МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
«_____»_____ 2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Онлайн середовище для організації спільної роботи над проєктами

Виконавець: Лузанов Є. О.

Керівник: к.т.н., доцент Росінська Г. П.

Нормоконтролер: : Тупота Є. В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Литвиненко О.Є.
«_____»_____ 2021 р.

ЗАВДАННЯ
на виконання дипломного проєкту
Лузанова Євгена Олександровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проєкту) Онлайн середовище для організації спільної роботи над проєктами
затверджена наказом ректора від « 04 » лютого 2021 р. № 135/ст.

2. Термін виконання роботи (проєкту): з 17.05.2021 по 20.06.2021

3. Вихідні дані до роботи (проєкту): програмний продукт розробляється на мові програмування *Python* та фреймоврку *Flask*

4. Зміст пояснювальної записки:

1) Аналіз предметної області;

2) Дослідження технологій розробки веб-додатків;

3) Реалізація веб-додатку.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) зображення головної сторінки веб-додатку;

2) форми авторизації користувача;

3) зображення панелі адміністратора;

4) форма додавання задачі;

5) зображення сторінки завдання.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомитись з постановкою задачі	17.05.2021	
2	Ознайомитись з літературою за темою роботи	18.05.2021-19.05.2021	
3	Аналіз мови програмування та її бібліотек	20.05.2021-28.05.2021	
4	Написати розділ 1	29.05.2021-31.05.2021	
5	Розробка клієнтського додатку	01.06.2021-02.06.2021	
6	Написати розділ 2.	03.06.2021-04.06.2021	
7	Розробка серверного додатку	05.06.2021-08.06.2021	
8	Написати розділ 3.	09.06.2021-10.06.2021	
9	Оформити пояснювальну записку	11.06.2021	
10	Підготувати графічний демонстраційний матеріал та доповідь	12.06.2021	

7. Дата видачі завдання: “ 17 " травня 2021 р.

Керівник дипломної роботи (проєкту)_____ Росінська Г.П.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання_____ Лузанов Є. О.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту “Онлайн середовище для організації спільної роботи над проєктами ”: 49 с., 23 рис., 1 табл., 17 літературних джерел, 1 додаток.

БАЗА ДАНИХ, *SQL*, *SQLITE*, ОНЛАЙН СЕРЕДОВИЩЕ, СУБД, ВЕБ-ДОДАТОК.

Об’єкт дослідження – веб-додаток.

Предмет дослідження – онлайн середовище для організації спільної роботи над проєктами.

Мета дипломного проєкту – дослідження можливостей створення веб-додатку засобами *Flask Framework*.

Методи дослідження – застосування фреймворку *Flask* для створення веб-додатку.

Прогнозні припущення щодо розвитку об’єкта дослідження – розробка працюючої програми та використання її для організації роботи над проєктами.

Результати дипломної роботи рекомендується використовувати при організації роботи над груповими проєктами.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ – програмне забезпечення

СУБД - система управління базами даних

БД – база даних

ООП - об'єктно-орієнтоване програмування

ВСТУП

Найважливіший ресурс для досягнення результату проекту - це люди, які залучені до проекту. Команда проекту - це група людей , які володіють необхідними, для досягнення єдиної мети, знаннями і вміннями, і спільно відповідають за досягнення результату. Внутрішнім людським ресурсом проекту в основному є команда або ті люди, які залучені до роботи протягом усього терміну проекту і роботу яких координує керівник проекту. У кожного члена команди певна мета, і всі знають, що від кожного очікувати. Члени команди доповнюють один одного, плавно обмінюються різними ролями. Діяльність координується. Кожен допомагає один одному. Виникає ініціатива і почуття відповідальності у всіх членів команди.

Для планування діяльності та організації командних дій необхідні такі компоненти:

- підбір людей виходячи із завдань;
- постановка цілей команди;
- розподіл ролей в команді (відбувається виходячи з процесів і процедур роботи);
- особистісна гармонія;
- внутрішня підтримка команди при виконанні завдань;
- ефективне використання ресурсів;
- організація взаємного спілкування між членами команди і керівниками.

Для ефективної організації спільної роботи під час розробки проекту слід виконувати наступні вимоги:

– конфіденційність – багато компаній мають важливу інформацію або особливі знання, завдяки яким вони домоглися свого успіху. Цінність такої інформації безпосередньо визначає ринкову вартість їх активів. Такою

інформацією може бути і список ділових контактів і клієнтів, методи та кроки розробки проекту;

– технічна можливість швидко отримати задачу – розподіл задач серед співробітників, у процесі розробки проекту, займає велику кількість часу. Слід назначити кожного працівника на відповідне для його спеціалізації місце та обмежити доступ до ресурсів, які заважають виконанню завдання;

– чіткий розподіл ролей і обов'язків. Одна з найсерйозніших перешкод для ефективної командної роботи – неправильне уявлення про те, в чому полягають обов'язки окремих учасників команди. Кожен співробітник повинен розуміти, чого від нього чекають, які саме завдання він повинен виконати і в який термін. Кожен повинен нести відповідальність за певну частину проекту і розуміти, яку важливість представляє його ділянка роботи для досягнення спільної мети;

– загальний обмін інформацією між співробітниками. Щоб командна робота була по-справжньому ефективною, керівнику слід ретельно відстежувати результати кожного учасника. Знаючи про прогрес кожного співробітника і про хід досягнення спільної мети, він зможе внести в робочий процес необхідні корективи.

На сьогоднішній день велику популярність набирає використання різних онлайн-сервісів: соціальні мережі, інтернет магазини , інформаційні, навігаційні, фінансові. Їх використання покращує та пришвидшує роботу користувачів: замість очікування в черзі, користувач може в пару кліків замовити продукти додому , спілкування з близькими та друзями можливо на великій дистанції , а робота займає менше часу.

Тому для реалізації середовища організації роботи над проектами було обрано веб-додаток. Він дозволяє поєднати динамічність та гнучкість процесу організації розробки проектів. У сучасних умовах розвитку інтернет-технологій, майже кожна людина має доступ до інтернету. А стрімкий розвиток технічних засобів робить доступним використання пристроїв з підтримкою браузерів та іншого програмного забезпечення для використання інтернету.

Об’єкт дослідження – веб-додаток.

Предмет дослідження –онлайн середовище для організації спільної роботи над проєктами.

Мета дипломного проєкту – дослідження можливостей створення веб-додатку засобами *Flask Framework*.

Відповідно до даної мети необхідно вирішити наступні завдання:

– Дослідити технології розробки веб-додатків.
– Розробити клієнтський додаток із зручним та зрозумілим інтерфейсом.
– Спроектувати повноцінну базу даних додатка.
– Розробити серверний додаток на мові *Python*.
– Зв’язати роботу бази даних із серверним додатком.
– Зв’язати роботу серверного додатку із клієнтським.
– Впровадити сповіщення за електронною поштою.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Веб-додатки

Веб-додаток - це додаток, що працює на платформі *Web*, тобто використовує для взаємодії з користувачем веб-сервер, працюючий по протоколу *HTTP* і браузер, що інтерпретує сторінки *HTML*.По-іншому можна сказати, що це певний сайт, зміст якого змінюється динамічно на основі взаємозв'язку з користувачем . Логіка веб-додатка розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є крос-платформними сервісами. Найбільш відомі приклади веб-додатків: веб-пошта(*http://mail.ru*), інтернет магазини (*http://amazon.com*), онлайн аукціони (*http://ebay.com*). Але область застосування веб-додатків набагато ширша , ніж електронний бізнес, вони використовуються в багатьох наукових та комерційних областях.

1.2. Особливості веб-додатків

1.2.1. Незалежність від операційної системи клієнта

Додаток може працювати на будь-якій операцій системі клієнта. Достатньо встановити браузер, для відображення інтерфейсу додатка. Але в залежності від реалізації *DOM, CSS, HTML* та інших деталей у браузері, існують різні варіанти поведінки програми.

1.2.2. Простота використання

На сьогоднішній день Інтернетом користується 4,66 мільярда чоловік, що складає 59,5 % від загального населення планети. Тому поведінка веб-додатків будується таким чином, щоб будь-який користувач зміг одразу повноцінно функціонувати із системою, не вимагаючи додаткових теоретичних знань . Розробники намагаються робити упор на синдикацію, а не координацію. Прості веб-сервіси - як сервіси на базі *REST* та *RSS* - займаються синдикацією даних, не намагаючись контролювати дані, що відбувається з інформацією на іншому кінці ланцюжка.

1.2.3. Великий оборот користувачів

Доступ до веб-ресурсу є відкритим для усіх користувачів Інтернету. Кількість користувачів є необмеженою, а кількість онлайн користувачів визначається характеристиками серверу на якому побудован додаток. При великому напливі користувачів виникає навантаження на сервер , що веде до затримок при передачі даних, а іноді і до падінню самого сервера.

1.2.4. Великі масиви даних

Бази даних які спроможні зберігати великі об’єми інформації є необхідною складовою веб-додатків. Бази даних надають інструменти для організації, додавання, пошуку, оновлення, видалення та виконання обчислень над даними. У більшості випадків сервери веб-додатків безпосередньо спілкуються з серверами завдань. Крім того, у кожної серверної служби може бути відповідна база даних, ізольована від решти частини програми.

1.2.5. Стійкість до атак

Оскільки доступ до веб-додатку є відкритим , кожний користувач може відправити дані , які можуть нашкодити або зупинити роботу серверу. Щоб запобігти відправці зайвих або шкідливих даних, у розробці веб-додатків використовують валідатори. Валідатор— комп'ютерний сервіс, програма, об'єкт, функція або оператор, який перевіряє відповідність деяких даних вимогам до типу, вмісту, формату або синтаксису. Процес перевірки називають валідацією, а дані, що відповідають вимогам — валідними.

1.3. Переваги та недоліки використання веб-додатків

До переваг використання веб-додатків відносяться:

– кросплатформеність - веб-додатки працюють на мобільних і десктопах, дозволяючи охопити максимально широку аудиторію;

– зручність оновлення - при оновленнях програми всі користувачі відразу ж отримують доступ до останньої версії вашого ПЗ;

– інтеграція - гнучкі можливості інтеграції з усіма сервісами, що підтримують обмін даних через *API*;

– ефективність - завдяки автоматизації внутрішніх процесів впровадження веб-додатки робить бізнес більш ефективним;

– єдина база даних- веб-додатки можуть виступати центральною базою даних для всіх інших сервісів і додатків компанії.

Недоліки використання веб-додатків:

– ослаблений захист інтелектуальної власності- велика частина коду програмного забезпечення є у відкритому доступі, а способів захистити дані мало. Стандартна функція браузера “Переглянути вихідний код” дозволяє легко скопіювати будь-яку веб-сторінку;

– великий час реакції на дії користувача – при формуванні сторінок сервер кожен раз звертається до бази даних, щоб отримати необхідну інформацію. Кожен запит займає певний час, і чим більше запитів, тим більше загальний час генерації сторінки;

– несумісність браузерів – для коректного відображення інформації , треба зазначити усі можливі варіанти використання веб-додатків у різних браузерах, що є доволі складною задачею.

1.4. Структура веб-додатку

Клієнтська частина програми - це скрипти, написані на мові програмування *Javascript* (*JS*) і виконувані в браузері користувача. Для реалізації клієнтської частини використовують *HTML, CSS, JavaScript, Ajax* . *HTML* застосовується для розмітки веб-сторінок. Вона потрібна браузерам, які перетворять гіпертекст і виводять на екран сторінку в зручному для людини форматі. *CSS*-стилі незамінні при оформленні сторінок сайтів: в одному файлі містяться відомості про відображення всіх елементів документа. *JavaScript* використовують для додання інтерактивності веб-сторінок.

