МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

ПРОЕКТУВАННЯ ТА РОЗРОБЛЕННЯ СОЦІАЛЬНОЇ МЕРЕЖІ ДЛЯ НАУКОВЦІВ

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти «бакалавр»

Виконав: студент 4 курсу 441 групи
Спеціальності: 121 Інженерія програмного забезпечення
Освітньо-професійної програми:
Інженерія програмного забезпечення
Залізко Дмитро Олександрович

Керівники: доктор педагогічних наук,
професор Круглик Владислав Сергійович,

Кандидат фізико-математичних наук, доцент
Єрмолаєв Вадим Анатолійович

Рецензент: Котова Ольга Володимирівна
доцентка, кандидатка фізико-математичних наук

Херсон – 2021

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol
XML	Extensible Markup Language
CSRF	Cross-site request forgery

ВСТУП

Актуальність. З появою інформаційних технологій та інтернету люди у всьому світі почали використовувати їх не тільки заради розваги а також і у робочих цілях.

Однією з найбільш широких сфер застосунку інформаційних технологій є соціальні мережі. Соціальна мережа, яку також називають віртуальною спільнотою, - це веб-сайт, який об'єднує людей з метою спілкування, задля того щоб ділитися ідеями та інтересами або заводити нові знайомства. На відміну від традиційних засобів масової інформації, які створюють відносно невелика кількість людей, сайти соціальних мереж містять контент що створений сотнями або навіть мільйонами різних людей. Незважаючи на те, що Facebook, Twitter та LinkedIn можуть бути першими соціальними мережами які приходять на думку, але ці популярні веб-сайти не репрезентують повний обсяг існуючих віртуальних спільнот. Підтримувати зв'язок з друзями та членами сім'ї - одна з найбільших переваг соціальних мереж. Серед таких сайтів можна виділити Facebook, MySpace. Соціальні мережі також дозволяють легко обмінюватися вмістом відео та фотографій в Інтернеті. Серед таких соціальних мереж є YouTube, Instagram, Flickr. Професійні соціальні мережі створені для забезпечення можливостей для кар'єрного зростання. Деякі з цих типів мереж забезпечують загальний форум для об'єднання професіоналів, тоді як інші зосереджені на конкретних професіях чи інтересах. Відомим прикладом такої соціальної мережі є LinkedIn. Станом на листопад 2011 року LinkedIn налічував понад 135 мільйонів учасників, що робить його найбільшою професійною мережею в Інтернеті. Учасники мають можливість будувати стосунки, встановлюючи зв'язки та приєднуючись до відповідних груп.

Тож Інтернет допомагає різним групам людей задовольняти різні потреби, як базові так і професійні. Ключовим інтересом для даної

роботи було обрано соціальні мережі для певної групи людей, а саме для наукової спільноти. Не дивлячись на те що у світі вже існують соціальні мережі для науковців, такі як ResearchGate або Academia.edu, українських аналогів таких сайтів немає. Тож з цієї причини було прийнято рішення створити українську соціальну мережу яка зможе об’єднати науковців нашої країни та допомогти їм створити не тільки комфортне середовище для соціальної комунікації, а й централізовану базу наукових публікацій та можливістю шукати, читати і коментувати роботи своїх колег.

Об'єкт дослідження - технології розробки соціальної мережі.

Предмет + - Web-додаток соціальної мережі.

Метою даної роботи є проектування та розроблення соціальної мережі.

Досягнення зазначеної мети здійснюється шляхом вирішення таких **основних завдань:**

1. Дослідити літературу по Django та HTTP
2. Налаштувати інтегроване середовище для розробки та спроектувати моделі в базі даних.
3. Проектування моделей, логіки архітектури проекту.
4. Налаштувати проект на сервері.

Структура роботи складається з вступу, двох розділів, висновку і списку літератури.

РОЗДІЛ 1
КОНЦЕПЦІЯ ПОБУДОВИ ВЕБ-ЗАСТОСУНКУ

1.1 Web-сервіси

Web-сервіс - це по суті системи, до якої можна отримати доступ через мережу, не обов'язково через інтернет. Це дає змогу розглядати, як свого роду шар стандартизації між кодом програми та клієнтськими додатками. Це створює велику перевагу - web-сервіси дозволяють незалежність платформи при інтеграції різних програм або частин більш складних додатків, тому що програма програма може використовувати web-сервіс без необхідності відповідати певній платформі, де сервіс працює. Web-сервіси можна розділити на 4 основні шари:

- Комунікаційний(communication) шар - відповідає за транспортування повідомлень за допомогою таких технологій, як протокол управління передачею(TCP), або протокол протокол передачі гіпертексту(HTTP), про який буде детальніше в наступному підрозділі.
- Пакувальний(packaging) шар - визначає спосіб структурування даних, що найчастіше є деяким типом розширюваної мови розмітки (XML), або JavaScript-нотацією об'єктів (JSON)
- Шари опису(description) та виявлення(discovery), що роблять можливим використання та пошук сервісу.

Існують два типа дизайну web-сервісів, які найчастіше використовуються в наші дні. Розглянемо їх далі.

1.2 SOAP (Simple Object Access Protocol)

SOAP дуже часто використовують, як пакувальний шар у Web-сервісах, - це полегшений протокол, призначений для обміну структурованою інформацією в децентралізованому розподіленому середовищі. Основна мова SOAP-повідомлень - це XML, що надає йому

широкі структурні можливості. Найчастіше SOAP використовує переваги HTTP, або SMTP(Simple Mail Transfer Protocol) для передачі повідомлень, однак, він може бути пов'язаний практично з будь-яким протоколом зв'язку.

Інтерфейси SOAP описують за допомогою мови опису Web-сервісів WSDL(Web Service Description Language), що налаштовує договір між клієнтом та сервером.

Найбільшими перевагами SOAP є те, що він може надати захист на рівні підприємства(enterprise-level security) та в нього вбудована обробка помилок. Біль того, завдяки різним розширенням він має можливість додавати нову функціональність досить легко.

Повідомлення SOAP складається з двох основних частин, що загорнуті в SOAP-“конверт”. А саме, в не обов'язковий заголовок SOAP та обов'язкове тіло SOAP. Блоки заголовків визначають метадані, які вказують, як слід обробляти повідомлення, наприклад, пункт призначення, термін повідомлення, інформацію про авторизацію, тощо. Тіло - це фактичний зміст повідомлення, це означає що дані, повернені значення, параметри та інша інформація, що виражена в XML формат.

