інших цілей.

Найпоширенішими типами клієнт-серверної архітектури є дворівнева, трирівнева та n-рівнева архітектури. Дворівнева архітектура передбачає собою прямийзв’язок між клієнтом і сервером, причому на сервері розміщено як сам додаток, так і база даних. В свою чергу трирівнева архітектура розділяє їх на три компоненти: клієнт, сервер де розміщений додаток і сервер де розміщена база даних. А ось n- рівнева архітектура узагальнює цю модель, допускаючи будь-яку кількість дискретнихрівнів.

Flask і Django, дві веб-платформи на основі Python, які були створені щоб полегшити створення додатків за архітектурою клієнт-сервер. Вони стали важливими гравцями у світі розробки веб-додатків завдяки своїй надійності та адаптивності. Фреймворки Flask і Django відіграють значну роль у впровадженні та просуванні цієї архітектури.

У контексті розробки програмного забезпечення термін «фреймворк» відноситься до базової, але настроюваної структури для розробки програмних додатків. Він забезпечує стандартний спосіб створення та розгортання програм, пропонуючи попередньо визначені класи та функції, які розробники можуть використовувати для обробки вхідних даних, керування апаратними пристроями та взаємодії із системним програмним забезпеченням. По суті, фреймворк — це

універсальне багаторазове програмне середовище, яке забезпечує певну функціональність як частину більшої програмної платформи для полегшення розробки програмних додатків, продуктів і рішеньТому саме на їх прикладі буде досліджуватися різниця в підходах до реалізації архітектури клієнт-сервер.

Актуальність роботи та підстави для її виконання. Актуальністьдослідження саме цих фреймворків, можна побачити на графіку, зображеному на рисунку 1.

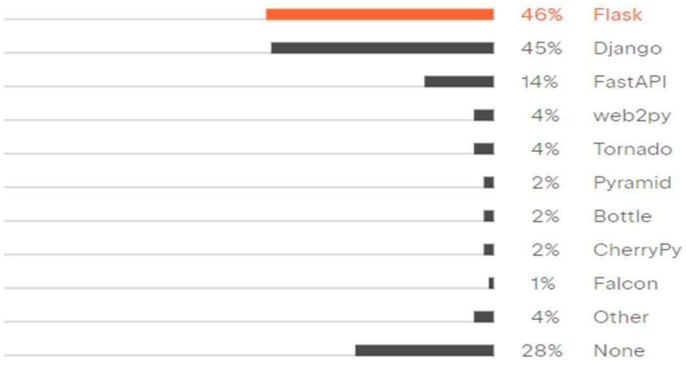

Рисунок 1 Популярність фреймворків для веб розробників [1]

Данні для цього графіку були зібранні компанією JetBrains на річному опитуванні Python розробників. Видно, що попит на технологію Flask та Django майже однаковий. В процесі дослідження буде зрозуміло чому в сфері як веб розробка немає явного фавориту.

Об'єкт та предмет дослідження. Об’єктом дослідження є методи і технології розробки інформаційних систем. Предметом дослідження є технології реалізації клієнт-серверного підходу за допомогою фреймворків Flask та Django. Тобтопредметом буде реалізація клієнт серверної архітектури за допомогою фреймворку Flask, та аналогічно для фреймворку Django.

Мета й завдання роботи. Метою є дослідження способів реалізації клієнт серверного підходу та інструментарію. Для того, щоб порівняти підходи фреймворків до реалізації клієнт серверної архітектури треба визначити критерії по яким буде проводитися дослідження. Критерії повинні дати зрозуміти різницю між цими фреймворками та показати чим виражається відмінність в підходах до самої реалізації клієнт серверної архітектури. Один із способів продемонструвати ці критерії це обрати

задачі, які дуже часто постають перед розробниками при розробці майже будь якого проекту, який має відношення до клієнт серверної архітектури.

Для досягнення цієї мети поставлено такі завдання:

- Створити програмний продукт з використання Django фреймворку.
- Створити програмний продукт використовуючи Flask фреймворк.
- Зробити порівняльний аналіз процесів створення продукту за допомогою Django та Flask.

Можливі сфери застосування.Програмні продукти на тему “Порівняльний аналіз сучасного інструментарію реалізації клієнт- серверноїархітектури” допоможе зрозуміти розробникам веб додатків, на етапі вибору інструментарію для створення продукту, який фреймворк в конкретному випадку краще застосувати та які переваги дає той чи інший інструмент.

Також у випадку, коли ці фреймворки стануть неактуальними, або по якісь причині треба буде використати інші інструмент розробки веб додатків, можна буде зрозуміти, на які критерії треба звертати увагу та які потенційні проблеми можуть виникнути в майбутньому, щоб спланувати їх вирішення заздалегідь.

Взаємозв’язок з іншими роботами.За методами розробки та інструментальними засобами робота виконувалася сумісно з побудовою таналаштуванням реляційних баз даних, та побудови макету дизайну продукту з подальшою реалізацією за допомогою технології Bootstrap.

Також для зручного контролю версій програмного продукту використовувався сервіс GitHub, та відповідна технології Git, яка забезпечувала зручне модифікування проекту в процесі розробки продукту.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Архітектура клієнт-сервер має різні форми, які виходять за межі веб- служб, кожна з яких задовольняє певні потреби на основі масштабу та складності завдань. Одна з таких реалізацій є сервери бази даних. В даній архітектурі сервер підтримує служби бази даних і керує ними, поки клієнти запитують дані. Клієнти можуть виконувати такі дії, як запит даних, оновлення записів або видалення записів. MySQL, PostgreSQL і Oracle Database є популярними прикладами серверів баз даних. Також існують файлові сервери.Вони забезпечують централізовану систему для зберігання та керування файлами, до яких клієнти мають доступ. Ці сервери можуть контролювати ці права доступу, підтримувати контроль версій і пропонувати послуги резервного копіювання. Приклади включають сервери FTP і хмарні служби зберігання,такі як Google Drive і Dropbox. Ще одним з прикладів реалізації архітектури клієнт сервер це ігрові сервери. У багатокористувацьких онлайн іграх ігрові сервери синхронізують дані між гравцями, підтримують правила гри та відстежують результати та стан гравців. Як не дивно, найбільша онлайн-гра, EVE Online, використовує клієнт-сервер архітектура, де єдина копія стану світу зберігається на централізованому сервері і доступ до неї здійснюють клієнтські програми, що працюють на консолях гравців або інші пристрої, це зазначено у джерелі [2]. Клієнтами в цьому випадку зазвичай є ігрові пристрої гравців. Досить часто також використовують сервери електронної пошти.Ці сервери керують і передають електронні листи через мережу. Клієнти надсилають, отримують і зберігають електронні листи через них. Одним з прикладів є SMTP сервери також ще є IMAP сервери або POP сервери. Також існують веб-сервери.Веб-сервери розміщують веб-сайти та надають запитувані веб-сторінки клієнтам, як правило виконується це через веб-браузер. Коли ви вводите URL-адресу у свій веб-браузер, він надсилає запит на відповідний веб-сервер, який потім надсилає запитану сторінку назад у ваш веб-переглядач. Ще гарним прикладом буде проксі-сервери. Вони діють як посередники між клієнтами та іншими серверами, надаючи такі функції, як кешування, безпека та балансування навантаження. Вони можуть допомогти зменшити навантаження на

сервери, підвищити продуктивність за рахунок кешування та забезпечити анонімність клієнтів. Сервери API: ці сервери надають набір API для клієнтів. Клієнти можуть бути будь якими, від інтерфейсу користувача до іншого сервера, взаємодіють із сервером через ці API. Цей тип клієнт-серверної архітектури зазвичай використовується в архітектурах мікросервісів, де кожна служба має власний сервер API.

Досить значну роль відіграють саме веб-сервери. Адже для будь якого бізнес проекту потребується як мінімум сайт візитка, де користувача або інвестори зможуть ознайомитися з тим, що пропонує розроблюваний сервіс. Тому практичний аспект ми продемонструємо саме за допомогою фреймворків Flask та Django, адже завдяки вбудованим DTL у Django та бібліотеки Jinja2 для Flask можна не тільки створити серверну частину, а й також опрацювати клієнтську.

Основним критерієм вибору предметної області буде не так бізнес ідея, як набір функціоналу, який присутній в більшості застосунків. Бізнес ідея проекту без сумнівів досить важлива частина, але в процесі порівняння та дослідження технічнихзастосунків вона являється лише орієнтиром. Ціль же це показати різницю в реалізації та у підході до цієї реалізації у двох фреймворках Flask та Django. Користуючись популярним штучним інтелектом ChatGPT та давши йому питання,

які функції на цей час є найпопулярнішими в більшості програм, я продемонструю який функціонал станом на вересень 2021 року є популярним:

- Аутентифікація та авторизація користувачів. Коли ви використовуєте певні програми, вам потрібно зареєструватися та ввійти за допомогою імені користувача та пароля, щоб захистити вашу інформацію.
- Обмін повідомленнями в режимі реального часу. Можливість відразу поговорити з друзями в додатку та працювати разом над речами.
- Персоналізація. Можливість юзеру працювати над своїм профілем для вираження себе як особистості. Це також допомагає додатку надати вам особливий досвід саме для вас.

Це не весь список, також в списку ще є система рекомендацій, мультиплатформенність та покупки в програмі та інтеграція платежів, але ці особливості все ж потребують або великої бази користувачів, щоб проводити аналітику та відстеження активності

користувачів для системи рекомендацій, мультиплатформенність не розкриє важливих нюансів в порівнянні фреймворку Flask та Django, а платежі хоч і цікава тема, але не релевантна для дипломної роботи.Тому зупинимося на аутентифікації та авторизація користувачів, обмін повідомленнями в режимі реального часу та персоналізації юзера. Також для демонстрації процесу створення, редагування, читання та видалення реалізуємо можливість юзеру додавати, редагувати та створювати об’єкти по типу заміток. Визначившись з популярними задачами, які виникають в процесі розробки більшості застосунків, та обравши три з них в список того, що буде реалізовуватися в наших двох застосунках можемо описати, які технологічні аспекти вони допоможуть відобразити в різниці Django та Flask:

- Можливість аутентифікувати та авторизувати користувачів, використовуючи підходи фреймворків як серверної частини так і клієнтської демонструє різницю в підходах фреймворку до опрацювання моделі користувача
- Створення видалення читання та редагування поста або заміток щоб продемонструвати різницю до CRUD у фреймворках
- Можливість почати чат з іншим юзером демонструє роботу з запитами типу вебсокет у фреймворках
- Модифікування свого профілю та опрацювання своєї галереї, де можна додавати фотографії, видалити їх або поставити на аву демонструє підхід роботи з різними типами даних на вході.

Тому тематика наших застосунків буде реалізація щось схоже на соціальну мережу.