Серверна частина веб-додатку. Веб-браузери взаємодіють з веб-серверами за допомогою гіпертекстового транспортного протоколу (*HTTP*). Коли користувач функціонує з додатком, а саме, заповнення реєстраційної форми, переходячи за посиланням або використання пошуку, *HTTP*-запит відправляється з вашого браузера на цільової сервер . Запит включає в себе *URL*, який визначає обраний ресурс, метод, який визначає необхідну дію (наприклад, отримати, видалити або опублікувати ресурс) і може включати додаткову інформацію, закодовану в параметрах *URL*, як *POST* запит (дані, відправлені методом *HTTP POST*) або в cookie-файлах. Веб-сервери приймають клієнтські запити, обробляють їх і відправляють веб-браузеру за допомогою *HTTP* повідомлення відповідь . Відповідь містить про стан *HTTP* повідомлення

та відповідний йому код. Усього Існують 5 видів кодів станів *HTTP* повідомлення:

– інформаційні - цей клас інформує про процес передачі. Зазвичай клієнт ігнорує ці повідомлення, або не відправляє відповідь серверу. Код складається з трьох цифр, перша цифра характеризує клас повідомлення та дорівнює “ 1 “, останні цифри характеризують тип повідомлення. Наприклад код 100 (*Continue*) - сервер задоволений початковими відомостями про запит, клієнт може продовжувати пересилати заголовки;

– успіх –цей клас інформує про успішний прийом та обробку запиту клієнта. Перша цифра дорівнює “2”. Найбільш частішими прикладами є 200(*Ok*), 201 (*Created*) , 202 (*Accepted*);

– перенаправлення – клас кодів , які повідомляють клієнта що для успішного виконання операції необхідно зробити інший запит, як правило, по іншому *URL*. Усього існує 5 кодів : 301, 302, 303, 305, 307;

– помилка клієнта- коди групи для вказівки помилок з боку клієнта. У відповідь клієнт отримує гіпертекстове посилання з причиною помилки. Перша цифра “4” характеризую цей клас;

– помилка сервера- помилки які починаються з цифри “5”, повідомляють про випадки необроблених винятків при виконанні операцій на стороні сервера.

База даних – програмне забезпечення , яке функціонує разом із сервером та, зберігає або відправляє дані. Бази даних використовуються для динамічності веб-додатків: вони дозволяють формувати унікальні веб-сторінки під час запиту користувача та відправляти їх клієнту. Однією з найчастіших областей застосування баз даних у веб-додатках є функція реєстрації користувачів, щоб зберегти інформацію про користувача та надати йому відповідний контент.

1.5. Висновки до розділу

В даному розділі було розглянуто основні поняття веб-додатку, його властивості, визначена загальна архітектура, переваги та недоліки використання. Враховуючи усі факти, використання веб-додатків є перспективним та оптимальним рішенням для інформаційних та практичних задач. Динамічна робота програми дозволяє керувати великим об’ємом даних , використовуючи серверні бази даних , в комфортному для клієнта вигляді. Можливість створювати інтерактивний інтерфейс покращує сприяння інформації та швидкість усвідомлення суті поставленої задачі. Функція комунікації співробітників робить процес виконання завдання швидшим та легше. А здатність функціонувати на будь-якій операційній системі охоплює велику частину потенційних користувачів.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ-ДОДАТКІВ

2.1. *HTML*

HTML, що означає *Hyper Text Markup Language* (мова гіпертекстової розмітки) - це технологія, яка дозволяє задавати структуру розташування візуальних елементів (іноді її називають користувальницький інтерфейс) веб-додатку. Текстові документи, що містять код на мові *HTML* (такі документи зазвичай мають розширення *.html* або *.htm*), обробляються спеціальними додатками, які відображають документ в його форматованому вигляді. Такі додатки, звані «браузерами» або «інтернет-оглядачами», зазвичай надають користувачеві зручний інтерфейс для запиту *web*-сторінок, їх перегляду (і виведення на інші зовнішні пристрої) і, при необхідності, відправки введених користувачем даних на сервер. Найбільш популярними на сьогоднішній день браузерами є *Internet Explorer, Mozilla Firefox ,Safari, Google ,Chrome, Opera*.

За допомогою *HTML* електронний документ може бути розділений на окремі частини, вміст кожної визначається індивідуально , в залежності від міркувань розробника. Функції *HTML* дозволяють прикріпляти мітки, ілюстрації, аудіо та відео-фрагменти, обирати різні рівні заголовків, списки, таблиці та багато іншого.

HTML-документ складається з трьох частин:

- оголошення типу документа;
- заголовка документа;
- тіла документа.

Елементи *HTML* називаються тегами. Теги бувають двох типів: парні й одиночні. Парні теги можуть містити в собі допоміжні теги, а одиночні теги можуть використовуватися самостійно. Елемент розмітки складається з трьох частин : початковий тег, зміст, закриваючий тег.

Елементи *HTML* можна умовно поділити на категорії:

– структурні, які задають структуру документа (наприклад, *html*, *head*, *body* і *title*);

– блокові елементи, призначені для форматування цілих текстових блоків (наприклад, *div*, *h1*, *p*, *pre*);

– текстові елементи, які задають розмітку тексту (*em*, *dfn*,*code*, *samp*);

– спеціальні елементи порожнього рядка (*br*, *hr*, *nobr*), якірний елемент a, вбудовані елементи (*embed*, *img*, *map*), елементи форм (*input select*), елементи таблиць (*table*) та інші.

Часто елементи розмітки містять додаткові елементи – атрибути. Допустимі різні атрибути, які розділяються між собою пробілом. Втім, є теги без будь яких додаткових атрибутів. Умовно атрибути можна розділити на обов'язкові, неодмінно повинні бути, і необов'язкові (їх додавання залежить від мети застосування тега). Атрибут записується після імені елемента розмітки перед закриваючою дужкою і складається, як правило, з пари «ім'я атрибут та – значення».

Головними елементами розмітки є структурні:

– <*head*> - містить інформацію про заголовок сторінки, опис, ключі слова для пошуку у браузері, закодовані скріпти, а також посилання на конфігураційні файли із властивостями;

– <*body*> - призначений для відображення основної інформації документа. У цьому тегу формується структура відображення веб-сторінки;

– <*html*> - тег є контейнером, який містить в собі весь вміст веб-сторінки, включаючи теги <*head*> і <*body*>.

До елементів заголовка відносяться :

– <*title*> - потрібен для відображення тексту в рядку заголовка веб-браузера;

– <*style*> - потрібен для вставки таблиці стилів *CSS*. Документ може містити декілька тегів;

– <*meta*> - технічна інформація про сторінку. Відповідає за пошукові слова для браузера;

– <*script*> - дозволяє додавати до веб-сторінки сценарії на мові *JavaScript*. Може містити або фрагмент коду сценарію , або посилання на файл з кодом.

До елементів тіла документа відносяться:

– <*h1-6*> - тег дозволяє додавати різні заголовки до веб-сторінки. Номер характеризую ієрархію підзаголовків;

– <*p*> - розбиває текст на абзаци;

– <*div*> - виділяє окремий блок, вміст якого керуються за допомогою стилів;

– <*ul*> - створює невпорядкований список. Між початковим та кінцевим тегом потрібен знаходиться тег <*li*>, що позначає окремі пункти списку;

– <*ol*> -створює впорядкований список. Між початковим та кінцевим тегом потрібен знаходиться тег <*li*>, що позначає окремі пункти списку. Атрибути *start* та *type* визначають особливості цього списку. *Start* дозволяє обрати перше число з якого почнеться нумерація списку. *Type* визначає стиль нумерації пунктів;

– <*a*> - потрібен для розміщення гіперпосилання у тексті;

– <*img*> - додає зображення до сторінки. Обов’язковим атрибутом є *src* в якому визначається адресу графічного файлу. До найбільш популярних графічних форматів відносять ***JPEG, PNG, GIF***;

– <*map*> - дозволяє створити зображення карти з активними областями.

2.2. *CSS*

CSS - це мова стилів. Вона використовується для того, щоб надати сторінкам на *HTML* - фундаментальна мова Всесвітньої павутини - досконалий вид. Каскадні таблиці стилів, або *Cascading Style Sheets (CSS)*, забезпечують творчу свободу в розмітці і дизайні веб-сторінок. Користуючись *CSS*, можливо прикрасити текст сторінок привабливими заголовками, буквицями і рамками, як в барвистих глянцевих журналах. Можна точно розмістити і позиціонувати зображення, зробити колонки і створити банери, виділити посилання динамічними ефектами. Можна також домогтися поступової появи і зникнення елементів, переміщення об'єктів по сторінці або повільної зміни кольору кнопки при проходженні над нею покажчика миші.

Визначення стилю в *CSS*, який встановлює зовнішній вигляд будь-якого елемента (Фрагмента) веб-сторінки, - це всього лише правило, яке повідомляє браузеру, що і яким чином форматувати: змінити колір шрифту заголовка на синій, виділити фото червоною рамкою, створити меню шириною 150 пікселів для списку гіперпосилань. Фактично визначення стилю складається з двох основних елементів: самого елемента веб-сторінки, який безпосередньо підлягає форматуванню браузером, - селектору, а також команд форматування - блоку оголошення. Селекторами можуть бути заголовок, абзац тексту, зображення і т.д.

Селектор повідомляє браузеру, до якого елементу або елементів веб-сторінки застосовується стиль: до заголовку, абзацу, зображенню або гіперпосиланню.

Блок оголошення. Код, розташований відразу за селектором, містить всі команди форматування, які можна застосувати до цього селектору. Блок починається з відкриваючої фігурної дужки і закінчується закриваючою.

Оголошення. Між відкриваючою і закриваючою фігурними дужками блока оголошення можна додати одне або кілька оголошень - команд форматування. Кожне оголошення має дві частини - властивість і значення.

Двокрапка відокремлює ім'я властивості від його значення, і все оголошення закінчується крапкою з комою.

Властивість. Каскадні таблиці стилів пропонують великий вибір команд форматування, званих властивостями. Властивість є слово або кілька написаних через дефіс слів, що визначають конкретний стиль. У більшості властивостей є відповідні прості для розуміння імена, такі як *font-size, margin-top, background-color* і т.д. (в перекладі з англійської: розмір шрифту, верхній відступ, колір фону). Після імені властивості потрібно додати двокрапку, щоб відокремити його від значення.

Таблиці стилів бувають двох видів - внутрішні і зовнішні – в залежності від того, де визначена інформація про стилі: безпосередньо в самій веб-сторінці або в окремому файлі, пов'язаному з веб-сторінкою.

Зовнішні таблиці стилів зосереджують всю інформацію про стилі в єдиному файлі, який потім приєднується до сторінки, написавши для цього всього один рядок коду. Можливо приєднати одну і ту ж зовнішню таблицю стилів до кожної сторінки сайту, створюючи таким чином єдиний дизайн. А оновлення зовнішнього вигляду всього сайту буде полягати лише в редагуванні єдиного текстового файлу - зовнішньої таблиці стилів. Зовнішні таблиці стилів дозволяють веб-сторінкам завантажуватися швидше. Коли використовується такі таблиці стилів, веб-сторінка містить тільки *HTML*-код, без коду громіздких вкладених таблиць для форматування, без елементів *font* і подібних, без коду внутрішніх каскадних таблиць стилів. Окрім того, коли браузер завантажить зовнішню таблицю стилів, він збереже цей файл на клієнтському комп'ютері відвідувача веб-сторінки (в спеціальній системній папці, яку називають кешем) для швидкого доступу до нього. Коли користувач веб-сторінки переходить до інших сторінок сайту, які використовують ту ж зовнішню таблицю стилів, браузеру немає необхідності знову завантажувати таблицю стилів. Він просто завантажує запитуваний *HTML*-файл і використовує зовнішню таблицю стилів зі свого кеша, що дає суттєву перевагу в часі завантаження сторінок.

Внутрішня таблиця стилів - це набір стилів, який є частиною коду веб-сторінки, яка завжди повинна знаходитися між відкриваючими і закриваючими тегами <*style*> і </*style*> *HTML*-коду в тілі веб-сторінки, тобто всередині елемента <*head*>.

Елемент *style* відноситься до *HTML*, а не до *CSS*, але саме він повідомляє браузеру, що дані, що містяться всередині, є кодом *CSS*, а не *HTML*. Створення внутрішньої таблиці стилів ідентично створення зовнішньої, з тією лише різницею, що перелік стилів не виноситься в окремий файл, а укладається між тегами елемента *style*.

