Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема: Розробка сервісу для автопостингу в соціальних мережах.
Розробка вебдодатку.

Спеціальність: 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-15.ПЗ
(позначення)

Рецензент
доцент Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Навроцький Б.П.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
доцент Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
ст. викладач Савка І.Я.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

доцент Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020
(рік)

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
_____
„_____” _____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Навроцькому Богдану Петровичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка сервісу для автопостингу в соціальних мережах. Розробка вебдодатку.
затверджена розпорядженням по факультету математики та інформатики від „11” вересня 2019 р. №7
2. Термін здачі студентом закінченого проекту 22 травня 2020 р.
3. Вихідні дані до дипломного проекту Стандарт кафедри інформаційних технологій , технологія програмування серверної частини – Python Flask, технології програмування клієнтської частини – HTML, Werkzeug, Jinja2. Технології розгортання програмного забезпечення – Heroku
4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1.Постановка задачі розробки сервісу автопостингу та аналіз предметної області
2. Аналіз основних характеристик, розробка моделей та алгоритмів для створення автопостингу
3. Розробка вебсервісу автопостингу,
4. Бізнес план розробки автопостингу
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень) 1. Титульний аркуш. 2. Загальний опис і мета. 3. Головні завдання автопостингу. 4. Актуальність, переваги та недоліки. 5. Формулювання завдання. 6. Покроковий алгоритм роботи сервісу. 7. ER-Diagram бази даних. 8. Use Case Diagram. 9. Створення вебдодатку. 10. Бізнес план розробки автопостингу.

6. Дата видачі завдання 11.09.2019 р.

Керівник _____ Савка І.Я.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Навроцький Б. П.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
Постановка задачі розробки сервісу автопостингу та аналіз предметної області	12.11.2019	Виконав
Аналіз основних характеристик, розробка моделей та алгоритмів для створення автопостингу	02.02.2020	Виконав
Розробка вебсервісу автопостингу	25.04.2020	Виконав
Бізнес план розробки автопостингу	3.05.2020	Виконав
Оформлення пояснювальної записки	18.05.2020	Виконав

Студент _____ Навроцький Б.П.
(підпис) (розшифровка підпису)

Керівник проекту _____ Савка І.Я.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 73 сторінки (без додатків), 42 рисунки, 33 джерела, 1 додаток на 41 сторінці.

Ключові слова: FLASK, БАЗА ДАНИХ, SQLALCHEMY, FLASK-ADMIN, FLASK-LOGIN, ER-МОДЕЛЬ, S3, REDIS, HEROKU, RQ, AUTOPOSTING..

Об`єктом дослідження є веб частина вебсервісу для автопостингу в соціальних мережах.

Мета роботи – спроектувати та розробити веб частина вебсервісу для автопостингу в соціальних мережах, набути практичних навичок у роботі з фреймворком Flask.

Стислий опис тексту пояснювальної записки:

У даному дипломному проекті описано основні етапи проектування та розробки вебдодатку на фреймворку Flask, такі як: створення архітектури проекту, моделі бази даних, створення і редагування форм, маршрутизація у проекті, а також підключення бота і публікація сервісу на віддалений сервер Heroku.

ABSTRACT

Explanatory note: 73 pages (without appendix), 42 figures, 33 sources, 1 appendix on 41 pages.

Keywords: FLASK, DATABASE, SQLALCHEMY, FLASK-ADMIN, FLASK-LOGIN, ER-MODEL, S3, REDIS, HEROKU, RQ, AUTOPOSTING.

The object of research is the web part of the service for auto-posting in social networks.

The purpose of the work is to design and develop a web part of a service for auto-posting in social networks, to acquire practical skills in working with the Flask framework.

Brief description of the explanatory note:

This graduate work describes the main stages of design and development of a web application on the Flask framework, such as: creating a project architecture, database models, creating and editing forms and routing in the project, as well as connecting the bot and publishing the service to a remote Heroku server.

ЗМІСТ

ВСТУП

Для сучасного світу робота з соціальними мережами і просуванням свого продукту в інтернеті є дуже важливою, оскільки такий вид реклами є дешевим і дуже зручним, оскільки можна напряму вести діалог з користувачами. Останнім часом все популярнішою стає професія SMM-спеціаліста [1], основна робота якого є просування того чи іншого бренду в соціальних мережах. Для простоти і автоматизації такої роботи створюються сервіси автопостингу [2]. Вони підходять як для професійних SMM-спеціалістів, так і для звичайних блогерів, які створюють контент і потім хочуть легко і швидко його розповсюдити, слідкуючи за його аналітикою.

Актуальність теми:

Існуючі сервіси можуть частково або зовсім не задовільнити користувача в тих чи інших завданнях. Основною проблемою цього є велика вартість послуг. Адже на популярних сервісах з автопостингу ціни за послуги можуть доходити до 50-100$ в місяць за користування ними. Іншою проблемою в менш популярних сервісах є низький рівень роботи, або невеликий набір послуг. Дана тематика є актуальною, з огляду на те, як швидко зараз розвиваються всі соціальні мережі і наскільки важливо стало просувати свій продукт саме там. Таким чином актуальність даної теми зумовлена необхідністю підвищити рівень і набір послуг, а також зменшити витрати на це.

Об’єкт дослідження: актуальні сервіси для автопостингу.

Предмет дослідження: головні функції і можливості актуальних сервісів для автопостингу.

Методи дослідження: основні результати і висновки зроблено на основі теоретичної інформації та порівняння різних можливостей сервісів.

Мета дипломної роботи: аналіз і розробка вебдодатку для автопостингу в соціальних мережах

Для досягнення мети дипломної роботи поставлено такі завдання:

- розглянути усі можливі сервіси і їхні набори послуг та особливості на ринку;
- проаналізувати можливе вдосконалення їхніх можливостей, а також знайти спосіб здешевлення послуг;
- на основі аналізу даних, розробити сервіс, який буде містити в собі найважливіші функції автопостингу для найзручнішої і дешевої роботи з соціальними мережами.

Завдання роботи: розробка сервісу для автопостингу в соціальних мережах. Розробка вебдодатку.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ СЕРВІСУ АВТОПОСТИНГУ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний опис автопостингу

Для спрощення роботи і ведення соціальних мереж у сучасному світі використовують спеціальні сервіси для автопостингу або кроспостингу.

Автопостинг – автоматична публікація матеріалу на сайти, блоги, мікроблоги чи соціальні мережі [2].

Кроспостинг – навмисне автоматичне, напівавтоматичне або ручне розміщення однієї і тої самої статті, посилання чи повідомлення на різні соціальні мережі, форуми, блоги і т. д. [3].

За допомогою таких сервісів полегшується робота для спеціалістів чи простих блогерів. Переліком завдань таких сервісів є:

- автопостинг;
- кроспостинг;
- збереження певних хештегів;
- публікація по розкладу;
- видалення по розкладу;
- автоматичне видалення рекламних постів;
- адаптація поста під кожну соціальну мережу;
- вибір найкращого часу для розміщення постів;
- збереження посту в чернетку, для подальшого редагування.

За допомогою такого сервісу можна автоматично виконувати всі перелічені завдання набагато швидше і простіше, ніж робити це все вручну. Можна заздалегідь вибрати час і дату для публікації потрібного матеріалу. Саме тому його актуальність зараз дуже висока.

1.2 Особливості існуючих сервісів для автопостингу

На даний момент існує багато сервісів для автопостингу. Для порівняння і дослідження особливостей їх основних можливостей буде розглянуто три з них.

Першим прикладом такого сервісу є “NovaPress publisher” [4]. Серед його переваг є зручність і простота в управлінні (див. Рисунок 1.1). Проте з мінусів його мала функціональність.

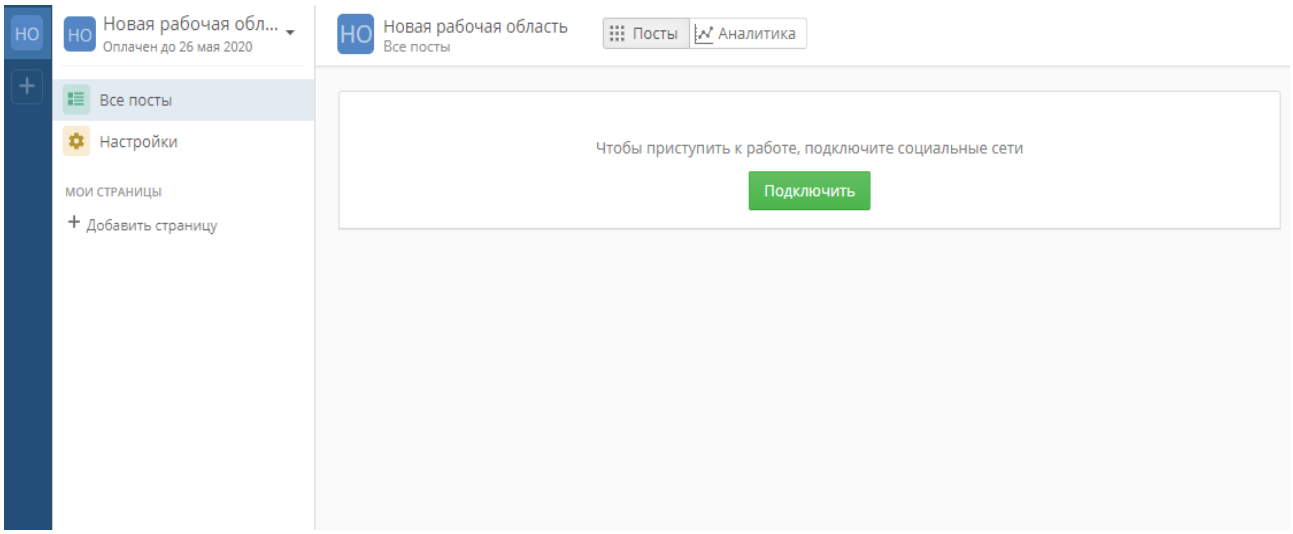

Рисунок 1.1 – Головна сторінка автопостингового сервісу NovaPress Publisher

Отже, основними можливостями цього сервісу є:

- відкладена публікація – сервіс публікує потрібний запис у заданий час;
- кроспостинг – функція, яка дозволяє один і той запис публікувати у різні соціальні мережі;
- хештеги – зберігати необхідну колекцію хештегів, для певного проекту;
- імпорт записів з сайту – функція, що дозволяє при додаванні запису чи статті на вашому сайті, зразу публікувати її у соціальні мережі;
- імпорт відео з YouTube, Rutube, vimeo – дозволяє ділитись відео з даних відеохостингів;
- також можна створювати команду або додавати водяний знак на дописи.

Водночас у даного сервісу також є багато недоліків, зокрема:

- відсутня функція «видалення рекламних постів», або по розкладу – функція, яка дозволяє певні пости, або історії видаляти через потрібний проміжок часу. Ця функція є корисною для того, щоб не засмічувати сторінку зайвою рекламою;
- відсутній «випадковий автопостинг» – функція, яка дозволяє автоматично вибрати дату і час певним вибраним дописам;
- відсутній «розумний автопостинг» – функція, яка аналізує найкращий час для постингу і публікує найважливіші пости саме в такій годині;
- відсутня функція «Зберігання в чернетку» – функція, необхідна для створення допису, з подальшим його доопрацюванням і публікацією.

Окрім цього, варто також вказати що ціна користування даним сервісом буде невеликою, за користування 5 сторінок 7.5$ в місяць. За користування 10 і більше від 15$ і + 1$ за кожну сторінку.

Отже, можна зробити висновок, що попри невелику кількість функцій, ціна залишається малою і тому сервіс залишається популярним на ринку.

Протилежністю до нього є сервіс «Amplifr» [5], оскільки він містить в собі більше можливостей, але оплата за його послуги є значно більшою (див. Рисунок 1.2).

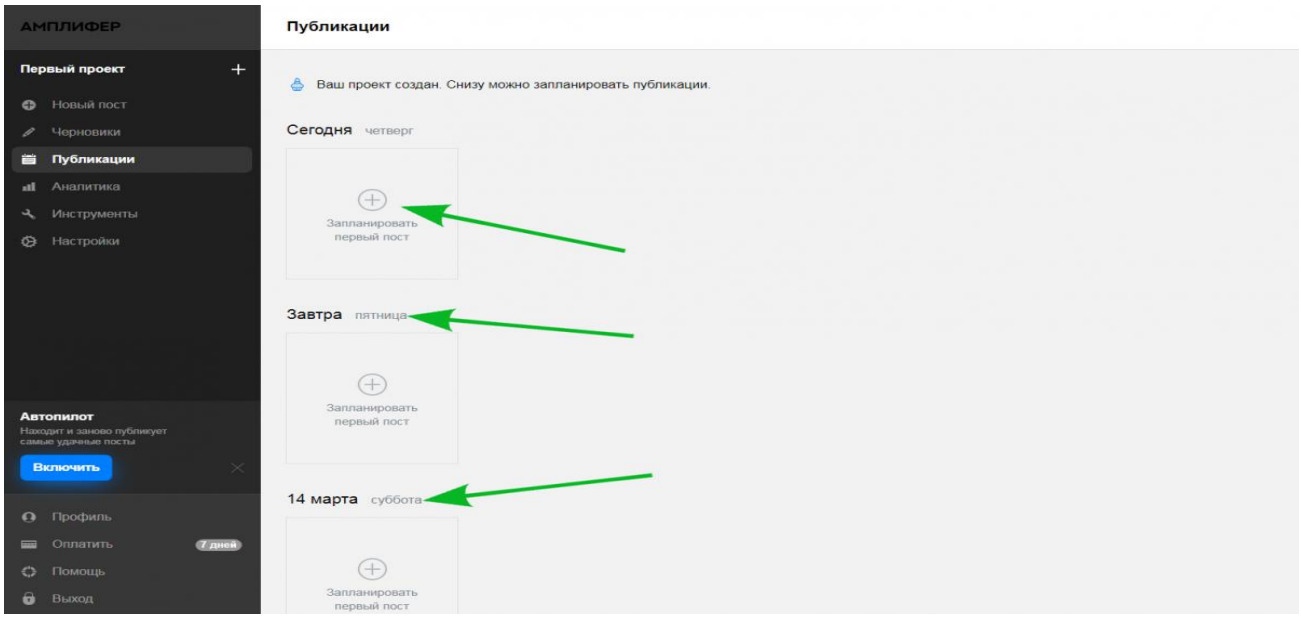

Рисунок 1.2 – Головна сторінка автопостингового сервісу Amplifr

Основними можливостями і перевагами є:

- відкладена публікація;
- кроспостинг;
- Видалення по розкладу;
- хештеги;
- також можна створювати команду, або додавати водяний знак на дописи;
- зберігання в чернетку;
- «розумний автопостинг» – функція, яка на основі аналітики раніше опублікованих дописів, дозволяє вирахувати найкращий час для публікації;
- «випадковий автопостинг» – функція, яка дозволяє певні пости публікувати у випадковому порядку й часі;
- аналітика і статистика;
- кращі пости за тиждень чи місяць.

Основними недоліками функціональності є:

- відсутність видалення рекламних постів – функція, яка дозволятиме видаляти рекламні пости відразу, як закінчиться відповідний час очікування;
- відсутність функції «адаптація під вид кожної соціальної мережі» – функція, яка допомагає користувачу автоматично адаптувати будь-який запис під довільну соціальну мережу автоматично, або вручну, якщо це необхідно користувачу.

Окрім цього, варто також вказати що ціна користування даним сервісом буде великою, за користування 5 сторінок і неповним набором функцій потрібно платити від 15$ в місяць. За користування 15-25 і більше сторінок від 50-100$ і більше в залежності від тарифу.

Третім сервісом, який буде розглянутим, є SMMplaner [6]. Він є міцним середнячком з середніми можливостями і доволі не дешевими цінами як для спектру своїх послуг (див. Рисунок 1.3).

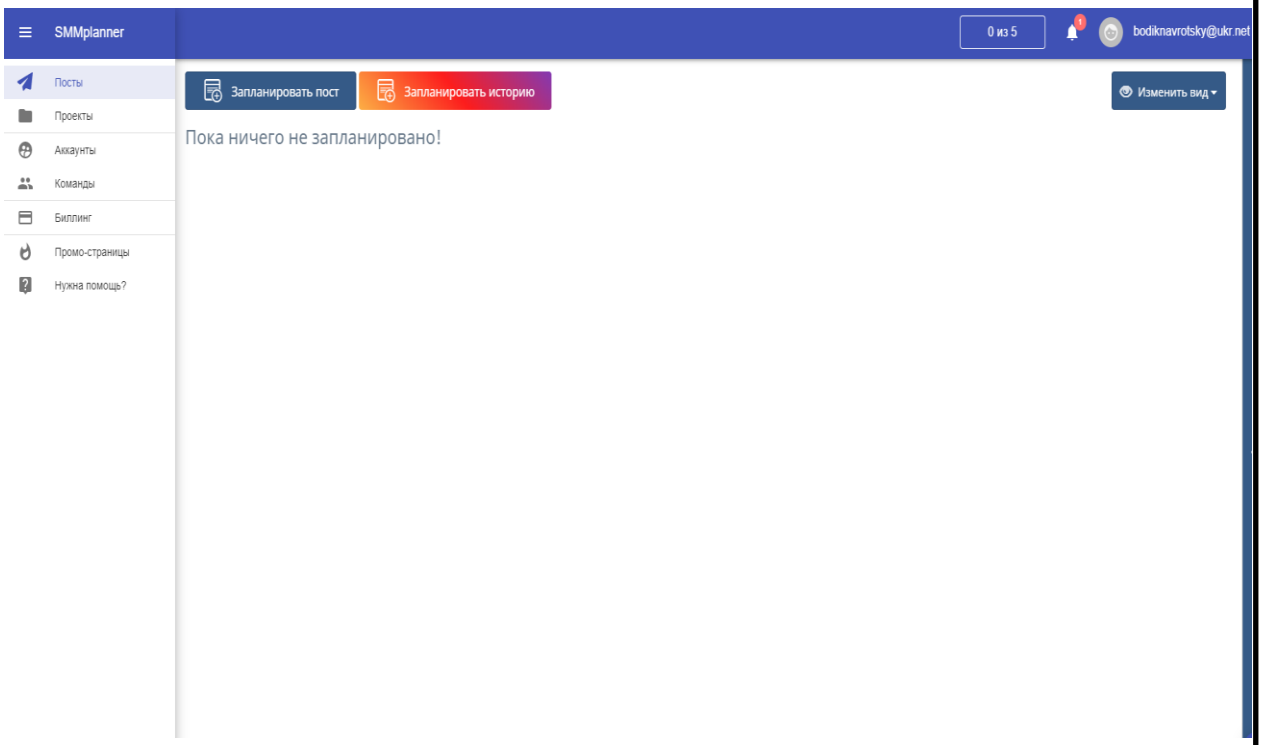

Рисунок 1.3 – Головна сторінка автопостингового сервісу SMMplaner

Основними можливостями і перевагами є:

- відкладена публікація;
- кроспостинг;
- постинг відео файлів – дозволяє загружати і постити автоматично відео файли;
- хештеги;
- також можна додавати водяний знак на дописи.

Основними недоліками функціональності є:

- відсутність функції «адаптація під вид кожної соціальної мережі» – функція, яка допомагає користувачу автоматично адаптувати будь-який

запис під довільну соціальну мережу автоматично, або вручну, якщо це необхідно користувачу;

- відсутність створення чернеток;

- відсутність функції «випадковий автопостинг»;

- відсутня функція «видалення рекламних постів», або по розкладу – функція, яка дозволяє певні пости видаляти через потрібний проміжок часу. Ця функція є дуже корисною, щоб не засмічувати сторінку зайвою рекламою.

Окрім цього, варто також вказати що ціна користування даним сервісом буде порівняно великою, за користування 5 сторінок і неповним набором функцій потрібно платити від 200 грн в місяць. За користування 25 і більше сторінок від 700 грн і більше в залежності від тарифу.

Отже, в результаті аналізу особливостей існуючих сервісів для автопостингу можна зробити висновок, що всі три сервіси мають свої переваги і недоліки, “NovaPress publisher” – дешевший у користуванні, але має набагато менше корисних функцій, в той же час як «Amplifr» має достатньо функцій, проте дуже дорогий в масовому використанні. Щодо сервісу SMMplaner, то він доволі дорогий у використанні, але при цьому має доволі обмежену кількість функцій у порівнянні з конкурентами.

1.3 Постановка задачі

Основною задачею дипломної роботи є розробка вебдодатку для автопостингу в соціальних мережах.

Вхідними даними повинні бути:

- дані облікового запису користувача;

- тема, основний текст і всі допоміжні елементи допису;

- вибір кроспостингу, хештегів і інших параметрів;

- вибір часу публікації, виду публікації, якщо потрібно – видалення.

Вихідними даними повинні бути:

- публікація певного допису на заданій сторінці соціальної мережі;
- повідомлення про успішну чи невдалу публікацію на самій сторінці вебдодатку.

Сервіс повинен приймати вхідні дані на вебдодатку, після чого передавати дані спеціальному боту, який публікуватиме ці дані у соціальних мережах, а назад на сайт передаватиме успішність чи неуспішність публікації.

Для створення вебдодатку потрібно:

1. розробити основну функціональну частину вебдодатку;
2. розробити дизайн сайту;
3. реалізувати базу даних;
4. створити адміністративну частину сайту;
5. створити функції для комунікації з ботом;
6. розробити сторінку реєстрації, авторизації, профілю, поста і всі допоміжні сторінки на сайті;
7. врахувати всі перечисленні мінуси і плюси аналогічних сервісів, і реалізувати весь необхідний функціонал, а саме:
    - «відкладена публікація»;
    - «кроспостинг»;
    - «видалення по розкладу»;
    - «хештеги»;
    - «зберігання в чернетку»;
    - «розумний автопостинг»;
    - «випадковий автопостинг»;
    - «видалення рекламних постів»;
    - «адаптація під вид кожної соціальної мережі».

Плюсами даного сервісу буде:

- простота використання;
- зручність використання;

- економія часу;

- можливість роботи через інтернет;

- можливість створення, редагування, видалення дописів в одному місці і їх розповсюдження з одного кабінету;

- майже монополія в україномовному сегменті;

- безплатне користування для університету, на відмінну від його платних аналогів.

Мінусами даного сервісу буде:

- неможливість роботи без підключення до мережі інтернет;

- відносно менший функціонал у порівнянні з набагато дорожчими конкурентами.