РОЗДІЛ 2. ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ КЛІЄНТ СЕРВЕРНОГО ПІДХОДУ НА БАЗІ ФРЕЙМВОРКІВ FLASK ТА DJANGO

2.1 Інтегроване середовище

Проект був розроблений в інтегрованому середовищі розробки програмного забезпечення Pycharm Professional 2022.3.2. PyCharm Professional — це інтегроване середовище розробки (IDE), спеціально розроблене для розробки Python. Це багатофункціональний і потужний інструмент, який пропонує широкий спектрфункцій для підвищення продуктивності розробників Python. Основними перевагами використання Pycharm Professional являється наявність багатьох розширень які допомагають писати код швидше не тільки на мові програмування python, а й наприклад html та css з можливістю синхронізуватися навіть з DTL.

2.2 Стек технологій

В процесі розробки серверної частини застосунків буде використовуватися мова Python. Мова Python досить добре зарекомендувала себе востанні часи, мова досить проста у вивченні, в гарних руках може бути корисним інструментом в вирішенні великого спектру задач.

Популярність цієї мови програмування можемо подивитися у таблицях Індексу PYPL(рис. 2.1) та TIOBE Індексу(рис. 2.2):

PYPL Index (Worldwide)

Jun 2022	Change	Programming language	Share	Trends
1		Python	27.61 %	-2.8 %
2		Java	17.64 %	-0.7 %
3		JavaScript	9.21 %	+0.4 %
4		C#	7.79 %	+0.8 %
5		C/C++	7.01 %	+0.4 %
6		PHP	5.27 %	-1.0 %
7		R	4.26 %	+0.5 %
8	↑↑↑	TypeScript	2.43 %	+0.7 %
9	↓	Objective-C	2.21 %	+0.1 %
10	↓	Swift	2.17 %	+0.4 %
11	↑↑	Matlab	1.71 %	+0.2 %
12	↓↓	Kotlin	1.57 %	-0.2 %
13	↓	Go	1.48 %	+0.0 %
14	↑↑	Rust	1.29 %	+0.4 %
15		Ruby	1.1 %	-0.0 %
16	↓↓	VBA	1.07 %	-0.2 %
17	↑↑	Ada	0.95 %	+0.4 %
18	↑↑↑	Scala	0.73 %	+0.2 %
19	↓↓	Visual Basic	0.65 %	-0.0 %
20	↓↓	Dart	0.64 %	+0.0 %
21	↑	Abap	0.58 %	+0.1 %
22	↓↓	Lua	0.51 %	-0.0 %
23	↑	Groovy	0.48 %	+0.1 %
24	↓	Perl	0.44 %	+0.0 %
25		Julia	0.41 %	+0.0 %
26		Cobol	0.34 %	+0.1 %
27		Haskell	0.29 %	+0.1 %
28		Delphi/Pascal	0.16 %	+0.1 %

Рисунок 2.1 Таблиця індексу PYPL[3]

TIOBE Index

Jun 2022	Jun 2021	Change	Programming language	Ratings	Change
1	2	↑	Python	12.20%	+0.35%
2	1	↓	C	11.91%	-0.64%
3	3		Java	10.47%	-1.07%
4	4		C++	9.63%	+2.26%
5	5		C#	6.12%	+1.79%
6	6		Visual Basic	5.42%	+1.40%
7	7		JavaScript	2.09%	-0.24%
8	10	↑	SQL	1.94%	+0.06%
9	9		Assembly language	1.85%	-0.21%
10	16	↑↑	Swift	1.55%	+0.44%
11	11		Classic Visual Basic	1.33%	-0.40%
12	18	↑↑	Delphi/Object Pascal	1.32%	+0.26%
13	8	↓↓	PHP	1.25%	-0.97%
14	23	↑↑	Objective-C	1.02%	+0.33%
15	20	↑↑	Go	1.02%	+0.07%
16	14	↓	R	0.98%	-0.22%
17	15	↓	Perl	0.76%	-0.41%
18	38	↑↑	Lua	0.76%	+0.43%
19	13	↓↓	Ruby	0.75%	-0.48%
20	26	↑↑	Prolog	0.74%	+0.18%

Рисунок 2.2 Таблиця індексу TIOBE[3]

Як було зазначено в цілях роботи, для розробки програмного забезпечення буде використовуватися фреймворк Django.

Django є фреймворком високого рівня, який заохочує швидкий розвиток і чистий, прагматичний дизайн. Він дотримується філософії «все в одному», забезпечуючи надійний набір функцій із коробки. Вбудовані функції Django включають рівень ORM для обробки операцій з базою даних, автентифікації користувачів і маршрутизації URL-адрес. Завдяки своїм можливостям Django може виконувати складні завдання, щоробить його придатним для великомасштабних програм корпоративного рівня. Надійність і ефективність Django робить його ідеальним для вирішення складних завдань клієнт-серверної моделі, таких як керування багатьма одночасними підключеннями та робота з великими базами даних.

Також ще буде використовуватися фреймворк Flask.

Flask являє собою легкий і гнучкий фреймворк, який часто використовується для невеликих програм або мікросервісів. Flask дотримується мінімалістичного підходу, надаючи базові функції та дозволяючи розробникамвикористовувати ті бібліотеки, які вони хочуть. Його простота та гнучкість роблять його популярним вибором для швидкого створення прототипів і невеликих веб- сервісів, які є загальною потребою в моделі клієнт-сервер. Природа Flask «підключай і працюй» дозволяє розробникам реалізувати певні функції, необхідні в моделі клієнт- сервер.

Так як Flask дає лише базові інструменти то додатковий функціонали ми будемо брати з інших бібліотек, адже це і є філософія Flask, що якщопотрібен якийсь функціонал то його запросто можна імплементувати з додаткових ресурсів.

SQLAlchemy, Flask надає лише основні інструменти і функції, які є необхідні для веб-розробки, але не надає можливість користуватися ORM, тому завжди Flask йде разом з бібліотекою SQLAlchemy.

У SQLAlchemy є навіть спеціальне розширення саме для розробки за допомогою Flask, Воно має назву Flask-SQLAlchemy. При розробці додатку з Flask ми будем користуватися SQLAlchemy. SQLAlchemy являє собою бібліотеку Python з відкритим кодом, яка надає гнучкий і потужний набір інструментів об’єктно-реляційного відображення для роботи з базами даних. Це дозволяє розробникам взаємодіяти з

базами даних за допомогою високорівневих об’єктів Python, полегшуючи керування та маніпулювання даними.

Jinja2 являє собою ще одну бібліотеку, що майже постійно йде разом з Flask. Jinja2 - це потужний і гнучкий механізм шаблонів, який легко інтегрується з Flask длявідтворення динамічного вмісту у веб-додатках.

Основні переваги використання jinja2 є можливість будувати ієрархічне дерево шаблонів, які являють собою html сторінки з додатковим функціоналом динамічно міняти сам шаблон в залежності від змінних які туди були задані.

WTForms бібліотека також для фреймворку Flask, дає можливість працювати з об’єктом форми, яке дозволяє зручно задавати форми для заповнення тих чи інших данних

Flask_socketio ще бібліотека для Flask, яка дає можливість працювати з протоколами ws:// та wss://

2.3 Організація баз даних

2.3.1 Postgres SQL Server

База даних була розроблена за допомогою Postgres SQL Server. PostgreSQL, яку часто називають Postgres, є потужною та багатофункціональною системою керування реляційною базою даних (RDBMS) із відкритим кодом. Вона забезпечує надійну та масштабовану платформу для зберігання, керування та отримання структурованих даних. Одна з важливих якістю даної бази даних полягає у її відповідності до принципів ACID. Принципи ACID полягають у чотирьох пунктах:

- Атомарність. Атомарність гарантує, що транзакція розглядається як єдина неподільна одиниця роботи. Або всі операції в рамках транзакції виконано успішно, або жодна з них. Якщо будь-яка частина транзакції виходить з ладу, вся транзакція повертається в той стан в якому починалася, так само як і база даних повертається до попереднього стану. Ця властивість допомагає підтримувати цілісність даних і запобігає неповним абонепослідовним змінам даних.
- Узгодженість. Узгодженість гарантує те що транзакція буде переводить

базу даних з одного дійсного стану в інший. Тобто забезпечує дотримання обмежень цілісності, правил і зв’язків, визначених у самій схемі бази даних. Або якщо описати іншими словами, транзакція повинна зберігати загальну послідовність даних, дотримуючись попередньо визначених правил і обмежень. А якщо транзакція порушує будь-яке обмеження узгодженості, то тоді вона переривається, а база даних залишається незмінною.

- Ізоляція. Сама по собі ізоляція гарантує, що кожна транзакція буде виконуватися ізольовано від інших одночасних транзакцій, іншими словами проміжні результати транзакції не будуть видимі для інших транзакцій, доки транзакція не буде завершена. Ізоляція запобігає інтерференції між транзакціями, підтримує цілісність даних і запобігаєконфліктам, що виникають через одночасний доступ до тих самих даних.
- Довговічність. Довговічність гарантує, що після здійснення транзакції її наслідки зберігаються навіть у разі системних збоїв або наприклад аварій на технічному обладнанні де була розташована база даних. Закріплені дані зберігаються на довговічному носії, наприклад на диску, і можуть бути відновлені навіть після перезавантаження системи. Довговічність гарантує, що зміни даних, внесені в рамках здійсненої транзакції, є постійними та витримають будь-які наступні збої.

Саме ці фактори являються одною з причин вибору саме реляційних баз даних, зокрема в нашому випадку це Postgres.

2.3.2 PgAdmin 4

Адміністрування бази даних Postgres відбувалося за допомогою додатка pgAdmin 4. PgAdmin 4 являє собою популярну платформу з відкритим кодом для адміністрування та розробки баз даних PostgreSQL. Він розроблений, щоб забезпечитизручний інтерфейс для керування та взаємодії з базою даних PostgreSQL. PgAdmin 4 має веб-інтерфейс, доступний через веб-браузер. Це дозволяє досить зручнимспособом керувати своїми базами даних PostgreSQL з будь-якого пристрою з підтримуваним браузером, що дає гнучкість і доступність.

2.3.3 ORM

В обох застосунках, як і розробленому за допомогою фреймворку Flasktак і розробленому за допомогою Django буде використовуватися технологія ORM,для роботи з базою даних Postgres. ORM це підхід, який використовується в розробці програмного забезпечення для об’єднання парадигми об'єктно-орієнтованого програмування та реляційних баз даних, як в нашому випадку реляційної бази даних Postgres. ORM надає можливість використовувати спосіб взаємодії з базою даних, використовуючи об’єктно-орієнтовані принципи, а не написання необроблених запитів SQL. Зазвичай підходи до роботи з базами даних є або надсилання напряму SQL запитів які після відправки обробляються сервером бази даних, або підхід ORM.
Основними перевагами ORM над SQL запитами є читабельність коду, адже, пишучи навіть звичайний запит SQL ми будемо працювати з так званими магічними константами, ось так наприклад виглядає звичайний запит SQL на вибір всіх юзерів з бази даних на python використовуючи бібліотеку psycopg2, cursor.execute("select * from User"), у ORM цей запит виглядав б приблизно так, User.objests.all().Також ще ORM забезпечує вищий рівень абстракції та об’єктно-орієнтований підхід до роботи з базами даних, що робить її простішою та інтуїтивно зрозумілішою для розробників.
З недоліків можна підмітити те, що рівень абстракції, який дає підхід ORM приховує механізм обробки запиту, через що використовуючи його функціонал запросто можна зробити поганий запит. Іншими словами ORM часто мають свою документацію, яку треба окремо вивчати кожен раз, це пов’язану з розумінням концепцій які реалізовані в той чи іншій ORM системі, їх конфігурації та використання. Розробникам потрібно ознайомитися з документацією та її умовностями, що може зайняти час і зусилля. А ось SQL напроти є усталеною та широко зрозумілою мовою, з якою багато розробників уже мають досвід.