Говорячи про web-сервіси SOAP, розуміється, що пакувальний шар web-сервісу використовує SOAP, як протокол. Тим не менше, SOAP сам по собі не є web-сервісом. Його також можна використовувати де-небудь ще, як протокол зв'язку(комунікації).

SOAP - це платформа для великих корпоративних інформаційних систем, де інтеграція повинна бути жорсткою. Він високо стандартизований, що робить його більш складним і потребуючим більш досвіду та знань. З іншого боку, він пропонує широкі налаштування. Незалежно від шарів, він досягає широкого використання коду і можливості дуже просто змінювати web-сервіс та дозволяє різним кінцевим точкам взаємодіяти за допомогою різних протоколів.

1.3 REST (Representational State Transfer)

Вперше REST був запропонований Роєм Томасом Філдінгом у 2000 році як частина його дисертації та став широко популярним в Інтернеті завдяки простоті використання. Вся ідея заснована на HTTP та його чотирьох методах GET, POST, PUT та DELETE (що буде детальніше буде описано в розділі 1.3). Як наслідок цього, REST обмежується використанням виключно з HTTP, що є компромісом для його простоти.

Метою REST є використання вже добре встановленого, перевіреного та загального призначення HTTP, що спрощує web-сервіси. Це можна назвати реакцією на куди складніший SOAP, який в основному складається з великої кількості стандартів, які доволі часто впроваджуються різними способами постачальниками програмного забезпечення, а саме через сумнівні та неоднозначні визначення. Це є основною слабкістю SOAP. Різні реалізації не завжди узгоджувались між собою. Така поведінка сильно підриває принципи web-сервісів про незалежність від реалізації, що і призводять до використання REST.

Незважаючи на те, що деякі аспекти важко порівняти безпосередньо з SOAP, оскільки REST - це не протокол, а архітектурний стиль, аналогічна функціональність може бути досягнута як для REST так і для SOAP. Ось чому розглядаються обидва підхода.

На відміну від SOAP, REST не обмежується використанням XML. Теоретично, REST може використовувати будь-який формат, хоча стандартним форматом вважають саме JSON. Це дозволяє об'єктам даних легко трансформуватись із представлення в пам'яті у відформатований текст в одній точці та перетворитися назад із із тексту в об'єкти даних. Хоча це не обмежується лише REST-ом. але це є аспектом, що поділяє обидва стилі. REST має перевагу в цьому відношенні, оскільки може давати відповіді у декількох форматах і

користувач може обрати з підтримуючих форматів, який йому більше подобається.

Також, важливою характеристикою REST є той факт, що сторони, що взаємодіють, не повинні пам'ятати жодного стану, оскільки все що потрібно для обробки запиту повинно бути у самому запиті. Цей аспект особливо зменшує складність, що знову ж таки піднімає якість програмного забезпечення, ремонтопридатність та масштабованість.

Отже, REST зазвичай швидший і потребує меншу пропускну здатність ніж SOAP, оскільки він використовує менш детальний JSON(на відміну від XML у SOAP). Крім того, повідомлення SOAP повинні передавати більше метаданих. Також, REST розглядається як більш вільно пов'язана архітектура, оскільки він використовує URI для доступу та логіки программ. З цього виграє розробник, отже йому не потрібно дотримуватися всієї специфікації інтерфейсу, на відміну від SOAP.

1.4 Django Framework

Django - це веб-фреймворк Python для “перфекціоністів з дедлайнами” , на якому базується шаблон дизайну MVC (Model – моделі, View – відображення, Controller – контроллер). Завдання розробки надзвичайно швидке, масштабоване, безпечне та неймовірно універсальне.

Так як, контроллер опрацьовується середою обробки і все найцікавіше в Django відбувається саме у моделях, шаблонах і відображеннях, на Django силаються як на MTV орієнтовану середу розробки.

MTV розшифровується як Model View Template(шаблон перегляду моделі) - це програмне забезпечення, паттерн, в архітектурі Джанго, який складається з трьох компонентів:

- Model, що відповідальна за базу даних(ORM)

- Template, що відповідає за рівень презентації та інтерфейс користувача
- View, що відповідає за логіку, як отримувати доступ до моделей і застосовувати відповідний шаблон. Ви можете розглядати його як міст між моделями і шаблонами.

Принцип роботи зображено на рис 1.1

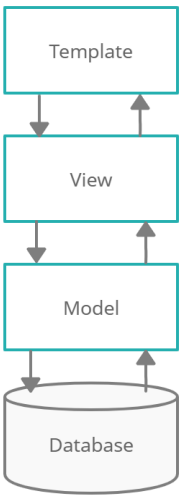

Рисунок 1.1 – принцип роботи MTV

Джанго приймає філософію вільного зчеплення - де різні частини системи слабо залежать одна від одної. Отже, одна частина системи має єдину роль, яку можна легко замінити на інші подібні функціональні компоненти, що роблять компоненти ортогональними.

1.2.1 Шари програми Django.

Програма Django складається з наступних частин:

- Request Middlewares
- URL Router
- Views
- Context Processors
- Template Renders
- Response Middlewares

Щоразу, коли запит надходить, він обробляється посередником запиту(Request middleware). Ми можемо мати кілька проміжних засобів, він знаходиться в налаштуваннях проекту, а саме в файлі settings.py. Проміжні засоби запиту Django слідують за порядком під час обробки запиту. Припустимо, якщо ми маємо запити проміжних засобів у порядку А, Б, В то запит обробляється проміжним програмним забезпеченням А, потім Б і після В. Django пропонує пакет проміжних засобів за замовчуванням. Ми також можемо писати власні проміжні засоби чи кастомизировать існуючі. Після обробки запиту він буде переданий до URL-маршрутизатору(URL Router) або диспетчеру URL-адрес.

URL-маршрутизатор приймає запит із middleware та бере шлях URL-адреси із запиту. На підставі URL-адреси маршрутизатор URL-адрес намагається зіставити шлях запиту з доступними шаблонами URL-адрес. Ці шаблони URL мають форму регулярних виразів. Після узгодження шляху URL-адреси із доступними шаблонами URL-адреси запит буде надіслано до View, яка пов'язане URL-адресою.

Як сказано вище, зараз ми перебуваємо в шарі бізнес-логіки View. Views обробляє бізнес-логіку використовуючи запити та дані запитів(дані, які надсилаються методом GET, POST і тд..). Після обробки запиту - запит надсилається в процессор контексту(context processors), за його допомогою
до запитів додають дані контексту, які допоможе Template Renderers відтворити шаблон для генерації відповіді HTTP.