2 АНАЛІЗ ОСНОВНИХ ХАРАКТЕРИСТИК, РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ДЛЯ СТВОРЕННЯ АВТОПОСТИНГУ

2.1 Аналіз основних характеристик

Для аналізу основних вимог і характеристик проекту існує багато джерел для виявлення їх. Основними джерелами для виявлення потрібних характеристик у проекті є:

- інтерв'ю – спілкування з замовником чи групою замовників, про те, яким необхідний кінцевий продукт, створення його моделі, а також обговорення можливого функціонування [7];
- анкетування – проведення опитування аудиторії, яка так чи інакше зв'язана з необхідною тематикою. Цей спосіб найменш ефективніший, оскільки часто респонденти бувають не зовсім компетентними, або просто не вмотивованими в тому, щоб інформативно і чітко заповнювати анкету [7];
- спостереження – отримання інформації про основні характеристики та моделі необхідного проекту, шляхом спостереження за іншими аналогічними проектами, а також співставлення з своїми існуючими розробленими планами та ідеями [7];
- прототипування – створення швидких прототипів, моделей, поведінки програмного забезпечення, яке виконуватиме усі поставлені завдання, а згодом редагування і від форматування його до потрібного вигляду [7]. Основним прикладом такої концепції є технологія RAD (Rapid Application Development) — швидка розробка за стосунків [8].

Проаналізувавши всі вище перелічені джерела, було прийнято рішення, що для аналізу даного проекту, використається два з перелічених джерела, а саме: спостереження і прототипування. Оскільки інтерв'ю і анкетування не

можливо до проведення через відсутність замовника і прямої цільової аудиторії, яка буде компетентною в питанні розробки саме цього проекту.

Щоб представити можливі сценарії, моделі, алгоритми виконання роботи існують діаграми UML (Unified Modeling Language) – уніфікована мова моделювання [9]. Серед варіантів використання цих діаграм є:

- Use Case Diagram – діаграма варіантів використання. Вони задумана так, щоб дати максимальне уявлення про можливості і функціональність створеної системи, а також загальне уявлення про варіанти використання [7];
- Sequence Diagram – діаграма послідовностей, яка відображає взаємодії об’єктів за часом їх використання і виконання. У ній відображають задіяні об’єкти, а також послідовність дій з ними [10];
- Activity Diagram – діаграма діяльності, візуальне представлення діяльностей за допомогою графа [11];
- Statechart Diagram – діаграма станів, візуально моделює зміну стану об’єкту [7];
- Class Diagram – діаграма класів, статичне представлення структури моделі. Відображає модель бази даних [7];
- Verticle Timeline Diagram – діаграма для покрокового алгоритму виконання завдання, або їх висвітлення у часовому чи покрокову просторі [12].

На основі отриманих результатів було розроблено наступні об’єкти:

- покроковий алгоритм функціонування вебсервісу для виконання поставленого завдання (Verticle Timeline Diagram);
- діаграма прецедентів, тобто діаграма варіантів використання сервісу (Use Case Diagram);
- діаграма послідовностей, яка відображає взаємодії об’єктів за часом їх використання і виконання (Sequence Diagram);
- діаграма бази даних (ER Diagram) [13].

2.2 Vertical timeline diagram

На даній діаграмі (див. рисунок 2.1) зображений покроковий алгоритм виконання основного завдання сервісу, а саме – автопостингу. Він чітко демонструє як саме взаємодіє користувач з сервісом.

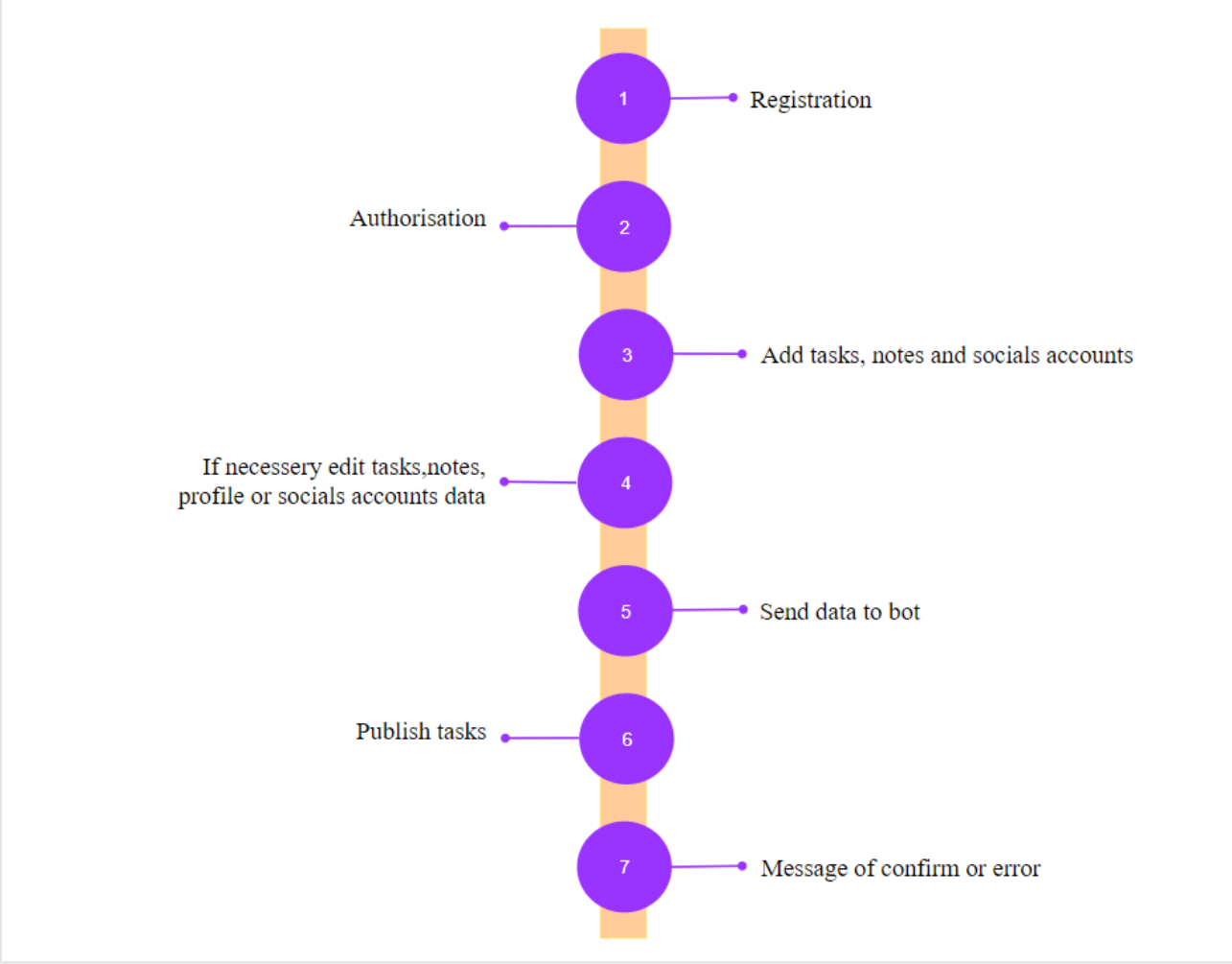

Рисунок 2.1 – Vertical timeline diagram. Покроковий алгоритм взаємодії користувача з сервісом

Алгоритм складається з семи кроків:

1. registration (реєстрація) – перший пункт алгоритму, процедура в якій кожен користувач може зареєструватись. Інформацію буде збережена в базу даних, що дозволить в подальшому в будь-який момент часу

переходити до наступного другого кроку – авторизації. Адже без реєстрації і авторизації користуватись даним сервісом буде не можливо;

2. authorization (авторизація) – керування засобами доступу до даного сервісу, здійснюється за допомогою ідентифікатора(в даному випадку email-адреси, та пароля, який створюється при проходженні першого крока, а саме реєстрації;

3. add tasks, notes, socials accounts (додавання завдань, чернеток, соціальних мереж) – це третій крок, який є одним з найголовніших в даному алгоритмі, адже саме тут здійснюється створення найважливіших об’єктів, а саме поста, чернеток та додавання облікових записів соціальних мереж. Завдання або пост – це набір даних які будуть опубліковані за заданим часом, у задані згодом соціальні мережі. Чернетка – недопрацьований пост, який можна редагувати скільки завгодно разів, до моменту його переведення в завдання. Облікові записи соціальних мереж – це дані від сторінок соціальних мереж, куди необхідно буде публікувати весь матеріал;

4. if necessary edit tasks, notes, profiles or socials accounts data (якщо потрібно редагування завдань, чернеток, чи даних облікових записів соціальних мереж) – цей пункт містить у собі можливість редагування і налаштування всієї інформацію, яку було створено на попередньому етапі. А саме, редагування завдань, в даному етапі можна відредагувати завдання, які досі не були опубліковані, додати чи видалити потрібні облікові записи соціальних мереж на які публікуватиметься дане завдання, а також редагувати час в який публікуватиметься задане завдання. Редагування чернеток дозволяє змінити будь-яку інформацію, яка розміщена в чернетці, а також додати при необхідності дату і час і перемістити даний об’єкт до завдань для подальшої публікації. Редагування облікових записів у соціальних мережах забезпечує можливість зміни даних до потрібного облікового запису;

5. send data to bot (надсилання інформації боту) – п’ятий крок, який забезпечує надсилання інформації до бота, який потім публікуватиме задану інформацію у виділені в завданні облікові записи соціальних мереж. Він виконується двома способами, або публікація за розкладом, тобто додається до черги і публікується у заданий час, або миттєва публікація, тобто миттєво передає інформацію боту, який виконає завдання публікації;

6. publish tasks (публікування завдань) – процес взаємодії бота з соціальною мережею і проведення публікації поста у ній;

7. message of confirm or error (повідомлення про успішність виконання чи помилку) – це останній, сьомий пункт, який дозволяє побачити користувачу статус успішності чи неуспішності публікування у заданих соціальних мережах.

2.3 Use case diagram

Ця діаграма (див. рисунок 2.2) розроблена для того, щоб дати максимально можливе розуміння про використання і функціональності створеного сервісу, про можливості користувачів а також їх ролі. Мета цієї схеми – продемонструвати варіанти взаємозв’язку кожної з ролей з всіма можливими варіантами використаннями. Діаграма прецедентів складається з таких головних аспектів як [7]:

- множина акторів – можливих ролей користувачів. В заданому випадку існує дві ролі: User (простий користувач) та Admin (адміністратор – користувач з певними привілеями) [7];
- прецедентів – варіантів використання. Використовуються для описання послуг чи функціоналу, які система надає актору, тобто користувачу. На діаграмі зображені блакитним кольором, розташовані вертикально [7];

- відношень між акторами та прецедентами. Зв'язки, стрілки, які з'єднують акторів з прецедентами, але при цьому не йдеться про те як саме взаємодіятиме актор з прецедентом [7].

Рисунок 2.2 – Use Case Diagram. Діаграма варіантів використання можливостей і ролей користувачів

Діаграма демонструє, що користувач User зможе використовувати наступні послуги:

- registration (реєстрація) – як описувалось раніше, можливість завести обліковий запис в даному сервісі, оскільки використання вебдодатку без реєстрації неможливе;

- authorization (авторизація) – для користування сервісом необхідно авторизуватись під своєю email-адресою та паролем, які були створені під час реєстрації;

- view profile and tasks (перегляд профілю і завдань) – можливість перегляду свого особистого профілю, зміна паролю в ньому. Окрім цього, є змога переглянути список всіх завдань чи чернеток, або перейти на сторінку якогось певного завдання, або чернетки, для того щоб згодом відредагувати його;

- create, update and delete tasks (створення, оновлення, або видалення завдань) – надає доступ до таких функцій сервісу як створення, редагування і видалення завдань. Створення, як зазначалось раніше, це одна з головних функцій проекту, адже саме завдяки цій функції можна створити нове завдання, яке потім може бути перетвореною в певний пост і розміщеним в різних соціальних мережах. Оновлення або редагування – для того щоб можна було змінити дату, текст, теги чи будь-яку іншу інформацію в потрібному завданні, яке досі не опубліковано. Видалення – видалення певного завдання з переліку всіх завдань і бази даних, при такій необхідності;

- add social accounts (додавання облікових записів у соціальних мережах) – можливість прикріплювати існуючі облікові записи у соціальній мережі, для того щоб у майбутньому постити в них необхідні матеріали.

Щодо можливостей користувача admin, то він може все перелічене вище, окрім реєстрації, оскільки просто так зареєструватись адміністратором неможливо. Також у користувача даної ролі розширюються повноваження, а саме додаються наступні можливості:

- create, delete and edit all posts from spam (створювати, видаляти, та редагувати всі пости від спаму, або недозволених матеріалів) – можливість створювати, а також змінювати чи видаляти інші будь які завдання з адміністративної панелі. Ця функція створена для того, щоб

можна було переглядати і редагувати раніше створені завдання, оскільки там можливо буде присутній спам, або інші недозволені речі;

- delete accounts (видалення облікових записів) – у випадку, якщо буде багаторазове порушення правил користування сервісом, можливе повне видалення облікового запису;

- create admin (створення адміністратора) – можливість створення іншого адміністратора, або надання прав адміністратора – користувачу. Ця функція розроблена для того, щоб можна було додати чергового адміністратора, який потім зможе відслідковувати чесність всіх користувачів, а також перевіряти їхні дописи на спам і недопустимі фотографії чи повідомлення.

2.4 Sequence diagram

Наступною є діаграма послідовності (див. рисунок 2.3), яка відображає взаємодії об’єктів за часом. У ній зображені об’єкти які безпосередньо беруть участь у взаємодії. Вона складається з таких елементів:

- об’єктів – елементів, які приймають участь у взаємодії. Зображується горизонтальними прямокутниками, з назвами самих об’єктів [14];

- лінія життя – зображується вертикальною пунктирною лінією, яка асоціюється з одним вказаним об’єктом і служить для позначення періоду часу протягом якого об’єкт існує і бере участь у взаємодіях системи. У випадку, якщо елемент існує в системі постійно, то його лінія життя має продовжуватись по всій діаграмі, від верху до низу [14];

- фокус управління – зображується у формі вузького прямокутника, який витягнутий вздовж лінії життя. Верхня частина позначає початок отримання фокусу управління, а його нижня сторона – закінчення. Він розміщується нижче об’єкта і може замінювати його лінію життя, якщо є

активним протягом всієї лінії. За допомогою нього і відрізняють активну фазу від пасивної [14];

- повідомлення і стрілки – опис взаємодії, яка відбувається в даний момент часу, прийом повідомлень яких ініціює запуск наступного завдання у життєвому циклі [14].

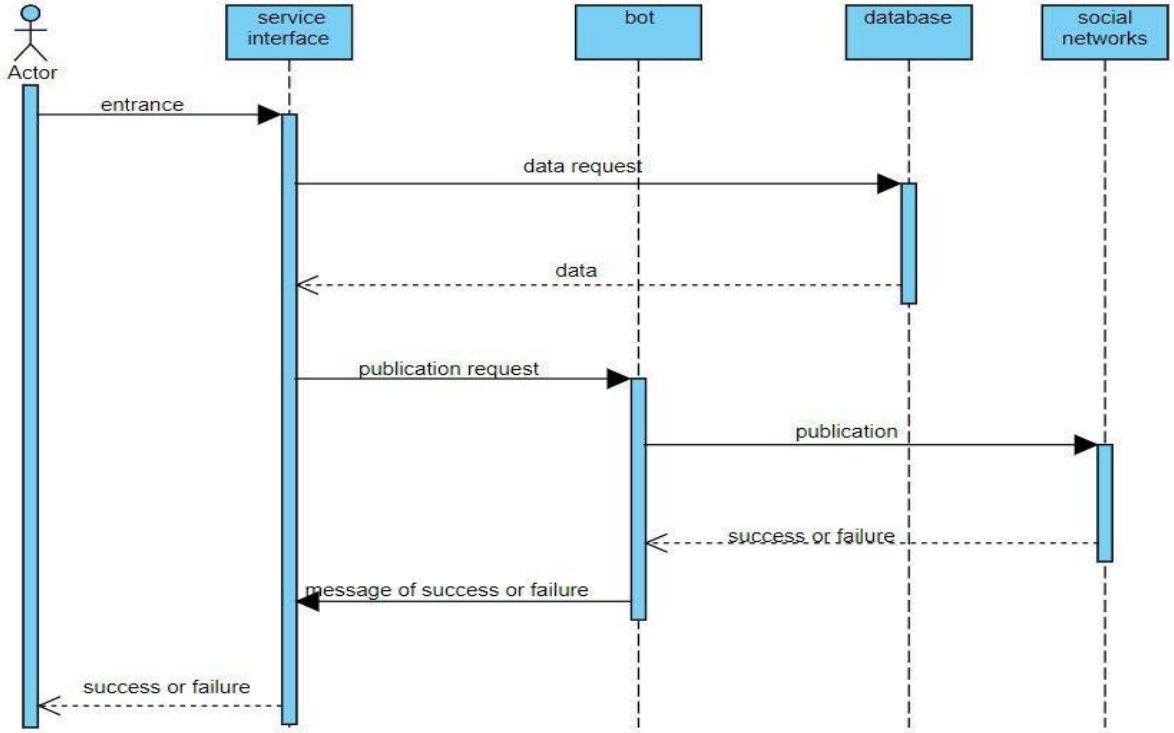

Рисунок 2.3 – Sequence Diagram. Діаграма послідовностей взаємодії об’єктів за часом.

На діаграмі зображено наступні чотири об’єкти:

- service interface (інтерфейс сервісу) – сам сервісний інтерфейс з яким взаємодіє користувач;

- bot (бот) – скрипт, який виконує бере дані і виконує саму роботу автопостингу;

- database (база даних) – база даних, де зберігаються всі дані про користувача, збережено його майбутні пости, фотографії, час відправки повідомлення і т. д.;

- social networks (соціальні мережі) – самі соціальні мережі на які публікується заданий матеріал.

Лінія життя проходить крізь весь життєвий шлях об’єкта інтерфейс, оскільки протягом всього часу, користувач так чи інакше стикається з ним, спочатку задля того щоб увійти (entrance). Тепер він взаємодіє з інтерфейсом. Для того щоб зареєструватись, чи створити завдання, або провести будь-які інші маніпуляції інтерфейс працює з базою даних вебсервісу. Для цього може здійснюватись запит (data request) до бази даних через інтерфейс для створення чи редагування. Також коли необхідно дістати щось з бази для того, щоб надіслати її боту повертається інформація (data). На цьому моменті закінчується життєвий цикл бази даних.

Після цього інформація через запит (publication request) потрапляє до бота. Його життєвий цикл розпочинається. Потім він публікує (publication) допис в соціальній мережі, після чого йому назад повертається відповідь(success or failure) про успішність або не вдаліть виконання публікації. Життєвий час соціальних мереж триває тільки в період перебування тут бота. Після того як бот отримує повідомлення, його життєвий цикл закінчується і він передає це повідомлення назад інтерфейсу вебсервісу, який в свою чергу також закінчує життєвий цикл. Тепер користувач бачитиме повідомлення про успішність або помилку при виконанні того чи іншого завдання, а це значить, що життєвий цикл послідовностей закінчується.

2.5 Опис ER-Diagram

При конструюванні структури бази даних прийнято використовувати ER-модель (або ER-діаграма) [13].

Модель «сутність-зв'язок» (ER-модель) (англ. Entity-relationship model або entity-relationship diagram) — модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. ER-

модель — це мета-модель даних, тобто засіб опису моделей даних. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення що його реалізує, є модель «сутність-зв'язок». Важливим є той факт, що з моделі «сутність-зв'язок» можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найзагальнішою [13].

Модель сутність-зв'язок є результатом систематичного процесу, який описує та визначає деяку предметну область. Вона не визначає сам процес, а лише візуалізує його. Дані представлені у вигляді компонентів (сутностей), які пов'язані між собою певними зв'язками, які виражають залежності і вимоги між ними, такі як: одна будівля може бути розділена на нуль або більше квартир, але одна квартира може бути розташована лише в одній будівлі. Сутності можуть мати різні властивості (атрибути), які характеризують їх. Діаграми, створені для представлення цих сутностей, атрибутів і зв'язків графічно, називають сутність-зв'язок діаграмами.

У разі реляційної бази даних, в якій зберігаються дані в таблицях, кожен рядок кожної таблиці являє собою один екземпляр сутності. Деякі поля даних в цих таблицях вказують на індекси в інших таблицях. Такі поля є покажчиками фізичної реалізації зв'язків між сутностями.

- **сутність** – це клас однотипних об'єктів, інформація про яких повинна бути врахована в моделі. Кожна сутність повинна мати найменування, виражене іменником в однині [13];
- **примірник сутності** – це конкретний представник даної сутності. Наприклад, представником сутності "storage" може бути "storage №23". Екземпляри сутностей повинні бути помітні, тобто сутності повинні мати деякі властивості, унікальні для кожного екземпляра цієї сутності [13];
- **атрибут сутності** – це іменована характеристика, що є деякою властивістю сутності. Найменування атрибута має бути виражене іменником в однині (можливо, з характеризують прикметниками).

Прикладами атрибутів сутності " storage " можуть бути такі атрибути як "id","name" і т. д. Атрибути зображуються в межах прямокутника, що визначає сутність [13];

- **ключ сутності** – це ненадлишковий набір атрибутів, значення яких у сукупності є унікальними для кожного екземпляра сутності. Ненадлишковий полягає в тому, що видалення будь-якого атрибута з ключа порушується його унікальність. Сутність може мати кілька різних ключів. Ключові атрибути зображуються на діаграмі з ключиком [13];
- **зв'язок** – це деяка асоціація між двома сутностями. Одна сутність може бути пов'язана з іншою сутністю або сама з собою. Зв'язки дозволяють по одній сутності знаходити інші суті, пов'язані з нею [13]:

1. зв'язок типу **один-до-одного** означає, що один примірник першої сутності пов'язаний з одним примірником другої сутності [15];
2. зв'язок типу **один-до-багатьох** означає, що один примірник першої сутності пов'язаний з декількома екземплярами другий сутності. Це найбільш часто використовуваний тип зв'язку. Одна сутність називається батьківською, інша – дочірньою [15];
3. зв'язок типу **багато-до-багатьох** означає, що кожен екземпляр першої сутності може бути пов'язаний з декількома екземплярами другої сутності, і кожен примірник другої суті може бути пов'язаний з декількома примірниками першої сутності [15].