2.4 Архітекутрний шаблон

Щодо архітектурних шаблонів то у проекті, який буде реалізований за допомогою Django фреймворку, так і у проекті, який буде реалізований за допомогою

Flask фреймворку буде використовуватися шаблон MVC.

MVC являє собою досить популярний шаблон програмування, який використовується для розбиття логіки програми на три компоненти:

- Модель(Model) представляє дані та бізнес-логіку програми. Вона по своїй суті інкапсулює структуру даних, зберігання, пошук і методи маніпулювання. Модель відповідає за підтримку цілісності та узгодженості даних. Також взаємодіє з базою даних та зовнішніми службами або будь-яким іншим джерелом даних для виконання операцій CRUD. Модель повідомляє View і Controller про зміни в даних, дозволяючи їм оновлюватися відповідно.
- Представлення(View) відповідає за представлення даних користувачеві та обробку логіки інтерфейсу користувача. Представлення відображає інформацію, отриману з моделі, і надає користувачеві засоби для взаємодії з програмою. Представлення генерує візуальний результат, наприклад HTML-сторінки, інтерфейси користувача або будь-які інші форми представлення. Головна його відповідальність полягає в тому щоб відтворити дані, обробити введені дані користувача та відобразитивідповідну інформацію для користувача.
- Контролер(Controller) діє як посередник між моделлю та представленням. Контролер відповідає за обробку перевірки введених користувачем даних, ініціювання оновлень моделі та оновлення представлення на основі змін у моделі.

Відношення між компонентами шаблону проілюструю на схемі, яка зображена на рисунку 3

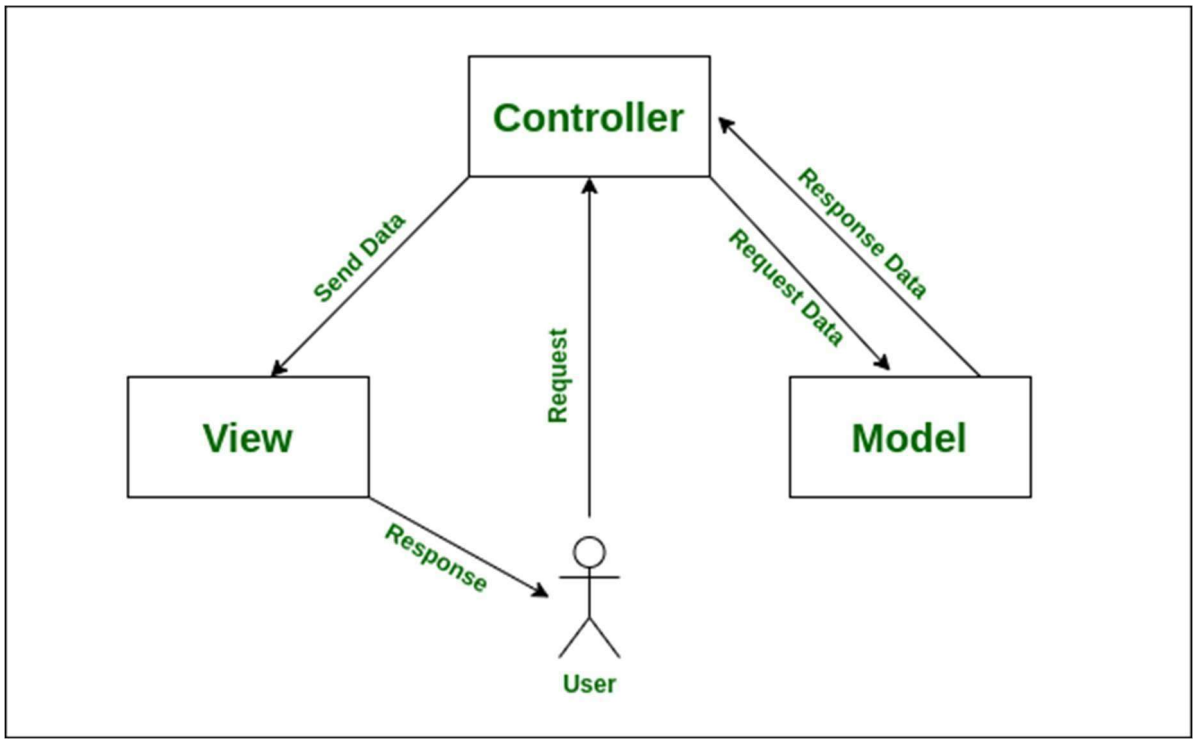

Рисунок 3 Схема шаблон MVC [4]

2.5 Клієнтська частина

Для реалізації клієнтської частини в проектах для Flask та Django окрім jinja2 та DTL
відповідно, буде також використані такі речі як
HTML. Це стандартна мова розмітки, яка використовується для створення структури та вмісту веб-сторінок. По суті HTML надає набір тегів і атрибутів, які визначають елементи та їхні властивості у веб-документі.

CSS. Це мова таблиць стилів, яка використовується для опису представлення та візуального форматування документів HTML і XML.Якщо описати більш детально то це мова, завдяки якій веб-сайти можуть виглядають гарно. Існує декілька, більш точніше всього три, види CSS: внутрішній, вбудований і зовнішній. Внутрішній CSS записується безпосередньо в коді конкретної сторінки. Вбудований CSS застосовується лише до однієї частини сторінки, а також записується безпосередньо в код. Зовнішній CSS зберігається в окремому файлі, і для роботи його потрібно зв’язатиз HTML-кодом. Найкращий спосіб використовувати CSS – використовувати зовнішні стилі, оскільки так легше вносити зміни та уникати повторення того самого коду зновуі знову.

Bootstrap. Bootstrap можна описати як потужний фреймворк, який легко інтегрується з шаблонами Jinja2 для покращення візуального дизайну та швидкості реагування веб-додатків. Так само як він інтегрується з Jinja2 то так же з DTL бо насправді DTL є послідовником Jinja2. З основних переваг bootstrap можу зазначити те, що він є адаптивний для майже всіх девайсів, наприклад телефони, планшети, комп'ютери, а також сумісний з усіма сучасними веб-браузерами, такими як Chrome, Microsoft Edge, Opera, Safari, Internet Explorer.

JavaScript. JavaScript можна описати як універсальну мову програмування, яка додає інтерактивність і динамічну функціональність веб-додаткам. Одна з головних переваг є можливість маніпулювати об'єктною моделлю документа (DOM), яка представляє структуру та вміст документа HTML. За допомогою JavaScript, фреймворки Flask і Django можуть динамічно змінювати й оновлювати елементи, стилі та атрибути в DOM. Ця можливість дозволяє створювати інтерактивні інтерфейси, де елементи можна додавати, видаляти або змінювати на основі дій користувача або відповідей сервера.

Разом з javascript також використовувалися бібліотеки для нього, наприклад socket.io. З точки зору розробника JavaScript, Socket.IO можна описати як потужну бібліотеку, яка забезпечує двонаправлений зв'язок у реальному часі між клієнтом і сервером. Це спрощує реалізацію додатків у реальному часі, забезпечуючи безперебійний і надійний рівень зв'язку на основі WebSocket протоколу. Також для надсилання звичайних запитів по http/https протоколу було використано функція fetch, ось приклад її використання

```
fetch('/chat/start_chat', {method: 'POST', body: JSON.stringify({user_id: userId})
})
```

РОЗДІЛ 3. РОБОТА З БАЗОЮ ДАНИХ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ DJANGO ТА FLASK

3.1 ActiveRecord vs DataMapper

Перше з чого варто почати, коли ми говоримо про розробку серверу це база даних

Адже спосіб, як ми будемо обробляти запити, напряму залежить від того, як ми будемо зберігати дані з запиту.Для зручності ми не будемо використовувати якісь специфічні сховища для даних, та обійдемося звичайною реляційною базою PostgreSQL, вона ідеально підходить як і для Flask, так і для Django.

Але незважаючи на те, що використання PostgreSQL зручне для двох фреймворків, підхід для роботи з базами даних у них кардинально різний, ця різниця і буде висвітлена в цьому розділі. Почнемо з того, шо в обох фреймворках робота з базами даних проходить через ORM.

Технологія ORM дозволяє нам інкапсулювати роботу з базою даних, та дати нам зручний інтерфейс, базований на представлені таблиці даних, як клас, де стовбці таблиці це атрибути класу, а рядок це об’єкт класу.

Але в Flask та Django для реалізації ORM технології використовують два різні шаблони проектування. А саме, Django використовую шаблон проектування ActiveRecord. Мартин Фаулер у книжці [5] описує шаблон ActiveRecord як об'єкт, що виконує роль оболонки для рядка таблиці або уявлення бази даних. Цей об’єкт інкапсулює доступ до бази даних і додає до даних логіку домену. ШаблонActiveRecord гарно себе показує, коли домена логіка не є складною.Тобто якщо більшість запитів є читання, створення, оновлення та видалення ActiveRecod є хорошим вибором. Що насправді покриває найбільш популярні типи запитів.Також ActiveRecord виграє тим, що вона проста, її просто побудувати та легко зрозуміти.Але недоліком ActiveRecord є те, що якщо ваша бізнес-логіка складна, ви незабаром захочете використовувати прямі зв’язки вашого об’єкта, колекції, успадкування тощо.І ось вже їх буде доволі непросто відобразити на ActiveRecord, і додавати їх по частинах стає дуже незручно.Також до практичних недоліків я б відніс те, що

наприклад ті ж запити з використанням ORM Django, повертають нам доволі складні об’єкти, що мають купу всяких метаданих,

які не потрібні в більшості випадків. Більш того, іноді складність об’єктів, що повертають ORM запити в Django може призвести до сповільнення роботи сервера, завдяки збільшуванню кількість запитів на сервер бази даних.

Наприклад якщо ми виконаємо такий запит для Django

users = User.objects.values('username', 'email')

зараз зміна users має в собі всіх юзерів зі стовбцями username, email але якщо нам наприклад знадобиться profile_picture юзера і ми зробимо до нього звернення таким чином

users.profile_picture

при умові, що в нас є таке поле в табличці User то ми отримаємо його значення, і хоча це виглядає зручно, то треба розуміти, що під капотом воно зробило ще одне звернення до бази даних, тобто воно знову підключилося до бази даних і зробило запит взяти всіх юзерів але з поле profile_picture. Гарно, що така можливість є, але кількість підключень до бази даних напряму впливає на продуктивність застосунку. Боце можна сприймати як те, що сервер робить окремий запит на інший сервер, який після обробляє це запит, і повертає відповідь в якій, як ми вже визначили, знаходяться складні об’єкти ORM створені по шаблону ActiveRecord. Тому такі речі погано впливають на сервер, якщо людина не знає цих нюансів в реалізації ORM у фреймворку Django. Схема реалізації шаблону ActiveRecord наведена на Рисунку 4.