Після цього запит буде відправлений до Response Middlewares для його обробки. Response Middlewares обробляє запит та модифікують чи додають інформацію заголовку(header) або про тіло (body) перед тим, як відправити його назад до браузеру. Після того, як браузер обробить його, він відобразить його кінцевому користувачу. Повний цикл зображено на рис 1.2.

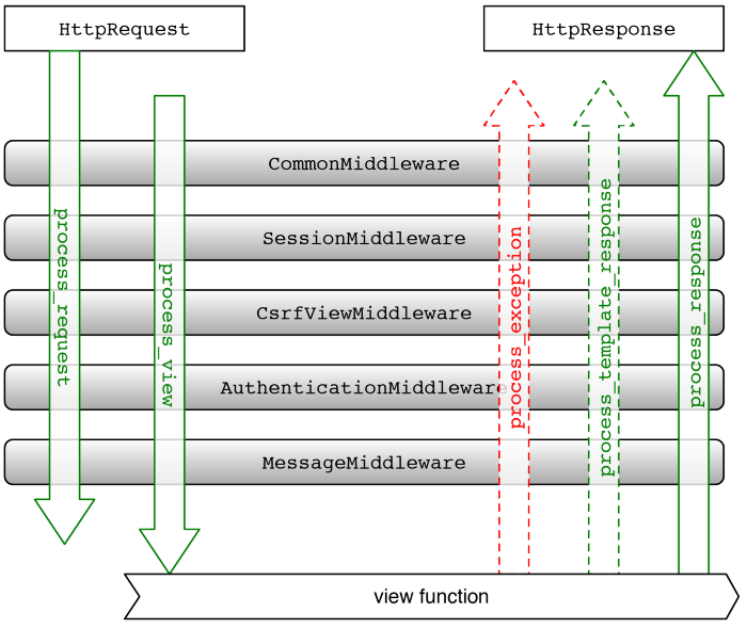

Рисунок 1.2 – Django Request/Response Cycle

1.5 Протокол передачі гіпертексту HTTP

HTTP - це протокол, який дозволяє отримувати ресурси, такі як документи HTML. Це основа будь-якого обміну даними в Інтернеті, і це протокол клієнт-сервер, що означає, що запити ініціюються одержувачем, як правило, веб-браузером. Повний документ реконструюється з різних отриманих під документів, наприклад тексту, опису макета, зображень, відео, сценаріїв тощо.

HTTP – базовий мережевий протокол, який не зберігає стан. Це означає, що сервер при здійсненні запиту до клієнта, не зберігає ніякої інформації про нього, включаючи вміст, метадані, заголовки тощо.

1.3.1 Основні методи HTTP

HTTP - визначає набір методів запиту, щоб вказати бажану дію, яку слід виконати для ресурсу. Ці методи запиту іноді називають

дієсловами HTTP. А саме, це GET (для отримання), POST (для створення), UPDATE (оновлення), DELETE (видалення).

Основні методи HTTP:

- GET - Метод GET вимагає представлення зазначеного ресурсу. Запити за допомогою GET повинні отримувати лише дані.
- HEAD - Метод HEAD вимагає відповіді, ідентичної відповіді запиту GET, але без тіла відповіді.
- POST - Метод POST використовується для подання сутності до вказаного ресурсу, часто спричиняючи зміну стану або побічні ефекти на сервері. Найчастіше використовується для створення нових даних.
- PUT - Метод PUT замінює всі поточні подання цільового ресурсу корисним навантаженням запиту.
- DELETE - Метод DELETE видаляє вказаний ресурс.
- CONNECT - Метод CONNECT встановлює тунель до сервера, визначений цільовим ресурсом.
- OPTIONS - Метод OPTIONS використовується для опису варіантів зв'язку для цільового ресурсу.
- TRACE - Метод TRACE виконує тест зворотного зв'язку із повідомленнями по шляху до цільового ресурсу.
- PATCH - Метод PATCH використовується для застосування часткових модифікацій до ресурсу.

1.3.2 Коди статусів

Код відповіді (стану) HTTP показує, чи був успішно виконаний певний HTTP запит. Коди згруповані в 5 класів:

1. Інформаційні відповіді 100 - 199
2. Успішні відповіді 200 - 299

3. Перенаправления 300 - 399
4. Клієнтські помилки 400-499
5. Серверні помилки 500-599

У таблиці 1.1 приведені основні коди.

Таблиця 1.1

Код відповіді	Назва	Опис	Версія HTTP
100	Continue	“Продовжити”. Це проміжкова відповідь, яка показує, що запрос успішно принят і клієнт може продовжити запит, або проігнорувати цей запит.	Тільки HTTP/ 1.1
101	Processing	“В процесі”. Цей код показує, що сервер получив запит і опрацьовує його, але обработка ще не завершена.	Тільки HTTP/ 1.1
200	OK	“Успішно”. Запрос успішно опрацьовано. Що саме значить “успішно” - залежить від методу HTTP, який був запитан. GET: “Отримати”. Запитаний ресурс був знайдений і переданий в тілі відповіді. HEAD: “Заголовок”. Заголовки передані у відповіді. POST: “Посилка”. Ресурс, що описує результат дії сервера на запит, переданий в тілі відповіді.	HTTP/ 0.9 та вище
201	Created	“Створено”. Запит успішно виконаний і в результаті був створений ресурс.	HTTP/ 0.9 та вище

202	Accepted	“Прийнятий”. Запит успішно прийнятий, але ще не оброблений.	HTTP/ 0.9 та вище
204	No Content	"Немає вмісту". Немає вмісту для відповіді на запит, але заголовки відповіді, які можуть бути корисні, надсилаються.	HTTP/ 0.9 та вище
302	Found	"Знайдено". Цей код відповіді значить, що запитаний ресурс тимчасово змінено. Нові зміни в URI можуть бути доступні в майбутньому. Таким чином, цей URI, повинен бути використаний клієнтом в майбутніх запитах.	Тільки HTTP/ 1.1
400	Bad Request	"Поганий запит". Ця відповідь означає, що сервер не розуміє запит через неправильний синтаксис.	HTTP/ 0.9 та вище
403	Forbidden	"Заборонено". У клієнта немає прав доступу до вмісту, тому сервер відмовляється дати належну відповідь.	HTTP/ 0.9 та вище
404	Not Found	"Не знайдений". Сервер не може знайти запитаний ресурс. Код цієї відповіді, напевно, найвідоміший з-за частоти його появи в інтернет	HTTP/ 0.9 та вище
500	Internal Server Error	"Внутрішня помилка сервера". Сервер не знає як обробити запрос.	HTTP/ 0.9 та вище

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБЛЕННЯ ВЕБ-ЗАСТОСУНКУ

2.1 Детальний опис функціоналу Web-застосунку

У Web-застосунку повинна бути можливість зареєструватися як користувач. У користувача є можливість редагувати свій персональний профіль, додавати інших користувачів до друзів, переглядати відправленні запроси до друзів (прийняти чи відклонити їх), переглядати профілі інших користувачів. Створювати пости, редагувати їх, коментувати та оцінювати.