2.6 Побудова ER-Diagram бази даних

Для побудови ER-Diagram потрібно проаналізувати весь необхідний функціонал майбутнього вебсервісу. Виходячи з всіх вище перелічених діаграм було зроблено висновки та розроблено таку модель бази даних, яка задовольнятиме всі вимоги і виконуватиме всі поставлені завдання (див. рисунок 2.4).

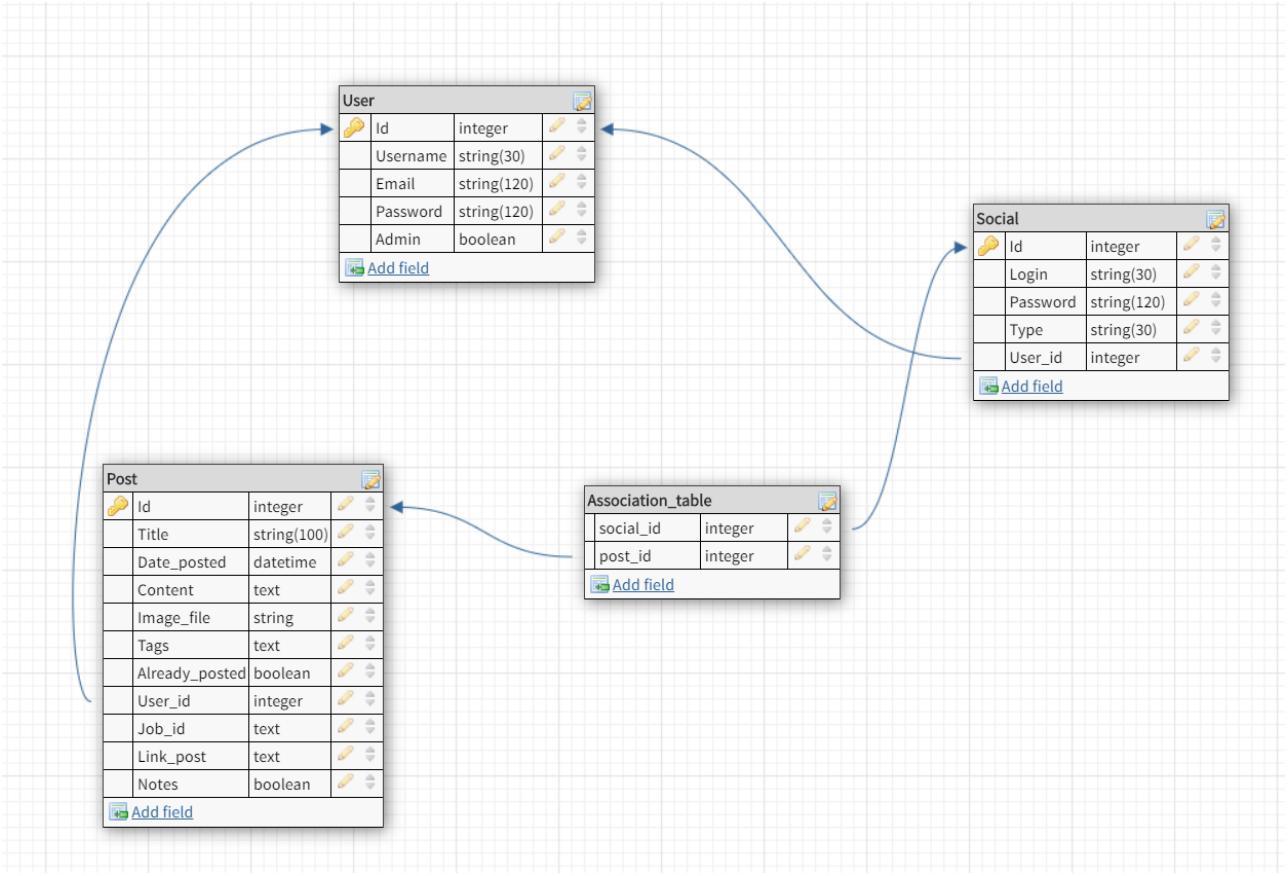

Рисунок 2.4 – ER-Diagram. Діаграма моделі бази даних вебсервісу

На цій діаграмі зображено три основні таблиці, та одна проміжна таблиця зв'язку багато-до-багатьох. Головною таблицею є User, яка відповідає за створення облікового запису користувача. Вона складається з п'яти наступних атрибутів:

- id – поле, яке відповідає за унікальний ідентифікатор користувача, який створюється автоматично при реєстрації. Його неможна задати вручну. Цей атрибут є первинний ключем(primary key) з типом змінної "integer", тобто цілого числа;
- username – унікальний атрибут, не може приймати нульове значення, відповідає за унікальне ім'я користувача на вебсервісі. Тип даних – "string", можлива довжина до 30 символів;
- email – унікальний атрибут, не може приймати нульове значення, відповідає за адресу унікальної електронної поштової скриньки користувача. Тип даних – "string", можлива довжина до 120 символів;

- password – пароль користувача. Тип даних – “string”, можлива довжина до 120 символів;

- admin – атрибут, який приймає значання істини або хиби. Якщо користувач є адміністратором, то він приймає значення істини. По замовчуванню стоїть хиба.

Наступною розглянутою таблице є Social:

- id – атрибут, який відповідає за унікальний ідентифікатор облікового запису соціальної мережі, який створюється автоматично при створенні нового об’єкту цієї сутності. Його неможна задати вручну. Цей атрибут є первинний ключем(primary key) з типом змінної “integer”, тобто цілого числа;

- login – атрибут, відповідає за логін з облікового запису потрібної соціальної мережі, з типом даних – “string”, можлива довжина до 30 символів;

- password – атрибут, що відповідає за пароль до облікового запису потрібної соціальної мережі з типом даних – “string”, можлива довжина до 120 символів;

- type – обов’язковий атрибут, який відповідає за те, який тип соціальної мережі створюється, має тип даних “string”, можливо довжина до 30 символів;

- user_id – вторинний ключ, тобто посилання на елемент з таблиці User, має тип даних “integer”.

Між таблицями User і Social існує зв'язок один-до-багатьох, тобто один елемент з таблиці User може містити декілька елементів з таблиці Social. Зв’язок для таких елементів відбувається по спеціальних ключах: первинному ключі в головній таблиці, та вторинним ключем у дочірній.

Останньою з головних таблиць розглядається таблиця Post, яка містить наступні атрибути:

- id – атрибут, який відповідає за унікальний ідентифікатор створеної публікації, який створюється автоматично при створенні нового об’єкту цієї сутності. Його неможна задати вручну. Цей атрибут є первинний ключем (primary key) з типом змінної “integer”, тобто цілого числа;

- title – обов’язковий атрибут, служить для створення заголовка до поста з типом даних – “string”, можлива довжина до 100 символів;

- date_posted – атрибут, який визначає дату і час публікації поста, тип даних “datetime”;

- content – обов’язковий атрибут, який створює основний вміст публікації. Тип даних “text”;

- image_file – атрибут, який зберігає посилання на файл на сервері є необов’язковим атрибутом, оскільки в деяких публікація можлива відсутність фотографії. Тип даних “string”;

- tags – атрибут, що відповідає за створення і додавання тегів до публікації. Типом даних для даного атрибута є “text”;

- already_posted – атрибут, який приймає значання істини або хиби. Якщо пост вже є опублікованим, то він приймає значення істини. По замовчуванню стоїть хиба;

- user_id – вторинний ключ, тобто посилання на елемент з таблиці User, має тип даних “integer”;

- job_id – атрибут, який відповідає за елемент “Job”, який додається в чергу при публікації, для подальшого його відслідковування. Тип даних “text”;

- link_post – атрибут, що зберігає в собі посилання на вже створену публікації в соціальній мережі. Тип даних “text”;

- notes – атрибут, який приймає значання істини або хиби. Якщо завдання немає вказаної дати, часу, або просто помічено як чернетка, то прийме значення істини. По замовчуванню і інших випадках набуватиме хиби.

Між таблицями User і Post існує зв'язок один-до-багатьох, тобто один елемент з таблиці User може містити декілька елементів з таблиці Post. Зв’язок для таких елементів відбувається по спеціальних ключах: первинному ключі в головній таблиці, та вторинним ключем у дочірній.

Між таблицями Post і Social існує зв'язок багато-до-багатьох, тобто кожен елемент з таблиці Post може містити декілька елементів з таблиці Social, а кожен елемент з таблиці Social може містити декілька елементів з таблиці Post. Зв’язок для таких елементів відбувається за допомогою спеціалізованої таблиці, яка з’єднує їх. Ця таблиця має назву Association_table і містить тільки два атрибути “social_id” та “post_id”, які за допомогою первинних і вторинних ключів здійснюють зв’язок з основним таблицями.

3 РОЗРОБКА ВЕБСЕРВІСУ АВТОПОСТИНГУ

3.1 Розробка моделі

Для розробки проекту була використана мова Python [16-18], а саме мікрофреймворк Flask [19,20]. Для створення самої моделі використовувався допоміжній інструментарій SQLAchemy [21,22]. Це об'єктно-реляційне відображення для мови Python. Це означає, що за допомогою цього інструменту база даних зв'язується з об'єктно-орієнтованими мовами програмування, в даному випадку з мовою Python. Модель розроблялась на основі ER-діаграми. Існує три головні таблиці: User, Social та Post. Окрім цього створена допоміжна таблиця, яка з'єднує таблиці Social та Post зв'язком багато-до-багатьох (див. Додаток А).

Розглядається даний приклад коду (див. Рисунок 3.1), який демонструє створення класу User, який відповідає за створення користувача. Усі його атрибути створені на основі ER-діаграми, що конструювалась раніше.

```
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(30), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)
    socials = db.relationship('Social', backref='owner', lazy=True)
    posts = db.relationship('Post', backref='author', lazy=True)
    admin = db.Column(db.Boolean())

    def __repr__(self):
        return self.username

    def __init__(self, username, email, password, admin=False):
        self.username = username
        self.email = email
        self.password = password
        self.admin = admin

    def is_admin(self):
        return self.admin
```

Рисунок 3.1 – Код класу User

Окрім атрибутів, тут також є декілька методів, а саме: “__repr__”, “__init__”, “is_admin”. Далі розглядатиметься створення кожного атрибута і метода детальніше. Цей клас складається з наступних атрибутів:

- id – унікальний ідентифікатор, який створюється автоматично з об’єктом класу, а також є первинним ключем. Тип даних “integer”;
- username – ім’я користувача, також унікальний атрибут, який не може бути пустим при створенні об’єкта цього класу. Може досягати до 30 символів. Тип даних “string”;
- email – електронна пошта користувача, унікальний атрибут, також не може бути пустим при створенні об’єкта класу. Може досягати довжини 120 символів. Тип даних “string”;
- password – пароль, не може бути пустим при створенні. Атрибут який відповідає за збереження пароля користувача. Може сягати до 120 символів. Тип даних “string”;
- socials – атрибут, який створює зв’язок з таблицею Social. Це означає, що об’єкт класу User зможе містити декілька елементів класу Social за зверненням User.Socials[n], де n – номер елемента. Або в той же час, може здійснюватись зворотній виклик – Social.owner. Тобто через об’єкт Social зможемо отримати доступ до власника об’єкту, а саме якогось з елементів класу User;
- posts – аналогічний до переднього атрибут, тільки він створює зв’язок з таблицею Post;
- admin – атрибут, що може приймати значення істини, або хиби, у випадку якщо користувач справді є адміністратором, то значення буде істиною, у випадку якщо користувач є простим то значення атрибуту буде – хиба.

Щодо методів цього класу то існують:

- метод “__repr__” – сприяє точному представленню об’єкта;

- метод “__init__” – конструктор, який автоматично приймає імена ключових слів, які відповідають стовпцям, які відображені;

- метод “is_admin” – метод, з допомогою якого можна визначити чи даний об’єкт класу є адміністратором чи ні, адже він повертає теперішнє значення атрибуту admin.

Інші класи цієї моделі створюються аналогічним чином, варто тільки відзначити зв’язок багато-до-багатьох таблиць Post і Social (див. Рисунок 3.2).

```
association_table = db.Table('association_table', db.Model.metadata,
    db.Column('Socials_id', db.Integer, db.ForeignKey('socials.id',ondelete="CASCADE"),
primary_key=True),
    db.Column('Post_id', db.Integer, db.ForeignKey('posts.id',ondelete="CASCADE"),
primary_key=True),
    db.UniqueConstraint('Socials_id', 'Post_id', name='UC_social_id_post_id')
)
```

Рисунок 3.2 – Код перехідної таблиці association_table

За допомогою цієї асоціативної таблиці з’єднуються зв’язком багато-до-багатьох таблиці Post і Social, атрибутами Post_id та Social_id відповідно.

3.2 Створення форм

Створення форм у проекті продемонструється на прикладі створення форми “AddTask”(див. Рисунок 3.3).

```
class AddTask(FlaskForm):
    title = StringField('Title', validators=[DataRequired()],
render_kw={"placeholder": "It is your content title"})
    content = TextAreaField('Content', validators=[DataRequired()],
render_kw={"placeholder": "This is your main content"})
    date_posted = DateField('Date Posted', format='%Y-%m-%d',validators=[Optional()],
render_kw={"placeholder": "Format date example: 2020-06-01"})
    time_posted = DateTimeField('Time Posted',format='%H:%M',validators=[Optional()],
render_kw={"placeholder": "Format Time example: 20:30"})
    image_file = FileField('Choose picture', validators=[FileAllowed(['jpg','png','mp4'])])
    image_file_url = StringField('Your picture now: ' )
    tags = TextAreaField('Tags',
render_kw={"placeholder": "Write here your tags. Format:  #something #test "})
    socials = SelectMultipleField('Socials', coerce=int,validators=[Optional()])
    notes = BooleanField('It is notes')
    submit = SubmitField('Add task')
```

Рисунок 3.3 – Код створення форми AddTask

Для створення форм використовується WTForms [23]. Це спрощує виконання завдання і побудови форми. Для того щоб створити форму за допомогою нього, необхідно створити відповідний клас, з потрібною назвою форми, цей клас містить атрибути, які і будуть елементами форми. В даному випадку є такі елементи:

- title – заголовок, рядкове поле для введення заголовка, з обов'язковою перевіркою на те чи введена інформація validators=[DataRequired()]. Окрім цього тут є значення яке висвітлюється, коли не введений ніякий текст. Описується воно за допомогою "render_kw={ "placeholder": "It is your content title"}". Тобто за замовчання, поки не буде введено жодної літери, писатиме фраза "It is your content title", яка означає "Це є заголовок твого контенту";
- content – поле для контенту, тобто основного змісту, всі параметри аналогічні до попереднього поля, тільки відрізняється тим що тут формат даних текстовий, а значить і формат поля є текстовим (TextAreaField) ;
- date_posted – відрізняється від попередніх полів тим що це є формат дати, а також тут вказаний певний формат дати яка вводиться;
- time_posted – відповідає формату згаданому вище, тільки поле типу DateTimeField, що дозволяє запам'ятовувати окрім дати ще і час. Потрібну нам годину і хвилину;
- image_file – поле для загрузки файлів, з введенням обмеження на потрібні нам формати файлів, а саме: "jpg", "png", "mp4";
- image_file_url – поле формату рядка, для виведення посилання на файловому сервері на збережену фотографію;
- tags – текстове поле для запису тегів;
- socials – особливе поле, яке створюється за допомогою SelectMultipleField тобто поле множинного вибору. Воно є необов'язково заданим. Це поле є для того щоб підключити до заданого завдання потрібні соціальні мережі, можна вибрати як одну так і декілька зразу;

- notes – булеве поле, яке має лише два значення істину і хибу. При значенні істини в даному випадку створиться чернетка, у випадку якщо значення дорівнюватиме хибі, то створиться завдання;
- submit – поле, яке відповідає за підтвердження і відправлення запиту на сервер.

Решту форм створюється аналогічно з невеликими змінами. Для того щоб не збільшувати розмір диплому, решту прикладів форм можна буде переглянути на github.

3.3 Реєстрація, авторизація та вихід

В розробленому сервісі передбачено такі функції як реєстрація, авторизація, та вихід з сервісу. Розробка цих елементів здійснювалась з допомогою Flask-login [24,25]. Для цього потрібно було створити відповідні форми (RegsitrationForm, LoginForm), а також маршрути і шаблони для всіх сторінок.

3.3.1 Реєстрація

Сторінка реєстрації (див. Рисунок 3.4) створювалась за допомогою форми RegistrationForm, шаблону “register.html”, а також маршруту “@app.route("/register", methods =['GET', 'POST'])”(див. Рисунок 3.5).

На рисунку можемо побачити 4 поля для введення тексту, у першому полі користувач вводить свій унікальний username, а бо ім’я користувача, на другому полі вводить свою email-адресу, а на третьому і четвертому пароль і повторення пароля відповідно.

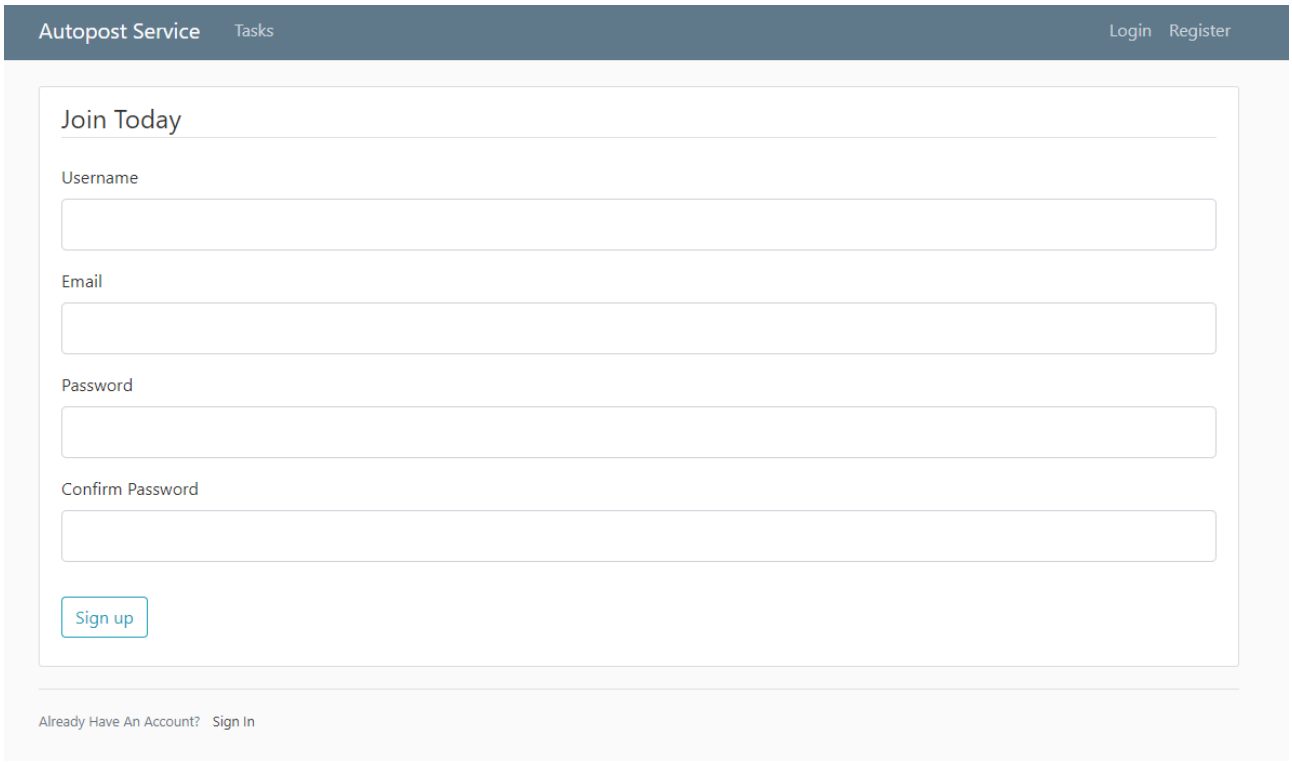

Рисунок 3.4 – Сторінка реєстрації у сервісі

Спочатку функції (див. Рисунок 3.5) перевіряється чи зараз користувач є авторизованим, якщо він авторизований, то йде переадресація на головну сторінку сама сторінка реєстрації відображена не буде. Якщо користувача немає, тоді створюється форма. Функція “validate_on_submit” відповідає за правильність даних у формі, тобто якщо всі дані введені правильно, тоді і лише тоді реєстрація пройде успішно. Далі генерується закриптований пароль, який береться з форми пароля. Після чого всі дані з форм, а також ось цей закриптований пароль зберігаються у відповідний полях об’єкта користувач (User). Після того як об’єкт створений, відбувається додавання його в базу даних, а також збереження транзакції за допомогою db.session.commit(). При успішному створенні облікового запису буде висвітленим повідомлення про те, що він створений і після чого переадресація на сторінку авторизації. Повертає ця функція створення шаблону реєстрації, передаючи туди заголовок і форму відповідно.

```python
@app.route("/register", methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form =RegistrationForm()
    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username = form.username.data, email = form.email.data,
                    password = hashed_password)

        db.session.add(user)
        db.session.commit()
        flash(f'Your account has been created. You are now able to log in', 'success')
        return redirect(url_for('login'))
    return render_template('register.html', title='Register',form=form)
```

Рисунок 3.5 – Код маршруту сторінки реєстрації у сервісі

3.3.2 Авторизація

Для авторизації (див. Рисунок 3.6) використовується такий самий набір інструментів як і для реєстрації. Сама сторінка авторизації складається з поля вводу email-адреси, поля для вводу пароля, кнопки підтвердження запиту “log in”, а також булевого значення “remember me”, яке створене для того щоб запам’ятати користувача у системі, щоб не доводилось заново заходити.

Рисунок 3.6 – Сторінка авторизації у сервісі

Щодо самого коду, то він схожий до коду реєстрації. Тільки тут порівнюється кожне задане значення з тим що вже існує в базі. Після вдалої

авторизації пройде переадресація на головну сторінку. У випадку якщо ж авторизація неуспішна, невірний email або пароль, то буде висвітлено повідомлення про, те що авторизація не пройшла і потрібно перевірити введені дані(див. Рисунок 3.7).