Внутрішні таблиці стилів можна легко додати в веб-сторінку, і так само просто перейти до редагування *HTML*-коду цієї ж веб-сторінки. Однак ці таблиці стилів аж ніяк не є найефективнішим способом для проектування дизайну сайту, що складається з безлічі сторінок. З одного боку, доведеться копіювати і вставляти код внутрішньої таблиці стилів в кожну сторінку сайту, а це не тільки трудомістка, але ще і безглузда робота. Цей код робить кожну сторінку сайту громіздкою. До того ж така сторінка повільно завантажується. Крім того, внутрішні таблиці стилів доставляють багато труднощів при оновленні дизайну сайту.

Кожен *CSS*-стиль складається з двох частин: селектора і блоку оголошення ,який, в свою чергу, містить правила форматування . Саме селектор контролює дизайн веб-сторінки ,визначаючи елемент, який потрібно змінити. Іншими словами, він використовується для форматування безлічі елементів одночасно. Селектори дозволяють вибрати один або кілька схожих елементів для їх подальшої зміни за допомогою властивостей в блоці оголошення.

Селектори, які використовуються для застосування стилів до певних *HTML* елементів, називаються селекторами тегів (або селекторами елементів). Вони являють собою дуже ефективний засіб проектування дизайну веб сторінок, оскільки визначають стиль всіх примірників конкретного *HTML*-елемента. З їх допомогою можна швидко змінювати дизайн веб-сторінки з невеликими зусиллями.

Селектори для окремих елементів називаються селекторами класів. Класи дозволяють вказати конкретний елемент веб-сторінки, незалежно від тегів. Можна застосувати зміни до безлічі елементів, оточених різними *HTML*-тегами. Браузери і мова *HTML* не мають жодних проблем у випадках, коли до одного елементу застосовано кілька класів. Все, що потрібно зробити, - додати атрибут *class* до *HTML*-елементу і як його значення - ім'я кожного класу, розділяючи їх комою. Браузер об'єднає властивості з різних класів, застосувавши закінчені стилі до елементів.

2.3. Мова програмування *Python*

Python - потужна і проста у використанні мова програмування, розроблена Гвідо ван Россум (*Guido van Rossum*). Перший реліз системи вийшов в 1991 році. На *Python* можна швидко написати невеликий проект, а взагалі вона застосовується до проектів будь-якого масштабу, в тому числі комерційних додатків і програм, націленим на відповідальні завдання.

З точки зору професійного програміста, легкість *Python* - запорука високої продуктивності праці: програми на *Python* короткі і вимагають менше часу на розробку, ніж програми на багатьох інших популярних мовах. *Python* має всі можливості, яких слід очікувати від сучасної мови програмування. Завдяки своїй потужності *Python* приваблює розробників з усього світу. Ним користуються найбільші компанії: *Google, IBM, Industrial Light + Magic, Microsoft, NASA, Red Hat, Verizon, Xerox* і *Yahoo!*. професійні розробники ігр також застосовують *Python*. *Electronic Arts, 2K Games і Disney Interactive Media Group* - всі ці компанії публікують гри з кодом на *Python*.

До основних переваг використання відносяться:

– якість програмного забезпечення - для багатьох концентрація *Python* на читабельності, узгодженості і якості програмного забезпечення (ПЗ) в цілому відрізняє його від інших інструментів в світі мов написання сценаріїв. Код *Python* за задумом повинен бути читабельним, а тому багато разів

використовуваним і супроводжуваним - в набагато більшому ступені, ніж традиційні мови написання сценаріїв. Узгодженість коду *Python* полегшує його розуміння, навіть якщо він написаний не вами. До того ж *Python* має розвинену підтримку механізмів багаторазового застосування ПЗ, таких як об'єктно-орієнтоване і функціональне програмування;

– продуктивність праці розробників - *Python* підвищує продуктивність розробників в багато разів ніж інші компільовані та статично типізовані мови , як *C, C++* та *Java*. Код *Python* зазвичай займає від однієї третьої до одної п’ятої частини розміру еквівалентного коду *C++* або *Java*. Отже потрібно менше набирати на клавіатурі, менше налагоджувати код та менше згодом супроводжувати. Окрім того , програми на *Python* запускаються одразу, без довгих кроків компіляції та зв’язування, необхідних у ряді інших інструментів, що додатково збільшує швидкість роботи програмістів;

– переносимість програм – більшість програм *Python* функціонує без змін на усіх основних комп’ютерних платформах. Наприклад, переніс коду *Python* з *Linux* та *Windows*, як правило, зводиться до копіювання коду сценарію між машинами;

– підтримувані бібліотеки – разом з *Python* поставляється велика колекція попередньо зібраної функціональності , яка називається стандартна бібліотека. Стандартна бібліотека підтримує безліч рішень програмних задач прикладного рівня, починаючи з зіставлення тексту зі зразком та закінчуючи сценарієм для мереж. Додатково *Python* можна розширювати бібліотеками власної розробки та широким набором прикладного підтримуваного ПЗ, створеного сторонніми розробниками;

– інтеграція компонентів – в сценаріях *Python* можна легко взаємодіяти з іншими частинами додатку, використовуючи різні механізми інтеграції. Така інтеграція дозволяє використовувати *Python* в якості інструмента для настройки та розширення продуктів. В даний момент із коду *Python* можна звертатися до бібліотек *C* та *C++*, інтегрувати з компонентами *Java* та *.NET*, взаємодіяти з фреймворками як *COM* та *Silverlight*.

Для веб-розробника, *Python* є необхідним інструментом у розробці серверної частити додатку. На ньому пишеться багато програмних платформ, які визначають структуру програмної системи; програмне забезпечення , яке полегшує розробку програмного проекту – фреймворки. Фреймворки визначають структуру, задають правила і надають необхідний набір інструментів для розробки. Вони поділяються на дві групи: синхронні та асинхронні. Синхронні фреймворки підтримають виконання запитів послідовно, з блокуванням потоку виконання, а асинхронні дозволяють оброблювати декілька запитів одночасно. До синхронних фреймворків відносять *Django, Flask , TurboGears, Web2py, Pyramid*. До асинхронних відносяться *Sanic, Tornado, FastApi, Aiohhtp*.

2.4. Мікрофреймворк *Flask*

Flask - це мікроплатформа для створення веб-додатків. Це означає, що *Flask* надає всі засоби, бібліотеки і технології, необхідні для створення веб-додатків. Цей веб-додаток може містити кілька веб-сторінок, блог, вікі-сайт або щось більш суттєве, наприклад додаток календаря або комерційний веб-сайт.

За замовчуванням, *Flask* не включає рівень абстракції баз даних, валідацію форм або інші потрібні для розробки бібліотеки . Замість цього, *Flask* підтримує розширення для додавання подібної функціональності в додаток, таким чином, як якщо б це було реалізовано в самому *Flask*. Численні розширення забезпечують інтеграцію з базами даних, валідацію форм, обробку завантажень на сервер, різні відкриті технології автентифікації і так далі.

Flask має багато параметрів конфігурації з статичними значеннями за замовченням, і мало попередніх угод. За угодою, шаблони і статичні файли зберігаються в піддиректоріях всередині дерева вихідних текстів на *Python* , названі *templates* і *static* відповідно.

Flask використовує внутрішні поточні об'єкти, тому в запиті , щоб зберігати безпеку потоку (*thread-safe*), необов'язково передавати об'єкти від

функції до функції. Такий підхід зручний, але вимагає дійсний контекст запиту для впровадження залежностей або при спробі повторного використання коду, який використовує значення, прив'язане до запиту.

Flask захищає вас від однієї і найпоширеніших проблем безпеки сучасних веб-додатків: від міжсайтового скриптинга (*cross-site scripting - XSS*). Якщо тільки ви не помітите небезпечний-ний *HTML*-код як безпечний, вас прикриє *Flask* і ніжній шаблонізатор *Jinja2*. Але існує і безліч інших способів викликати проблеми з безпечністю.

Flask залежить від деяких зовнішніх бібліотек - таких, як *Werkzeug* і *Jinja2*. *Werkzeug* - це інструментарій для *WSGI* - стандартного інтерфейсу *Python* між веб-додатками і різними серверами, призначений як для розробки, так і розгортання. *Jinja2* займається відображенням шаблонів.

Flask використовує декоратори маршрутів для реєстрації функцій з конкретними *URL*-адресами.

2.5. База даних *SQLite*

SQLite - це бібліотека, яка використовується для роботи з *SQL*. Відмінністю *SQLite* від інших реалізацій *SQL* є те, що дані беруться не від *SQL* сервера, а безпосередньо з носія інформації. Це відрізняє *SQLite* від інших реалізацій *SQL*, що використовують концепцію клієнт-сервер.

Вбудованість бази даних означає, що вона існує не як процес, окремий від обслуговуємого процесу, а є його частиною – частиною деякого прикладного застосунку. Зовнішній спостерігач не помітить ,що додаток використовує СУБД. Додаток просто робить його роботу, движок БД просто міститься всередині. Це позбавляє від необхідності мережевих налаштувань і адміністрування. Дозволяє відмовитися від засобів безпеки, мережевих адрес, користувачів і конфліктів їх прав доступу. І клієнт, і сервер працюють в одному процесі. Це позбавляє від проблем конфігурації.

На поточному ринку вбудованих баз даних представлено багато продуктів від різних виробників, але тільки один з них поставляється з відкритим кодом, не вимагає ліцензійних зборів і спроектований виключно як вбудована БД - це *SQLite*.

На рис. 2.1. представлено роботу додатків із використання *SQLite*. *SQLite* вбудований в адресний простір кожного з них. Кожен з них стає незалежним сервером бази даних. І, крім того, хоча кожен процес представляє незалежний сервер, вони можуть виконувати операції на одному і тому ж файлі бази даних. *SQLite* дозволяє їм справлятися з синхронізацією і блокуванням.

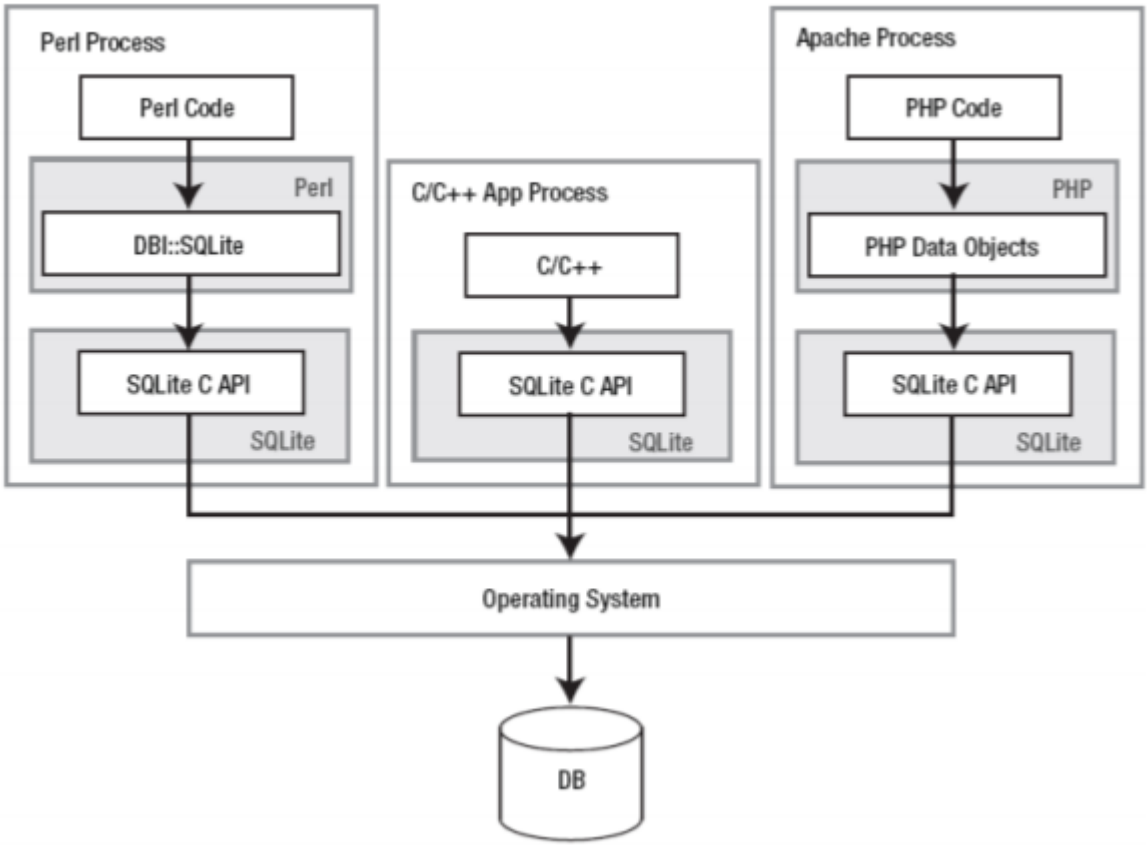

Рис. 2.1. Принцип роботи *SQLite* із різними додатками

SQLite має елегантну модульну архітектуру, яка буде показувати унікальні підходи до управління реляційними базами даних. Вісім окремих модулів згруповані в три головних підсистеми (рис. 2.2). вони поділяють обробку запиту на окремі завдання, які працюють подібно конвеєру. Верхні модулі компілюють запити, середні виконують їх, а нижні управляються з диском і взаємодіють з операційною системою.