Також у соціальній мережі буде запроваджено можливість створювати та зберігати научні статті, свої публікації в наукових журналах і книгах. Особливістю цього аспекту є те, що буде спроектовано відношення один до багатьох, а саме що одну публікацію, книгу, статтю може створювати декілько авторів.

Найбільшу увагу звернемо саме до створення персональних профілів, взаємодії користувачів та створенню постів.

Загальна увага буде приділена саме серверній розробці.

2.2 Технології, які будуть використані при створенні Web-застосунку

Для розробки проекту будуть використані різноманітні технології та інструменті. Їх опис буде в цьому підрозділі.

2.2.1 База даних SQLite

SQlite - у Django, SQLite установлена за дефолтом. Вона надзвичайно легко налаштовується у проекті за допомогою двух команд:

ру manage.py makemigrations - відповідає за створення нових міграцій на основі змін, внесених у models(моделі). Якщо застосувати дану команду на щойно створений проект, то воно створить звичайну базу даних

py manage.py migrate - відповідає за прийнятя та відмову міграцій.

Також, за допомогою команди py manage.py showmigrations можно подивитись перелік міграцій Web-застосунку та їх статус.

Особливістю SQLite є те, що Django підтримує технологію ORM(Object Relational Mapping) - це означає, що можно взаємодіяти з об'єктами в базі даних можна за допомогою методів і класів у Django, що надзвичайно полегшує створення та моделювання проекту без необхідності собсно руч писати запити SQL.

2.2.2 HTML і CSS

HTML (Hypertext Markup Language) - мова розмітки, яка дозволяє переглядати веб-сторінку у браузері таку, яка вона є. Кожна сторінка містить у собі наступні базові елементи, що зображені на рис 2.1

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
    </body>
</html>
```

Рисунок – 2.1 базові елементи

CSS (Cascading Style Sheets) - це мова , яку ми використовуємо для оформлення веб-сторінки. Вона описує спосіб відображення елементів HTML в браузері. За допомогою CSS можно контролювати оформлення одраз декількох сторінок. Всі зовнішні стилі зберігаються у файлах .css, використовуються тільки з HTML.

2.2.3 JavaScript

JavaScript (JS) - це полегшена, інтерпретована або “just-in-time” вчасно компільована мова програмування з першокласними функціями. Хоча вона найбільш відома як мова сценаріїв веб-сторінок, її

використовують також багато середовищ, що не належать до браузера, такі як Node.js, Apache CouchDB та Adobe Acrobat. JavaScript - це багатопарадигмова, однопотокова динамічна мова, що підтримує об’єктно-орієнтований, імперативний та декларативний (наприклад, функціональне програмування) стилі що робить її дуже популярною мовою програмування.

Javascript відповідає за:

- Оновлення даних на сторінці, без її перезавантаження.
- Зберігання інформації у кеші браузера.
- Гнучкий доступ до елементів на сторінці.

2.2.4 Git

Git - це популярна система контролю версій, що записує зміни у файл або набір файлів протягом деякого часу, так що ви можете повернутися до повної версії пізніше.У Git щоразу, як ви створюєте коміт, тобто зберігаєте стан вашого проекту, Git запам’ятовує як виглядають всі ваші файли в той момент і зберігає посилання на цей знімок. Для ефективності, якщо файли не змінилися, Git не зберігає файли знову, просто робить посилання на попередній ідентичний файл, котрий вже зберігається. Git вважає свої дані більш як **потік знімків**. Це дуже важлива різниця між Git та майже всіма іншими системами контролю версій. З цієї причини в Git було заново переосмислено майже кожен аспект контролю версій, які інші системи просто копіювали з попереднього покоління. Це зробило Git більш схожим на мініатюрну файлову систему з деякими неймовірно потужними вбудованими інструментами на додаток, а не просто систему контролю версій.

2.2.5 Heroku

Heroku - один з найтриваліших і популярних хмарних сервісів PaaS. Спочатку він підтримував тільки додатки Ruby, але тепер його

можна використовувати для розміщення додатків з багатьох середовищ програмування, включаючи Django.

Обрано використання Heroku з кількох причин:

- У Heroku є вільний(для хоббі) рівень, який дійсно вільний (хоча і з деякими обмеженнями)
- Як PaaS, Heroku піклується про великий веб-інфраструктуру для нас. Це значно полегшує роботу, тому що ви не турбуєтеся про сервери, балансирах навантаження, зворотних проксі або будь-який інший веб-інфраструктурі, яку Heroku надає нам під капотом.

Heroku запускає сайти Django всередині одного, або більш, ізольованих один від одного "Dynos", які є віртуальними Unix-контейнерами, які надають необхідне оточення для вашого застосування. Дані dynos повністю ізольовані і мають ефемерну файлову систему ("короткоживуча" файлова система, яка повністю очищається і оновлюється кожного разу, коли dyno перезапускается). Єдиною сутністю, яку надає dynos за замовчуванням, є додаток по конфігурації змінних. Heroku всередині себе застосовує балансувальник завантаження для розподілу веб-трафіку серед усіх "веб" - dynos. Оскільки dynos ізольовані, Heroku може масштабувати додаток горизонтально, просто додаючи більше dynos (хоча, звичайно, у вас може виникнути потреба розширити базу даних для обробки додаткових з'єднань).

2.3 Налаштування проекту

У цьому розділі розглядається питання про налаштування проекту з метою безпосередньо розпочати частину з кодуванням.

Для написання коду використовується IDE VSC.

VSC(Visual Studio Code) - це (IDE)інтегроване середовище розробки, що було створено компанією Microsoft, яке використовується для програмування майже на будь-якій мові програмування.

Першим кроком потрібно створити папку для проекту, після цього за допомогою командної строки потрібно встановити віртуальне оточення для проекту.

Virtualenv - створює ізольоване та незалежне середовища для розробки на Python. Створюється віртуальне оточення за допомогою команди:

py -m venv env - це створить наступну папку env, архітектура папки зображена на рис. 2.2

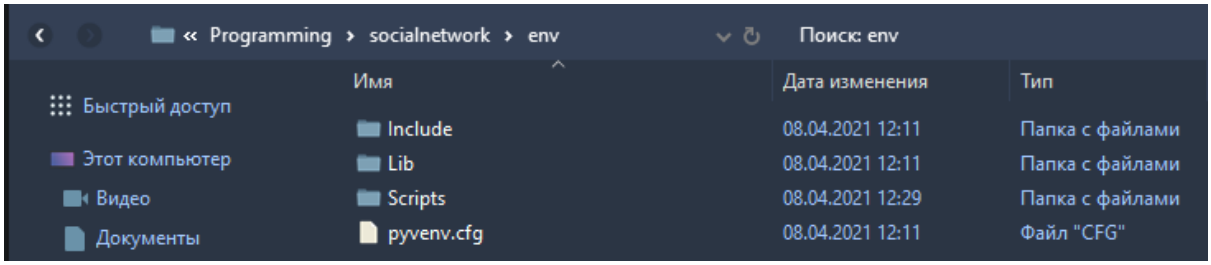

Рисунок 2.2 – архітектура папки env

Після цього, нам потрібно активувати віртуальне оточення, для цього потрібно перейти до папки Scripts за допомогою командної строки та активувати оточення, це робиться за допомогою наступних команд:

```
D:\Programming\socialnetwork\env>cd Scripts

D:\Programming\socialnetwork\env\Scripts>Activate

(env) D:\Programming\socialnetwork\env\Scripts>
```

Рисунок 2.3 – створення віртуального оточення

Як ми бачимо на рис 2.3, ми активували віртуальне оточення (env). Після цього, потрібно встановити необхідні пакети для проекту. А саме :

- Django
- Pillow
- pytz
- django-crispy-forms

Всі необхідні пакети можно додати до текстового файлу ”requirements.txt”, що ми і зробимо. Для встановлення пакетів з файлу потрібно використати наступну команду :

pip install -r requirements.txt

Отже, віртуальне оточення створено та налаштовано, а це значить, що час налаштувати Django.

Створюємо проект наступною командою:

django-admin startproject socialnetworkscientists - створює ядро проекту, що зображено на рис 2.3

```
(env) D:\Programming\socialnetwork>django-admin startproject socialnetworkscientists

(env) D:\Programming\socialnetwork>dir
 Том в устройстве D имеет метку Новый том
 Серийный номер тома: 764F-BDA4

 Содержимое папки D:\Programming\socialnetwork

08.04.2021  12:35    <DIR>          .
08.04.2021  12:35    <DIR>          ..
08.04.2021  12:11    <DIR>          env
08.04.2021  12:29                24 requirements.txt
08.04.2021  12:35    <DIR>          socialnetworkscientists
               1 файлов             24 байт
               4 папок  177 730 658 304 байт свободно
```

Рисунок 2.3 – створення Django проекту

Подальшим кроком кроком буде створення модулів чи компонентів проекту, а саме нам необхідно створити для профілів та постів. Це робимо послідуючими командами:

py manage.py startapp profiles - створюємо для профілів;

py manage.py startapp posts - створюємо для постів.

Тепер наш проект має наступну структуру, що зображено на рис 2.4 :

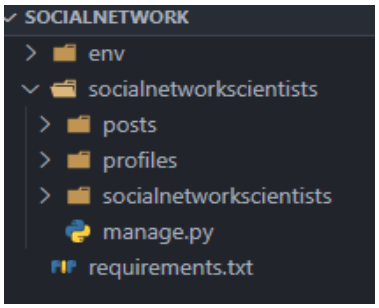

Рисунок 2.4 – Структура проекту

Також створимо суперюзера, щоб мати доступ до адмін-панелі

py manage.py createsupseruser

Та створюємо базу даних:

python manage.py makemigrations - створення міграцій для бази даних.

python manage.py migrate - прийняття міграції даних у базу даних.

Додаємо до корневого файлу settings.py створенні раніше модулі до файлу налаштування “settings.py”.

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

Перевіримо налаштування проекту наступною командою :

py manage.py runserver

Як ми можемо спостерігати, локальний сервер запустився за адресою 127.0.0.1:8000. В браузері веб-сторінка виглядає наступним чином рис 2.5:

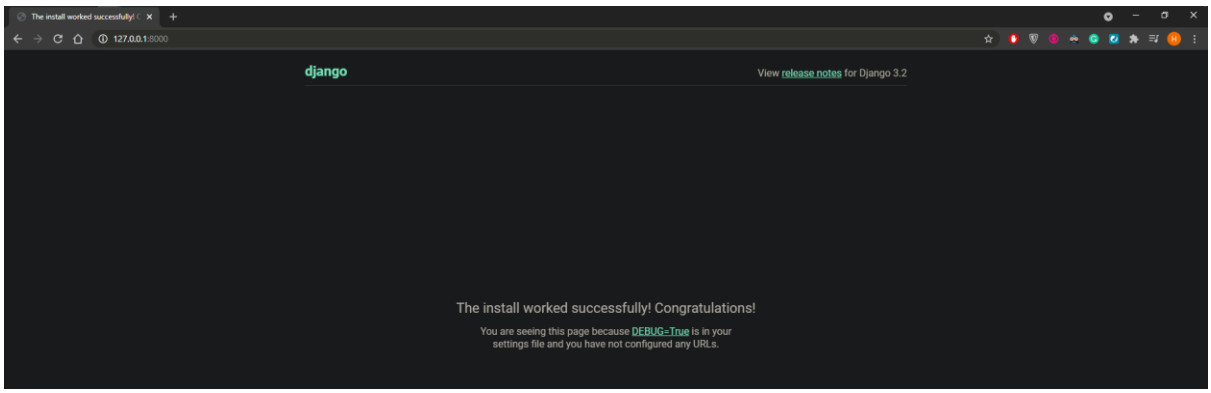

Рисунок 2.5 – Стартова сторінка Django

Наступним кроком налаштування зміна TEMPLATES в файлі settings.py додавши до нього абсолютний шлях BASE_DIR та його ім'я.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates'),],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Та додамо до settings.py конфігурації статичних(static) та медійних(media) файлів та створимо відповідні папки в корні проекту.

```
STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static_project")
]

STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_cdn", "static_root")

MEDIA_URL = '/media/'
```

MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR), "media_cdn", "media_root")

Також, додаємо до файлу urls.py наступні імпорти та модифікуємо паттерни.

```
from django.conf import settings
from django.conf.urls.static import static



urlpatterns += static(
   settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
urlpatterns += static(
   settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

В командній строці прописуємо команду
py manage.py collectstatic

На цьому налаштування проекту можна вважати успішно виконаним.