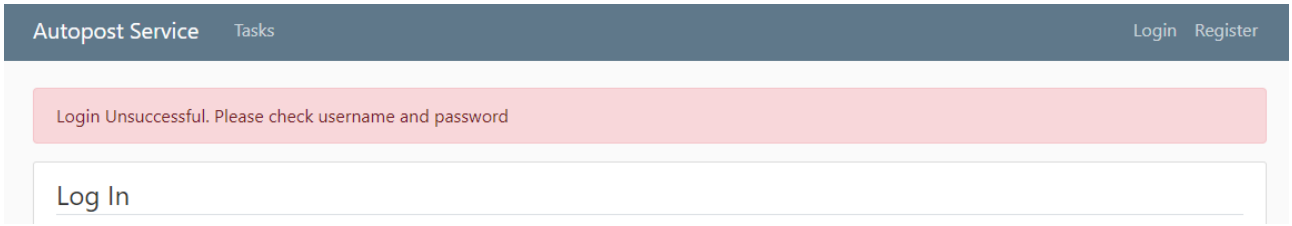

Рисунок 3.7 – Повідомлення про неуспішну авторизацію

3.3.3 Вихід

Для того щоб вийти з облікового запису необхідно натиснути відповідну кнопку справа “Log out”(див. Рисунок 3.8), після чого буде викликана функція виходу з облікового запису і пройде переадресація на основну сторінку.

Рисунок 3.8 – Меню сервісу.

3.4 Створення, редагування та видалення завдань та чернеток

Одним з основних завдань сервісу є можливість створювати, редагувати чи видаляти завдання і чернетки. Завдання – це майбутні публікації, які створені з вказуванням дати і часу публікації, їх можна згодом редагувати і прикріплювати до них облікові записи соціальних мереж. Чернетки – це неповністю заповнені завдання, тобто, там обов’язковими даними є тільки заголовок і контент, дата, час та інші параметри є необов’язковими. Для того щоб не збільшувати розмір дипломної роботи, буде розглянуто всі маніпуляції

над завданнями і чернетками разом, не розбиваючи їх на окремі підрозділи, оскільки вони самі по собі схожі.

3.4.1 Створення завдання або чернетки

Для того щоб перейти на вікно створення завдання необхідно натиснути на “Add Task”, на меню сервісу (див. Рисунок 3.8). Після чого буде відкрито форма для створення нового завдання, або чернетки (див. Рисунок 3.9).

На цій формі є наступні елементи:

- title – заголовок публікації;
- content – саме повідомлення публікації;
- date Posted – дата публікації, якщо дата відсутня по замовчуванню створиться чернетка, а не завдання;
- time Posted – час публікації, якщо час відсутній то по замовчуванню створиться чернетка, а не завдання;
- tags – список тегів, які необхідно використати у публікації;
- notes – булеве поле, для примусового створення саме чернетки;
- файлове поле – поле для вибору зображення необхідного для публікації;
- submit – кнопка підтвердження створення завдання або чернетки.

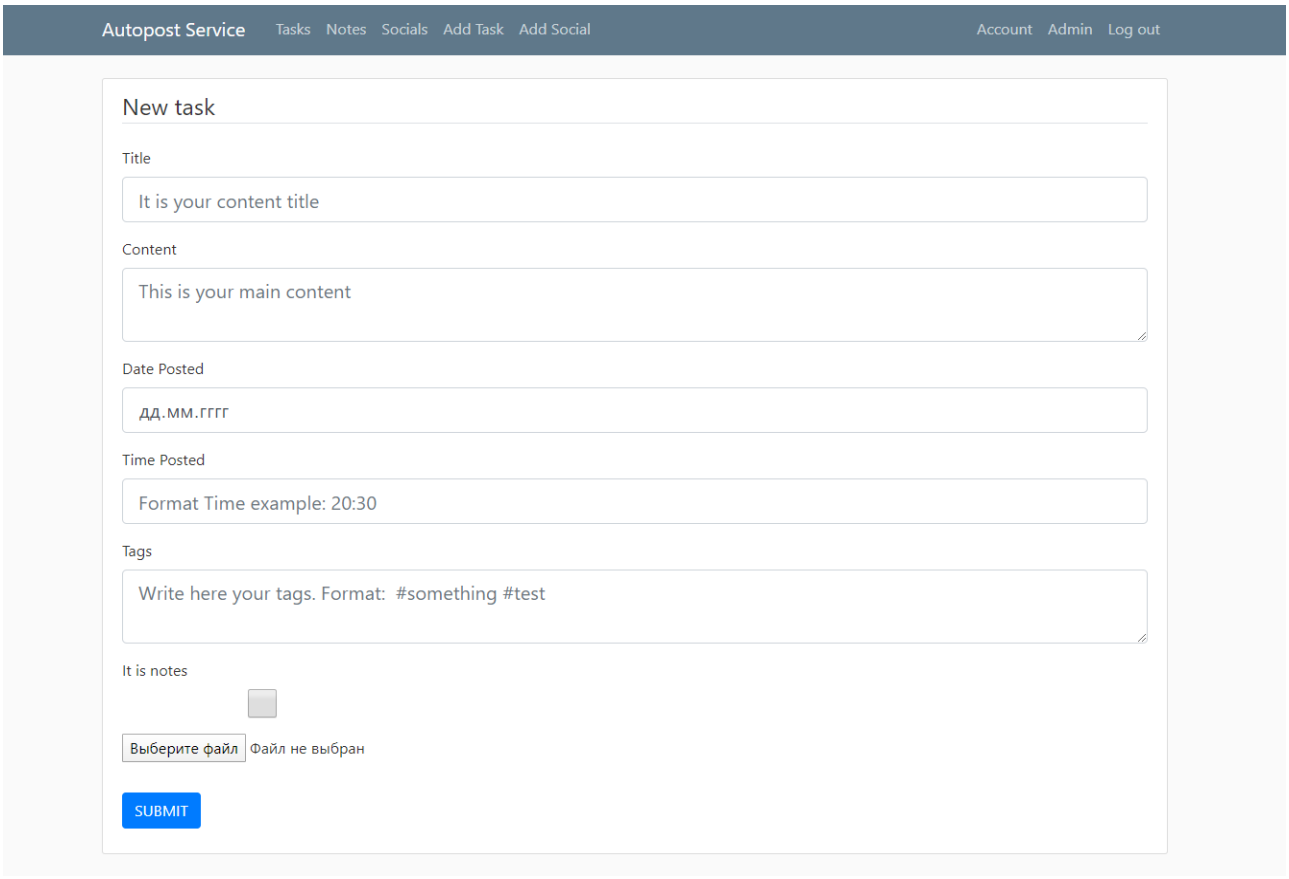

Рисунок 3.9 – Форма створення нового завдання або чернетки.

Необхідно також зауважити те, що для вибору дати використовується спеціальне поле, за допомогою якого, зручно вибрати ту чи іншу дату, не записуючи її вручну (див. Рисунок 3.10).

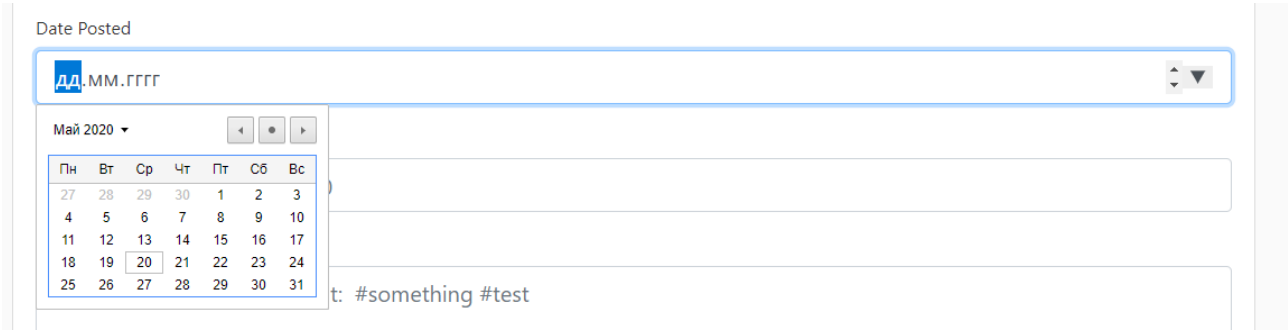

Рисунок 3.10 – Поле для вибору дати публікації.

Після того як всі потрібні дані введені і натиснута кнопка підтвердження відбудеться перевірка форми. Якщо все вірно введено, то створиться завдання або чернетка, у випадку, якщо необхідні поля не будуть заповненні (заголовок і

контент), то буде сповіщення про те, що дане поло обов’язково потрібно заповнити (див. Рисунок 3.11).

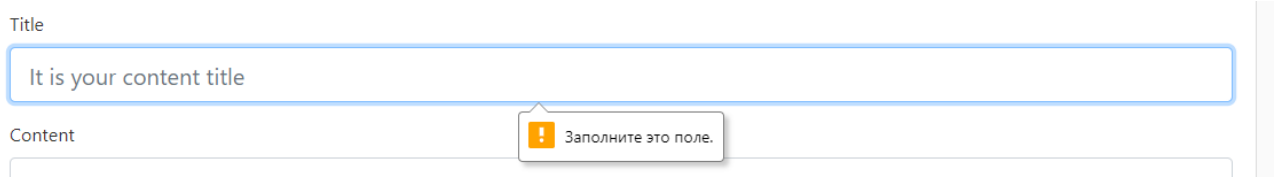

Рисунок 3.11 – Сповіщення про обов’язковість заповнення поля.

Робота самої функції створення завдання розпочинається з перевірки форми. Після того як вона успішно пройдена перевіряється наявність файлу у полі для добавлення файлу, якщо він присутній, то йде загрузка його на файловий сервер і в базу даних зберігається посилання на нього, якщо він відсутній то його значенню в базі даних присвоюється “None”. Далі беруться усі введені поля, і присвоюються до відповідний елементів класу Post. Тобто до елемента “post.title” присвоюється значення поля “Title” і т. д.

По замовчуванню в елемент дати створюється елемент “None”. Далі у випадку якщо введена дата, тоді підставляється потрібна дата з часом 00:00, якщо введений час, але не введена дата, тоді час буде потрібним а дата автоматично ставиться 2020/01/01. В обох перелічених випадках створиться саме чернетка, оскільки не вказані правильні дата або час. А у випадку запису часу і дати одночасно, то в базу даних записується повний формат дати з часом, також створиться саме завдання, а не чернетка.

3.4.2 Редагування завдання або чернетки

Редагування завдання виглядає подібним чином як і створення, але тут додається декілька нових полів(див. Рисунок 3.12). А саме поле для вибору декількох значень Socials, воно призначене для того, щоб вибрати одного або декількох облікових записів, на які здійснюватиметься публікація. А також поле для перегляду існуючого файлу, якщо такий є при створенні завдання, в

ньому вказане посилання на файловий сервер, де знаходиться відповідний завантажений файл. Про кнопки, а також маніпуляції буде у наступних пунктах розділу.

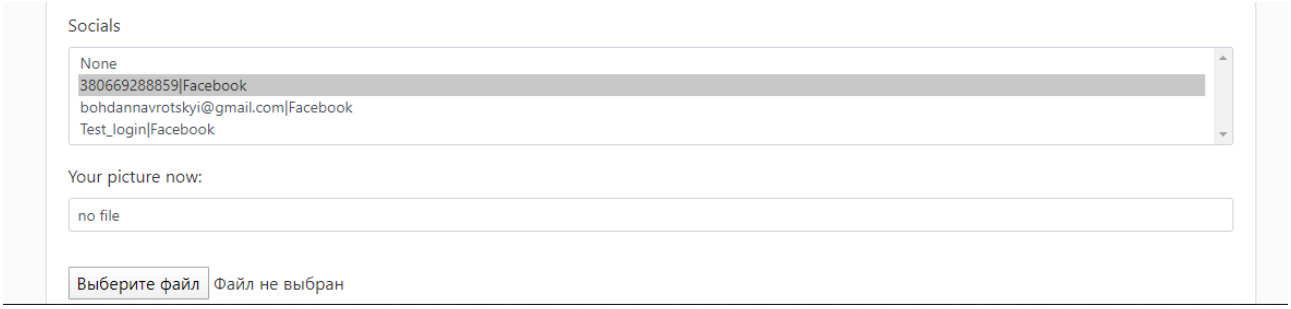

Рисунок 3.12 – Нові поля у редагуванні завдання.

У чернетці немає поля Socials, але присутня адреса на поточний файл на файловому сервері. Також є наступні кнопки (див. Рисунок 3.13):

- SUBMIT – для підтвердження змін;
- Add to task – для перетворення чернетки в завдання(можливе тільки якщо всі дані збережено і всі обов'язкові поля вказано (тобто заголовок, контент, дата і час) ;
- Delete – видалення чернетки;
- Exit – вихід назад до перегляду всіх чернеток.

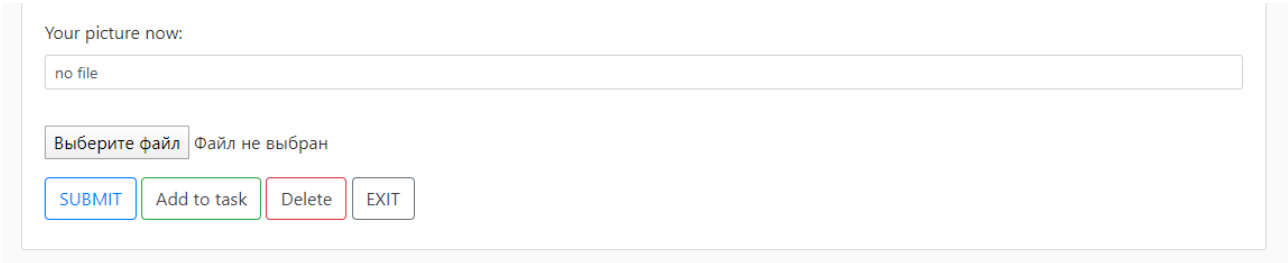

Рисунок 3.13 – Нові поля і кнопки у редагуванні чернетки.

3.4.3 Видалення завдання або чернетки

Видалення завдання або чернеток можна зробити двома способами, перший спосіб це видалення в режимі редагування, як було раніше показано з чернеткою, а другим способом є видалення напряму з перегляду всіх завдань, або чернеток. В даному випадку є можливість видалення завдання за допомогою відповідної кнопки “Delete” (див. Рисунок 3.14). У чернеток повністю аналогічно.

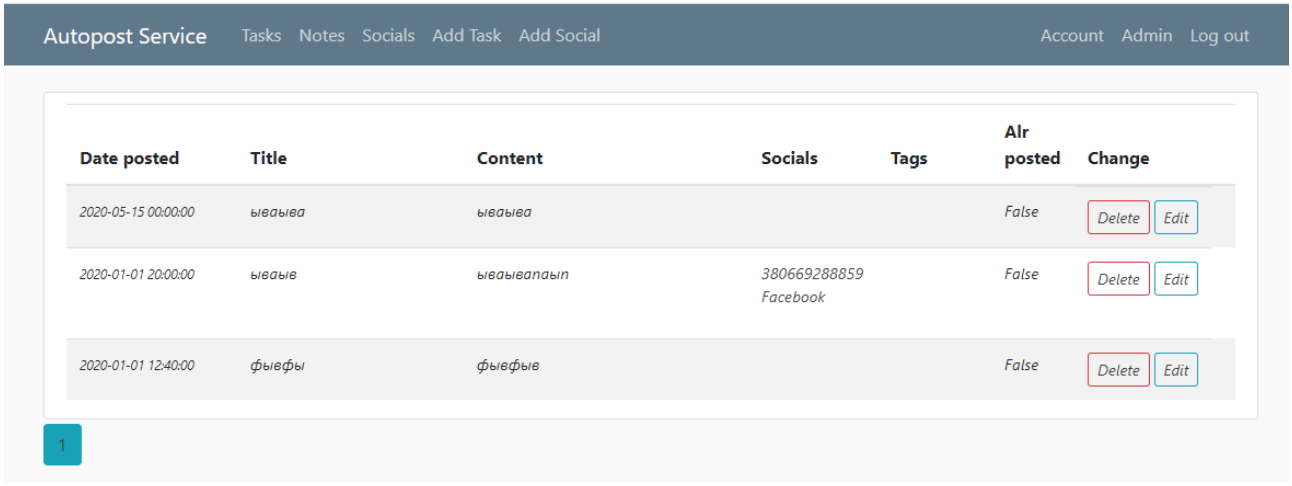

Рисунок 3.14 – Відображення всіх створених завдань.

3.4.4 Список всіх завдань і взаємодія з ними

Список всіх створених завдань(див. Рисунок 3.14) можна побачити на головній сторінці, або натискаючи на пункт меню “Tasks”. Тут відображено коротка інформація про кожен запис, а також можлива взаємодія з ними, а саме видалення, або редагування. Нажимаючи на кнопку видалення (Delete), дане завдання повністю видалиться з бази даних. А натискаючи на кнопку редагування (Edit), сайт переадресується на відповідну сторінку з редагуванням потрібного поста.

3.4.5 Список всіх чернеток і взаємодія з ними

Щодо чернеток, то тут аналогічно до попереднього пункту, тільки для доступу до них необхідно натиснути на пункт мені “Notes”. Тоді буде можливість переглянути весь список існуючих чернеток і проводити потрібну взаємодію з ними. А саме видаляти або редагувати, де в редагуванні як описувалось раніше, зможе дану чернетку відправити до вже готових, сформованих завдань.

3.5 Додавання, редагування та видалення облікових записів соціальних мереж

Основною функцією сервісу є взаємодія з соціальними мережами, а саме публікація туди потрібної інформації. Для цього і необхідне додавання, редагування, та видалення облікових записів соціальних мереж. Адже окрім потрібного матеріалу, який було створено раніше, потрібно ще також додати облікові записи на які цей матеріал публікувати. В подальшому також планується запровадження функції публікування в групи, власниками яких є якраз таки дані облікові записи.

3.5.1 Додавання облікового запису

Для додавання нового облікового запису соціальної мережі, необхідно натиснути у меню (див. Рисунок 3.8) на кнопку “Add Social”. Після чого буде переадресація на відповідну форму з додаванням облікового запису соціальної мережі (див. Рисунок 3.15). У цій форма є всього лише три поля. Першим з них є поле для введення логіну з соціальної мережі, другим є поле для вводу паролю звідти, а третім є поле для вибору типу потрібної соціальної мережі.

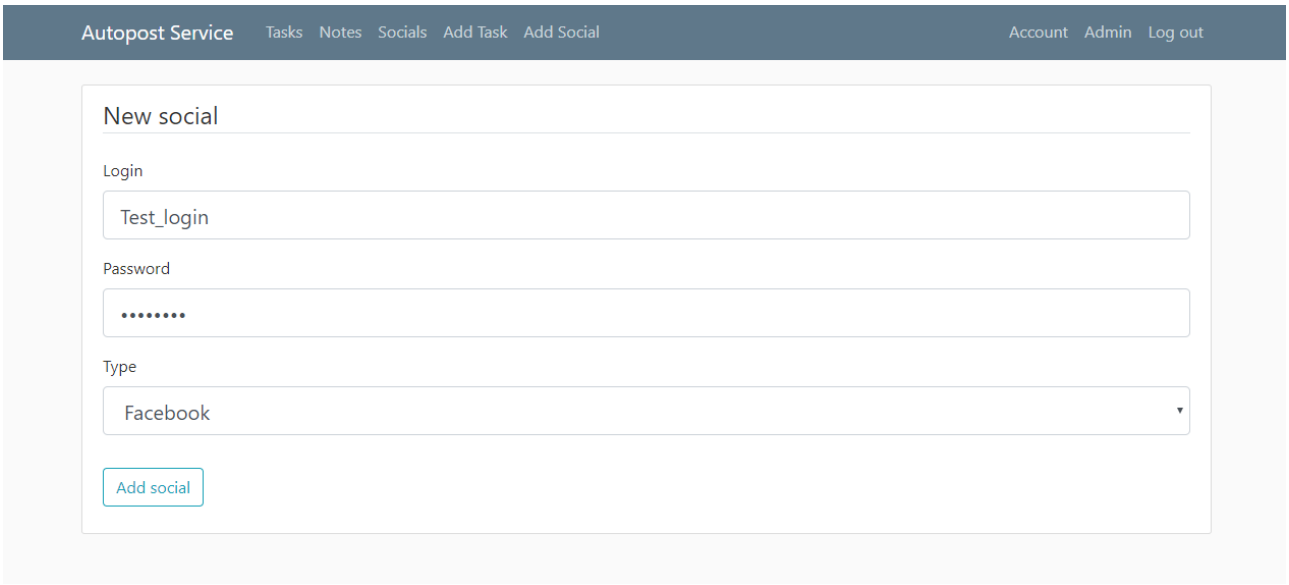

Рисунок 3.15 – Додавання нового облікового запису потрібної соціальної мережі.

Після того як всі поля заповнені правильно, при натисканні на кнопку “Add social” відбудеться створення нового об’єкту класу Social, в який будуть записані всі дані з форм. В базу даних запишуться та збережуться потрібні дані і висвітлиться повідомлення з успішно доданим обліковим записом (див. Рисунок 3.16). У випадку якщо не всі дані будуть заповнені, буде помилка в обробці форми, тоді потрібно буде відредагувати форму.

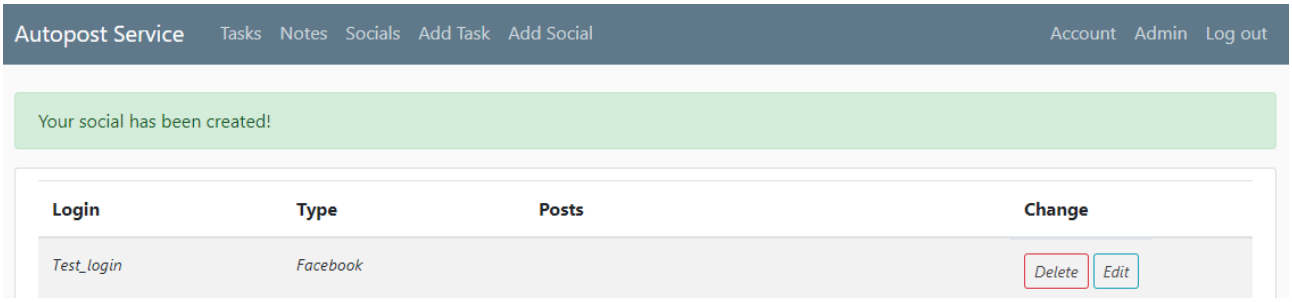

Рисунок 3.16 – Повідомлення про успішно доданий обліковий запис

3.5.2 Список всіх облікових записів

Для перегляду списку з всіма обліковими записами (див. Рисунок 3.17), які вже створені, необхідно перейти в меню до пункту “Socials”.