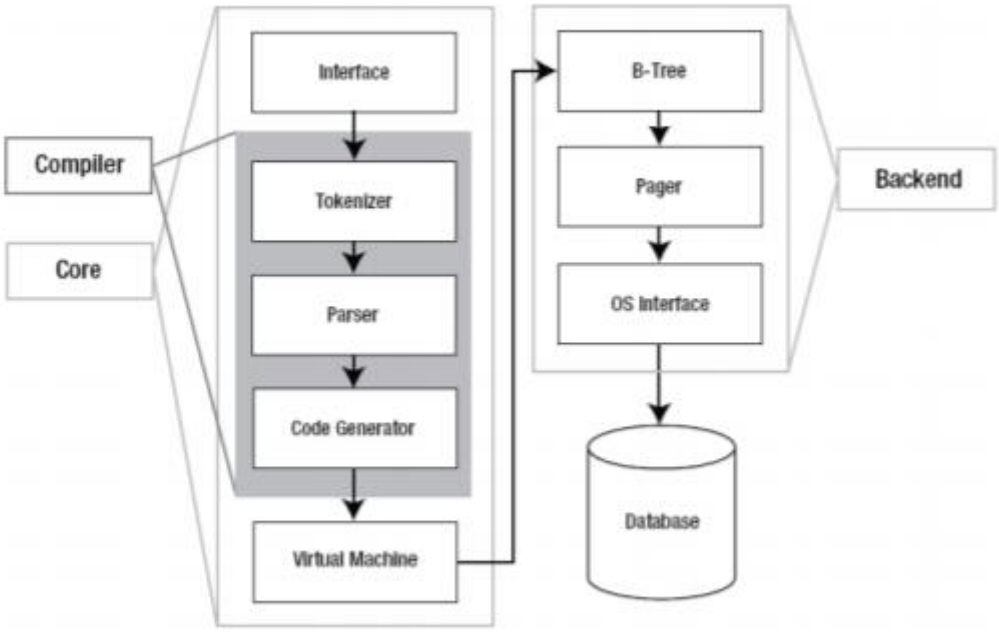

Рис. 2.2. архітектура *SQLite*

Файли бази даних мають розширення *.db* і можуть буди відкритими за допомогою *DB Browser for SQLite*, який використовує звичний табличний інтерфейс, тому не потрібно вивчати складні *SQL*-команди. Інтерфейс програми зображений на рис. 2.3.

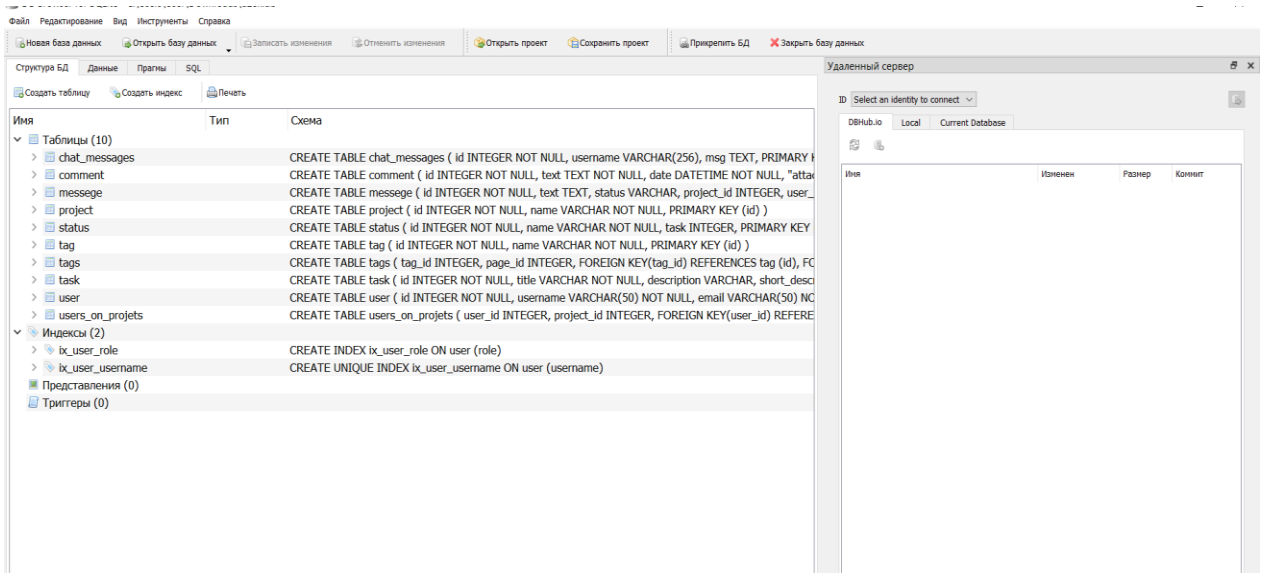

Рис. 2.3. Інтерфейс програми *DB Browser for SQLite*

2.6. Об'єктно-реляційне відображення *SQLAlchemy*

ORM або *Object-relational mapping* - це технологія програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема, між сховищем даних і об'єктами програмування. *ORM* використовується для спрощення процесу збереження об'єктів в реляційну базу даних і їх вилучення, при цьому *ORM* сама піклується про перетворення даних між двома несумісними станами. Більшість *ORM*-інструментів значною мірою покладаються на метадані бази даних і об'єктів, так що об'єктам нічого не потрібно знати про структуру бази даних, а базі даних - нічого про те, як дані організовані в додатку. *ORM* забезпечує повне розділення завдань в добре спроектованих додатках, при якому і база даних, і додаток можуть працювати з даними, кожен у своїй вихідній формі.

SQLAlchemy - це популярний інструментарій *SQL* і *Object Relational Mapper*. Він написаний на *Python* і надає повну потужність і гнучкість *SQL* для розробника додатків. Це багатоплатформне програмне забезпечення з відкритим вихідним кодом.

SQLAlchemy славиться своїм об'єктно-реляційних відображенням (*ORM*), за допомогою якого класи можуть бути співставленні з базою даних, що дозволяє з самого початку чітко розв'язати об'єктну модель і схему бази даних.

Оскільки розмір і продуктивність баз даних *SQL* починають мати значення, вони ведуть себе не так, як колекції об'єктів. З іншого боку, коли абстракція в колекціях об'єктів починає мати значення, вони ведуть себе не так, як таблиці і рядки. *SQLAlchemy* прагне враховувати обидва ці принципи.

Для роботи з додатками на основі фреймворку *Flask* , використовують бібліотеку *Flask SQLAlchemy*. Для роботи з базою даних треба задати шлях до неї. *Flask-SQLAlchemy* очікує знайти цей параметр в конфігурації по ключу *SQLALCHEMY_DATABASE_URI*. Бібліотека містить клас *Model* який використовується для опису моделі. На основі цього класу створюються власні класи, які описують роботу додатків. Атрибути класів відповідають стовбцям

таблиць із даними, властивості дозволяють реалізовувати перевірку атрибутів за параметрами, а методи обробляють дані обраного класу.

Зв'язки записуються за допомогою функції *relationship()*. Однак зовнішній ключ повинен бути оголошений окремо за допомогою класу *sqlalchemy.schema.ForeignKey*.

2.7. Середовище розробки *Visual Studio Code*

Для швидкої розробки програмного забезпечення, використовують спеціальні середовища розробки, які покращають якість та швидкість написання програмного коду.

Visual Studio Code - це сервіс, який позиціонує себе як «легкий» редактор коду для багатоплатформової розробки веб-додатків.

Особливості *Visual Studio Code*:

– *VS Code* дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології *Windows Forms*, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ;

– у редакторі присутні вбудований відладчик, інструменти для роботи з *Git* і засоби рефакторинга, навігації по коду, автодоповнення типових конструкцій і контекстної підказки;

– продукт підтримує розробку для платформ *ASP.NET* і *Node.js*, і вважається легковажним рішення, яке дозволяє обійтися без повного інтегрованого середовища розробки;

– великим плюсом редактора є підтримка великої кількості мов, таких як *C++, C #, Python, PHP, JavaScript* та інших.

Можливості *Visual Studio Code*:

– вбудовані інструменти інтеграції з *GitHub, GIT*, а також *Visual Studio Team Services* для швидкого тестування, складання, упаковки та розгортання різних типів додатків;

– зручність роботи з *Unity*-проектами;

– робота з *Mono* і *Node.js* за допомогою вбудованого відладчика;

– підтримка *TypeScript* і *JavaScript*;

– публікація створених додатків в *Microsoft Azure* через сервіс *Visual Studio Team Services*;

– підтримка практично всіх мов програмування;

– написання коду для конкретного завдання з його подальшою інтеграцією в проект ;

– велика бібліотека шаблонів, готових фрагментів коду і сніпетів з можливістю додавання своїх елементів;

– одночасна робота з декількома проектами (в декількох вікнах);

– інтерфейс можна розділити на дві панелі для порівняння коду;

– функція налагодження.

2.8. Бібілотека *Flask-WTForms*

WTForms - це потужна бібліотека, написана на *Python* і незалежна від фреймворків. Вона вміє генерувати форми, перевіряти їх та попередньо заповнювати інформацією (зручно для редагування) і багато іншого. Також вона пропонує захист від *CSRF*. Для установки *WTForms* використовується *Flask-WTF*.

Flask- WTF - це розширення для *Flask*, яке інтегрує *WTForms* у *Flask*. Воно також пропонує додаткові функції, такі як завантаження файлів, *reCAPTCHA*, інтернаціоналізація та інші. Для установки *Flask-WTF* потрібно ввести наступну команду:

pip install flask-wtf

Пакет *wtform* пропонує кілька класів, що представляють собою наступні поля: *StringField, PasswordField, SelectField, TextAreaField, SubmitField*. З їх допомогою можливо створювати різні поля форм та коректувати їх поведінку.

Для перевірки даних поля форми , існує клас *Validator* , який перевіряє дані введені користувачем.

2.9. Висновки до розділу

В даному розділі було досліджено технології розробки веб-додатків та їх особливості. Для розробки клієнтського додатку найефективнішим варіантом є використання *HTML*. За допомогою *HML* можливо вибудувати структуру веб-сторінок та їх вміст. Щоб подати солідне зовнішнє оформлення і доступний спосіб подачі інформації потрібно використати таблиці каскадних стилів *CSS*. Поєднання цих двох інструментів дозволить зробити веб-додаток інтуїтивно-зрозумілим для користувача і найбільш ефективним у організації роботи з проектами.

SQLite дозволяє реалізувати основні сутності «Організація роботи над проектами» для простого веб-додатку. За допомогою *SQLAlchemy* можливо працювати з базою даних у вигляді класів та їх атрибутів.