2.4 Проектування моделей для бази даних.

Django дотримується парадигми CRUD(Create, Read, Update, Delete), що э загальною для побудови веб-додатків, оскільки вона забезпечує пам'ятну основу, яка нагадує розробникам про те, як створювати повні та придатні до використання моделі.

В першій частині вже було описано про ORM концепцію. Всі моделі визначені в файлі models.py. У моделей ми можемо охарактеризувати наступні 3 види відношень:

- Many-to-many relationship(багато-до-багатьох) - багатьом користувачам може сподобатися багато публікацій.

- Many-to-one relationship(багато-до-одного) - конкретний коментар може бути пов'язаний лише з однією публікацією, але багато коментарів може бути пов'язано з даною публікацією. Як ще приклад, декілько користувачів можуть створити одну статтю чи книгу.
- One-to-one relationship(один-до-одного) - один користувач може мати лише один профіль.

Відобразимо наші моделі та їх зв'язки на наступній діаграмі класів рис 2.6

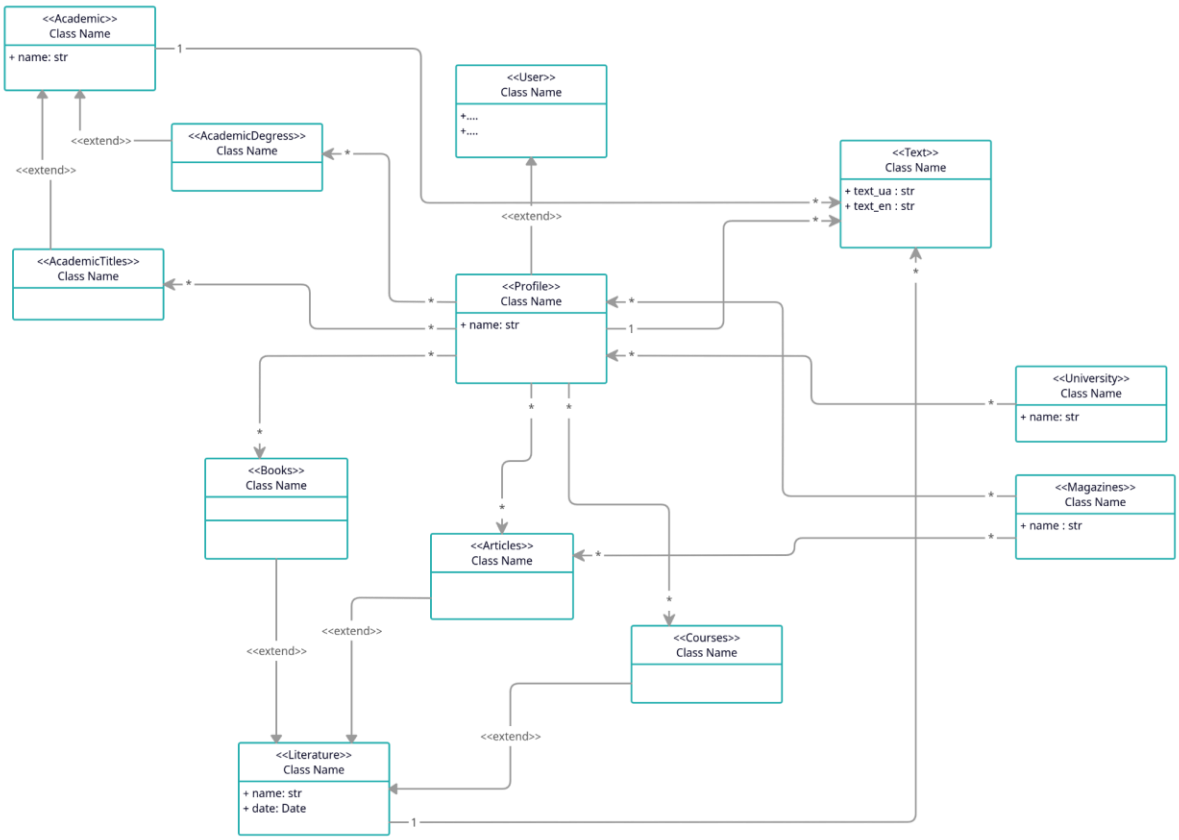

Рисунок 2.6 – діаграма класів

Як вже описувалось в попередньому абзаці, про функціональний опис додатку, в нашому додатку звернено більше уваги на відношення багато-до-багатьох. Наприклад користувач може бути кандидатом одразу декількох наук.

Отже перейдемо до реалізації на прикладі моделі Profile. В директорії profiles/models.py створимо наступний клас, що відображено на рис 2.7

```python
class Profile(models.Model):
    first_name = models.CharField(max_length=200, blank=True)
    last_name = models.CharField(max_length=200, blank=True)
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField(default='there are no bio at the moment', max_length=300)
    email = models.EmailField(max_length=200, blank=True)
    country = models.CharField(max_length=200, blank=True)
    avatar = models.ImageField(default='avatar.png', upload_to='avatars/')
    university=models.ManyToOne(University, blank=True, related_name='universities')
    academic_degree= models.ManyToManyField(Degree, blank=True, related_name='degrees')
    book = models.ManyToManyField(Book, blank=True, related_name='Books')
    slug = models.SlugField(unique=True, blank=True)
    friends = models.ManyToManyField(User, blank=True, related_name='friends')
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)
def __str__(self):
    return f"{self.user.username}-{self.created}"
```

Рисунок 2.7 – Клас Profile

Для внесення змін до бази даних потрібно виконати вже відомі команди:

1. py manage.py makemigrations - генерує sql команди для створення таблиці кожного класу, визначеного в models.py
2. py manage.py migrate - команда, відповідальна за виконання наведених вище команд (застосування міграцій). Він створює таблиці у файлі бази даних.