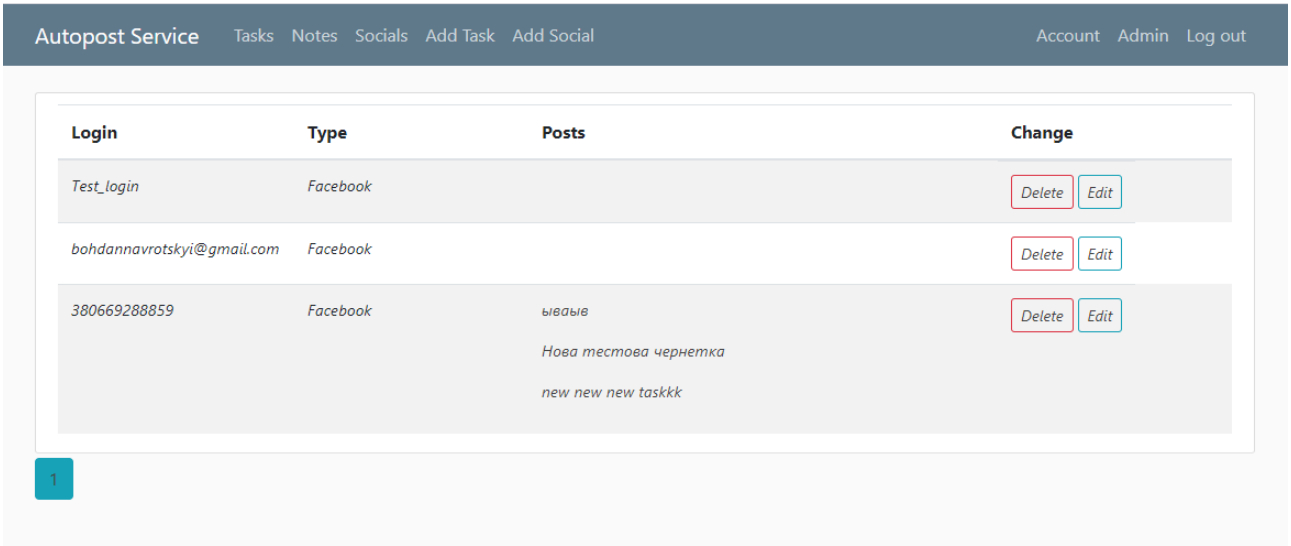

Рисунок 3.17 – Список усіх доданих облікових записів

3.5.3 Редагування даних облікового запису

Для редагування даних облікового запису необхідно зайти на сторінку редагування. Для цього необхідно у списку всіх записів (див. Рисунок 3.17), натиснути відповідну кнопку “Edit”, після чого пройде переадресація на сторінку редагування облікового запису (див. Рисунок 3.18).

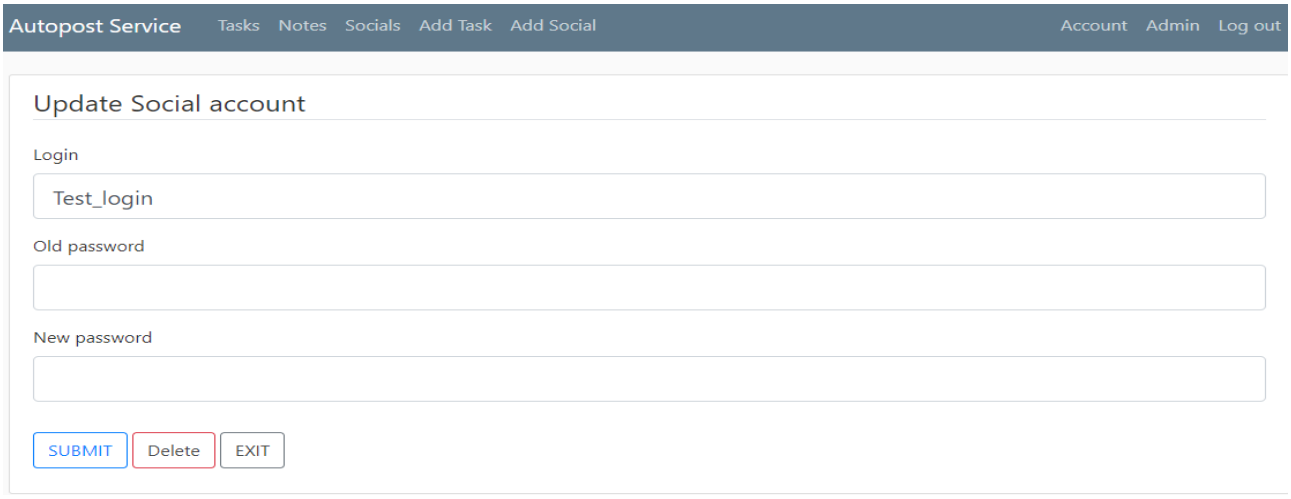

Рисунок 3.18 – Форма редагування доданих облікових записів

3.5.4 Видалення даних про обліковий запис

Для видалення даних, необхідно так само використати форму редагування даних (див. Рисунок 3.18). Або у списку всіх облікових записів (див. Рисунок 3.17). натиснути відповідну кнопку “Delete”. Після цього пройде запит до бази даних, що цей обліковий запис соціальної мережі (з відповідний ідентифікатором) потрібно видалити з бази даних. Після видалення збережуться зміни в базі і пройде переадресація на сторінку зі списком всіх добавлених облікових записів з повідомленням про успішне видалення облікового запису соціальної мережі (див. Рисунок 3.19).

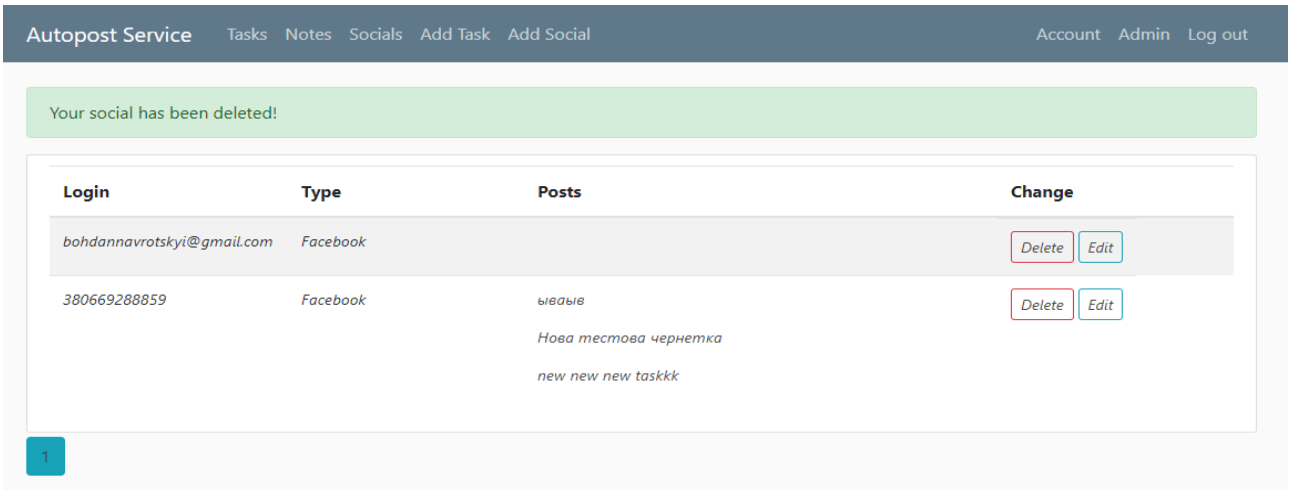

Рисунок 3.19 – Список облікових записів після видалення та сповіщення про успішне видалення

3.6 Маніпуляція з створеними завданнями

Під маніпуляцією з створеними завданнями мається на увазі весь можливий функціонал з вже створеним і відредагованим завданням. Тобто коли завдання створене у нього є такий функціонал можливостей (див. Рисунок 3.20)

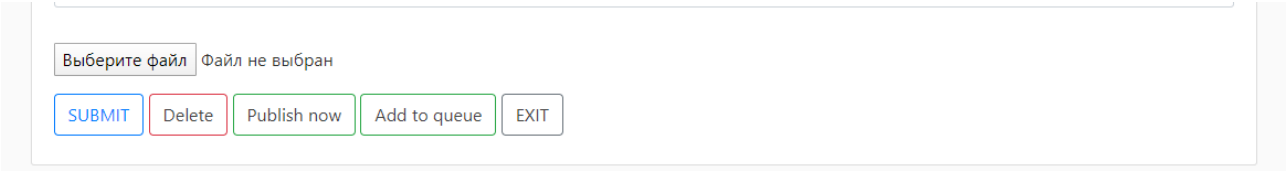

Рисунок 3.20 – Функціонал можливих маніпуляцій з створеним, але не опублікованим завданням

Доступні п’ять кнопок маніпуляції з створеним неопублікованим завданням:

- “SUBMIT” – кнопка для підтвердження змін;
- “Delete” – видалення даного завдання з бази даних;
- “Publish now – кнопка для виклику функції опублікування зараз, тобто миттєвого публікування;
- “Add to queue” – додавання в чергу, тобто завдання додасться в чергу і за заданим часом, виконається його публікація;
- “Exit” – вихід до перегляду всіх завдань.

У випадку коли завдання створене і вже опубліковане, тоді функціонал змінюється і стає таким (див. рисунок 3.21).

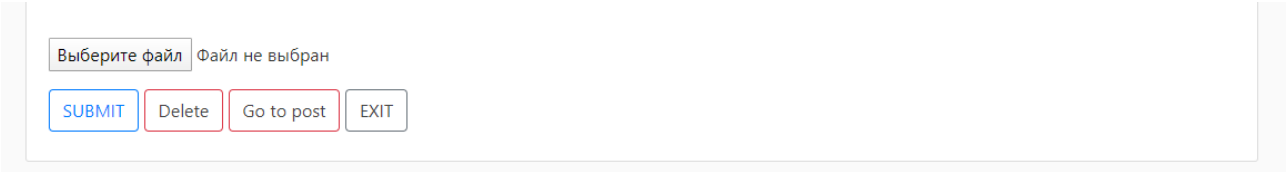

Рисунок 3.21 – Функціонал можливих маніпуляцій з створеним та опублікованим завданням

В цьому випадку видаляється дві функції публікації, проте додається можливість перейти на перегляд самого завдання, тобто переглянути всі пости у соціальних мережах які були створені за допомогою цього завдання (див. Рисунок 3.22). Тут відображається деяка інформація про публікація, але

найголовнішим є посилання на саму публікацію. З цієї сторінки можна прямо перейти на готову публікацію, або видалити її, що буде описано пізніше.

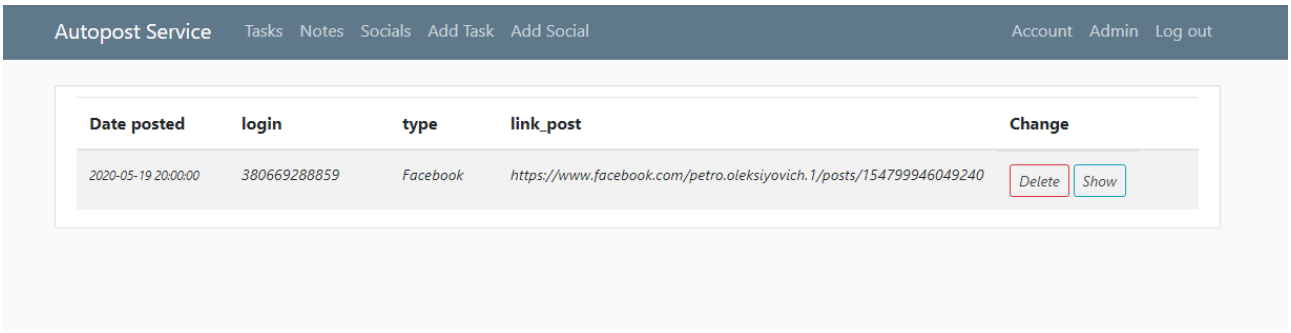

Рисунок 3.22 – Сторінка відображення опублікованого завдання

3.6.1 Миттєве публікування

Дана функція передбачає в собі можливість миттєвого публікування потрібного завдання. Вона викликається за допомогою натискання кнопки “Publish now” (див. Рисунок 3.20). При виклику дістаються дані з бази даних про потрібний пост, потрібний список облікових записів соціальної мережі, а також додається в чергу миттєвого виконання за допомогою даної команди (див. Рисунок 3.23). Ця команда дозволяє за допомогою допоміжних сервісів RQ (бібліотека для створення черги) та REDIS (розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання за бажанням користувача) додати на виконання на сервері будь-які функції. В нашому випадку, це одна з функцій бота, а саме створення поста [26,27].

```
job = queue.enqueue(facebook_create_post,soc.login,soc.password,test_publish,test )
```

Рисунок 3.23 – Команда для миттєвої публікації

3.6.2 Додавання завдання до черги для відкладеної публікації

Наступною можливістю є можливість відкладеної публікації, в якій використовуються ті ж технології, але в цьому випадку потрібно натискати кнопку “Add to queue”(див. Рисунок 3.20). Тоді буде викликано іншу функцію, яка подібна за функціонал, тільки цього разу до аргументів функції додається дата і час потрібної публікації(див. Рисунок 3.24). В даному прикладі також вказано, що вона запишеться не в звичайну чергу, а саме в чергу “ScheduledJobRegistry”, а це означає, що вона не перетинатиметься з нею.

```
job = queue.enqueue_at(datetime(int(year), int(month), int(day), hour, int(minute)),
                       facebook_create_post,soc.login,soc.password,test_publish,test)
registry = ScheduledJobRegistry(queue=queue)
print(job in registry)
```

Рисунок 3.24 – Команда для відкладеної публікації

Обидві функції публікації, а також функцію видалення потрібно виконувати саме на стороні серверу, для того щоб, поки бот виконує свою роботу, користувачу не доводилось чекати весь необхідний час. Особливо у випадку якщо завантажене зображення велике, або великий наплив людей і сервіс може довго завантажуватись, тому для цього і створена черга, яка просто виконуватиме всі завдання на стороні сервера поступово.

3.6.3 Видалення публікації

Видалення публікації робиться у за допомогою переходу на сторінку відображення опублікованого завдання (див. Рисунок 3.22). Саме там є кнопка, яка відповідає за видалення опублікованого запису з соціальної мережі. Нажимаючи на неї відбувається виклик функції додавання в чергу, у якій викликається функція з бота, про видалення поста з соціальної мережі і передаються потрібні параметри (див. Рисунок 3.25).

```
job = queue.enqueue(facebook_delete_post,social_log,social_pas,need_url)
```

Рисунок 3.25 – Команда для видалення публікації

3.7 Адміністративна частина

Адміністративна частина проекту створювалась за допомогою Flask-admin, а також відповідних форм і маршрутів [28]. У ній є доступні такі функції як Create, Read, Update, Delete. Тобто весь функціонал CRUD [28]. Головна її сторінка має простий дизайн і такий вигляд (див. Рисунок 3.26).

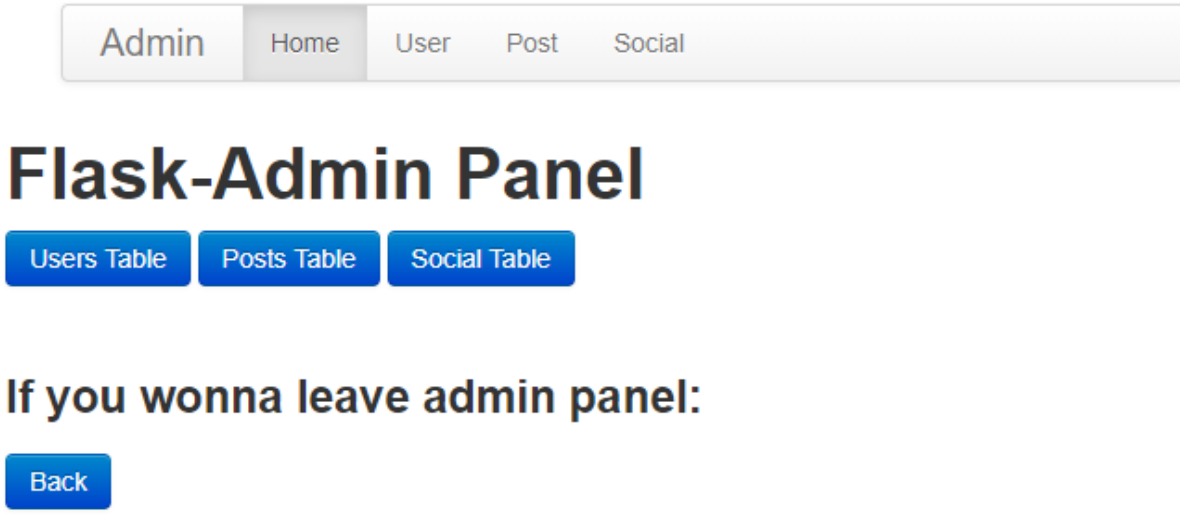

Рисунок 3.26 – Вигляд головної сторінки адміністративної частини

В неї є такі функції як повернення до самого сайту, а також перегляд трьох головних таблиць, а саме “User”, “Post” і “Social”. В кожній з яких є можливість створювати, редагувати чи видаляти всі екземпляри даних класів, або їх атрибути.

Першою з адміністративних сторінок які буде розглянуто є сторінка Post (див. Рисунок 3.27). Вона створена за допомогою Flask-admin, а також форми “PostAdminView”. На ній можна переглядати існуючі записи, тобто завдання. Окрім цього можна створювати нові, за допомогою кнопки “Create”. Крім двох

переліченних функцій також є можливість редагувати, або видаляти об’єкти. Для редагування потрібно нажати на олівець, який відкриє поле для коригування потрібного елемента таблиці. Для видалення елементу з таблиці і бази даних зовсім, потрібно натиснути на кошик. Оскільки в користувача є адміністративні права, то він може видаляти або редагувати всі записи, незалежно від того чиїми вони є.

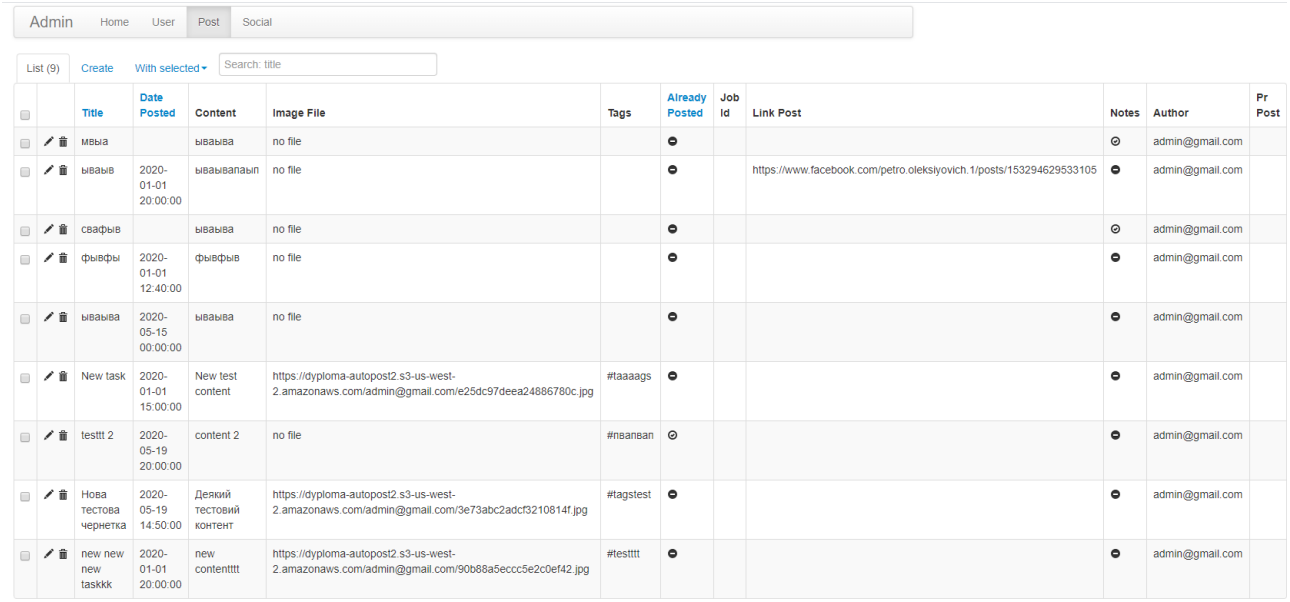

Рисунок 3.27 – Вигляд Post сторінки адміністративної частини

Наступним буде розглянуто таблицю Social (див. Рисунок 3.28), яка створювалась аналогічно до передньої. Для її створення використано Flask-admin та форма “SocialAdminView”. Всі доступні функції, які були в таблиці Post, є і у даній таблиці. А саме:

- можна переглянути основну інформацію про вибраний обліковий запис;
- видалити або редагувати потрібний обліковий запис;
- створити новий обліковий запис і прив’язати його до користувача.

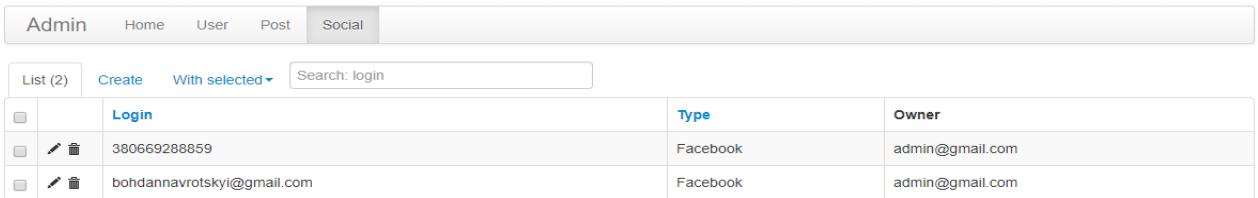

Рисунок 3.28 – Вигляд Social сторінки адміністративної частини

Останньою таблицею у адміністративній частині буде розглянуто таблицю User (див. Рисунок 3.29). Її створено аналогічно до двох попередніх, використовуючи Flask-admin та форму “PostAdminView”. Весь функціонал ідентичний, тобто створення, читання, редагування і видалення.