Для розробки серверного додатку можливо використання фреймворку *Flask* на мові *Python*. Він дозволяє реалізувати усі необхідні функції , що необхідні у роботі веб-додатку: обмін інформацією з користувачем, взаємодія з базою даних, відправка повідомлень на пошту. А властивість фреймворку до розширення за допомогою зовнішніх бібліотек дозволяє розробнику самостійно проектувати архітектуру проекту.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

3.1. Створення проекту

На початку створюється віртуальне оточення для зберігання усіх бібліотек та засобів програмування , що будуть використанні при розробці.

Налаштовано список зовнішніх залежностей за допомогою файлу *requirements.txt*. Цей файл містить інформацію про усі необхідні бібліотеки для запуску проекту на інших серверах. Вміст файлу наведений на рис. 3.1.

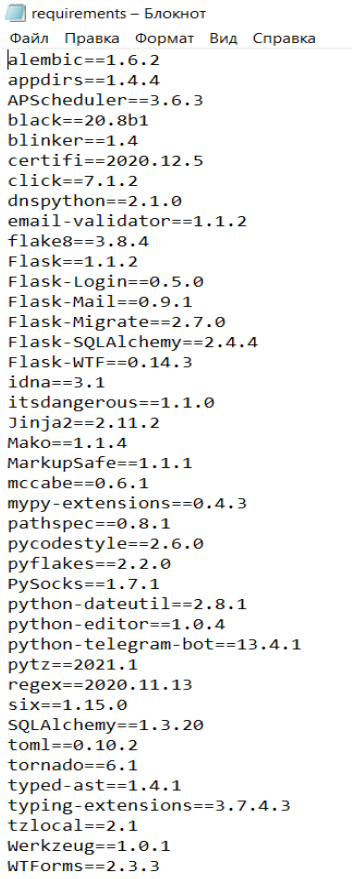

Рис. 3.1. Перелік необхідних бібліотек для роботи додатку

Запуск додатку починається з команди :

set FLASK_APP=webapp && set FLASK_ENV=development&& set FLASK_DEBUG=1 && flask run

Для полегшення процесу запуску додатку створено скрипт-файл *run.bat*, вміст якого зображений на рис. 3.2.

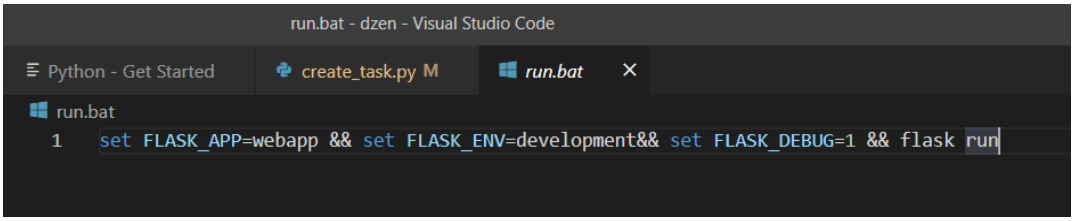

Рис. 3.2. Файл *run.bat*

Оскільки використовується система контролю версій *Git*, створюється файл *.gitignore* у який заносяться типи файлів що не будуть відслідковуватися, для забезпечення конфіденційності та безпеки додатку. Вміст файлу зображений на рис. 3.3.

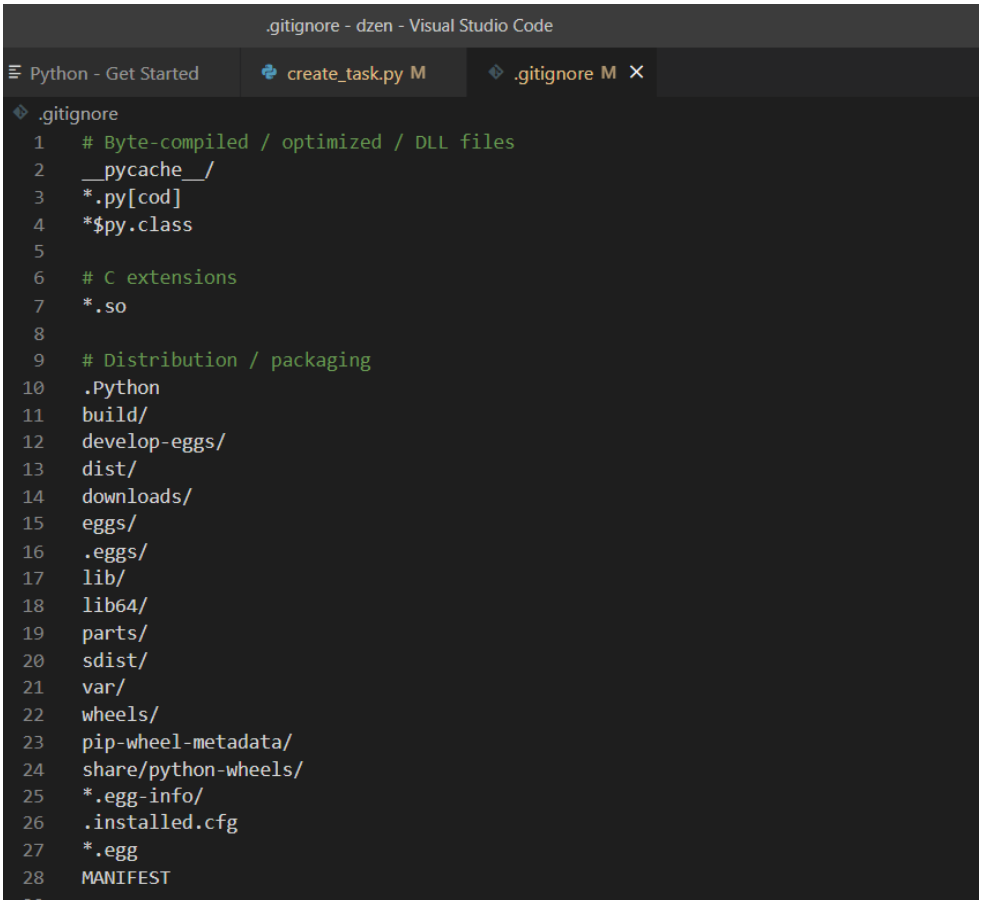

Рис. 3.3. Вміст файлу *.gitignore*

Flask використовує концепцію *blueprint*'ов для створення компонентів додатків і підтримки загальних шаблонів всередині програми або між додатками. *Blueprint*'и можуть як значно спростити великі програми, так і надати загальний механізм реєстрації в додатку операцій з розширень *Flask*. Об'єкт *Blueprint* працює аналогічно об'єкту додатку *Flask*, але насправді він не є додатком. Зазвичай це лише ескіз для збірки або розширення програми.

В даному випадку виділено чотири головних компонентів додатку :

– *admin;*

– *user;*

– *task;*

– *project.*

Для взаємодії з іншими модулями додатку у кожній папці компоненту створюється файл *__init__.py* , який повідомляє іншим модулям *Python* про можливість використовувати даний компонент та його атрибути.

У кожного компонента є власні обробники, які зберігаються у файлі *views.py*. Також цей файл містить назву свого *Blueprint*'а , за якою додаток отримає дані про методи компонента. Приклад компоненту *admin* наведено на рис. 3.4.

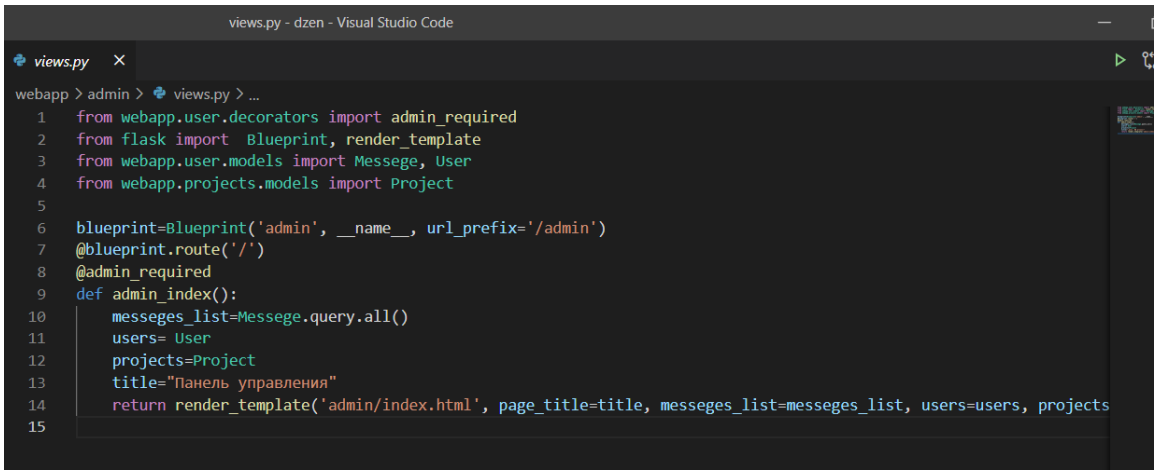

Рис. 3.4. Приклад обробника подій компоненту *admin*

В залежності від типу компонента він може мати різні стеки даних:

– *models* – відповідають за вміст за вигляд бази даних відносно обраного компонента.

– *decorators* – містять функції-декоратори. Декоратор - це функція, яка дозволяє обернути іншу функцію для розширення її функціональності без безпосереднього зміни її коду.

– *forms* – місять інформацію про форми які використовуються в додатку.

3.2. Створення веб-сторінок додатку

Під час розробки веб-сторінок додатку було використано методи розширення шаблонів та шаблонізатор *Jinja*. У якості *CSS* стилів були взяті стилі *Bootstrap* та власні стилі. Основний макет веб-сторінки міститься у файлі *base.html*, а усі інші успадковуються від нього.

На головній сторінці веб-додатку розміщено слайдер із проектами. Для переміщення між існуючими проектами потрібно використовувати стрілки вліво та вправо на цьому ж слайдері. По центру зображення знаходиться посилання до задач цього проекту. У правому верхньому куті знаходиться головне меню додатку. В залежності від типу користувача(адміністратор або звичайний користувач) меню змінюється. На рис. 3.5. зображена головна сторінка веб-додатку з огляду адміністратора , а на рис. 3.6. зображена головна сторінка з боку звичайного користувача.