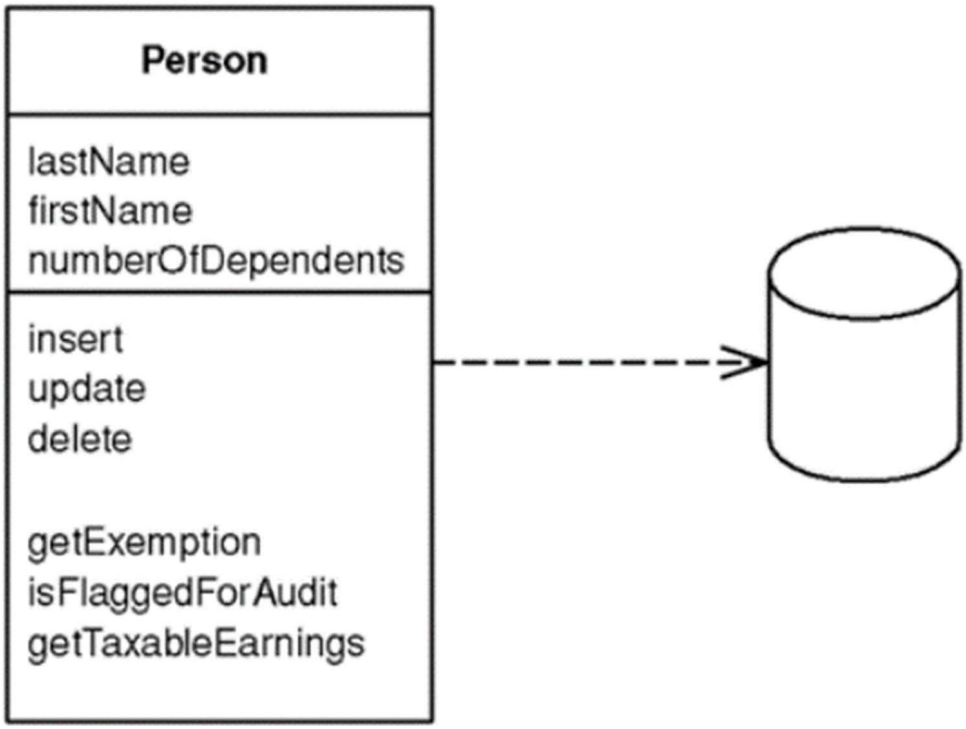

Рисунок 4 схема шаблону ActiveRecord [5]

У фреймворку Flask використовується інший підхід до реалізації ORM системи.
А саме шаблон DataMapper. Його схему зображено на Рисунку 5.

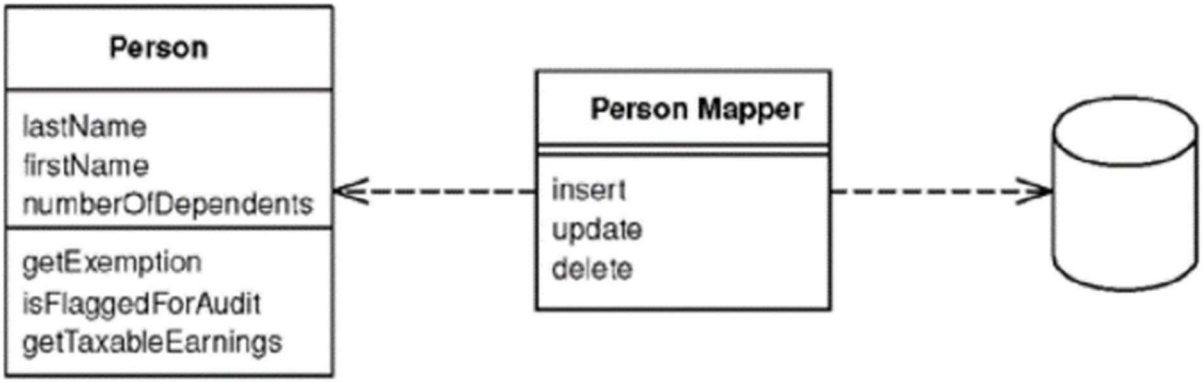

Рисунок 5 схема шаблону
DataMapper [5]

DataMapper являє собою шар програмного забезпечення, який відокремлює об'єкти, розташовані в оперативній пам'яті, від бази даних. У функції DataMapper входить передача даних між об'єктами та базою даних і ізоляція їх один від одного. Завдяки використанню цього підходу об'єкти, розташовані в оперативній пам'яті, можуть навіть не підозрювати про сам факт присутності бази даних. Їм не потрібен SQL- інтерфейс і тим паче схема бази даних. У свою чергу, схема бази даних ніколи не знає

про об'єкти, що її використовують. Більше того, оскільки DataMapper є різновидом Mapper, він повністю прихований від рівня домену.

Це дуже допомагає у коді, оскільки ви можете розуміти об’єкти домену та працювати з ними без необхідності зрозуміти, як вони зберігаються в базі даних.

По суті працювати з простими структурами даних, звичні для мови Python, а не з ускладненими об’єктами.

Зі складними відображеннями, особливо з наявними і вже існуючими базами даних, це є дуже гарний спосіб.

Ціною за це буде додатковий рівень, який ви не отримуєте з Active Record, тому перевірка на те, який шаблон використати - це складність бізнес-логіки. Якщо у вас досить проста бізнес-логіка, то ймовірно є більше сенс в Active Record, аніж в DataMapper. Більш складна логіка все ж приведе вас до DataMapper.

3.2 Реалізація таблиць бази даних в Flask SQLAlchemy ORM

Діаграма бази даних розташована на Рисунку 6

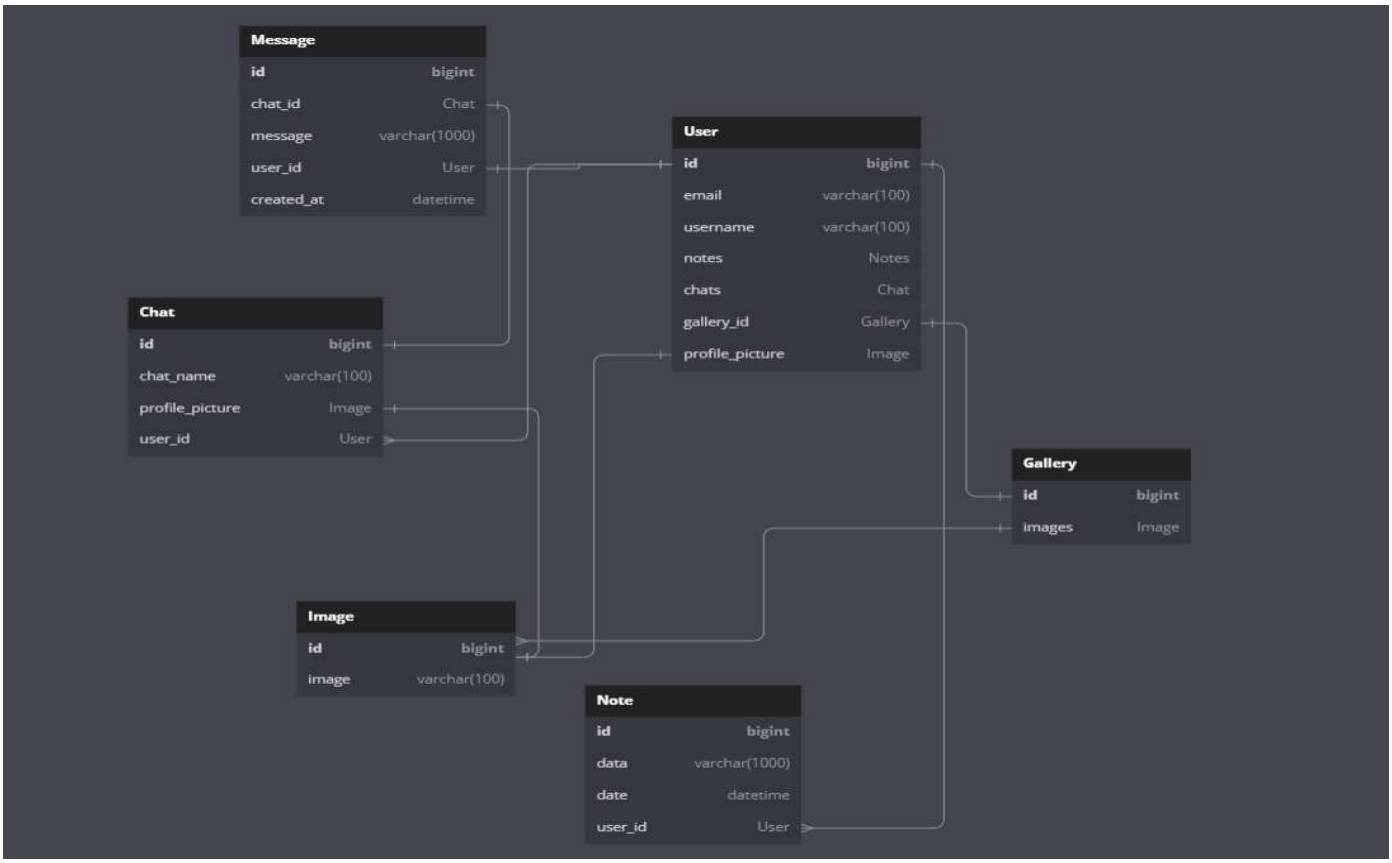

Рисунок 6 UML
діаграма бази даних

Враховуючи ще проміжні таблиці запишемо їх остаточний вигляд

Таблиця 1 – Опис наявних схем у проекті

1	User	Схема користувача
2	Chat	Схема чату
3	Chat_Members	Схема, що описує зв'язок користувача та чату
4	Gallery	Схема галереї
5	Gallery_images	Схема, що описує зв’язок галереї та об’єкту фотогрфії
6	Image	Схема фотографії
7	Message	Схема повідомлення
8	Note	Схема нотатку для юзера

Таблиця 2 – User

id	Integer	Ідентифікатор користувача
email	Character varying	Електрона пошта користувача
password	Character varying	Пароль користувача
username	Character varying	Ім’я користувача
gallery_id	Integer (Foreign Key)	Зв’язок з галереєю користувача
profile_picture	Integer (Foreign Key)	Зв’язок з фотографією, яка є аватаром користувача

Таблиця 3 – Chat

id	Integer	Ідентифікатор чату
chat_name	Character varying	Назва чату
profile_picture	Integer (Foreign Key)	Зв'язок з фотографією, яка є аватаром чату
user_id	Integer (Foreign Key)	Зв'язок з тим хто створив чат

Таблиця 4 – Chat_members

id	Integer	Ідентифікатор зв’язку між чатом та користувачем
chat_id	Integer (Foreign Key)	Чат до якого належить користувач
user_id	Integer (Foreign Key)	Користувач який є у чату