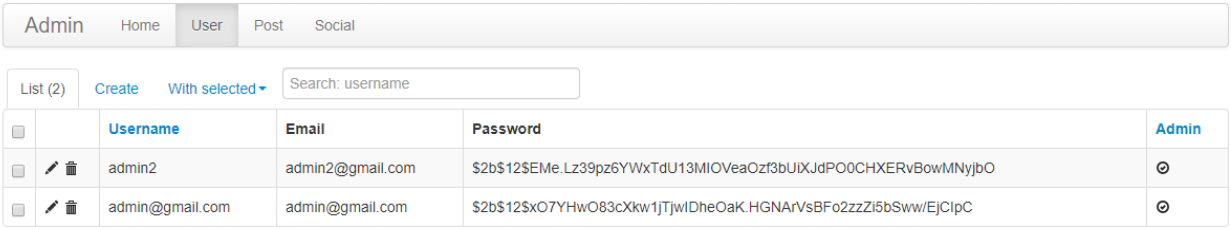

Рисунок 3.29 – Вигляд User сторінки адміністративної частини

3.8 Налаштування проекту і публікування його на Heroku

Для успішної публікації сервісу на Heroku необхідно провести процедуру налаштування і підключення його до допоміжних сервісів, зокрема до файлового сервера AMAZON S3 [29]. Це сервіс який дозволяє зберігати різні дані, зокрема для цього проекту буде зображення, які надсилає користувач, а також підключення деяких пакетів, для успішної і безперебійної роботи сервісу.

3.8.1 Налаштування проекту(settings) та змінні середовища

Для використання проекту на сервері використовується допоміжний файл “settings.py” (див. Рисунок 3.30) у який записується всі головні змінні середовища. Змінні середовища – це набір важливих значень, які впливають на поведінку всього проекту в цілому. Тобто тут є перелік важливих змінних, без яких проект не буде існувати, зокрема це:

- SQLALCHEMY_DATABASE_URI – доступ до бази даних, який отримується за допомогою змінної “ DATABASE_URL”;
- SECRET_KEY – секретний ключ;
- S3_BUCKET – назва кошика у сервісі S3;
- S3_KEY – ключ у сервісі S3;
- S3_SECRET – секретний ключ від сервісу S3;
- REDISTOGO_URL – адреса допоміжного сервісу REDIS;
- CHROMEDRIVER_PATH – шлях до браузера;
- GOOGLE_CHROME_BIN – шлях до запуску браузера на сервері.

```
SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL')
SECRET_KEY = os.environ.get('SECRET_KEY')
SQLALCHEMY_TRACK_MODIFICATIONS = False
S3_BUCKET = os.environ.get('S3_BUCKET')
S3_KEY = os.environ.get('AWS_ACCESS_KEY_ID')
S3_SECRET = os.environ.get('AWS_SECRET_ACCESS_KEY')
REDISTOGO_URL=os.environ.get('REDISTOGO_URL')
CHROMEDRIVER_PATH = os.environ.get('CHROMEDRIVER_PATH')
GOOGLE_CHROME_BIN = os.environ.get('GOOGLE_CHROME_BIN')
```

Рисунок 3.30 – Код settings.py

Самі ж змінні середовища при локальному використанні записуються у файл “.env”. При серверному використанні, як на Heroku, розміщуються в вкладці “Config Vars” (див. Рисунок 3.31) і звідси підключаються до проекту, через файл описаний вище.

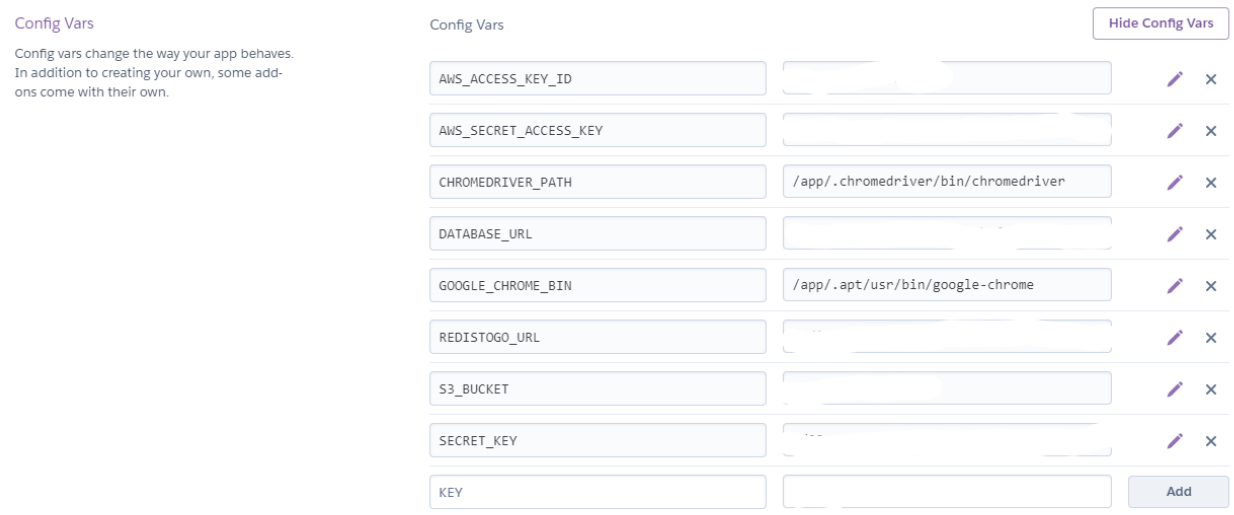

Рисунок 3.31 – Змінні середовища на сервісі Heroku

3.8.2 Ресурси та підключення файлового сервера

Для підключення файлового сервера необхідно використати файл “resources.py”, у який підключається сам сервіс S3 (див. Рисунок 3.32).

```
import boto3
from autopost.settings import S3_BUCKET, S3_KEY, S3_SECRET
```

Рисунок 3.32 – Код підключення сервісу S3

У ньому з налаштувань імпортується змінні середовища і потім з ними проходить наступна взаємодія у вигляді двох головних функцій “_get_s3_resource” (див. Рисунок 3.33) та “get_bucket” (див. Рисунок 3.34)

```
def _get_s3_resource():
    if S3_KEY and S3_SECRET:
        return boto3.resource(
            's3',
            aws_access_key_id=S3_KEY,
            aws_secret_access_key=S3_SECRET
        )
    else:
        return boto3.resource('s3')
```

Рисунок 3.33 – Код функції отримання ресурсу S3

Функція “_get_s3_resource” надає доступ до ресурсу сервісу S3, по заданим ключу і секретному коду.

```
def get_bucket():
    s3_resource = _get_s3_resource()
    if 'bucket' in session:
        bucket = session['bucket']
    else:
        bucket = S3_BUCKET
    return s3_resource.Bucket(bucket)
```

Рисунок 3.34 – Код функції отримання доступу до корзини з сервісу S3

Функція “get_bucket” надає доступ до корзини з сервісу S3, по заданому ресурсу, який було підключено раніше.

3.8.3 Публікація на Heroku

Для розгортання сервісу на вебсервер, використовується допоміжній сервіс Heroku, який є потужним інструментом, що може допомогти у легкому і зручному розгортанні [30,31]. Щоб розгорнути додаток потрібно створити файл “requirements.txt” у який записати всі залежності і модулі які використовуються в проекті, а також підключити базу даних на Postgre, для того щоб вона синхронізовувалась на сервері а не локально [32,33]. Після чого потрібно створити допоміжні файли “resources.py” та “settings.py” , про які йшлось в попередніх розділах. Потім на сервісі Heroku створити новий проект. Далі потрібно записати всі змінні середовища і підключити потрібні “Bulidpacks”. Коли всі ці речі виконано, підключити репозиторій з github до свого проекту і після цього завантажити з локального репозиторію проект до глобального на github і у разі якщо немає помилок, даний сервіс з автопостингу буде розміщено на вебсервері Heroku (див. Рисунок 3.35).

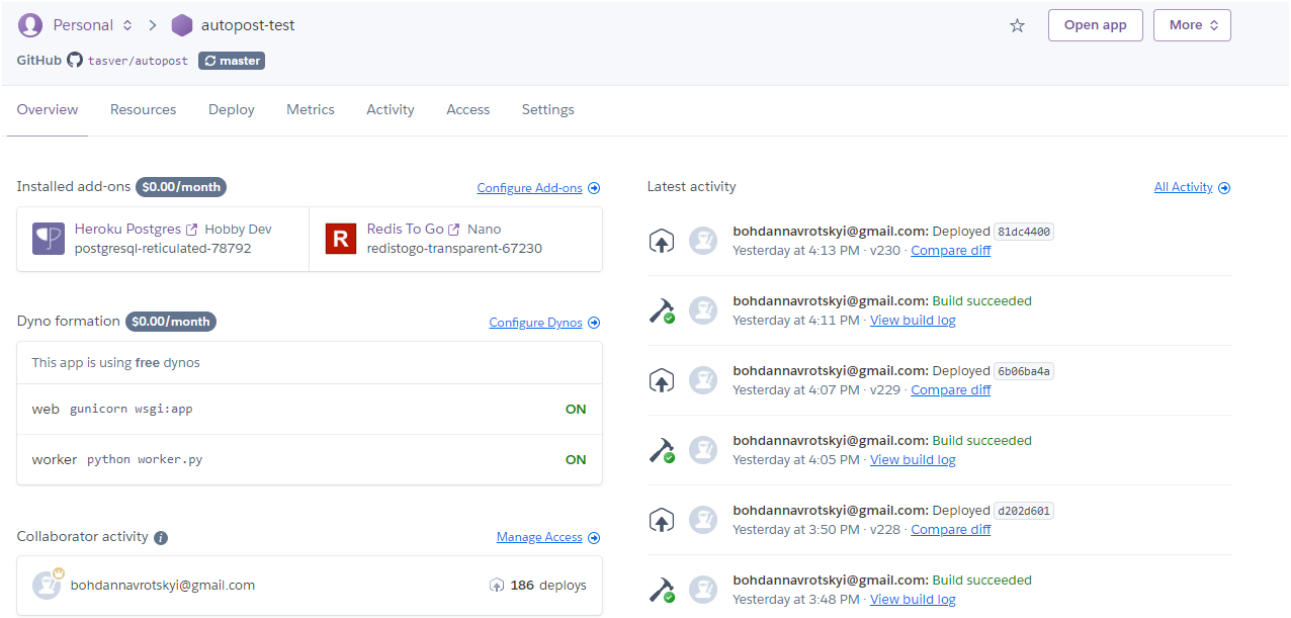

Рисунок 3.35 – Розгорнутий вебсервіс з автопостингу на вебсервері Heroku

4 БІЗНЕС ПЛАН РОЗРОБКИ АВТОПОСТИНГУ

4.1 Резюме

Розроблений вебсайт буде частиною сервісу автопостингу, який надаватиме пакет послуг у сфері соціального маркетингу, зокрема в автоматизації публікації постів у соціальних мережах.

Цільовою аудиторією є різні компанії, навчальні заклади, блогери і багато інших різних сфер, де використовуються соціальні мережі для комунікації чи реклами у соціальних мережах. Також цільовою аудиторією є адміністратори і SMM-спеціалісти (спеціаліст з роботи в соціальних мережах), які зацікавленні в швидкому і зручному веденні потрібних їм сторінок в мережі.

Для того щоб реалізувати даний проект потрібна сума від 30 000 до 40 000 грн.

Фінансування здійснюватиметься з вкладень інвесторів, а також особистого прибутку проекту.

Для реалізації необхідно залучити команду розробників, а саме дизайнера, back-end програміста, front-end програміста, а також розробник бота або скрипта, для взаємодії з соціальними мережами.

Очікуваний місячний дохід залежить від кількості клієнтів, але орієнтовно близько 10 000 грн.

Чистий прибуток за рік очікується у розмірі 80 000 - 90 000 грн.

4.2 Маркетинг

Вид послуги

Розроблений сервіс пропонуватиме користувачам послуги автоматичного публікування постів у різні соціальні мережі. Окрім цього буде доступні

декілька важливих можливостей, а саме відкладена публікація, тобто публікування матеріалу у потрібний вибраний час, а також редагування і видалення його з одного місця на сайті або смартфоні у мобільній версії даного сайту.

Сервіс має задовольнити потребу у швидкій, зручній і дешевій публікації одного і того ж матеріалу у різні соціальні мережі за використання мінімального часу та коштів.

Проект призначений для соціальної сфери та частково сфери соціального маркетингу.

Унікальністю даного проекту буде безплатність для університету, у подальшому повністю україномовність, а також непоганий функціонал для виконання поставлених завдань.

Недоліком на ранніх стадіях буде порівняно невеликий функціонал з дорожчими і серйознішими конкурентами. Але попри це функціонал згодом буде розширюватись.

Джерела вивчення ринку

Для вивчення ринку даних сервісів, необхідно використати мережу Інтернет, а також соціальні мережі. Окрім цього використовується декілька сервісів з відгуками, аби визначити які сервіси мають хороші сторони і що з цього найважливіше почерпнути для використання і імплементації у власному проекті.

Оцінка очікуваного попиту

За оцінками попит на даний сервіс є дуже великим, оскільки на його функціонал можуть претендувати як великі так і малі компанії, а також будь-які магазини, сервіси для надання послуг, блогери, власники чи адміністратори будь-яких груп чи сторінок у соціальних мережах. Потенційними клієнтами є особи, яким зручно за невеликі кошти використовувати сервіси для автоматизації публікації, не витрачаючи на це свій дорогоцінний час, або просто люди, які можуть створювати контент наперед.

Мінімальна ціна за подібні або ідентичні послуги складає від 100 грн за певну кількість публікацій або підписка на сервіс на місяць. Середньою ціною є близько 200 грн, що є в декілька разів більше ніж пропонуватиме наш сервіс.

Сервіси з аналогічним функціоналом зараз набувають великої популярності, адже в геометричній прогресії збільшується вплив цифрових технологій на повсякденне життя. Оскільки в сучасному суспільстві тільки розвивається все що пов'язано з інтернет маркетингом, блогерством, то попит на ці послуги тільки ростиме.

Конкуренти

Дана сфера послуг насичена великою кількістю подібних сервісів. Але переважна більшість з них розрахована на російськомовну або англійськомовну аудиторію, тому на них немає сильного попиту саме в Україні. А ще вагомим фактором є ціни, які зазвичай виступають в доларах і через валютну різницю вони надзвичайно дорогі для звичайного користувача в Україні. Тому конкурувати потрібно буде лише з деякою частиною з них. У конкурентних сервісах ціна на подібні послуги становитиме близько 200 грн за підписку на місяць, або певну кількість публікацій.

Для просування і розповсюдження свого сервісу конкурентні сервіси використовують в першу чергу мережу Інтернет, а також різні сервіси для накручування позитивних відгуків, цільову і загалом комплексну рекламу, яка націлене саме на потенційних клієнтів, блогерів чи малих підприємців.

Даний сервіс матиме декілька переваг перед іншими сервісами такої чи навіть дешевшої цінової категорії, а саме це дешевизна, зручний і зрозумілий функціонал, а також у подальшому наявність української мови.

Сценарії розвитку

Загалом існує багато можливих сценаріїв розвитку і існування проекту. Існує декілька з можливих варіантів. А саме два з них, оптимальний та негативний.

При оптимальному сценарії розвиток і існування цього проекту приблизно відбуватиметься так:

- спочатку через невідомість для аудиторії буле дуже мало клієнтів і мало хто користуватиметься сервісом;
- планується замовити рекламу в невеликих, але доволі впливових і популярних у вузьких кругах блогерів;
- окрім реклами в блогерів планується введення спеціально промо-акцій, за яку за приведення клієнтів буде даватись бонус, а також всім новим клієнтам буде нараховуватись деякий час безкоштовного користування сервісом і скидка на подальші послуги;
- після проведення даних кроків очікується великий приріст аудиторії, яку найважливіше буде втримати, адже далі піде вже так зване «сарафанне радіо», тобто люди самі будуть розповсюджувати інформацію між собою, про те що даний сервіс є хорошим і надає якісні послуги з автопостингу і кроспостингу.

Щодо негативного сценарії розвитку, то потенційно за негативним сценарієм очікується наступний хід подій:

- спочатку все що мало людей буде користуватись сервісом;
- якщо після того як купимо рекламу в блогерів і після проведення всіх промо-акцій очікуваного результату не буде, то доведеться шукати інших потенційних клієнтів;
- складатиметься список потенційних зацікавлених компаній чи підприємств, а також складатиметься список груп, каналів, сторінок, які потенційно могли б бути зацікавленими в даних послугах;
- після чого особисте спілкування і продемонстрування всіх можливостей сервісу і звісно надання пробного безкоштовного періоду;
- якщо даний сервіс не влаштовуватиме вже пряму аудиторію, то отримавши відповідні вказівки він буде змінений і доопрацьований під потрібні цілі і готовий до подальшої експлуатації;

- після чого вже якраз таки очікується покращення ситуації і знову ж таки потенційно запуститься «сарафанне радіо», яке дозволить просунути проект без великих затрат на рекламу.

Встановлення цін

Діапазон цін буде встановлений помітно меншим, ніж у аналогів з подібним функціоналом. У сервісів, які мають подібний функціонал, ціни будуть значно дорожчі, а у тих сервісів, які співпадають по цінах буде менший спектр можливостей. Також передбачено проведення багатьох промо-акцій, таких як безкоштовне використання сервісу новим користувачам, або певні скидки.

Сервіс повинен на ранніх етапах майже повністю окупити вкладені в себе гроші і згодом велика частина обороту вже приноситиме саме чистий дохід, а ще частину з доходу вкладатиметься назад в рекламу сервісу і його обслуговування.

Ціна за продукт відрізнятиметься від виду користування чи це за підписку чи за кількість постів. За підписку на місяць становитиме 100 грн за 4 облікових записи без акцій і скидок. Звісно по мірі збільшення кількості підключених соціальних мереж облікових записів ціна зростатиме, але все одно залишатиметься меншою ніж в прямих конкурентів.

План збуту та комплексу просування послуги

План збуту неодноразово оголошувався вище, а саме залучення до тестового періоду великої кількості груп і сторінок у соціальних мережах. А також по рекламі в різних блогерів, після чого повинне початись «сарафанне радіо» і сервіс розвиватиметься, або буде доопрацювання певних моментів і тоді також подальша його експлуатація.

4.3 Обґрунтування фінансових вкладів

Опис виробничих потужностей

Сервіс буде розміщений в мережі Інтернет на платному хостингу. На початкових стадіях адмініструвати і шукати клієнтів можна самим розробникам, але в подальшому буде найнято 2 спеціаліста адміністратора, а також менеджер який займатиметься питаннями пошуку нових клієнтів і залучення нової аудиторії.

Також розглядається можливість в майбутньому співпраця з якимись компаніям, або сервісами з промокодами чи будь-якими іншими сервісами, які підходять по тематиці, для взаємного розміщення реклами і залучення додаткової аудиторії. Сервіс по автопостингу працюватиме цілодобово, адже він розміщений на віддаленому сервері, який працює завжди.

Таблиця 4.1 – Витрати на сервіс атопостингу за розробку і перший місяць

Витрати на розробку сервісу автопостингу	20000 грн.
Витрати на хостинг	500 грн. в місяць
Найм системного адміністратора(на початку проекту тільки одного)	5000 грн в місяць
Разом:	25500 грн.

Підбір персоналу та оплата праці.

На початковій стадії розвитку найманих працівників не буде, з першими постійними клієнтами доведеться найняти одного постійного системного адміністратора, який слідкуватиме за тим, щоб не порушувались правила публікування і щоб все працювало коректно. Після того як буде достатньо багато клієнтів буде найнятий другий системний адміністратор, а також менеджер, який слідкуватиме і займатиметься рекламою і всім зв’язаним з просуванням.

Підбір персоналу спершу акцентуватиметься увага на активних студентах, оскільки це зменшить витрати на обслуговування. Якщо даний метод провалиться, то набиратиметься за звичайною співбесідою і тестом володіння навичок адміністрування і менеджменту відповідно. Про співбесіду претенденти взнаватимуть з оголошень на відповідних сайтах по пошуку роботи, групах, а також навчальних закладах, де можна облаштувати такі оголошення.

Оплата праці відбуватиметься двічі на місяць, при успішному і безпомилковому адмініструванні буде виплачуватись певний бонус до 15%. При гіршому ж розкладі, коли багато клієнтів буде не задоволеними і буде багато проблем будуть певні стягнення у розмір до 20%.

Календарний план

Таблиця 4.2 – Календарний план для розробки сервісу автопостингу

Місяць	Заплановані заходи	Особи
1	Розробка архітектури бота та сайту	Навроцький Б. Харун Ю.
2.5	Представлення робочого прототипу сайту та бота та їх тестування	Навроцький Б. Харун Ю.
3	Реліз сервісу	Навроцький Б. Харун Ю.

4.4 Нормативно-правові нюанси

Організаційно-правова форма бізнес-проекту

Сервіс автопостингу буде приватним, оскільки розробка проводилась особисто. У майбутньому буде можлива інтеграція або продаж з більшими проектами, або потенційно можливо викуп певного функціоналу соціальними мережами або іншими сервісами.

Організаційний план

Організацією і керуванням даного проекту будуть займатись два власники, у яких будуть права керувати і розпоряджатись ним, приймати будь-які рішення. Після відносного успіху планується залучити системних адміністраторів і менеджерів для подальшого зменшення активної діяльності в житті і функціонуванні саме прикладного рівня проекту, а зорієнтуватись на рівень рекламний і саме комерційний.

4.5 Складання фінансового бюджету

Визначення джерел фінансування

Джерелами фінансування очікуються вкладення інвесторів, а також особисті заощадження**.**

Графік початкового етапу реалізації проекту

Тривалість організаційного періоду проекту: 2 місяці

Тривалість розробки і тестування проекту: 3 місяці

Тривалість підготовки і введення в експлуатацію: 1 місяць

Очікувана сума загального фінансування: 30 000 - 40 000 грн.

Повна сума буде отримана після під час періоду введення в експлуатацію.

План очікуваних прибутків та збитків

Прибуток від реалізації послуг: на початкових етапах від 1 000 грн, починаючи від 3-го місяця очікується від 7 000 грн, а починаючи з 6 і далі місяця більше 20 000 грн.

Щомісячний податок: від 200 до 4000 грн, в залежності від доходу.

План очікуваних прибутків та збитків

Прибуток від реалізації послуг: в середньому 10000 грн. за місяць протягом першого року.

Чистий прибуток: від 4000 грн. за місяць

Постійні витрати: близько 5000 грн. за місяць

Змінні витрати: 1000 грн. за місяць

План руху грошових коштів

З різних доступних джерел фінансування інвестиції і особисті вкладення потрібно отримати разом 30 000-40 000 грн.