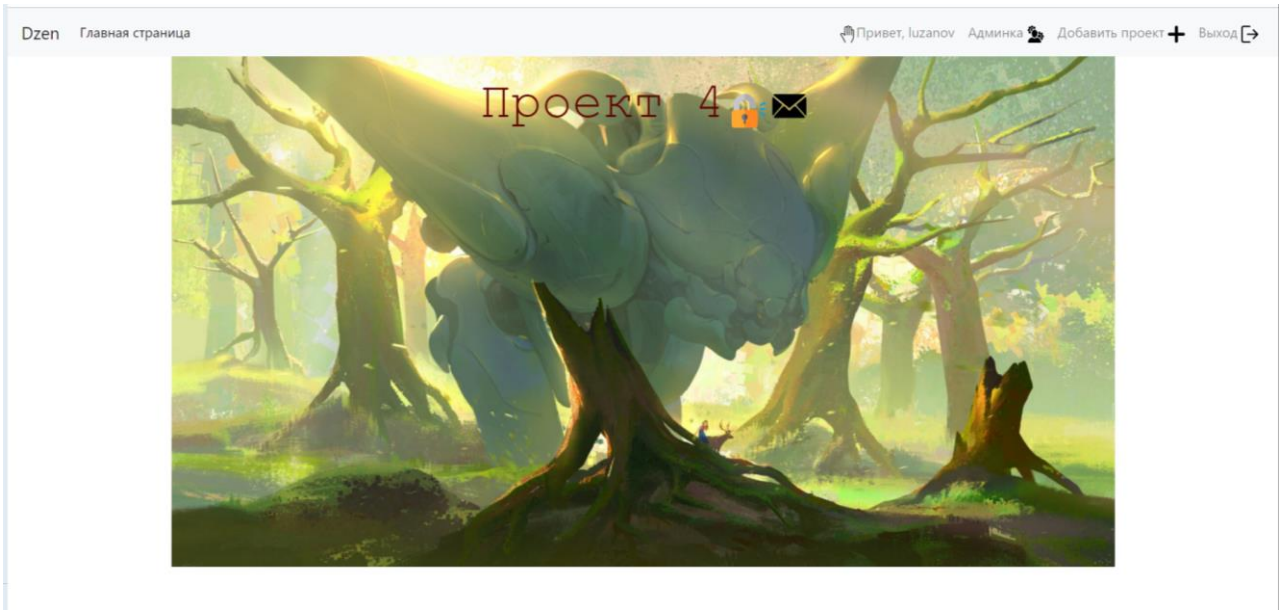

Рис. 3.5. Головна сторінка веб-додатку для адміністратора

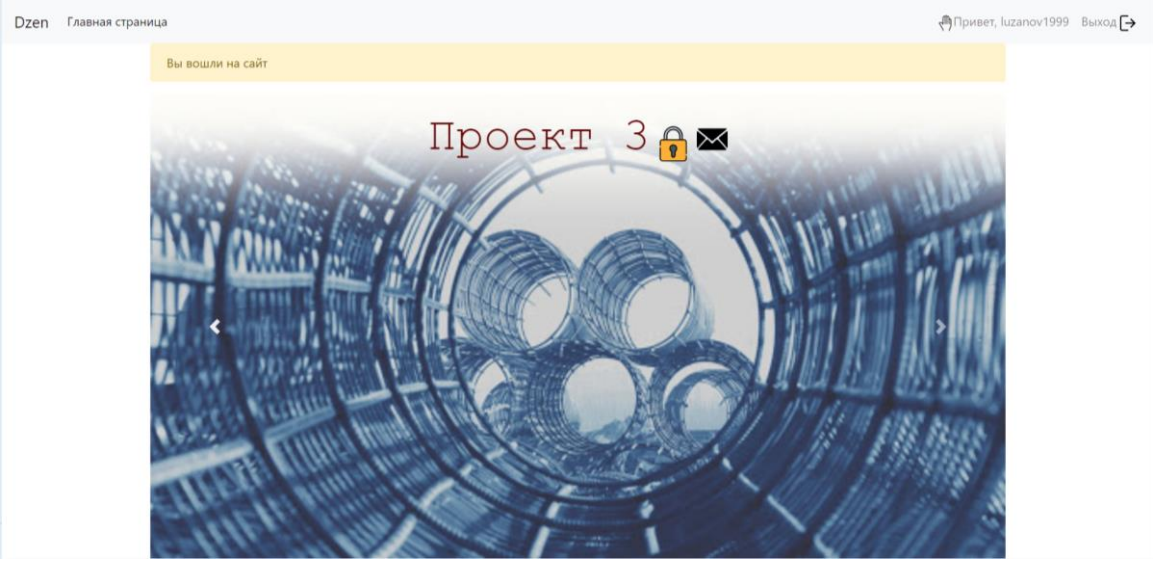

Рис. 3.6. Головна сторінка веб-додатку для звичайного користувача

В залежності від того має користувач доступ до проектів чи немає, зображення ключу на головному екрані змінюється.

Для зображення значків елементів на веб-сторінці було використано зображення *svg* формату. Масштабна векторна графіка (*Scalable Vector Graphics*) - це мова для опису двовимірної графіки, яка заснована на *XML*.

Даний векторний формат зображення можна відображати на великих дозволах без втрати якості, через це він і користується достатньою популярністю.

При авторизації користувача відкривається привітальне вікно з його ім’ям (рис. 3.7.). Для реалізації цього елементу було використано *cookies* та *css* стилі. *Cookie* – це фрагмент даних , який відправляється веб-сервером користувачу та зберігається на комп’ютері користувача. Веб-клієнт кожний раз коли відкриває веб-додаток відправляє ці дані назад до серверу.

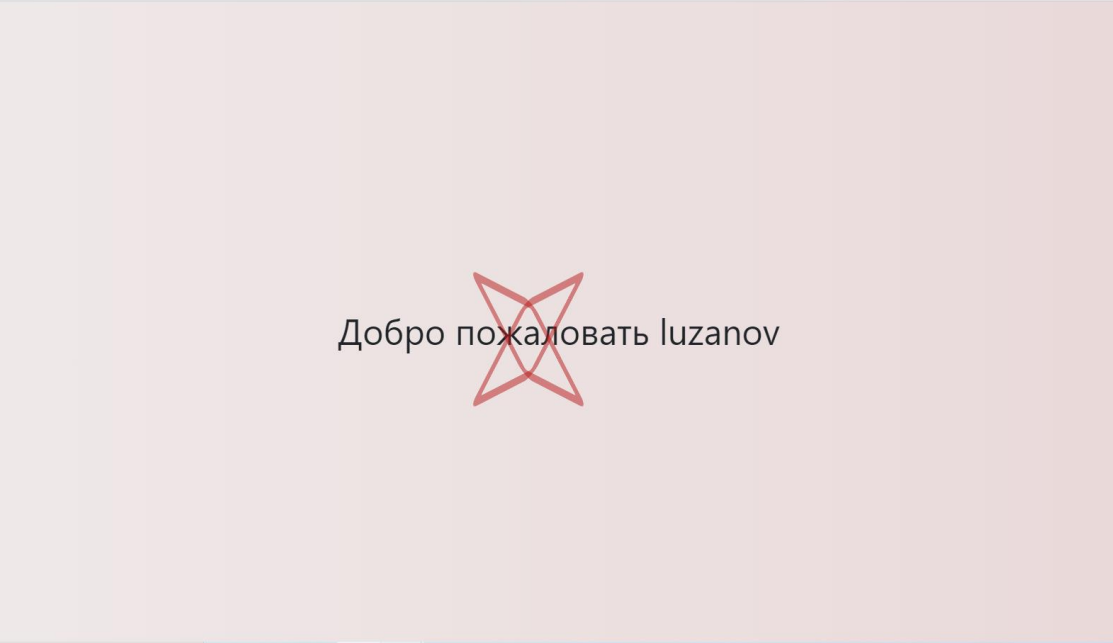

Рис. 3.7. Привітальне вікно користувача

В додатку використовується система розділення на адміністратора та звичайних користувачів. Їх можливості зображенні на діаграмі прецедентів (рис. 3.8).

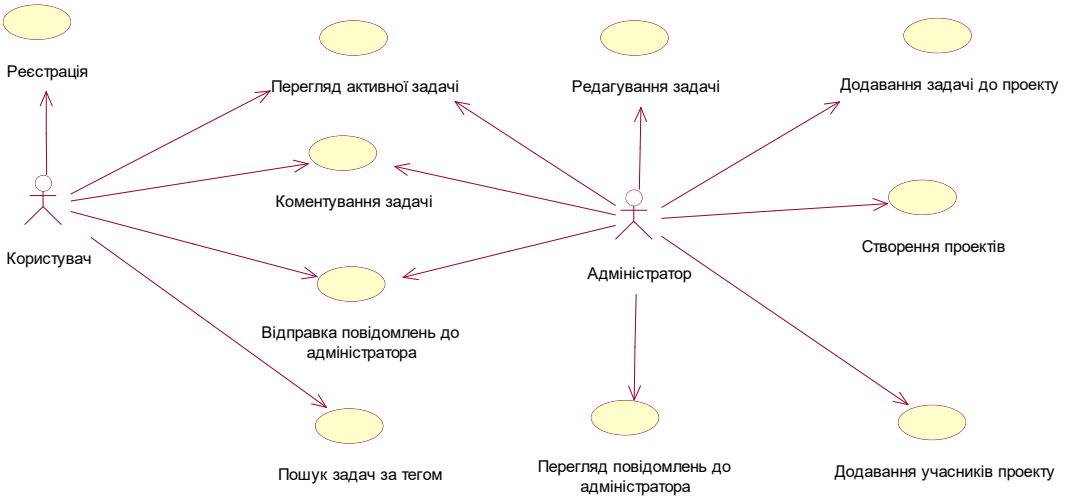

Рис. 3.8. Діаграма прецедентів веб-додатку

У вікні проекту (рис. 3.9.) адміністратор може додати користувачів, додати задачу, подивитися список користувачів з доступом до цього проекту, відкрити вікно задачі. Задачі мають три статуси: активні, неактивні та в режимі очікування. Звичайний користувач може бачити лише задачі зі статусом активні та в режимі очікування, а адміністратор ще й неактивні.

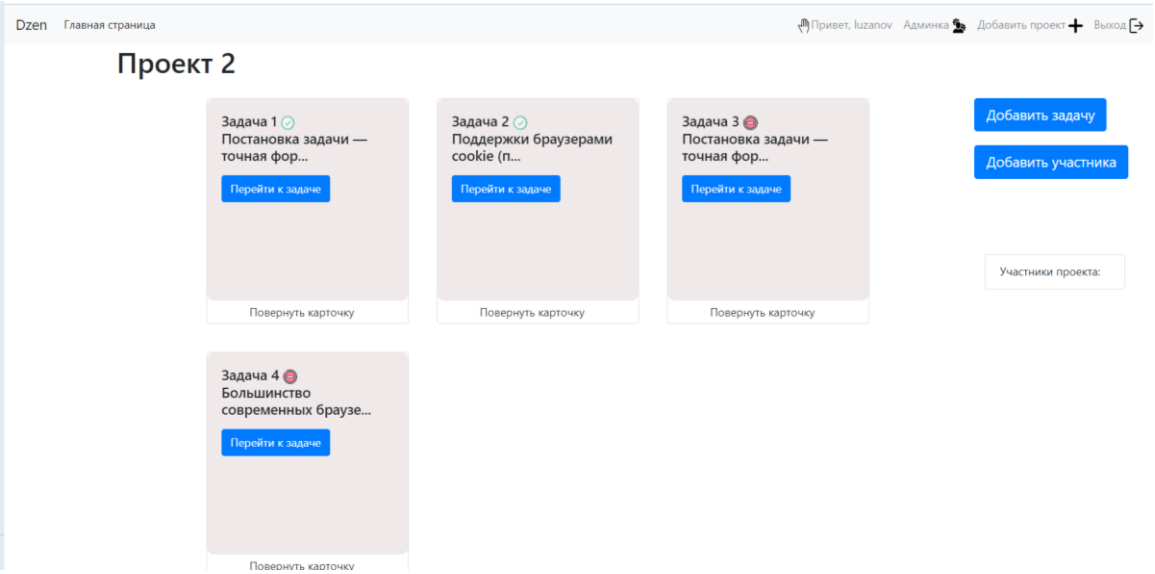

Рис. 3.9. Вікно проекту

Усі задачі оформлені у вигляді карток, на яких зображено посилання до вікна задачі, короткий опис задачі, назва задачі. Якщо натиснути нижній край картки то вона повернеться зворотною стороною на якій розміщенні теги цієї задачі. Натиснувши один з тегів , можна побачити усі задачі обраного проекту пов’язані з ним (рис. 3.10).

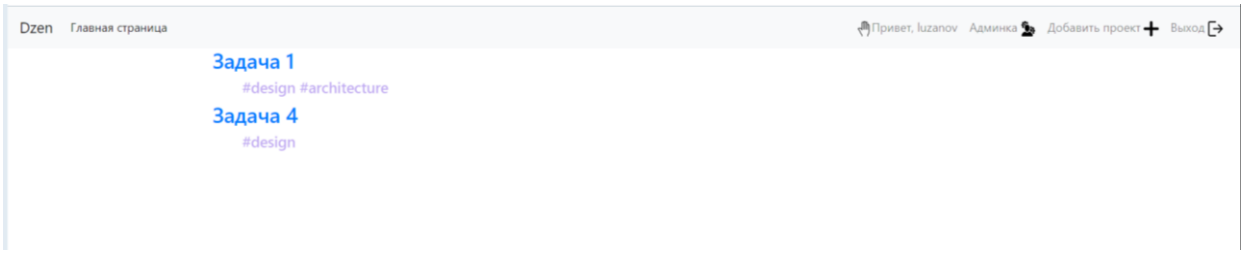

Рис. 3.10. Список задач відносно обраного тегу

Вікно задачі містить назву задачі, її опис, дату закінчення, коментарі , та поле для коментування (рис. 3.11).