Таблиця 5 – Gallery

id	Integer	Ідентифікатор галереї

Таблиця 6 – Gallery_images

id	Integer	Ідентифікатор звязку між галереєю та фотографіями
gallery_id	Integer (Foreign Key)	Галерея до якої належать фотографії
Img_id	Integer (Foreign Key)	Фотографії які належать до галереї

Таблиця 7 – Image

id	Integer	Ідентифікатор фотографії
image	Character varying	Посилання на фотогрфію

Таблиця 8 – Message

id	Integer	Ідентифікатор повідомлення в чаті
chat_id	Integer (Foreign Key)	Чат до якого відноситься повідомлення
message	Character varying	Зміст повідомлення
user_id	Integer (Foreign Key)	Користувач, який відправив повідомлення
created_at	Timestamp with time zone	Дата відправки повідомлення

Таблиця 9 – Note

id	Integer	Ідентифікатор нотатку юзера
data	Character varying	Зміст нотатку
date	Timestamp with time zone	Дата відправлення нотатку
user_id	Integer (Foreign Key)	Користувач, який відправив нотаток

3.3 Реалізація таблиць бази даних в Django ORM

Як ми вже знаємо підходи до реалізації ORM різняться для Django та Flask, як мінімум різниця полягатиме в тому, шо Django автоматично створює купу додаткових таблиць для реалізації функціоналу, який йде в Django з коробки

Таблиця 10 – Опис наявних схем у проекті

1	auth_user	Схема користувача
2	auth_group	Схема групи юзерів
3	auth_permission	Схема прав
4	auth_group_permission	Схема що зв’язує групи юзерів та права, які вони мають
5	auth_user_groups	Схема зв’язку юзерів та групи
6	auth_user_user_permissions	Схема що напряму зв’язує юзера та права які він має
7	django_admin_log	Схема логів для адмінів
8	django_content_type	Схема типів моделі
9	django_migrations	Схема міграцій
10	django_session	Схема сесії

Таблиця 11 – auth_user

id	Integer	Ідентифікатор користувача
password	Character varying	Пароль користувача
last_login	Timestamp with time zone	Час останнього логіну
is_superuser	Bool	Чи користувач супер юзер
username	Character varying	Інтернет ім’я користувача
first_name	Character varying	Ім’я користувача
last_name	Character varying	Фамілія користувача
email	Character varying	Електрона скринька користувача
is_staff	Bool	Чи має можливість користувач працювати з панеллю адміністратора
is_active	Bool	Чи може користувач виконати запит аутентифікації
date_joined	Timestamp with time zone	Час, коли користувач був зареєстрований

Таблиця 12 – auth_group

id	Integer	Ідентифікатор Групи
name	Character varying	Назва групи

Таблиця 13 – auth_permission

id	Integer	Ідентифікатор права
name	Character varying	Назва права
content_type_id	Integer	На який тип об’єкту працює це правило
codename	Character varying	Ім’я права читабельне для машини

Таблиця 14 – auth_group_permission

id	Integer	Ідентифікатор зв’язку групи користувачів та прав
group_id	Integer	Група на яку впливають права
permission_id	Integer	Права, які впливають на групу користувачів

Таблиця 15 – auth_user_groups

id	Integer	Ідентифікатор зв’язку користувача та групи
user_id	Integer	Користувач який належить до групи
group_id	Integer	Група до якої належить користувач

Таблиця 16 – auth_user_user_permissions

id	Integer	Ідентифікатор зв’язку користувача та права
user_id	Integer	Користувач який має право
permission_id	Integer	Право яке обмежує користувача

Таблиця 17 – django_admin_log

id	Integer	Ідентифікатор логу
actions_time	Timestamp with time zone	Час коли лог був записаний
object_id	Text	Ідентифікатор об’єкту про який відбувся лог
object_repr	Character varying	Строкове представлення об’єкту з яким оперує лог
action_flag	Smalint	Вид адміністративної дії, яка була вчинена
change_message	Text	Додаткові відомості або повідомлення, пов’язані з адміністративною дією
content_type_id	Integer	Тип об’єкта з яким оперуємо
user_id	Integer	Адмін який зробив відповідну дію, яка створила цей лог

Таблиця 18 – django_content_type

id	Integer	Ідентифікатор типу контенту
app_label	Character varying	Назва Додаток в якому знаходиться модель об’єкту
model	Character varying	Назва Модель

Таблиця 19 – django_migrations

id	Integer	Ідентифікатор типу контенту
app	Character varying	Назва Додаток в якому знаходиться файл міграції
name	Character varying	Назва Міграції
applied	Timestamp with time zone	Час, коли міграція була застосована

Таблиця 20 – django_session

session_key	Character varying	Ідентифікатор сесії
session_data	Text	Дані щодо сесії
Expired_date	Timestamp with time zone	Час коли сесія буде вичерпана

Інші таблиці будуть майже таку саму структуру як і були в Flask додатку. Що треба підмітити у додаткових таблицях Django додатку, та частина з них невикористовується розробником напряму. Що є однозначним недоліком, адже виходить, що структура бази даних в Django апріорі складніша аніж у додатку створеному за допомогою Flask. Це є ціна великої кількості функціоналу, який Django надає зразу. Більш детально функціонал, який надає Django та Flask описано в джерелах [6] та [7].

Ще можна відмітити те, що модель користувача згенерована автоматично, це неоднозначний плюс, адже дійсно, більшість варіантів використання цієї моделі покрито в даній моделі, але є нюанс. Якщо нам знадобиться переписати цю модель, під свої потреби, то це буде зробити значно складніше аніж аналогічно в Flask.

Нам буде потрібно унаслідуватися від базової моделі юзера та описувати менеджер який буде опрацьовувати ORM запити, у Flask ми пишемо модель користувача з нуля як забажаємо і її модифікація є легкою задачею.

Також багато полів згенерованої таблиці юзерів іноді є надлишковим та не використовуються в процесі розробки, тим самим ускладнюючи схему бази даних без причини.

Тому Flask з SQLAlchemy значко гнучкіший в сенсі конструювання бази даних, хоча він не дає багато функціоналу з його налаштуванням, але він дає можливість це зробити самому за потреби, хоча ціна в цьому випадку буде час та використання додаткових бібліотек з вивченням їх документації.

Декілька слів про терміни в цих таблицях, наприклад в таблиці номер 16 в нас є поле app_label, додаток в проекті Django являє себе окремий модуль який складається з свого власного відображення, моделей та інших додатків, те як виглядає модулі у проекті Django можна побачити на рисунку 6

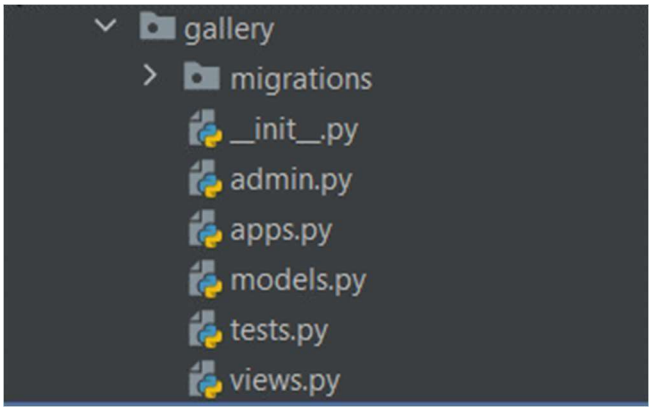

Рисунок 7 Файлова система додатку Django

Всі ці модулі прив’язані в основному ядрі проекту, таким чином ми можемо не витрачати час на відокремлення різної бізнес логіки в окремі модулі, за нас це зробить команда

python manage.py startapp <app_name>

Але така сама ситуація як з моделями, частіше за все ми отримаємо більш ніж нам потрібно, наприклад ті ж тести можуть бути не реалізованими в даному додатку, або адміністрація моделей додатку не потребується.

Тому на цьому етапі видно недолік того різноманіття, яке дає Django – не все треба.

3.4 Порівняння фреймворків Flask та Django

Таблиця 21 демонструє результати порівнянь підходу реалізації роботи з ORM для фреймворків Flask та Django.

Таблиця 21. Порівняння реалізації ORM для фреймворків Flask та Django

Django	Flask
Підхід до реалізації ORM розроблений по шаблону ActiveRecord, що дає можливість працювати з об’єктами, які містять багато мета даних	Підхід до реалізації ORM розроблений по шаблону DataMapper, що дає можливість працювати конкретно з даними без зайвих мета даних
Включає в себе купу додаткових таблиць, які потрібні для роботи деякого функціоналу Django, але не завжди потрібні розробнику	Все налаштування та створення таблиць бази даних, від початку да до кінцялежить на розробнику, що дає гнучкість, але потребує знання та час на розробку
Функціонал, який вбудований в Django іноді буває надлишковим	Дає лише базовий функціонал, вся інша додаткова реалізація залежить від вибору розробника

РОЗДІЛ 4 ПРОГРАМНА РЕАЛІЗАЦІЯ ОСНОВНИХ МОДУЛІВ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ DJANGO ТА FLASK

4.1 Програмна реалізація Flask

З початку створимо ядро нашого проекту, це буде init.py файл у папці website яка буде вмістилищем нашого проекту.

В цьому __init__.py файлі припишемо створення нашого застосунку Виглядати того як це буде реалізовано в коді можна побачити у Додатку А.

Функція os.getenv() бере змінні з .env файлу, це зроблено для безпеки вразливих констант, як наприклад SQLALCHEMY_DATABASE_URI, яка зберігає в собі данні про те, до якої саме бази даних треба під'єднатися.

Далі почнемо з реєстрації та логіну користувача за допомогою Flask

4.1.1 Реєстрація та логін

Для початку створимо окремий модуль auth.py де буде реалізація реєстрації та логіну користувача, створимо всередині об'єкт Blueprint. Blueprint це збірник маршрутів та інші пов'язані з програмою функції, які можна зареєструвати в програму пізніше.

Виглядає це ось так

```
auth = Blueprint('auth',__name__)
```

Тепер цей модуль додамо в наш__init__файл таким чином

```
from .auth import auth
app.register_blueprint(auth, url_prefix='/')
```

Тепер ми зареєстрували наш модуль до проекту і можемо написати наше представлення sign_up

```
@auth.route('/sign_up', methods=['GET', 'POST'])
def sign_up():
    if request.method == 'POST':
    …
```

```
 return redirect(url_for('views.home'))
return render_template("sign_up.html", user=current_user)
```

Як можемо побачити, ми використовуємо обєкт Blueprint нашого модуля, щоб додатинаше представлення до шляху '/sign_up' і обмежити запити методами GET та POST. Схожим способом ми будемо так робити для всіх інших представлень.

Далі в самому представлені sign_up ми залежно від методу запиту генеруємо сторінку, яка відобразиться у користувача. При методі POST нас повинно перенаправити на шлях для відображення ‘views.home’ це зроблено через функцію url_for() яка є частиною функціоналу фреймворку Flask, ми б могли написати просто redirect(“url”), але недоліктакого підходу очевидний, якщо ми змінемо назву шляху до представлення то тоді нам буде потрібно міняти його і в кожному перенаправленні, в цьому ж випадку воно завждизнайде представлення по імені, як би ми не міняли шлях.