Після покупки усього обладнання і роботи буде сплачено 25500 грн.

Розрахунок показників проекту

Чистий прибуток за рік: від 50 000 грн.

Рентабельність: 125 %

Термін окупності: 9-10 місяців

4.6 Оцінка можливих ризиків

Одними з найочевидніших ризиків є:

1. велика потенційна конкуренція;
2. неуспішність рекламної компанії;
3. недовіра до нового сервісу без репутації;
4. невиконання плану закладеного доходу в перші місяці чи півріччя;
5. зміни в правилах соціальних мереж;
6. різке падіння попиту.

Попри перелічені можливі ризики, обґрунтувавши всі перелічені плюси, розробка даного автопостингу здається актуальною і затребуваною, особливо на молодому українському ринку соціальних мереж.

ВИСНОВКИ

За підсумками дипломного проекту, було розроблено вебдодаток для автопостингу, який взаємодіє з ботом і разом вони формують повноцінний вебсервіс з автопостингу. Для цього переглянуто і проаналізовано чимало варіантів виконання роботи, використано багато технології. У підсумку функціоналом створеного сервісу є:

- миттєвий автопостинг;
- відкладений автопостинг;
- збереження певних хештегів;
- публікація по розкладу;
- видалення по розкладу;
- адаптація поста під кожну соціальну мережу;
- збереження посту в чернетку, для подальшого редагування.

Можливе застосування такого сервісу має безліч варіантів. Зокрема, першочергове застосування – це для університетських потреб, для швидкої і зручної публікації потрібного контенту у декілька облікових записів соціальних мереж одночасно. Окрім цього, звісно можна використовувати цей вебсервіс у комерційних цілях, оскільки такі можливості дуже актуальні в сучасному світі блогерам і різним компаніям для успішного і зручного ведення своїх сторінок чи груп.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Визначення SMM-спеціаліст. URL: https://creativesmm.com.ua/smm-specialist/ (дата звернення: 12.05.2020).
2. Що таке автопостинг. URL: https://ru.wikipedia.org/wiki/Автопостинг (дата звернення: 12.05.2020).
3. Що таке кроспостинг. URL: https://en.wikipedia.org/wiki/Crossposting (дата звернення: 12.05.2020).
4. Сервіс автопостингу “NovaPress publisher”. URL: https://novapress.com/ (дата звернення: 12.05.2020).
5. Сервіс автопостингу “Amplifr”. URL: https://amplifr.com/ru/ (дата звернення: 12.05.2020).
6. Сервіс автопостингу “SMMplaner”. URL: https://smmplanner.com/home/ (дата звернення: 12.05.2020).
7. Козак О. Л. Аналіз вимог до програмного забезпечення : Опорний конспект лекцій. Тернопіль : ТНЕУ, 2011. 56 с.
8. E. S. Carruthers Rules For Rad: Rapid Application Development And Project Management. London: 2016, 106 p.
9. E. Walters Using UML Activities to model Business Processes: A Handbook for Practitioners (Handbooks for Modeling the Enterprise). Washington: Independently published, 2019, 100 p.
10. Про Sequence Diagram. URL: https://uk.wikipedia.org/wiki/Діаграма_послідовності (дата звернення: 15.05.2020).

11. Про Activity Diagram. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/ (дата звернення: 15.05.2020).
12. Про Verticle Timeline Diagram. URL: https://www.presentationgo.com/presentation/vertical-timeline-diagram-powerpoint/ (дата звернення: 15.05.2020).
13. Про ER Diagram. URL: https://www.smartdraw.com/entity-relationship-diagram/ (дата звернення: 15.05.2020).
14. Про Sequence diagram. URL: http://flash.retejo.info/cxefpagxo/uml/diagrama-poslidovnosti (дата звернення: 15.05.2020).
15. About relationship. URL: https://fmhelp.filemaker.com/help/18/fmp/en/index.html#page/FMP_Help%2Frelationships.html%23 (дата звернення: 15.05.2020).
16. Про Language Python. URL: https://www.python.org/ (дата звернення: 18.05.2020).
17. H. Hayes Python Programming: The Ultimate Crash Course for Beginners with All the Tools and Tricks to Learn Coding with Python (with Practical Examples). Ontario: 2019, 276 p.
18. R. Mitchell Web Scraping with Python: Collecting Data from the Modern Web 1st Edition. Boston: 2015, 256 p.
19. Про Micro Framework Flask. URL: https://flask.palletsprojects.com/en/1.1.x/ (дата звернення: 18.05.2020).
20. Sh. Aggarwal Flask Framework Cookbook: Over 80 proven recipes and techniques for Python web development with Flask, 2nd Edition Toronto: 2019, 302 p.
21. Про SQLAchemy. URL: https://www.sqlalchemy.org/ (дата звернення: 18.05.2020).

22. J. Myers, R. Copeland Essential SQLAlchemy: Mapping Python to Databases 2nd Edition New York: 2015, 208 p.
23. Про WTForms. URL: https://wtforms.readthedocs.io/en/2.3.x/ (дата звернення: 18.05.2020).
24. Про Flask-login. URL: https://flask-login.readthedocs.io/en/latest/ (дата звернення: 18.05.2020).
25. M. Kozlenko, V. Tkachuk, and M. Dutchak, "Software implementation of microcomputer based intrusion detection and prevention system with binary neural network," in Proc. 2nd International Scientific-Practical Conference "Problems of Cyber Security of Information and Telecommunication Systems" (PCSITS), O. Oksiiuk et al, Eds. Taras Shevchenko National University of Kyiv. Kyiv: 2019, pp. 371-373.
26. Про RQ. URL: https://python-rq.org/ (дата звернення: 18.05.2020).
27. Про REDIS. URL: https://redis.io/ (дата звернення: 18.05.2020).
28. Про Flask-admin. URL: https://flask-admin.readthedocs.io/en/latest/ (дата звернення: 18.05.2020).
29. G. Wong Amazon S3 Programming Guide: Beginner’s guide book on how to get started with Amazon Simple Storage Service. Toronto: 2016, 90 p.
30. Про Heroku. URL: https://dashboard.heroku.com/ (дата звернення: 18.05.2020).
31. A. Hanjura Heroku Cloud Application Development. New York: 2014, 336 p.
32. Про PostgreSQL. URL: https://uk.wikipedia.org/wiki/PostgreSQL (дата звернення: 18.05.2020).
33. S. Juba, A. Volkov Learning PostgreSQL 11: A beginner's guide to building high-performance PostgreSQL database solutions, 3rd Edition. New York: 2019, 556 p.

ДОДАТОК А

Лістинг програми (models.py)

```
from datetime import datetime
from autopost import db, login_manager
from flask_login import UserMixin
from hashlib import md5
from flask import Flask, request, jsonify, make_response
import uuid

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(30), unique=True,
nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)
    socials = db.relationship('Social', backref='owner',
lazy=True)
    posts = db.relationship('Post', backref='author', lazy=True)
    projects = db.relationship('Project', backref='own_project',
lazy=True)
    admin = db.Column(db.Boolean())
    def __repr__(self):
        return self.username
    def __init__(self, username, email, password, admin=False):
        self.username = username
        self.email = email
        self.password = password
        self.admin = admin
```

```
    def is_admin(self):
        return self.admin

class Project(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(60), nullable=False)
    socials = db.relationship('Social', backref='pr_owner',
lazy=True)
    posts = db.relationship('Post', backref='pr_post', lazy=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'),
nullable=False)
    def __repr__(self):
        return self.name
association_table = db.Table('association_table',
db.Model.metadata,
    db.Column('Socials_id', db.Integer,
db.ForeignKey('socials.id',ondelete="CASCADE"),primary_key=True),
    db.Column('Post_id', db.Integer,
db.ForeignKey('posts.id',ondelete="CASCADE"),primary_key=True),
    db.UniqueConstraint('Socials_id', 'Post_id',
name='UC_social_id_post_id')
)

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    date_posted = db.Column(db.DateTime)
    content = db.Column(db.Text, nullable=False)
    image_file = db.Column(db.String(100))
    tags = db.Column(db.Text)
    already_posted = db.Column(db.Boolean())
    project_id = db.Column(db.Integer,
db.ForeignKey('project.id'))
    job_id = db.Column(db.Text(1000))
```

```
    link_post = db.Column(db.Text(1000))
    notes = db.Column(db.Boolean())
    __tablename__ = 'posts'
    socials = db.relationship('Social', secondary =
association_table,back_populates='posts',  lazy='dynamic')
    #social_id = db.Column(db.Integer, db.ForeignKey('social.id'))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'),
nullable=False)
    def __repr__(self):
        return self.title

class Social(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    login = db.Column(db.String(30), nullable=False)
    password = db.Column(db.String(120), nullable=False)
    type = db.Column(db.String(30), nullable=False)
    __tablename__ = 'socials'
    posts = db.relationship('Post',
secondary=association_table,back_populates='socials',
lazy='dynamic')
    project_id = db.Column(db.Integer,
db.ForeignKey('project.id'))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'),
nullable=False)
    def __repr__(self):
        return self.login +" social: "+ self.type
```

Лістинг програми (forms.py)

```
from flask_login import current_user
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed
```

```
from wtforms import SelectField, widgets, DateTimeField,
StringField, PasswordField, SubmitField,
BooleanField,TextAreaField,SelectMultipleField
from wtforms.validators import InputRequired, DataRequired,
Length, Email, EqualTo, ValidationError, Optional
from autopost.models import User, Post, Project, Social
from flask import flash, redirect, url_for, Markup
from wtforms.fields.html5 import DateField
#from wtforms_components import TimeField, TimeRange
from wtforms.widgets.html5 import TimeInput
from flask_admin.contrib.sqla import ModelView
from flask_admin import BaseView, expose, AdminIndexView
from flask_admin.form import rules
from autopost import bcrypt
import datetime

class RegistrationForm(FlaskForm):
    username=StringField('Username',
validators=[DataRequired(),Length(min=2,max=20)])
    email=StringField('Email', validators=[DataRequired()])
    password = PasswordField('Password',validators =
[DataRequired()])
    confirm_password = PasswordField('Confirm Password',validators
= [DataRequired(), EqualTo('password')])
    submit= SubmitField('Sign up')
    def validate_field(self,field):
        if True:
            raise ValidationError('Validation message')
    def validate_username(self, username):
        user =
User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken. Please
choose a different one')
```

```
    def validate_email(self,field):
        if User.query.filter_by(email=field.data).first():
            raise ValidationError('Email already registered.')
    def validate_username(self,field):
        if User.query.filter_by(username=field.data).first():
            raise ValidationError('Username already in use.')

class LoginForm(FlaskForm):
    email =StringField('Email', validators =
[DataRequired(),Email()])
    password = PasswordField('Password',validators =
[DataRequired()])
    remember = BooleanField('Remember Me')
    submit= SubmitField('Log in')

class UpdateAccountForm(FlaskForm):
    username=StringField('Username',
validators=[DataRequired(),Length(min=2,max=20)])
    email=StringField('Email',
validators=[DataRequired(),Email()])
    submit= SubmitField('Update')
    new_password = PasswordField('New password',validators =
[DataRequired()])
    passwordcheck = PasswordField('Old password',validators =
[DataRequired()])
    def validate_username(self, username):
        if username.data != current_user.username:
            user =
User.query.filter_by(username=username.data).first()
            if user:
                raise ValidationError('That username is taken.
Please choose a different one')
    def validate_email(self,email):
        if email.data != current_user.email:
```

```
            user = User.query.filter_by(email=email.data).first()
            if user:
                raise ValidationError('That email is taken. Please
choose a different one.')

class AddTask(FlaskForm):
    title = StringField('Title',
validators=[DataRequired()],render_kw={"placeholder": "It is your
content title"})
    content = TextAreaField('Content',
validators=[DataRequired()],render_kw={"placeholder": "This is
your main content"})
    date_posted = DateField('Date Posted', format='%Y-%m-
%d',validators=[Optional()],render_kw={"placeholder": "Format date
example: 2020-06-01"})
    time_posted = DateTimeField('Time
Posted',format='%H:%M',validators=[Optional()],render_kw={"placeho
lder": "Format Time example: 20:30"})
    image_file = FileField('Choose picture',
validators=[FileAllowed(['jpg','png','mp4'])])
    image_file_url = StringField('Your picture now: ' )
    tags = TextAreaField('Tags',render_kw={"placeholder": "Write
here your tags. Format:  #something #test "})
    #my_choices = [('1', 'Choice1'), ('2', 'Choice2'), ('3',
'Choice3')]
    socials = SelectMultipleField('Socials',
coerce=int,validators=[Optional()])
    notes = BooleanField('It is notes')
    submit = SubmitField('Add task')

class AddSocial(FlaskForm):
    login = StringField('Login', validators=[DataRequired()])
    password = PasswordField('Password',validators =
[DataRequired()])
```

```
    type = SelectField(u'Type', choices=[('Facebook',
'Facebook'),\
                        ('Instagram', 'Instagram'), ('Twitter',
'Twitter')])
    submit = SubmitField('Add social')

class UpdateSocial(FlaskForm):
    login = StringField('Login', validators=[DataRequired()])
    new_password = PasswordField('New password',validators =
[DataRequired()])
    passwordcheck = PasswordField('Old password',validators =
[DataRequired()])
    submit = SubmitField('Add social')

class AdminUserCreateForm(FlaskForm):
    username = StringField('Username', [InputRequired()])
    password = PasswordField('Password', [InputRequired()])
    admin = BooleanField('Is Admin ?')
    posts = StringField('Posts', [InputRequired()])
class AdminUserUpdateForm(FlaskForm):
    username = StringField('Username', [InputRequired()])
    admin = BooleanField('Is Admin ?')

class MyAdminIndexView(AdminIndexView):
    def is_accessible(self):
        return current_user.is_authenticated and
current_user.is_admin()

class CKTextAreaWidget(widgets.TextArea):
    def __call__(self, field, **kwargs):
        kwargs.setdefault('class_', 'ckeditor')
        return super(CKTextAreaWidget, self).__call__(field,
**kwargs)
```

```
class CKTextAreaField(TextAreaField):
    widget = CKTextAreaWidget()

class UserAdminView(ModelView):
    column_searchable_list = ('username',)
    column_sortable_list = ('username', 'admin')
    create_template = 'create.html'
    edit_template = 'edit.html'

    def is_accessible(self):
        return current_user.is_authenticated and
current_user.is_admin()

    def scaffold_form(self):
        form_class = super(UserAdminView, self).scaffold_form()
        form_class.new_password = PasswordField('New Password')
        form_class.confirm = PasswordField('Confirm New Password')
        return form_class

    def create_model(self, form):
        model = self.model(
            form.username.data, form.password.data,
form.admin.data
        )
        form.populate_obj(model)
        model.password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        self.session.add(model)
        self._on_model_change(form, model, True)
        self.session.commit()
        return redirect(url_for('home_admin'))

    def update_model(self, form, model):
        form.populate_obj(model)
```

```
        if form.new_password.data:
            if form.new_password.data != form.confirm.data:
                return flash('Passwords must match')
            model.password =
bcrypt.generate_password_hash(form.new_password.data).decode('utf-
8')
        self.session.add(model)
        self._on_model_change(form, model, False)
        self.session.commit()
        return redirect(url_for('home_admin'))

class PostAdminView(ModelView):
    column_searchable_list = ('title',)
    column_sortable_list = ('title',
'already_posted','date_posted')
    details_template = 'details.html'
    column_details_list = ('type','login','socials')
    create_template = 'create.html'
    edit_template = 'edit.html'

    def is_accessible(self):
        return current_user.is_authenticated and
current_user.is_admin()

class SocialAdminView(ModelView):
    column_searchable_list = ('login',)
    create_template = 'create.html'
    edit_template = 'edit.html'
    details_template = 'details.html'
    #can_view_details = True
    column_details_list = ('type','login','posts')
    #column_exclude_list = ('password',)
    #form_excluded_columns = ('password',)
    form_overrides = dict(
```

```
        type=SelectField
    )
    form_args = dict(
        type=dict(
            choices=[('Instagram', 'Instagram'),\
                           ('Facebook', 'Facebook'), ('Twitter',
'Twitter')]
        )
    )

    def _get_list_value(self, context, model, name,
column_formatters,
                        column_type_formatters):
        """
            Returns the value to be displayed.
            :param context:
                :py:class:`jinja2.runtime.Context` if available
            :param model:
                Model instance
            :param name:
                Field name
            :param column_formatters:
                column_formatters to be used.
            :param column_type_formatters:
                column_type_formatters to be used.
        """
        column_fmt = column_formatters.get(name)
        if column_fmt is not None:
            value = column_fmt(self, context, model, name)
        else:
            value = self._get_field_value(model, name)

        choices_map = self._column_choices_map.get(name, {})
        if choices_map:
```

```
            return choices_map.get(value) or value

        type_fmt = None
        for typeobj, formatter in column_type_formatters.items():
            if isinstance(value, typeobj):
                type_fmt = formatter
                break
        if type_fmt is not None:
            value = type_fmt(self, value)
        ### overwritten here
        if name == 'posts':

            html_string  = '<table>'
            for item in value.split(','):
                html_string += '<tr><td> {}
</td></tr>'.format(item)
            html_string += '</table>'

            value = Markup(html_string)

        return value




    def create_model(self, form):

        model = self.model(
            login = form.login.data, password =
form.password.data, \
            type=dict(form.type.choices).get(form.type.data)
        )
        form.populate_obj(model)
```

```
        model.password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        self.session.add(model)
        self._on_model_change(form, model, True)
        self.session.commit()
        return redirect(url_for('home_admin'))

    def update_model(self, form, model):
        form.populate_obj(model)
        if form.password.data:
            model.password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        self.session.add(model)
        self._on_model_change(form, model, False)
        self.session.commit()
        return redirect(url_for('home_admin'))


    def is_accessible(self):
        return current_user.is_authenticated and
current_user.is_admin()
```

Лістинг програми (settings.py)

```
import os


SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') #
'sqlite:///' + os.path.join(basedir, 'site.db')
SECRET_KEY = os.environ.get('SECRET_KEY')
#'5791628bb0b13ce0c676dfde280ba245'
SQLALCHEMY_TRACK_MODIFICATIONS = False

S3_BUCKET = os.environ.get('S3_BUCKET')
```

```
S3_KEY = os.environ.get('AWS_ACCESS_KEY_ID')
S3_SECRET = os.environ.get('AWS_SECRET_ACCESS_KEY')

REDISTOGO_URL=os.environ.get('REDISTOGO_URL')

CHROMEDRIVER_PATH = os.environ.get('CHROMEDRIVER_PATH')
GOOGLE_CHROME_BIN = os.environ.get('GOOGLE_CHROME_BIN')
```

Лістинг програми (resources.py)

```
import boto3
from autopost.settings import S3_BUCKET, S3_KEY, S3_SECRET
from flask import session,Response
import os
import errno
import botocore
from pathlib import Path

def _get_s3_resource():
    if S3_KEY and S3_SECRET:
        return boto3.resource(
            's3',
            aws_access_key_id=S3_KEY,
            aws_secret_access_key=S3_SECRET
        )
    else:
        return boto3.resource('s3')


def get_bucket():
    s3_resource = _get_s3_resource()
    if 'bucket' in session:
        bucket = session['bucket']
    else:
```

```
        bucket = S3_BUCKET

    return s3_resource.Bucket(bucket)


def get_buckets_list():
    client = boto3.client('s3')
    return client.list_buckets().get('Buckets')

def make_sure_path_exists(path):
    try:
        os.makedirs(path)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise

def download(key):
    s3 = boto3.client('s3', region_name='us-west-2')
    key_dir,key_name = key.split('/')

    make_sure_path_exists('tmp/'+key_dir)
    local_path = os.path.join("tmp/", key)
    try:
        with open(key_name, 'wb') as data:
            s3.download_fileobj(S3_BUCKET, key, data)

        print('success download')
    except:
        print("The object does not exist.")
    local_path_test = '/tmp/' + key
    templateDir = os.path.dirname(__file__)
    key_name_test = os.path.dirname(key_name)
    templateDir=templateDir[:-9]
    last_test = templateDir+local_path_test
```

```
    print('last_test')
    print(last_test)
    return last_test
```

Лістинг програми (__init__.py)

```
from flask import Flask, request, jsonify, make_response
import uuid
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt
from flask_login import LoginManager
from flask_migrate import Migrate, MigrateCommand
from flask_script import Manager
import os
from flask_admin import Admin, BaseView, expose
from flask_admin.contrib.sqla import ModelView
from flask_ckeditor import CKEditor
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

config_file='settings.py'
app = Flask(__name__)
#basedir = os.path.abspath(os.path.dirname(__file__))'

chrome_options = webdriver.ChromeOptions()
chrome_options.binary_location                                    =
os.environ.get("GOOGLE_CHROME_BIN")
chrome_options.add_argument("--headless")
chrome_options.add_argument("--disable-dev-shm-usage")
chrome_options.add_argument("--no-sandbox")
prefs  =  {"profile.default_content_setting_values.notifications"  :
2}
chrome_options.add_experimental_option("prefs",prefs)
```

```
driver                                                              =
webdriver.Chrome(executable_path=os.environ.get("CHROMEDRIVER_PATH
"), chrome_options=chrome_options)

app.config.from_pyfile(config_file)
db = SQLAlchemy(app)

migrate = Migrate(app,db)
manager = Manager(app)
manager.add_command('db', MigrateCommand)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message_category = 'info'

app.config['CKEDITOR_PKG_TYPE'] = 'basic'
ckeditor = CKEditor(app)

import autopost.forms as views
admin = Admin(app, index_view=views.MyAdminIndexView())
admin.add_view(views.UserAdminView(views.User, db.session))
admin.add_view(views.PostAdminView(views.Post, db.session))
admin.add_view(views.ProjectAdminView(views.Project, db.session))
admin.add_view(views.SocialAdminView(views.Social, db.session))

from autopost import routes
```