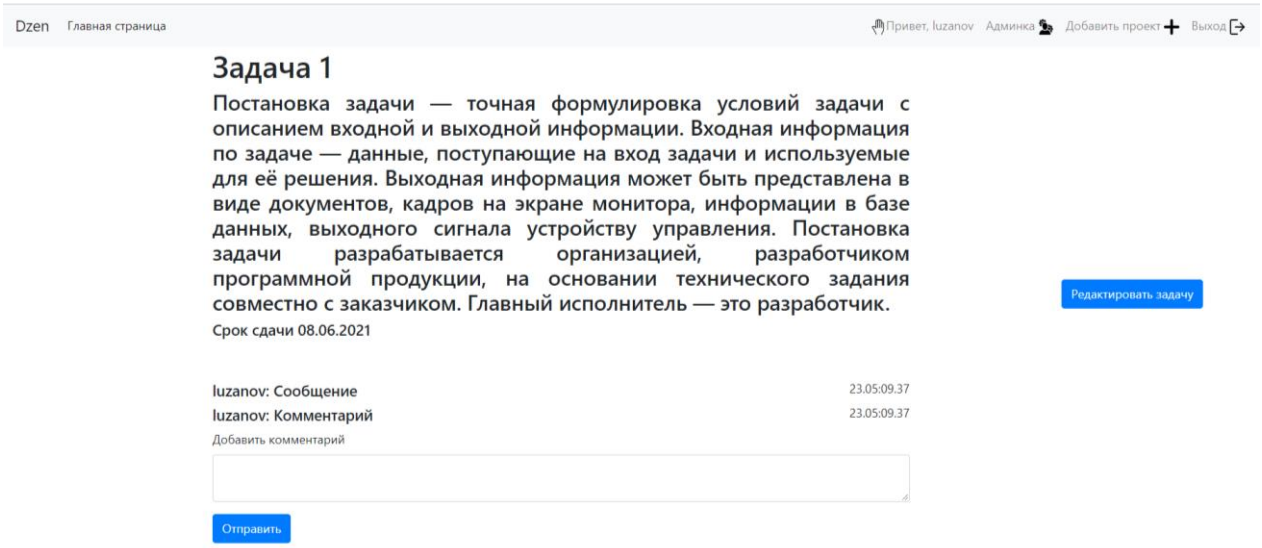

Рис. 3.11. Вікно задачі

Однією з переваг додатку є використання елементів комунікації між адміністратором та користувачами. Для цього на головній панелі розміщено посилання до адміністративне меню. Перейшовши за ним відкривається вікно адміністратора, яке містить повідомлення користувачів стосовно окремого проекту (рис. 3.12).

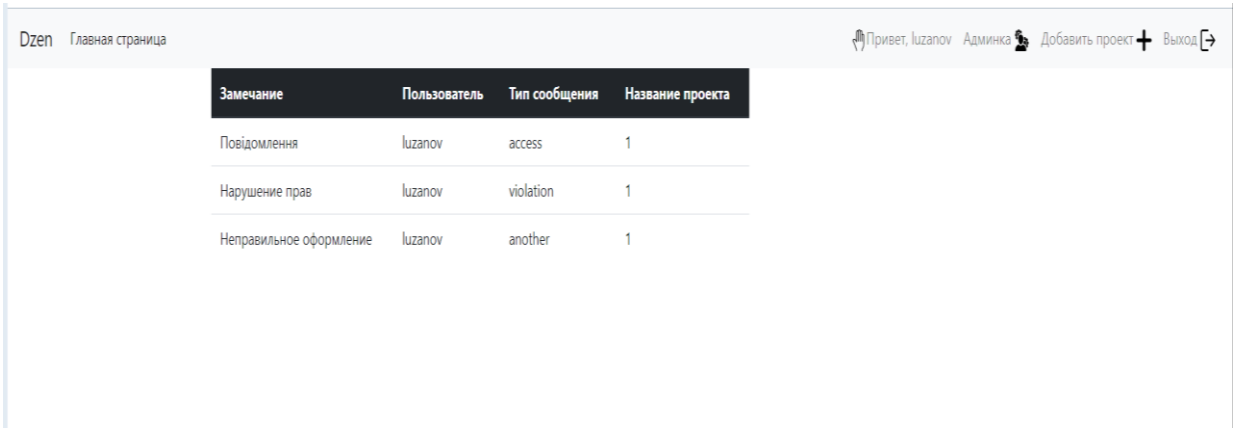

Рис. 3.12. Меню адміністратора

3.3. Створення бази даних веб-додатку

До основних сутностей бази даних належать:

– *User* –Користувач;
– *Message* – Повідомлення користувача;
– *Task* – Завдання;

– *Comment* – Коментар ;

– *Project* –Проект;

– *Tag* – Тег.

Для кожної таблиці визначаються її атрибути (табл. 3.1)

Таблиця 3.1

Поля та типи даних

Найменування атрибута	Тип даних
User	
id	числовий
Username	Текстовий
Email	Текстовий
Password	Текстовий
Commnets	Числовий
Messages	Числовий
Role	Текстовий
Userpic_url	Текстовий
Message	
Id	Числовий
Text	Текстовий
Status	Текстовий
Project_id	Числовий
User_id	Числовий
Task	
Id	Числовий
Title	Текстовий
Description	Текстовий
Short_description	Текстовий
Due_date	Дата та час
Project_id	Числовий

Закінчення таблиці 3.1

Status	Текстовий
Tag	Числовий
Comment	
Id	Числовий
Text	Текстовий
Date	Дата та час
Task	Числовий
Author	Числовий
Tag	
Id	Числовий
Name	Текстовий
Project	
Id	Числовий
Name	Текстовий
Tasks	Числовий
Users	Числовий

3.4. Створення форм

1) Форма реєстрації користувача (рис. 3.13).

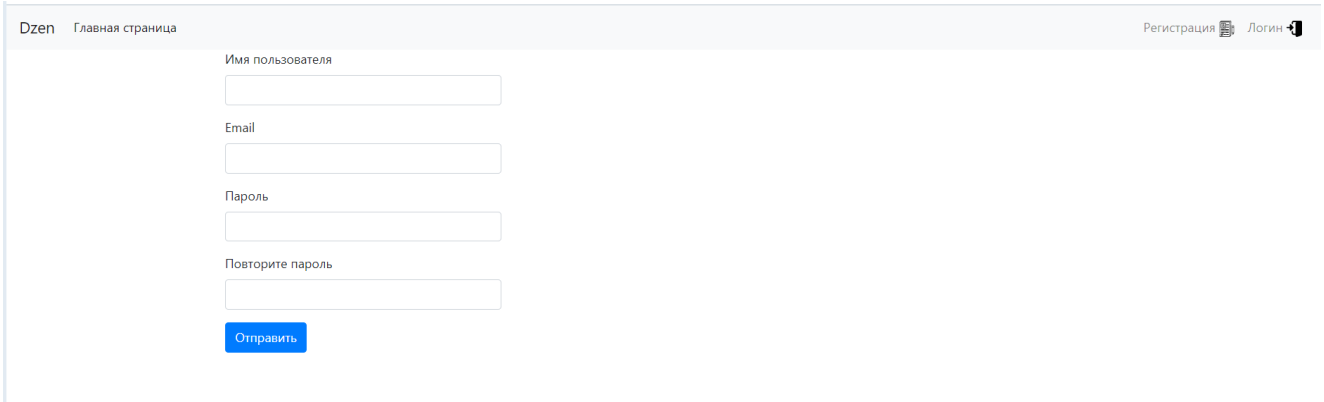

Рис. 3.13. Форма реєстрації нового користувача

2) Форма авторизації користувача(рис. 3.14).

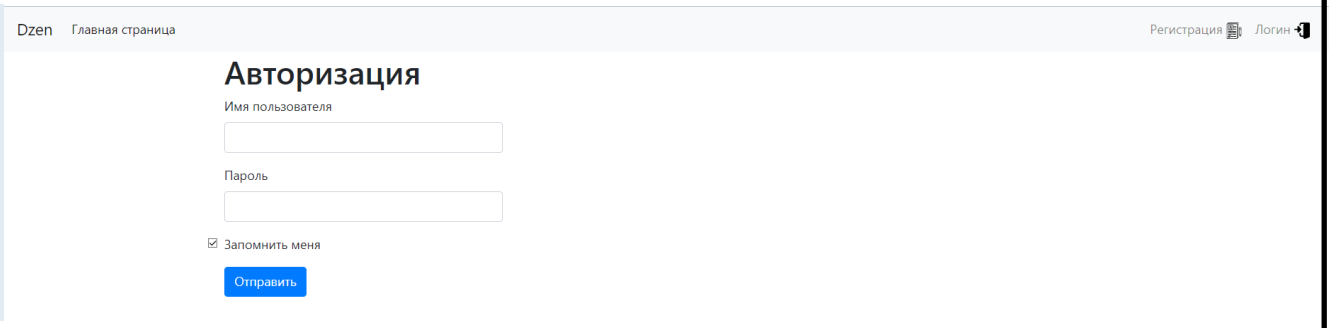

Рис. 3.14. Форма авторизації користувача

3) Форма додавання нової задачі до проекту(рис. 3.15).

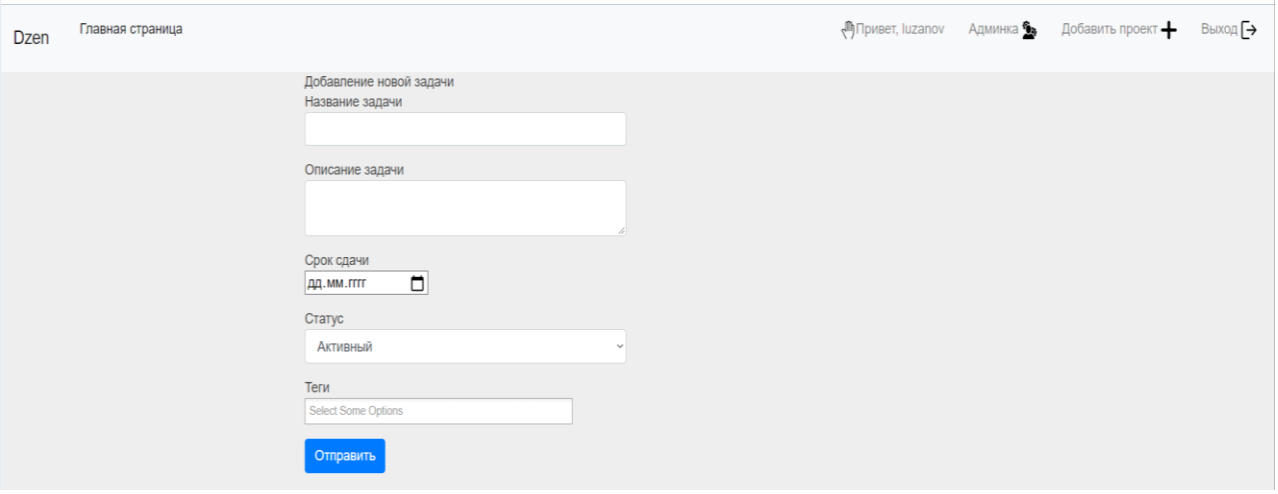

Рис. 3.15. Форма додавання нової задачі

4) Форма відправлення повідомлення до адміністратора (рис. 3.16.)

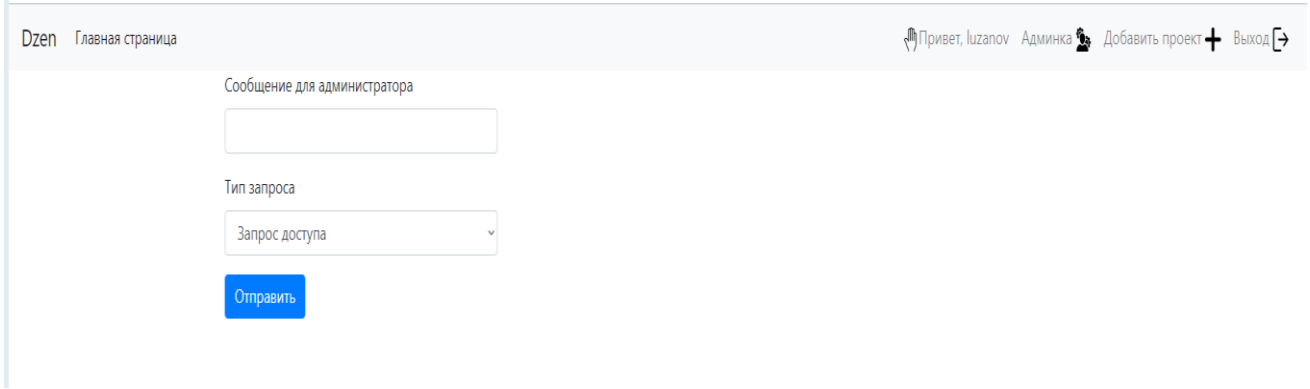

Рис. 3.16. Форма відправлення повідомлення

5) Форми додавання проекту, учасник та коментаря (рис. 3.17-19).