Якщо ж запит був GET то ми відображаємо шаблон sign_up.html передаючі до нього дані що user=current_user.

Щодо реалізації POST, далі ми збираємо дані які були надісланні в sign_up.htm формiТаким чином

```
email = request.form.get('email', '')
username = request.form.get('username', '')
password1 = request.form.get('password1', '')
password2 = request.form.get('password2', '')
```

Далі йде перевірка чи існує юзер з такою електронною поштою чи з таким інтернет ім’ям , також перевіряється те що email, username , password1 та password2 валідні, це відбувається за допомогою Flask-wtf.

Далі, якщо валідація пройшла успішно то створюється об’єкт нового користувача

```
new_user = User(email=email, username=username,
password=generate_password_hash(password1, method='sha256'))
db.session.add(new_user)
db.session.commit()
login_user(new_user, remember=True)
```

flash('Account created!', category='success')

return redirect(url_for('views.home'))

Пароль для нового юзера шифрується.

Після чого ми додаємо об’єкт нового юзера до сесії нашою бази даних і робимоcommit наших змін.

Далі використовуючи бібліотеку flask-login ми імпортуємо функцію login_user(), яка буде зберігати об’єкт нашого користувача в сесію, тим самим це дає можливість у майбутньому отримати юзера, який пройшов реєстрацію, або логін в майбутніх запитах. Сам Flask не має такої можливості без додаткової бібліотеки, в Django такий функціонал присутній одразу.

Опрацювання логіну дуже схоже, але в цьому випадку нам потрібно лише отримати пароль та інтернет ім’я користувача взявши ввідповідного юзер з таблиці юзера по йогоімені та використовуючи функцію check_password_hash() розшифрувати пароль

і звірити його, якщо все гаразд та пароль співпадає то знову залогінити юзера до нашоюсесії за допомогою функції login_user().

Щоб додати цю сесію до нашого застосунка треба ще внести зміни до нашого ядра. Ці зміни наведені у Додатку А

Тепер ми можемо застосовувати декоратор login_required який буде перенаправлятикористувача на реєстрацію, якщо той не має запису в поточній сесії.

Також якщо користувач зареєстрований то представлення може оперувати з об’єктомсаме цього юзера.

Те як будуть виглядати реєстрація та логін можно побачити на рисунках 7 та 8 відповідно.

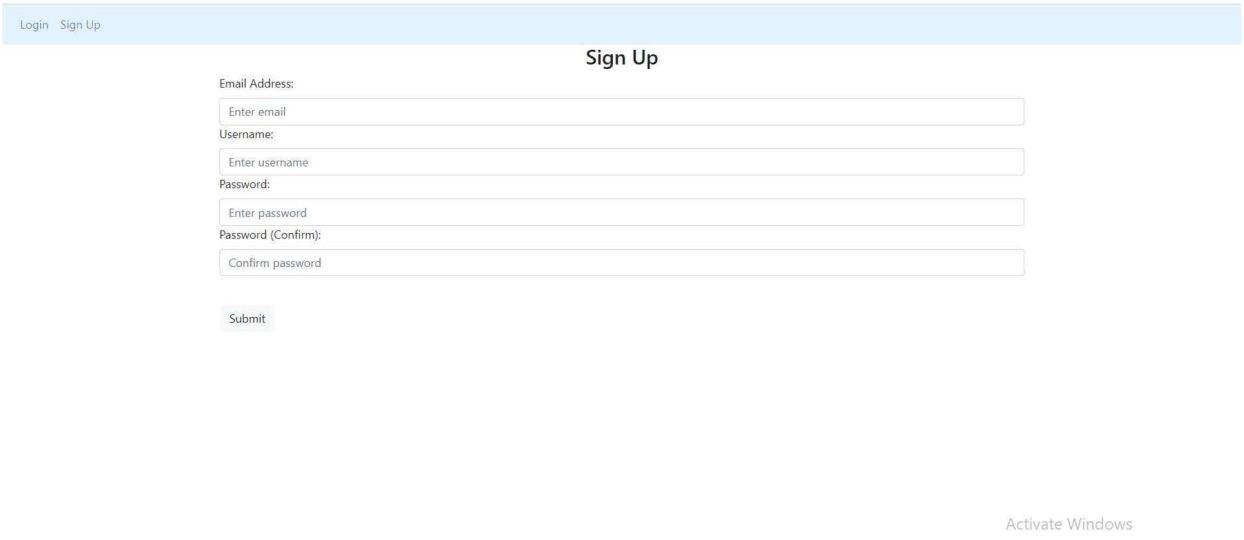

Рисунок 8 Сторінка реєстрації користувача

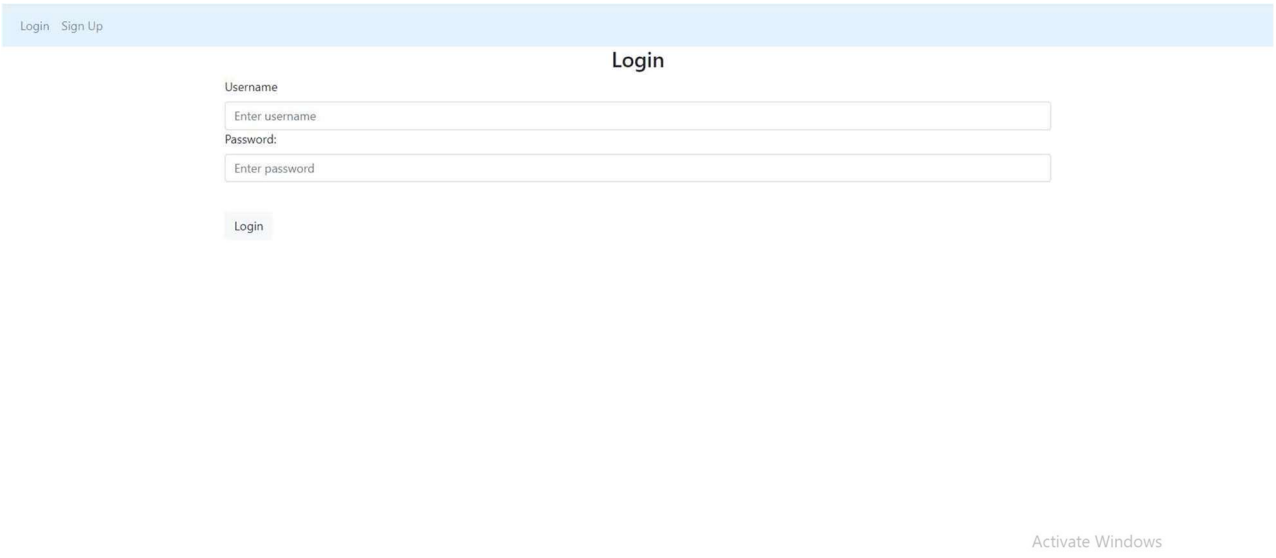

Рисунок 9 Сторінка логіну користувача

4.1.2 Реалізація CRUD

За основу будемо брати об’єкти моделі Note, це будуть нотатки, якікористувач може зберігати, редагувати, читати та видаляти.

Створення об’єкту Note відбувається в іншому додатку, у представлені homeСаме представлення можна побачити у Додатку А

Виконання даного представлення можливе лише якщо користувач авторизувався, цезабезпечує декоратор login_required,

При виконанні POST запиту ми беремо параметр note, що представляє собою самузамітку, та оперуючи з об’єктом current_user, який є ауторизований користувач в

нашій сесії, створюємо об’єкт Note. Далі завантажуємо ту саму сторінку, яка оновить список наших заміток, сам шаблон написаний ось так

Ця частина , це наслідування від основного шаблону, так як візуально ми міняємо не всіелементи то ті які в нас статичні ми виносимо в base.html а сам content виносимо в окремий блок, який вже завантажується відносно нашого шаблону

```
{% extends "base.html" %}
{% block title%}Home{% endblock %}
{% block content%}
```

Сам контент складається з двох речей

Це відображення всіх вже існуючих заміток юзера

```
<h1 align="center">Notes</h1>
<ul class="list-group list-group-flush" id="notes">
        {% for note in user.notes  %}
        <li class="list-group-item ">{{note.data}}
```

Функціонал, який викликає зміну нотатку

```
<button type="button" class="close" onClick="editNote({{ note.id }})">
        <span aria-hidden="true">Edit</span>
```

Та видалення

```
<button type="button" class="close" onClick="deleteNote({{ note.id }})">
        <span aria-hidden="true">&times;</span>
</button>
</li>
{% endfor %}
</ul>
```

Та частина відправки нового запису

```
<form method="POST">
        <textarea name="note" id="note" class="form-control"></textarea>
        <br />
        <div align="center">
        <button type="submit" class="btn btn-light">Add Note</button>
```

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

Видаленняяк і редагування відбувається через виклик javascript коду який підключений до base.html, а тому працює і в нашому home.html.

Цей код виглядає так

```
function deleteNote(noteId){
fetch('/delete-note', {method: 'POST', body: JSON.stringify({noteId: noteId})
}).then((_res) => {
            window.location.href = "/";});
}
```

Таким чином ми викликаємо представлення delete_note, аналогічно для edit_note, самі зміни відбуваються в представленнях майже однаково, в випадку з реагуванням це буде процес пошуку об’єкту нотатку, який відповідає даному ідентифікатору, після чого зміна його полів таким чинном

note.<назва_поля> = значення

після чого зміни треба зберегти у базу даних таким чином

```
db.session.add(note)
db.session.commit()
```

для видалення ще простіше, просто беремо той нотаток, який відповідає ідентифікатору та викликаємо методи

```
db.session.delete(note)
db.session.commit()
```

Спосіб видалення найбільш цікавий в цьому випадку, адже ми бачимо, що сам об’єкт Note не має методу delete(), його має лише об’єкт бази даних, це важливо помітити адже, відповідна логіка в Django буде відрізнятися, там у об’єкта Note буде власний метод delete()

4.1.3 Чат

Робота з чатом буде виконуватись завдяки бібліотеки flask-socketio.

Основні події, які будуть опрацьовуватися можна розбити на три функції.

Ця функція відповідає, за підключення юзера до чату, перед її викликом юзер повинен натиснути кнопку яка створить об’єкт чату та потім передає id моделі Chat, по якому буде ідентифікуватися чат, в самому connect() буде виконуватися функція join_room(), яка є вбудована функція в бібліотеку flask-socketio.

```
@socketio.on("connect")
def connect(auth):
    …
```

Наступна функція відповідає за від’єднання користувача, та викликає вбудовану функцію leave_room() в бібліотеку flask-socketio.

```
@socketio.on("disconnect")
def disconnect():
    …
```

І найголовніша функція message. Вона викликається коли юзер натисне кнопку відправити повідомлення, з відповідним повідомленням. В процесі виконання цієї функції відбудеться створення об’єкту моделі Message, та додавання цього об’єкту до повідомлень відповідного чату, а далі буде виклик send(content=, to=) де content це саме повідомлення, а to це айді чату до якого воно відправлене, теж вбудована функціяв flask-socketio.

```
@socketio.on("message")
def message(data):
    …
```

Важливо підмітити той факт, що опрацювання цих подій відбувається в окремих функціях, бо наприклад у Django схожий функціонал буде представлений в класовій реалізації, з іншим синтаксисом.