Всі наступні запити до бази даних будуть оброблятися через

Model Manager, це такий інтерфейс що відповідає за зв'язок з базою даних.До кожного класу в model за замовчуванням додано Manager з ім'ям об'єкта. Тобто можна отримати доступ до даних моделі Profile за наступними запитами Profile.objects.all(), Profile.objects.filter() або створити власні, що ми і будемо використовувати. Для цього лише потрібно розширити базовий клас Manager у визначеній моделі. Тому перепишемо останні строки наступним чином рис 2.8

```python
class Profile(models.Model):
    first_name = models.CharField(max_length=200, blank=True)
    last_name = models.CharField(max_length=200, blank=True)
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField(default="no bio...", max_length=300)
    email = models.EmailField(max_length=200, blank=True)
    country = models.CharField(max_length=200, blank=True)
    avatar = models.ImageField(default='avatar.png', upload_to='avatars/')
    friends = models.ManyToManyField(User, blank=True, related_name='friends')
    slug = models.SlugField(unique=True, blank=True)
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)

    objects = ProfileManager()

    def __str__(self):
        return f"{self.user.username}-{self.created.strftime('%d-%m-%Y')}"
```

Рисунок 2.8 – Model Manager для моделі Profile

Це потрібно для полегшення розробки та ініціалізування запитів до бази даних, коли ми створимо відношення між користувачами, їх публікації, пости та інше.

Також розглянемо створення моделі для публікацій на сайті , для цього необхідно перейти в вже створену директорію posts/models.py. Зміст посту буде складатися з тексту та зображення. Отже підключимо необхідну бібліотеку для валідації файлів.

from django.core.validators import FileExtensionValidator

Та реалізуємо саму модель, на рис 2.9

```python
from django.db import models
from django.core.validators import FileExtensionValidator
from profiles.models import Profile
# Create your models here.

class Post(models.Model):
    content = models.TextField()
    image = models.ImageField(upload_to='posts', validators=[FileExtensionValidator(['png', 'jpg', 'jpeg'])], blank=True)
    liked = models.ManyToManyField(Profile, blank=True, related_name='likes')
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)
    author = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='posts')

    def __str__(self):
        return str(self.content[:20])

    def num_likes(self):
        return self.liked.all().count()

    def num_comments(self):
        return self.comment_set.all().count()

    class Meta:
        ordering = ('-created',)
```

Рисунок 2.9 – Модель Posts

Описувати процес реалізації до інших моделей - недоцільно, тому що їх реалізація є подібною.

2.3.1 Реалізація views

Після створення models для Profile проекту потрібно імпортувати бібліотеки :

1. from django.shortcuts import render, redirect, get_object_or_404 - для опрацювання запитів до серверу та відправлення пакету HTTP до назад.
2. from .models import Profile - для імпорту щойно створеної моделі.
3. from django.contrib.auth.decorators import login_required - для перевірки на авторизованість.

Отже, коли необхідні бібліотеки підключені, можна приступити до реалізації. Views буде повертати html-сторінку з профілем створеного користувача з перевіркою на авторизованість в системі(за запитом до файлу urls.py в який передається view), представлено на рис 2.10

```
@login_required
def my_profile_view(request):
    profile = Profile.objects.get(user=request.user)
    form = ProfileModelForm(request.POST or None, request.FILES or None, instance=profile)
    confirm = False

    if request.method == 'POST':
        if form.is_valid():
            form.save()
            confirm = True

    context = {
        'profile': profile,
        'form': form,
        'confirm': confirm,
    }

    return render(request, 'profiles/myprofile.html', context)
```

Рисунок 2.10 – posts/views.py

Для реалізації views до модуля Posts потрібно додати такі бібліотеки:

from .forms import PostModelForm, CommentModelForm - для створення форми, яка дозволяє користувачу надсилати сам пост, та комментарі до нього.

from django.views.generic import UpdateView, DeleteView - для редагування та видалення постів.

Реалізація views для Posts зображена на рис 2.12, після опрацювання моделі нам буде повертати HTML-сторінку для постів(згідно з аналогію для Profiles - за запитом до файлу urls.py в який передається view), наглядно зображено буде на рис 2.11

```python
@login_required
def post_comment_create_and_list_view(request):
    qs = Post.objects.all()
    profile = Profile.objects.get(user=request.user)

    # initials
    p_form = PostModelForm()
    post_added = False

    profile = Profile.objects.get(user=request.user)

    if 'submit_p_form' in request.POST:
        print(request.POST)
        p_form = PostModelForm(request.POST, request.FILES)
        if p_form.is_valid():
            instance = p_form.save(commit=False)
            instance.author = profile
            instance.save()
            p_form = PostModelForm()
            post_added = True
    context = {
        'qs': qs,
        'profile': profile,
        'p_form': p_form,
        'post_added': post_added,
    }

    return render(request, 'posts/main.html', context)
```