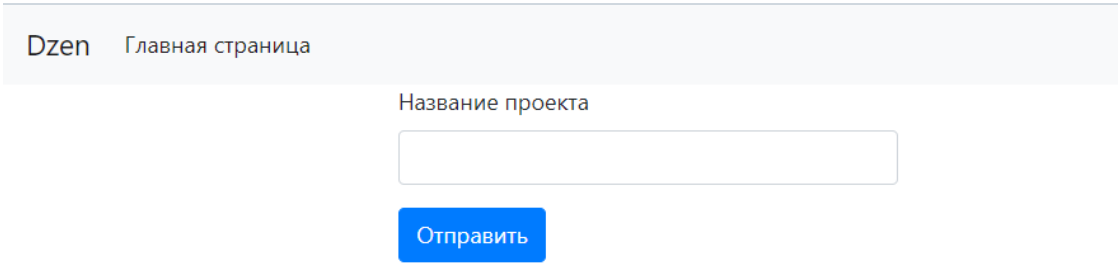

Рис. 3.17. Форма додавання проекту

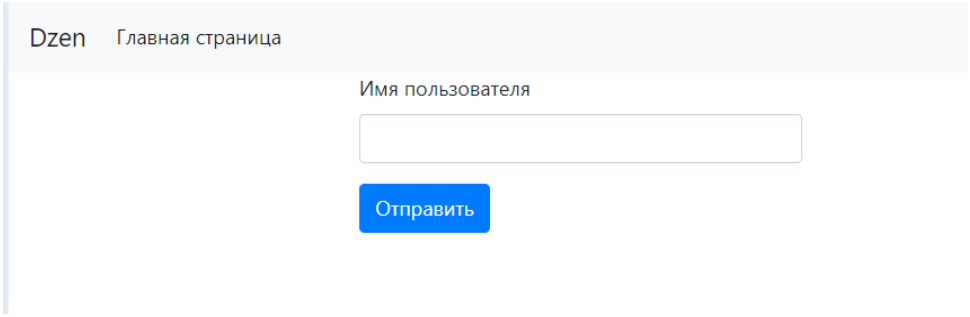

Рис. 3.18. Форма додавання учасників проекту

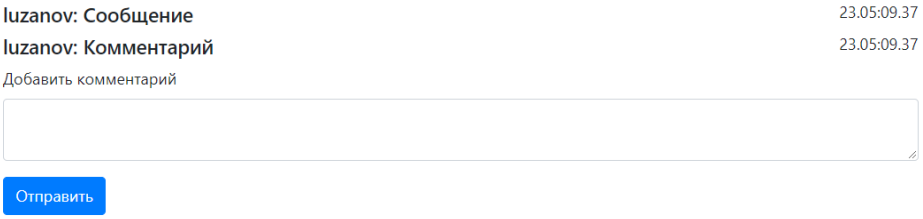

Рис. 3.19. Форма додавання коментарю до задачі

Використання форм у веб-додатку, дозволяє користувачу взаємодіяти з сервером , відправляти та отримувати дані. За допомогою форм було реалізовано реєстрацію та авторизацію користувача, функцію додавання проекту та задачі, відправлення повідомлення до адміністратора, коментування задачі.

3.5. Створення додаткового функціоналу

У якості додаткового функціоналу було реалізовано:

– відправка на електрону пошту адміністратора повідомлень від користувачів;

– редагування змісту задачі;

– створення веб-сторінки з посиланням на головну у разі виникнення помилок веб-додатку (рис. 3.20).

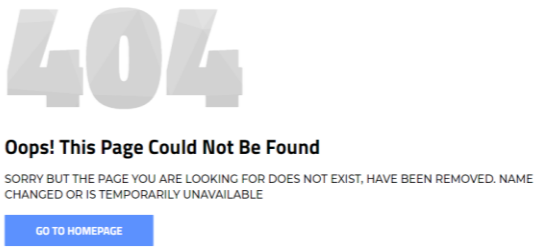

Рис. 3.20. Веб-сторінка помилки додатку

3.6. Висновки до розділу

Цей розділ демонструє процес розробки веб-додатку. На початку налаштовується середовище розробки та допоміжні інструменти (*Git* та *requirements.txt*). Контроль процесу розробки дозволяє усунути екстрених помилок та втрати даних. Завдяки файлу контролю бібліотек можливо розміщувати веб-додаток на інших серверах або пристроях.

Наступним кроком є створення головного файлу ініціалізації додатку. Цей файл містить функцію початку роботи серверу. В нього записується усі модулі, що будуть використовуватися.

Концепція *blueprint* вимагає реєстрацію усіх компонентів додатку у головному файлу ініціалізації. У структурі додатку було виділено 4 основні сутності-компоненти:

- Адміністратор;
- Користувач;
- Проект;
- Завдання.

Кожний компонент був оформлений у якості модуля для взаємодія один з одним. Для роботи сервера було створено функції-обробники, що реалізують обмін інформацією між сервером та базою даних, сервером та користувачем, відображення веб-сторінок і відправлення повідомлень на пошту.

Було налаштовано процес авторизації користувача та зберігання його у мережі. Для допуску нових користувачів до системи створено функцію реєстрації.

Розроблено систему користувачів: адміністратор та звичайний користувач. В залежності від типу , користувачі мають різні функції та обмеження. Адміністратор може створювати проекти та завдання, а також приєднувати до них співробітників.

Створено веб-сторінки з використанням зовнішнього набору інструментів *Bootstrap* і власних стилів. Додано функції *JavaScript* для відображення привітального вікна при авторизації користувача.

Наприкінці було поєднано клієнтський та серверний додаток за допомогою функцій-*url*.

ВИСНОВКИ

Під час виконання дипломної роботи було детально розібрано поняття веб-додатку. Веб-додаток - це додаток, що працює на платформі *Web*, тобто використовує для взаємодії з користувачем веб-сервер, працюючий по протоколу *HTTP* і браузер, що інтерпретує сторінки *HTML*.

Зазвичай веб-додаток складається з 3 трьох частин:

– клієнтський додаток;

– серверний додаток;

– база даних.

Клієнтський додаток відповідає за інтерфейс в браузері користувача. Розробка інтерфейсу виконується за допомогою інструментів *CSS, HTML, JavaScript*.

Серверний додаток використовується для взаємодії користувача із базою даних, обрахунок операцій, перехід між веб-сторінками додатку. Для розробки серверного додатку використовують мови програмування *Python, PHP, Ruby, GO*.

В другому розділі було досліджено технології розробки веб-додатків. Ознайомлено з основними тегами *HTML,* властивостями CSS. До основних компонентів *HTML*-сторінки відносяться: <*head*>, <*body*>, <*html*>, <*title*>. Для розділення сторінки на частини використовується тег . За допомогою тегів <*h1-6*> і <*p*> можливо коректувати оформлення тексту. Проаналізовано можливості фреймворку *Flask* та мови програмування *Python*. Концепція фреймворку *Flask* будується на розподіленні додатку на компоненти *blueprint*. Це допомагає виділити основні сутності та організувати взаємодію між ними. Розглянуто базу даних *SQLite* та інструмент її використання *SQLAlchemy*.

В третьому розділі було виконано поставлені задачі, а саме:

1. Розроблено клієнтський додаток із інтуїтивно зрозумілим інтерфейсом користувача-адміністратора системи.

2. Було запроектовано базу даних додатка та наведено її атрибути.

3. Розроблено серверний додаток на мові *Python* за допомогою фреймворку *Flask.*

4. Було налагоджено роботу між серверним та клієнтським додатком. Зв’язано базу даних із сервером.

5. Створено систему автентифікації користувачів.

6. Застосовано методи шифрування паролів для підвищення безпеки користувача.

7. Використано сповіщення через електрону пошту для адміністратора.

8. Розроблено меню адміністратора для контролю над користувачами системи.

9. Для кращого розуміння інтерфейсу було використано допоміжні елементи.

Описано процеси розробки з використанням графічного матеріалу для більшої наочності. Сутності бази даних представлені у вигляді таблиці, де зазначено усі атрибути .

Відповідно до практичної частини можна зробити висновок, що реалізація веб-додатку поділяється на чотири частини: створення бази даних, створення клієнтського додатку, створення серверного додатку, об’єднання усіх елементів в єдину працюючу систему. Кожен процес має безліч варіантів реалізації в залежності від вимог до програмного продукту, а також здібностей розробника.

Одержані результати дають підстави вважати, що завдання реалізоване, мета досягнута.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основы разработки веб-приложений. — СПб.: Питер, 2015. — 272 с.: ил. — (Серия «Бестселлеры *O'Reilly*»).
2. Изучаем *HTML, XHTML* и *CSS*. 2-е изд. — СПб.: Питер, 2014. — 720 с.: ил. — (Серия «*Head First O'Reilly*»).
3. Изучаем *Python*, том 1, 5-е изд.: Пер. с англ. — СПб.: ООО "Диалектика", 2019. — 832 с. : ил. — Парад, тит. англ.
4. Программирование на *Python* 3. Подробное руководство. – Пер. с англ. – СПб.: СимволПлюс, 2009. – 608 с., ил.
5. Бейдер Д. Чистый *Python*. Тонкости программирования для профи. — СПб.: Питер, 2018. — 288 с.: ил. — (Серия «Библиотека программиста»).
6. Молинаро Э. SQL. Сборник рецептов. – Пер. с англ. – СПб: Символ!Плюс, 2009. – 672 с., ил.
7. *Python*. Карманный справочник, 5-е изд. : Пер. с англ. – М. : ООО "И.Д. Вильямс", 2015. – 320 с. : ил. – Парал. тит. англ.
8. ГрантКит *CSS* для профи. — СПб. : Питер , 2019.—496 с. : ил . — (Серия «Библиотека программиста»).
9. Харрисон Мэтт Как устроен *Python*. Гид для разработчиков, программистов и интересующихся. — СПб.: Питер, 2019. — 272 с.: ил. — (Серия «Библиотека программиста»).
10. Прохорецок Н. А. *Python* 3 и PyQt. Разработка приложений. - : - СПб.: БХВ-Петербург, 2012.-704 с.: ил.
11. Новая большая книга *CSS*. — СПб.: Питер, 2016. — 720 с.: ил. — (Серия «Бест-селлеры *O'Reilly*»).
12. Слаткив Бретт. Секреты *Python*: 59 рекомендаций по написанию эффективного кода.: Пер. с англ. -М.: ООО ии.д. Вильяме", 2016. - 272 с.: ил. -Парал. тит. англ.

13. Бизли Д., Джонс Б. К.Б59 *Python*. Книга рецептов / пер. с анг. Б. В. Уварова. – М.: ДМК Пресс, 2019. – 648 с.: ил.
14. Дронов В. *HTML5*, *CSS3* и *Web* 2.0. Разработка современных *Web*-сайтов. — СПб.: БХВ-Петербург, 2011. —416с.: ил. — (Профессиональное программирование).
15. Гринберг М. Рзработка веб-приложений с использованием *Flask* на языке *Python* / пер. с англ. А.Н. Киселева. – М.: ДМК Пресс, 2014. – 272 с.: ил.
16. ДСТУ 3008-95 Документація. Звіти у сфері наук і техніці Структура і правила оформлення.
17. Положення про дипломні роботи(проекти) випускників Національного авіаційного университету СМЯ НАУ П 03.01(10) – 02 – 2017.