4.1.4 Обробка даних типу файл

При редагуванні свого аккаунту, юзер може змінити собі аватар або додати нову фоту в галерею, для опрацювання таких даних в формі ми будемо використовувати flask-wtf. З його допомогою ми можемо створити форму, приклад цієї форми наведено в Додатку А.

Ця форма дозволить нам задавати profile_pic як файл, який потім буде

завантажуватися в папку static/ , щоб відобразити цю форму в шаблоні jinja2 нам буде достатньо написати таке

```
{{ form.hidden_tag() }}
    {{ form.username.label(class="form-label") }}
    {{ form.username(class="form-control", value=user.username) }}
    <br/>
    {{ form.email.label(class="form-label") }}
    {{ form.email(class="form-control", value=user.email) }}
    <br/>
    {{ form.profile_pic.label(class="form-label") }}
    {{ form.profile_pic(class="form-control", value=user.profile_pic) }}
    <br/>
    {{ form.submit(class="btn btn-light") }}
```

Таким чином ми можемо побачити цю форму на рисунку 9

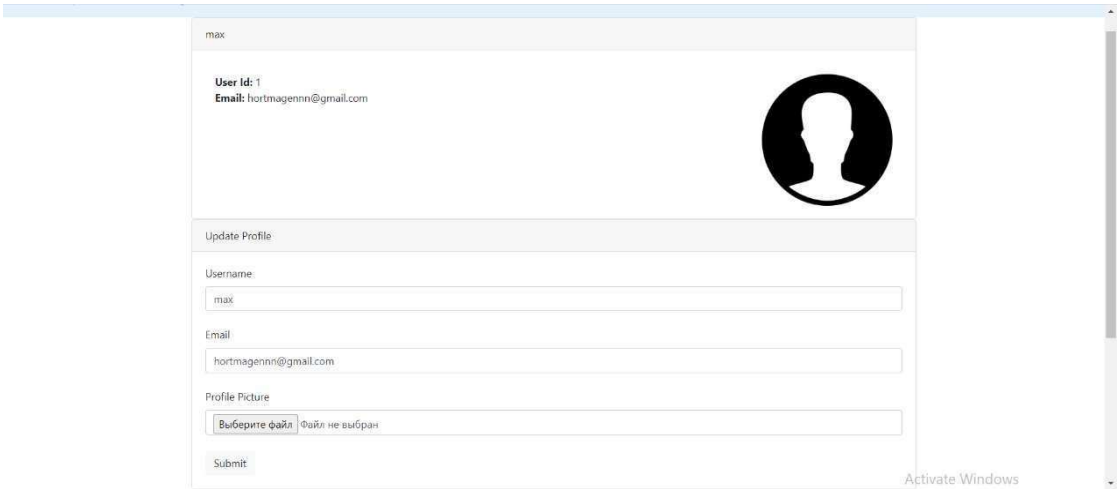

Рисунок 10 Сторінка профілю

Представлення, яке буде опрацьовувати цю форму буде зберігати файл таким чиномprofile_img_filename = secure_filename(profile_image.filename)

```
profile_img_name = str(uuid.uuid1()) + "_" + profile_img_filename
path = os.path.join(os.getenv('UPLOADED_FOLDER'), profile_img_name)
profile_image.save(os.path.join(path))
```

secure_filename() є функцією бібліотеки werkzeug

Передаючі йому назву файлу, функція поверне його безпечну версію. Цю

назву файлу можна безпечно зберегти у звичайній файловій системі та передати до :func:os.path.join. Повернене ім’я файлу є лише рядком ASCII для максимальної мобільності.

А функція uuid.uuid1() яка є вбудована в мову програмування Python згенерує випадковий набір символів, який залежить від часу, коли його викликали, забезпечуючи унікальність кожної назви фотографії.

Після збереження наш профіль одразу зміниться. Це видно на рисунку 10.

Також у файлову систему в папку static/images збережеться файл, його ім’явиглядатимеприблизно ось так

0e6a4b14-f27c-11ed-befa-dcf505d55bb0_photo.jpg

А в моделі Image з’явиться відповідний запис, який після додається до галереї користувача

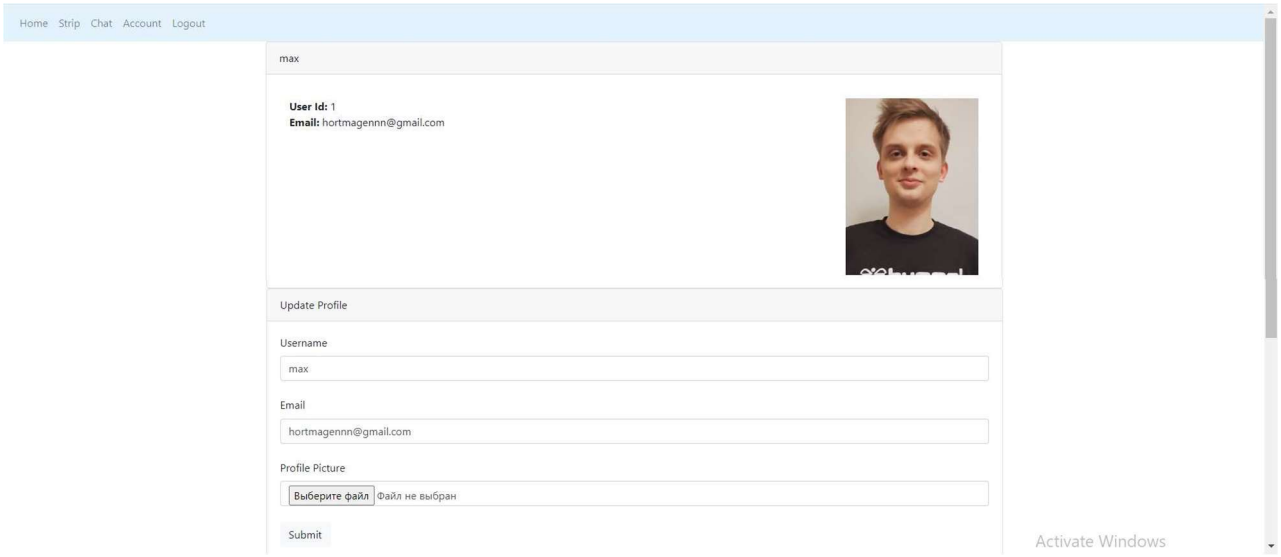

Рисунок 11 Сторінка оновленого профілю користувача

Також є спосіб додавання просто в галерею користувача яка відображається трохинижче.

Форма для цього виглядатиме трохи інакше, адже в галерею можна завантажити зразуне одну фотографію, а декілька,

```
class GalleryUserForm(FlaskForm):
images = FileField('Files', render_kw={'multiple': True})
submit = SubmitField("Submit")
```

Працює вона аналогічно встановленню аватару

4.2 Програмна реалізація Django

Для створення проекту в Django нам не потрібно нічого робити вручну, все зробить команда

django-admin startproject projectname

Коли ви запускаєте команду django-admin startproject projectname для створення проекту Django, вона створює такі файли та каталоги:

- projectname/: це головний каталог проекту, ім'я якого було вказано під час виконання команди. Він служить кореневим каталогом для проекту Django;
- projectname/__init_.py: цей порожній файл необхідний для обробки каталогу projectname як пакета Python;
- projectname/asgi.py: цей файл є точкою входу для протоколу ASGI (Asynchronous Server Gateway Interface), який використовується для асинхронного запуску програм Django;
- projectname/settings.py: цей файл містить параметри та конфігурації проекту. Він містить налаштування бази даних, проміжне програмне забезпечення; встановлені модулі, конфігурацію статичних файлів, налаштування шаблонів та інші параметри, що стосуються проекту, по суті являється ядром проекту;
- projectname/urls.py: цей файл містить конфігурацію URL адреси проекту. Він зіставляє URL адреси з відповідними представленнями чи наборами представлень у проекті Django.
- projectname/wsgi.py: цей файл є точкою входу для протоколу WSGI (Web Server Gateway Interface), який використовується для запуску програм Django на серверах WSGI.
- manage.py: цей файл є утилітою командного рядка, яка дозволяє взаємодіяти з проектом Django. Включає в себе різні команди для керування проектом, такі як запуск сервера розробки, створення таблиць бази даних, виконання міграцій і виконання тестів.

Створення проекту в Django зручніше, адже не треб витрачати час на відбудову

архітектури проекту і одразу почати писати потрібний код.
Також файл manage.py є зручною утилітою для виконання команд.

4.2.1 Реєстрація та логін

Невідмінно від Flask фреймворк Django має вбудовані функцію auth_login, яка записує юзера до сесії, після за допомогою request.user можна взяти вжезалогіненого користувача. В базі даних інформація про сесію зберігається в моделі django_session, яка створюється при першій міграції. Міграції в Django це свого роду контроль версії для моделей бази даних, досить зручний механізм, який дозволяєзберігати версії стану бази даних, та викликати функції при зміні стану бази даних. У Flask цієї можливості немає, хоча її також можна додати за допомогою сторонніх бібліотек.
Для створення представлення, що буде обробляти реєстрацію та логін користувача створимо окремий модуль за допомогою команди
python manage.py startapp auth
цей модуль треба додати до INSTALLED_APPS в налаштуваннях нашого проекту
після створимо в ньому файл urls.py в якому пропишемо наступне

```
from django.urls import path
from . import views
urlpatterns = [
path('', views.home),
path('login', views.login),
path('sign_up', views.sign_up),
path('logout', views.logout),
]
```

У проекті, в файлі urls.py напишемо нові шляхи з модуля auth за допомогою вбудованої функції include()
Як бачимо процес додавання нового модулю трохи довший та потребує знань у вже згенерованому проекті.
Далі реалізація sign_up та login відрізняється від аналогічної у Flask вбудованими в Django функціями, auth_login(request, user) який зберігає інформацію про об’єкт юзера

до об’єкту request, який передається до кожного запиту, та функції authenticate(request, username=username, password=password) яка повертає користувача, якщо вказаний username та пароль були правильними.

Далі робиться запис в модель User та за допомогою перенаправленню користувач попадає на наступну сторінку

Головна різниця в тому, що механізм реєстраціяї та логіну вбудований для заздалегідь створеної моделі User. У Flask потребувалося самому прописати модель користувача та перевірки паролю при логіні та використовувати додаткові модулі щоб зберігати користувача впродовж сесії.

4.2.2 Реалізація CRUD

Реалізація створення читання та видалення схожа на аналогічну в Flask, але є одна кардинальна різниця, вона полягає в різниці підходів фреймворків до реалізації ORM.

Так як Django використовує шаблон Active Record то видалення обєкту викликається не з окремої сутності, а з самого запису.

Код представлення буде виглядати ось так

```
def delete_note(request):
    data = json.loads(request.body)note_id =
    data['noteId']
    note = Note.objects.get(id=note_id)if note:
    if note.user == request.user:
                        note.delete() return
    HttpResponse(status=200)
```

Можна побачити що саме об’єкт note викликає метод delete() для моделі Note таробить видалення.