Лістинг програми (routes.py)

```
from flask import json,jsonify, render_template,url_for, flash,
redirect, request,abort, session,Response
from autopost import app, db, bcrypt
from PIL import Image
import json, facebook
```

```
from autopost.forms import *
from autopost.models import User, Post, Project, Social
from flask_login import login_user, current_user,
logout_user,login_required
import os
import secrets
import errno
from datetime import datetime
import shutil
from functools import wraps
from flask_admin import BaseView, expose
import uuid
import boto3
#from autopost.settings import S3_BUCKET,S3_KEY,S3_SECRET
from botocore.exceptions import ClientError
import logging
from autopost.resources import *
from pathlib import Path
from time import sleep
import time
from autopost.test_bot import *
from wtforms.utils import unset_value

from autopost import driver
from worker import *
from utils import *

@app.route("/")
@app.route("/home")
def home():
    if current_user.is_authenticated:
        username = current_user.username
        user =
User.query.filter_by(username=username).first_or_404()
```

```
        page = request.args.get('page', 1, type=int)
        posts =
Post.query.filter_by(user_id=user.id).filter_by(notes=False).order
_by(Post.date_posted.desc()).paginate(page, 10, False)
        return render_template('home.html', user=user,
posts=posts)
    else:
        return render_template('home_dev.html')

@app.route("/notes")
def notes():
    if current_user.is_authenticated:
        username = current_user.username
        user =
User.query.filter_by(username=username).first_or_404()
        page = request.args.get('page', 1, type=int)
        posts =
Post.query.filter_by(user_id=user.id).filter_by(notes=True).order_
by(Post.date_posted.desc()).paginate(page, 10, False)
        return render_template('notes.html', user=user,
posts=posts)
    else:
        return redirect(url_for('home'))

@app.route("/about")
def about():
    return render_template('about.html', title='About')

@app.route("/add_task", methods=['GET', 'POST'])
@login_required
def add_task():
    form = AddTask()
```

```
    if form.validate_on_submit():
        if form.image_file.data:
            file = request.files['image_file']
            nameee = request.files['image_file'].filename
            extensions = Path(nameee).suffixes
            ext = "".join(extensions)
            random_hex = str(secrets.token_hex(10))
            usern = str(current_user.username)
            name_file = usern + "/"+ random_hex + ext

            my_bucket = get_bucket()
            my_bucket.Object(name_file).put(ACL='public-read',
Body=file,ContentType ='image/png')

            file_path_3 = 'https://dyploma-autopost2.s3-us-west-
2.amazonaws.com/' + name_file

        else:
            file_path_3 = "no file"
        date_posted2 = None

        post = Post(title = form.title.data, content =
form.content.data, \
                    author= current_user, date_posted =
date_posted2, \
                    image_file = file_path_3, tags =
form.tags.data, \
                    already_posted=False,notes= form.notes.data
                    )

        if form.date_posted.data:
            date_test = str(form.date_posted.data)
            year,month,day = date_test.split('-')
            hour_ser = 0
```

```
            minute = 0
            date_posted2 = datetime(int(year), int(month),
int(day), int(hour_ser), int(minute))
            post.date_posted=date_posted2

        if form.time_posted.data:
            time_test = str(form.time_posted.data)
            time_te = time_test[11:]
            hour_ser,minute,seconds = time_te.split(':')
            year = 2020
            month = 1
            day = 1
            date_posted2 = datetime(int(year), int(month),
int(day), int(hour_ser), int(minute))
            post.date_posted=date_posted2

        if form.date_posted.data and form.time_posted.data:
            date_test = str(form.date_posted.data)
            time_test = str(form.time_posted.data)
            time_te = time_test[11:]
            year,month,day = date_test.split('-')
            hour_ser,minute,seconds = time_te.split(':')
            date_posted2 = datetime(int(year), int(month),
int(day), int(hour_ser), int(minute))
            post.date_posted=date_posted2

        if not form.time_posted.data or not form.date_posted.data:
            post.notes=True

        db.session.add(post)
        db.session.commit()
        test_publish = post.title + '\n\n'+ post.content +
'\n\n'+post.tags
        test = None
```

```
        if post.image_file!=None and post.image_file!="no file":
            key = post.image_file
            test = file_path_3
            print(test)
        else:
            test = None
        flash('Your task has been created!', 'success')
        return redirect(url_for('home'))
    return render_template('create_task.html', title='New Task',
form = form, legend = 'New task')

@app.route("/add_social", methods=['GET', 'POST'])
@login_required
def add_social():
    form = AddSocial()
    if form.validate_on_submit():
        #type2 = dict(form.type.choices).get(form.type.data)
        #hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        social = Social(login=form.login.data,
password=form.password.data\

,owner=current_user,type=dict(form.type.choices).get(form.type.dat
a))
        db.session.add(social)
        db.session.commit()
        flash('Your task has been created!', 'success')
        return redirect(url_for('socials'))
    return render_template('create_social.html', title='New
social', form = form, legend = 'New social')

@app.route("/register", methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
```

```
        return redirect(url_for('home'))
    form =RegistrationForm()
    if form.validate_on_submit():
        hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username = form.username.data, email =
form.email.data, password = hashed_password)

        db.session.add(user)
        db.session.commit()
        flash(f'Your account has been created. You are now able to
log in', 'success')
        return redirect(url_for('login'))
    return render_template('register.html',
title='Register',form=form)

@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password,
form.password.data):
            login_user(user, remember = form.remember.data)
            next_page = request.args.get('next')
            return redirect (next_page) if next_page else
redirect(url_for('home'))
        else:
            flash('Login Unsuccessful. Please check username and
password', 'danger')
    return render_template('login.html', title='Login', form=form)
```

```
@app.route("/logout")
def logout():
    logout_user()
    flash('You have been logged out.')
    return redirect(url_for('home'))

@app.route("/account", methods=['GET', 'POST'])
@login_required
def account():
    form = UpdateAccountForm()
    if request.method == "POST" and form.validate():
        user = User.query.filter_by(email=form.email.data).first()

        if bcrypt.check_password_hash(current_user.password,
form.passwordcheck.data):
            hashed_password =
bcrypt.generate_password_hash(form.new_password.data).decode('utf-
8')
            current_user.password = hashed_password
            current_user.username = form.username.data
            current_user.email = form.email.data
            user.password = hashed_password
            db.session.commit()
            flash('Success. You change your account', 'success')
        else:
            flash('Unsuccess. Please check correct of yuor input
data', 'danger')
    elif request.method == 'GET':
        form.username.data = current_user.username
        form.email.data = current_user.email
    return render_template('account.html', title = 'Account', form
= form)
```

```
def admin_login_required(func):
    @wraps(func)
    def decorated_view(*args, **kwargs):
        if not current_user.is_admin():
            return abort(403)
        return func(*args, **kwargs)
    return decorated_view


def get_res(job, post):
    time.sleep(120)
    result_job = job.result
    tessstid = post.job_id
    post.job_id = str(tessstid) +  str(job.id)
    lennnnk = post.link_post
    post.link_post = str(lennnnk) + str(result_job)
    #return str(result_job)

@app.route("/task/<int:post_id>/update", methods=['GET', 'POST'])
@login_required
def update_task(post_id):

    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)
    user = current_user
    need_socials = Social.query.filter_by(user_id = user.id).all()
    socials_list = [(i.id, i.login +"|"+ i.type ) for i in
need_socials]
    form = AddTask(obj=user)
    choo_noth = [(0,None)] + socials_list
    form.socials.choices = choo_noth

    if request.method == 'POST' and form.validate_on_submit():
```

```
        sommm = post.socials
        for elem in sommm:
            print(elem)
            post.socials.remove(elem)
            elem.posts.remove(post)
        db.session.commit()

        file_path = post.image_file

        if form.image_file.data:
            file = request.files['image_file']
            nameee = request.files['image_file'].filename
            extensions = Path(nameee).suffixes
            ext = "".join(extensions)
            random_hex = str(secrets.token_hex(10))
            usern = str(current_user.username)
            name_file = usern + "/"+ random_hex + ext

            my_bucket = get_bucket()
            my_bucket.Object(name_file).put(ACL='public-read',
Body=file,ContentType ='image/png')

            file_path_3 = 'https://dyploma-autopost2.s3-us-west-
2.amazonaws.com/' + name_file

        else:
            file_path_3 = file_path
        date_test = str(form.date_posted.data)
        time_test = str(form.time_posted.data)
        time_te = time_test[11:]
        year,month,day = date_test.split('-')
        hour_ser,minute,seconds = time_te.split(':')
```

```
        date_posted2 = datetime(int(year), int(month), int(day),
int(hour_ser), int(minute))
        post.title = form.title.data
        post.content = form.content.data
        post.date_posted = date_posted2
        post.image_file = file_path_3
        post.tags = form.tags.data
        soc = form.socials.data

        for elem in soc:
            test_int = int(str(elem))
            elem_soc = Social.query.get(test_int)
            print(elem_soc)
            print(test_int)
            if test_int != 0:
                post.socials.append(elem_soc)
                print('maybe success')
                db.session.commit()
            elif test_int ==0:
                sommm = post.socials
                for elem in sommm:
                    print(elem)
                    post.socials.remove(elem)
                    elem.posts.remove(post)
                #elem_soc.posts.remove(post)
                #post.socials.remove(elem_soc)
                #elem_soc.posts.remove(post)
                db.session.commit()
                print("valu = 0")
            else:
                #flash('Your can not choose nothing', 'danger')
                print("not value")
```

```
            db.session.commit()
            flash('Your task has been updated!', 'success')
            return redirect(url_for('home'))

        elif request.method == 'GET':
            test_default = []
            teeeeest_post = Post.query.filter_by(id = post.id).all()
            print(teeeeest_post)
            #len_soc = len(post.socials)
            #if len_soc>0:
                #print(post.socials)
            for elem in post.socials:
                tmp_default = str(elem.id)
                test_default.append(tmp_default)
            print(test_default)
            if len(test_default)>0:
                form.socials.default = test_default
                form.process()

            form.title.data = post.title
            form.content.data = post.content
            form.date_posted.data = post.date_posted
            form.time_posted.data = post.date_posted
            form.image_file.data = post.image_file
            form.tags.data = post.tags
            form.image_file_url.data = post.image_file


        return render_template('update_task.html', title='Update Task'
, form  = form, legend = 'Update Task',post = post)

@app.route("/note/<int:post_id>/update", methods=['GET', 'POST'])
@login_required
def update_note(post_id):
```

```
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)
    user = current_user
    need_socials = Social.query.filter_by(user_id = user.id).all()
    socials_list = [(i.id, i.login +"|"+ i.type ) for i in
need_socials]
    form = AddTask(obj=user)
    choo_noth = [(0,None)] + socials_list
    form.socials.choices = choo_noth

    if request.method == 'POST' and form.validate_on_submit():

        sommm = post.socials
        for elem in sommm:
            print(elem)
            post.socials.remove(elem)
            elem.posts.remove(post)
        db.session.commit()

        file_path = post.image_file

        if form.image_file.data:
            file = request.files['image_file']
            nameee = request.files['image_file'].filename
            extensions = Path(nameee).suffixes
            ext = "".join(extensions)
            random_hex = str(secrets.token_hex(10))
            usern = str(current_user.username)
            name_file = usern + "/"+ random_hex + ext

            my_bucket = get_bucket()
```

```
            my_bucket.Object(name_file).put(ACL='public-read',
Body=file,ContentType ='image/png')

            file_path_3 = 'https://dyploma-autopost2.s3-us-west-
2.amazonaws.com/' + name_file
            post.image_file = file_path_3
        else:
            file_path_3 = file_path
            post.image_file = file_path_3

        if form.date_posted.data and form.time_posted.data:
            date_test = str(form.date_posted.data)
            time_test = str(form.time_posted.data)
            time_te = time_test[11:]
            year,month,day = date_test.split('-')
            hour_ser,minute,seconds = time_te.split(':')
            date_posted2 = datetime(int(year), int(month),
int(day), int(hour_ser), int(minute))
            post.date_posted = date_posted2
        if form.date_posted.data:
            date_test = str(form.date_posted.data)
            year,month,day = date_test.split('-')
            hour_ser = 0
            minute = 0
            date_posted2 = datetime(int(year), int(month),
int(day), int(hour_ser), int(minute))
            post.date_posted = date_posted2
        if form.time_posted.data:
            time_test = str(form.time_posted.data)
            time_te = time_test[11:]
            hour_ser,minute,seconds = time_te.split(':')
            year = 2020
            month = 1
            day = 1
```

```
            date_posted2 = datetime(int(year), int(month),
int(day), int(hour_ser), int(minute))
            post.date_posted = date_posted2

        if form.title.data:
            post.title = form.title.data
        if form.content.data:
            post.content = form.content.data
        if form.tags.data:
            post.tags = form.tags.data

        soc = form.socials.data

        for elem in soc:
            test_int = int(str(elem))
            elem_soc = Social.query.get(test_int)
            print(elem_soc)
            print(test_int)
            if test_int != 0:
                post.socials.append(elem_soc)
                print('maybe success')
                db.session.commit()
            elif test_int ==0:
                sommm = post.socials
                for elem in sommm:
                    print(elem)
                    post.socials.remove(elem)
                    elem.posts.remove(post)
                #elem_soc.posts.remove(post)
                #post.socials.remove(elem_soc)
                #elem_soc.posts.remove(post)
                db.session.commit()
                print("valu = 0")
            else:
```

```
                #flash('Your can not choose nothing', 'danger')
                print("not value")

        db.session.commit()
        flash('Your note has been updated!', 'success')
        return redirect(url_for('notes'))

    elif request.method == 'GET':
        test_default = []
        teeeeest_post = Post.query.filter_by(id = post.id).all()
        print(teeeeest_post)
        #len_soc = len(post.socials)
        #if len_soc>0:
            #print(post.socials)
        for elem in post.socials:
            tmp_default = str(elem.id)
            test_default.append(tmp_default)
        print(test_default)
        if len(test_default)>0:
            form.socials.default = test_default
            form.process()

        form.title.data = post.title
        form.content.data = post.content
        form.date_posted.data = post.date_posted
        form.time_posted.data = post.date_posted
        form.image_file.data = post.image_file
        form.tags.data = post.tags
        form.image_file_url.data = post.image_file


    return render_template('update_note.html', title='Update Task'
, form  = form, legend = 'Update Task',post = post)
```

```
@app.route("/note/<int:post_id>/add_to_task", methods=['GET',
'POST'])
@login_required
def add_to_task(post_id):
    if current_user.is_authenticated:
        post = Post.query.get_or_404(post_id)
        if post.author != current_user:
            abort(403)



        if post.date_posted:
            post.notes = False
            db.session.commit()
            flash('You add a new task from note!', 'success')
            return redirect(url_for('home'))
        else:
            flash('You did no choose datetime!', 'danger')
            return redirect(url_for('notes'))
    else:
        pass
    return redirect(url_for('notes'))

@app.route("/task/<int:post_id>/delete", methods=['GET', 'POST'])
@login_required
def delete_task(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)
    db.session.delete(post)
    db.session.commit()
    flash('Your post has been deleted!', 'success')
    return redirect(url_for('home'))

@app.route("/note/<int:post_id>/delete", methods=['GET', 'POST'])
```

```
@login_required
def delete_note(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)
    db.session.delete(post)
    db.session.commit()
    flash('Your note has been deleted!', 'success')
    return redirect(url_for('notes'))

@app.route("/socials")
def socials():
    if current_user.is_authenticated:
        username = current_user.username
        user =
User.query.filter_by(username=username).first_or_404()
        page = request.args.get('page', 1, type=int)
        socials =
Social.query.filter_by(user_id=user.id).order_by(Social.id.desc())
.paginate(page, 5, False)
        return render_template('socials.html', user=user,
socials=socials)
    else:
        return redirect(url_for('home'))

@app.route("/social/<int:social_id>/update", methods=['GET',
'POST'])
@login_required
def update_social(social_id):
    social = Social.query.get_or_404(social_id)
    form = UpdateSocial()
    if request.method == 'POST' and form.validate_on_submit():
        if social.password == form.passwordcheck.data:
```

```
            #hashed_password =
bcrypt.generate_password_hash(form.new_password.data).decode('utf-
8')
            social.password = form.new_password.data
            social.login = form.login.data
            db.session.commit()
            flash('Success. You change your social account',
'success')
        else:
            flash('Unsuccess. Please check correct of yuor input
data', 'danger')
        return redirect(url_for('socials'))
    elif request.method == 'GET':
        form.login.data = social.login
        #form.type.choices
    return render_template('update_social.html', title='Update
Social account' , form  = form, legend = 'Update Social
account',social = social)

@app.route("/social/<int:social_id>/delete", methods=['GET',
'POST'])
@login_required
def delete_social(social_id):
    social = Social.query.get_or_404(social_id)
    if social.owner != current_user:
        abort(403)

    db.session.delete(social)
    db.session.commit()
    flash('Your social has been deleted!', 'success')
    return redirect(url_for('socials'))

@app.route("/post/<int:post_id>/publish", methods=['GET', 'POST'])
@login_required
```

```
def publish_task(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)

    test_publish = post.title + '\n\n'+ post.content +
'\n\n'+post.tags
    test = None
    if post.image_file!=None and post.image_file!="no file":
        #key = post.image_file
        test = post.image_file
        print(test)
    else:
        test = None

    test_datetime = post.date_posted
    take_day,take_time = str(test_datetime).split(' ')
    year,month,day = take_day.split('-')
    hour_ser,minute,seconds = take_time.split(':')
    hour = int(hour_ser) - 3
    if hour<0:
        hour=24+hour
    tmp = 0
    for soc in post.socials:
        tmp = tmp+1

    if tmp>0:
        for soc in post.socials:
            if soc.type =='Facebook':
                print('its facebook')
                #url =
facebook_create_post(soc.login,soc.password,test_publish,test)
```

```
                job =
queue.enqueue(facebook_create_post,soc.login,soc.password,test_pub
lish,test )
                post.job_id = str(job)
                print('success')
                print(soc)
            if soc.type =='Instagram':
                alr_posts = str(post.link_post)
                url = 'test'
                instagram_url = "instagram: " + url + " "
                post.link_post = alr_posts + instagram_url
                print(alr_posts + instagram_url)
                print('its instagram')
            if soc.type =='Twitter':
                url="test"
                alr_posts = str(post.link_post)
                twitter_url = "twitter: " + url + " "
                post.link_post = alr_posts + twitter_url
                print(alr_posts + twitter_url)
                print('its twitter')
        post.already_posted = True
        flash('Your post will be publish now!', 'success')
    else:
        post.already_posted = False
        flash('Your post not publish now, please choose your
social!', 'danger')
    db.session.commit()
    return redirect(url_for('home'))

@app.route("/post/<int:post_id>/addqueue", methods=['GET',
'POST'])
@login_required
def add_to_queue_task(post_id):
    post = Post.query.get_or_404(post_id)
```

```
    if post.author != current_user:
        abort(403)
    test_publish = post.title + '\n\n'+ post.content +
'\n\n'+post.tags
    test = None
    if post.image_file!=None and post.image_file!="no file":
        #key = post.image_file
        test = post.image_file
        print(test)
    else:
        test = None

    test_datetime = post.date_posted
    take_day,take_time = str(test_datetime).split(' ')
    year,month,day = take_day.split('-')
    hour_ser,minute,seconds = take_time.split(':')
    hour = int(hour_ser) - 3
    if hour<0:
        hour=24+hour

    tmp = 0
    for soc in post.socials:
        tmp = tmp+1

    if tmp>0:
        for soc in post.socials:
            if soc.type =='Facebook':
                print('its facebook')
                #url =
facebook_create_post(soc.login,soc.password,test_publish,test)
                job = queue.enqueue_at(datetime(int(year),
int(month), int(day), hour, int(minute)),
facebook_create_post,soc.login,soc.password,test_publish,test)
                registry = ScheduledJobRegistry(queue=queue)
```

```
                print(job in registry)
                post.job_id = str(job)
                print('success')
                print(soc)
            if soc.type =='Instagram':
                alr_posts = str(post.link_post)
                url = 'test'
                instagram_url = "instagram: " + url + " "
                post.link_post = alr_posts + instagram_url
                print(alr_posts + instagram_url)
                print('its instagram')
            if soc.type =='Twitter':
                url="test"
                alr_posts = str(post.link_post)
                twitter_url = "twitter: " + url + " "
                post.link_post = alr_posts + twitter_url
                print(alr_posts + twitter_url)
                print('its twitter')
        post.already_posted = True
    else:
        post.already_posted = False
        flash('Your post not publish now, please choose your
social!', 'danger')
    db.session.commit()
    flash('Your post will add to queue!', 'success')
    return redirect(url_for('home'))

@app.route("/post/<int:post_id>/go_to_post")
def go_to_post(post_id):
    if current_user.is_authenticated:

        post = Post.query.get_or_404(post_id)
        if post.author != current_user:
            abort(403)
```

```
        tmp = 0
        for soc in post.socials:
            print(soc)
            tmp = tmp+1
        url = post.link_post
        url_len = len(str(url)) / tmp
        print(str(url))
        print(url_len)
        url_len = int(url_len)
        test_url = url[:url_len]
        print(test_url)
        test_url_list = []
        n = 0
        while tmp>0:
            try:
                test_url = url[n:url_len]
                test_url_list.append(test_url)
                n = n+url_len
                url_len = url_len+url_len
                tmp=tmp-1
            except:
                print("no one links")
        print(test_url_list)
        #posts =
Post.query.filter_by(user_id=user.id).filter_by(notes=False).order
_by(Post.date_posted.desc()).paginate(page, 10, False)
        return render_template('go_to_post.html',
post=post,test_url_list=test_url_list)
    else:
        return redirect(url_for('home'))

@app.route("/post/<int:post_id>/<int:url_count>/delete_post_on_soc
ial")
def delete_post_on_social(post_id,url_count):
```

```
    if current_user.is_authenticated:
        post = Post.query.get_or_404(post_id)
        if post.author != current_user:
            abort(403)
        tmp = 0
        for soc in post.socials:
            print(soc)
            if tmp == url_count:
                soc_del = soc
                social_log = soc.login
                print(social_log)
                social_pas = soc.password
                print(social_pas)
            tmp = tmp+1
        url = post.link_post
        url_len = len(str(url)) / tmp
        url_len = int(url_len)
        test_url = url[:url_len]
        test_url_list = []
        n = 0
        while tmp>0:
            try:
                test_url = url[n:url_len]
                test_url_list.append(test_url)
                n = n+url_len
                url_len = url_len+url_len
                tmp=tmp-1
            except:
                print("no one links")
        print(test_url_list)
        need_url = test_url_list[url_count]
        print(need_url)
        job =
queue.enqueue(facebook_delete_post,social_log,social_pas,need_url)
```

```
        test_url_list.remove(need_url)

        str1 = ""
        for ele in test_url_list:
            str1 += ele
        print("str1:")
        print(str1)
        post.link_post = str1
        post.socials.remove(soc_del)
        db.session.commit()
        flash('Your post on facebook will be deleted', 'success')

        return redirect(url_for('home'))
    return redirect(url_for('home'))
```