Рисунок 2.11 – views для Posts

Приклад розробленої сторінки профілю зображено на рис 2.12

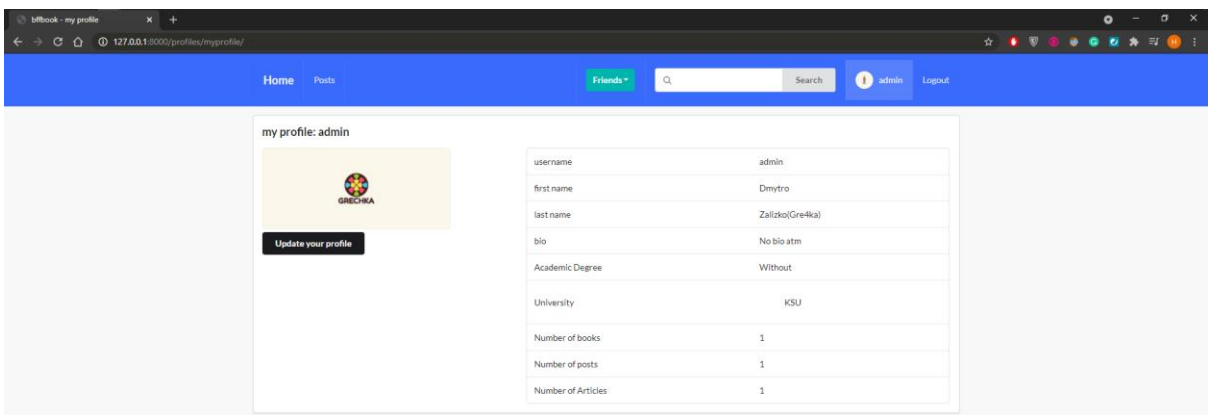

Рисунок 2.12 – Сторінка профілю

Приклад розробленої сторінки постів зображено на рис 2.13

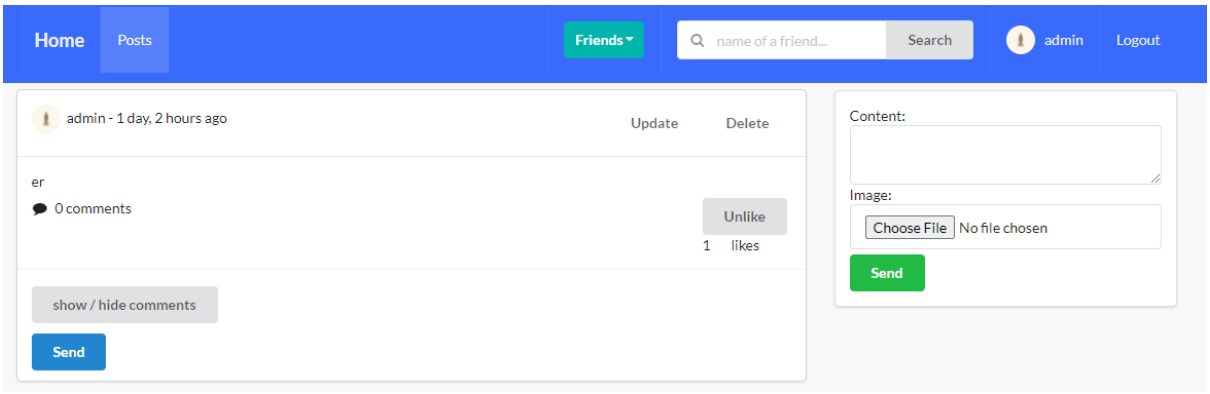

Рисунок 2.13 – Сторінка з постами та їх створенням

2.5 Налаштування проекту на платформі Heroku

Підготовка веб-додатка до публікації відбувається наступним кроком в файлі settings.py :

- DEBUG - при розгортанні сайту повинен бути установлен в False(Debug=False). Тим самим зупинив відображення виводу отладочной інформації.
- SECRET_KEY - Це велике випадкове число, яке використовується для захисту від CSRF. Важливо, щоб ключ, який використовується в продакшені, не вказувався у вихідному коді, та / або не запитувався з іншого сервера. Django рекомендує розміщувати значення ключа або у змінній оточення, або в файлі з доступом тільки на читання.

Для розгортання серверу на Heroku - першим кроком потрібно створити у корні проекту файл Procfile та додати до нього наступний текст “web: gunicorn locallibrary.wsgi --log-file -”. «Web:» повідомляє Heroku, що це веб динамічний і може бути відправлений HTTP-трафік. Процес, який почнеться в цьому динамічно, - це gunicorn, який є популярним сервером веб-додатків, який рекомендує Heroku. Ми запускаємо Gunicorn, використовуючи конфігураційну інформацію в

модулі locallibrary.wsgi (створений за допомогою нашого скелета програми: /locallibrary/wsgi.py). Для наступного кроку нам необхідно встановити деякі пакети:

- pip install gunicorn - Це чистий python http сервер для WSGI додатків які можуть запускати безліч паралельних python процесів
- pip install dj-database-url - для взаємодії з базою даних.
- pip install psycopg2(Python Postgresql database support) - Django потребує psycopg2 для роботи з базами даних Postgres, що використовує Heroku.
- pip install whitenoise - для коректної роботи статичних файлів.

Створимо проект Heroku:

heroku create socialnetwork

Підключимо Heroku до Git:

heroku git:remote –a socialnetwork

Додамо файли в git та внесемо зміни

git add

git commit -m "Initial commit"

Завантажимо проект до Heroku та зробимо міграції бази даних:

git push heroku master

heroku run python manage.py migrate

Після чого можна відкрити браузер на новоствореному веб-сайті використовуючи команду:

heroku open

Це був останній крок створення соціальної мережі за допомогою Django.

ВИСНОВКИ

У ході кваліфікаційної роботи були виконані наступні завдання:

1. Досліджено літературу про Web-сервіси, Django, HTTP, що було представлено та описано в першому розділі.
2. Налаштовано віртуальне середовище Virtualenv, досліджено та встановлено всі необхідно пакети, фреймворк Django для створення соціальної мережі.
3. Спроектовано та створено моделі для бази даних SQLite. Створено та опрацьована логіка архітектури проекта. Що детально описано у другому розділі.
4. Налаштовано проект на платформі Heroku. Що дає можливість кожному бажаючому використовувати соціальну мережу.

Отже, можна стверджувати, що мета роботи була виконана.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans., 2011.
2. Arana S. Web Services Platform Architecture / S. Arana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. – New Jersey : Prentice Hall, 2005
3. Список кодів HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP.
4. Kumar S. Django: Request/Response Cycle [Електронний ресурс] / Sarthak Kumar. – 2019. – Режим доступу до ресурсу: https://medium.com/@ksarthak4ever/django-request-response-cycle2626e9e8606e
5. SQLite Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://www.sqlite.org/docs.html.
6. HTML/CSS [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://en.wikibooks.org/wiki/HyperText_Markup_Language/CSS.
7. JavaScript [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/JavaScript
8. Heroku [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Heroku.
9. Git [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Gi
10. Документація Django 3.0 на російській мові [Електронний ресурс]. – Режим доступу до ресурсу: https://django.fun/docs/django/ru/3.0/.
11. Nield T. Getting Started with SQL: A Hands-On Approach for Beginners / Thomas Nield., 2016. – 134 с.
12. DataTables [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://datatables.net.

13. Gunicorn - Python WSGI HTTP Server for UNIX [Електронний ресурс]. – Режим доступу до ресурсу: https://gunicorn.org.
14. Getting Started on Heroku with Python [Електронний ресурс]. – Режим доступу до ресурсу: https://devcenter.heroku.com/articles/gettingstarted-with-python
15. Дронов В. А. Django: практика создания Web-сайтов на Python / В. А. Дронов. – Петербург: Профессиональное программирование, 2016. – 528 с.
16. Библиотека Django [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/w3prog/GoodTravel/wiki/Библиотека-Django.
17. Введение Django Girls Tutorial [Електронний ресурс]. – Режим доступу до ресурсу: https://tutorial.djangogirls.org/ru/
18. Getting Started on Heroku with Python [Електронний ресурс]. – Режим доступу до ресурсу: https://devcenter.heroku.com/articles/getting-started-with-python.
19. DRF Authentication [Електронний ресурс]. – Режим доступу до ресурсу: https://www.django-rest-framework.org/api-guide/authentication/.
20. Arana S. Web Services Platform Architecture / S. Arana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. – New Jersey : Prentice Hall, 2005.
21. Філдінг Р. Т. Архітектурні стилі та дизайн архітектури програмного забезпечення / Р. Т. Філдінг. – Ірвін : каліфорнійський університет, 2000.

ДОДАТКИ

Додаток А

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Залізко Дмитро Олександрович, учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

- не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;
- не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

_____	_____	_____
(дата)	(підпис)	(ім'я, прізвище)