Також збереження даних в базу даних також відбувається за допомогою моделей, а не окремого об’єкту. Це можна побачити в представлені створення заміток. З цим представленням можна ознайомитися в Додатку А.

Ознайомившись можна помітити що note після заповнення викликає метод .save() який і робить запис в базу даних, додаткових структур для цього не потребється. Це виглядає більш лаконічно

4.2.3 Чат

Створення чату в Django робиться за допомогою бібліотеки channels. Для роботи з чатом окрім нового модуля створимо всередині ще файл consumers.py.
Основна різниця в порівнянні з Flask буде полягати в тому, що бібліотека channels дає нам цілий клас для роботи з протоколом ws:// wss://. Виглядає він приблизно так

```
class ChatConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        …
    async def disconnect(self, close_code):
        …
    async def receive(self, text_data):
        …
    async def chat_message(self, event):
        …
```

AsyncWebsocketConsumer клас який дає бібліотека channels, він в якомусь сенсі є абстрактним класом, при наслідувані від нього можна прописати свою реалізацію методів connect(), який відповідає за те як ми підключаємось до чату, методу disconnect(), який відповідає за обробку, коли юзер лишає чат, метод receive(), відповідає за обробку отримання повідомлення та метод send() який відправляє повідомлення в чат, в нашому випадку він використовується в методі chat_message().
В receive() ми використовуємо group_send() який відправляє отримане повідомлення всім учасникам чату, цей метод також реалізований в класі AsyncWebsocketConsumer.
Шлях, який буде використовуватися для підключення чату виглядатиме приблизно так,

```
path(r"ws/v1/chat/<int:user_id>", consumers.DialogueConsumer.as_asgi()).
```

Приклад часткової реалізації класу AsyncWebsocketConsumer наведений в Додатку А.

4.2.4 Обробка даних типу файл

Файли в Django поділяються на статичні та медіа, різниця в цих типах полягає в таких хапактеристиках:

a) Статичні файли

1. Використовуються для функціональності веб-додатку, наприклад файли CSS, файли JavaScript, зображення, шрифти та тому подібні.
2. Ці файли не змінюються часто і зазвичай доступні всімкористувачам.
3. Django надає вбудований механізм для керування статичними файлами за допомогою параметрів STATICFILES_DIRS і STATIC_URL.

b) Медіа файли

1. Частіше за все це файли завантажені користувачами, наприклад зображення профілю користувача, завантажені відео або будь- який інший файл, створений користувачами.
2. Ці файли, як правило, є унікальними для кожного користувачата не надаються спільно для різних користувачів.
3. Django надає вбудований механізм для керування медіафайлами за допомогою налаштувань MEDIA_ROOT і MEDIA_URL.

Flask на відмінно від Django не має вбудованої обробки media файлів. У Flask це потрібно оброблювати самому, в Django все вбудовано, що доволі зручно.

Збереження схоже як у Flask з різницею, що використовуємо ми media

```
filename = str(self.id) + "." + filename[filename.rfind(".") + 1 :]
path = "profile_pictures/{0}/thumb/{1}".format(self.id, filename)
file_path = os.path.join(os.path.abspath(settings.MEDIA_ROOT), path)
```

для безпеки це досі важливо розділяти media файли від статичних і Django надає такий механізм одразу, у Flask це потрібно реалізовувати самому.

4.3 Порівняння фреймворків Flask та Django

Таблиця 22 демонструє результати порівнянь фреймворків Flask та Django.

Таблиця 22. Порівняння фреймворків Flask та Django

Flask	Django
Являє собою WSGI(Web Server Gateway Interface) фреймворк, не має свою мову реалізації шаблонів, але це виправляється додатковими бібліотеками по типу Jinja2, chameleon	Повноцінно клієнт серверний фреймворк, має свою мову реалізацію шаблонів під назвою DTL
Немає вбудованого класу форми, але за допомогою бібліотеки WTForms отримає схожі можливості як і у вбудованих формах Django	Має вбудований клас форми, за допомогою якої можна зручно написати обробку заповнення полів, якоїсь моделі, також інтегрується з самими моделями
Об’єкт request є глобальною змінною	Request відправляється для кожного представлення як індивідуальна змінна
Можна обирати підхід як забажаєш	Пропонує більше монолітний підхід до розробки
Не має власної адміністративної панелі	Має свою власну адміністративну панель для зручного інтерфейсу адміна
Більш складніше для вивчення, адже потрібно працювати з багатьма бібліотеками та з їх документаціями, яки не завжди так гарно прописані	Легкий для вивчення, адже майже не потребує додаткових бібліотек для роботи, а сама документація до Django одна з найкращих
Відомі продукти написані за допомогою Flask: Netflix , Uber, Reddit	Відомі продукти написані за допомогою Django: Instagram , Spotify, YouTube

В джерелі [8] наведено додаткові спостереження щодо відмінностей в цих фреймворках.

ВИСНОВКИ

Розроблено два програмних застосунки, один за допомогою Flask інший за допомогою Django. Продемонстровано різницю в підходах до реалізації клієнт серверної архітектури за допомогою фреймворків Django та Flask. Результатом є характеристика підходів реалізації кожного з фреймворків до поставлених задач.

З відмінностей наведена різниця до підходу реалізації ORM структури. Продемонстровано, що різниця в шаблонах AciveRecord та DataMapper постійно постає в процесі розробки, і є невід'ємною частиною роботи, якщо ви використовуєте Django та Flask. У роботі показано, що DataMapper надає доволі зручний та легкий підхід, але ActiveRecord продемонстрував більш інформативний підхід. Всі ці аспекти було продемонстровано через два додатки, один з яких був розроблений за допомогою фреймворку Django, інший за допомогою фреймворку Flask.Розробка включала в себе написання за допомогою фреймворків Django та Flask такого функціоналу як реєстрацію та логін користувача, реалізацію CRUD, реалізація чату у реальному часі та реалізації обробки файлових даних фреймворками.

Архітектура клієнт-сервер є наріжним каменем сучасних обчислень. Вона є невід'ємною частиною безперебійної роботи та оптимізації спільних ресурсів, а дворівневі, трирівневі та n-рівневі моделі дозволяють створювати індивідуальні рішення для різноманітних технологічних потреб. Фреймворки Flask і Django відіграють значну роль у впровадженні та просуванні цієї архітектури. Їх відповідні властивості —простота й гнучкість Flask, а також надійність і ефективність Django — роблять їх незамінними інструментами для управління складністю взаємодії клієнт-сервер і реалізації повного потенціалу цієї архітектури.

ПЕРЕЛІКИ ДЖЕРЕЛ ПОСИЛАННЯ

1. JetBrains annual survey [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/lp/devecosystem-2021/python/.
2. Distributed Systems: Concepts and Design / G.Coulouris, J. Dollimore, T. Kindberg, G. Blair. – United States of America: Pearson Education, 2011. – 1040 с. – (Pearson Education).
3. Top Computer Languages [Електронний ресурс] – Режим доступу до ресурсу: https://statisticstimes.com/tech/top-computer-languages.php.
4. Benefit of using MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/benefit-of-using-mvc/.
5. Мартин Фаулер Шаблони ActiveRecord та DataMapper – Бостон, Массачусетс, США.: Addison-Wesley Professional, 2002. – 560 с..
6. Django documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/4.2/.
7. Flask documentation [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/2.2.x/.
8. Flask vs Django – Difference Between Them [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/flask-vs-django.html.

ДОДАТОК А

Код застосований в процесі розробки двох додатків

__init__.py Flask застосунку виглядає ось так:
db = SQLAlchemy() – створення обьекту бази даних

```
def create_app():
app = Flask(__name__)
app.config['SECRET_KEY'] = os.getenv('SECRET_KEY')
app.config["SQLALCHEMY_DATABASE_URI"] = os.getenv('DATABASE_URI')
app.config['UPLOADED_FOLDER'] = os.getenv('UPLOADED_FOLDER')
app.config['TEMPLATES_AUTO_RELOAD'] = True
return app
app = create_app() – створення застосунку
db.init_app(app)
with app.app_context():
print('Creating database')
db.create_all()
```

Щоб додати сесію до нашого застосунка у Flask треба внести додаткові зміни до нашого __init__.py файлу
__init__.py

```
# redirect after call of login_required
login_manager = LoginManager()
login_manager.login_view = 'auth.login'
login_manager.init_app(app)
@login_manager.user_loader
def load_user(id):
return User.query.get(int(id))
return app
```

Створення об’єкту Note реалізується даним представленням

```
@views.route('/', methods=['GET', 'POST'])
@login_required
def home():
if request.method == 'POST':
note = request.form.get('note', '')
if len(note) < 1:
flash('Note is too short.', category='error')
else:
new_note = Note(data=note, user_id=current_user.id)
db.session.add(new_note)
db.session.commit()
flash('Note added!', category='success')
return render_template("home.html", user=current_user)
```

За допомогою flaskwtf ми можемо створити форму, яка буде обробляти форму для зміни даних користувача

```
class UserForm(FlaskForm):
    username = StringField("Username", validators=[DataRequired()])email =
    StringField("Email", validators=[DataRequired()])
    password_hash = PasswordField('Password', validators=[DataRequired(),
    EqualTo('password_hash2', message='Passwords does not match')]) password_hash2
                                = PasswordField('Confirm Password',
    validators=[DataRequired()])
    profile_pic = FileField("Profile Picture")submit =
    SubmitField("Submit")
```

Представлені створення заміток для додатку Django.

```
@login_required
def home(request):
user_notes = Note.objects.filter(user=request.user)
if request.method == 'POST':
note = request.POST.get('note', '')
```

```
if len(note) < 1:
messages.add_message(request, messages.WARNING, 'Note is too short.')
else:
note = Note(data=note, user=request.user)
note.save()
messages.add_message(request, messages.SUCCESS, 'Note added!')
template = loader.get_template('todo/home.html')
return HttpResponse(template.render({'user_notes': user_notes}, request))
```

Реалізація класу AsyncWebsocketConsumer в одному з комерційних проектів.

```
class DialogueConsumer(AsyncWebsocketConsumer, MessagesRestrictionMixin):

async def connect(self):
self.receiver_id = self.scope["url_route"]["kwargs"]["user_id"]
user = self.scope["user"]
self.room_group_name = None
if user == AnonymousUser():
raise DenyConnection("Wrong user")
self.room_group_name = self.make_group_name((user.id, self.receiver_id))
# Join room group
await self.channel_layer.group_add(self.room_group_name, self.channel_name)
await self.accept()
…
async def disconnect(self, close_code):
# Leave room group
if self.room_group_name is not None:
await self.channel_layer.group_discard(
self.room_group_name, self.channel_name
```

```
)
# Receive message from WebSocket
async def receive(self, text_data):
try:
text_data_json = json.loads(text_data)
message_type = text_data_json.pop("type", "")
sender = self.scope["user"]
handler = MessageHandlerFactory.get_handler(
                message_type,
                text_data_json,
                sender,
                self.receiver_id,
                self.channel_layer,
            )
```