ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА WEB-ПЛАТФОРМИ ДЛЯ УКРАЇНСЬКИХ КОНТЕНТ-МЕЙКЕРІВ МОВОЮ JAVASCRIPT»**

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Лисенко О.А.
(прізвище та ініціали)
Керівник Золотухіна О.А.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Напрям підготовки - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
О.В. Негоденко
“ ” 2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
Лисенку Олександру Анатолійовичу
(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка web-платформи для українських контент-мейкерів мовою JavaScript»
Керівник роботи к.т.н., доцент Золотухіна О.А.

(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від “ 24 ” лютого 2023 року №26

2. Строк подання студентом роботи “ 1 ” червня 2023 року

3. Вхідні дані до роботи:
- 3.1 Науково-технічна література, пов’язана з розробкою web-платформ.
- 3.2 Технічна документація бази даних MongoBD.
- 3.3 Технічна література до мови програмування Javascript.
- 3.4 Технічна документація засобу збору проєкта Gulp.

4. Зміст розрахунково-пояснювальної записки(перелік питань які потрібно розробити).
- 4.1 Аналіз задач які повинні виконуватися на розроблюваній web-платформі.
- 4.2 Аналіз та дослідження доступних аналогів.
- 4.3 Розробка в web-платформи для контент-мейкерів.
  - 4.3.1 Розробка серверної сторони.
  - 4.3.2 Розробка клієнтської сторони.
- 4.4 Проведення тестування web-платформи.
- 4.5 Підведення висновків.

5. Перелік демонстраційного матеріалу

5.1 Мета, об’єкт та предмет дослідження.
5.2 Задачі дипломної роботи.
5.3 Аналіз аналогів.
5.4 Вимоги до застосунку.
5.5 Програмні засоби реалізації.
5.6 Діаграма варіантів використання.
5.7 Діаграма класів.
5.8 Карта сайту.
5.9 Екранні форми.
5.10 Апробація результатів дослідження.
5.11 Висновки.

6. Дата видачі завдання “ 25 ” лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	27.02.23-05.03.23	Виконано
2	Аналіз та дослідження доступних аналогів	06.03.23-10.03.23	Виконано
3	Аналіз задач які повинні виконуватися на розроблюваній web-платформі	13.03.23-24.03.23	Виконано
4	Розробка в web-платформи для контент-мейкерів	27.03.23-30.04.23	Виконано
5	Тестування системи	01.05.23-05.05.23	Виконано
6	Вступ, реферат, висновки	06.05.23-13.05.23	Виконано
7	Розробка основних демонстраційних матеріалів	13.05.23-19.05.23	Виконано
8	Попередній захист роботи	25.05.23	Виконано
9	Здача роботи	01.06.23	Виконано

Студент _____ Лисенко О.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Золотухіна О.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 50 с., 1 табл., 46 рис., 2 дод., 13 джерел.

Об'єкт дослідження - формування рекомендацій постів українських контент-мейкерів для цільової аудиторії.

Предмет дослідження - програмне забезпечення для автоматизованого формування рекомендацій постів українських контент-мейкерів на основі характеристик цільової аудиторії.

Мета роботи - спрощення процесу формування рекомендацій постів українських контент-мейкерів для цільової аудиторії за рахунок використання web-платформи, розробленої мовою JavaScript.

В роботі було проведено аналіз задач, необхідних для виконання на web-платформі в результаті чого були отриматі функціональні та нефункціональні вимоги. Також було проведено аналіз наявних аналогів, визначені позитивні й негативні сторони кожного аналогу. Створена web-платформа для українських контент-мейкерів складається з клієнтської та серверної сторони та дозволяє виконувати: рееєстрацію та авторизацію; введення та зміну парамерів на яких базуються рекомендації постів; перегляд постів у стрічці та перегляд конкретного поста у розгорнутому вигляді; перегляд інформації про інших користувачів та їх постів; проходження верифікаційного тесту з яким пости користувача будуть мати найвищий пріоритет.

JAVASCRIPT, BCRYPTJS, EXPRESS, JSONWEBTOKEN, MONGOOSE, DOTENV, CORS, HTML, CSS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DOM-дерево – Document Object Model, представлення HTML-документа в структурі дерева з вузлами

ID - Identity Document

IDE - Integrated Development Environment.

JS – JavaScript

JWT – JSON Web Token

VS Code – Visual Studio Code

БД – база даних

ВСТУП

Актуальність створення web-платформи для українських контент-мейкерів в тому, що надання єдиної майданчика, де українські блогери зможуть розміщувати свій контент, а їх підписники або користувачі, зі схожими інтересами, будуть бачити ці пости, призведе до популяризації українських контент-мейкерів, що в свою чергу ослабить наявну проблему присутності російсьокого контенту.

В умовах стрімкого зростання популярності та аудиторії веб-застосунків створення власної платформи, з зрозумілими правилами та з аудиторію яка шукає гарний український продукт, є дуже важливим.

Мова програмування JavaScript вибрана через свою унікальну можливість – створення клієнтської та серверної сторони однією мовою програмування. Також важливи чинниками є популярність цієї мови, що в майбутньому полегшить пошук та залучення спеціалістів до проєкту з метою покращення його складових.

Створення WEB-платформи для українських контент-мейкерів мовою JavaScript є актуальним завданням, оскільки вона відповідає потребам та вимогам сучасного ринку контенту. Ця платформа полегшить пошук проукраїнської аудиторії для українських контент-мейкерів та надасть користувачам інформаційну сторону соцмереж, а не матиме за ціль затримання як найдовше користувача в системі.

Об'єкт дослідження - формування рекомендацій постів українських контент-мейкерів для цільової аудиторії.

Предмет дослідження - програмне забезпечення для автоматизованого формування рекомендацій постів українських контент-мейкерів на основі характеристик цільової аудиторії.

Мета роботи - спрощення процесу формування рекомендацій постів українських контент-мейкерів для цільової аудиторії за рахунок використання web-платформи, розробленої мовою JavaScript.

Для досягнення поставленоїмети в роботі вирішено наступні задачі:

1. Аналіз задач які повинні виконуватися на web-платформі.

2. Аналіз та дослідження доступних аналогів.
3. Розробка серверної сторони.
4. Розробка клієнтської сторони.
5. Проведення тестування web-платформи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Соціальні мережі та їх вплив на думки людей

В наші часи соціальні мережі являються одним з основних джерел комунікації для великої кількості активного населення. Політичний кандидат, служби новин, медійне лице чи звичайна людина, яка не має сторінки на одній з популярних платформ виглядає дуже дивно.

Цікаво що сам термін соціальна мережа з’явилася в 1954 році, запропонував його Джеймс Барнс. Тоді його значення було іншим, соціальною мережею називали структуру, яка містила групу зв'язків, між соціальними об’єктами (організації або люди) і вже в 1969 р. з появою інтернету, ця концепція отримала друге дихання. Згодом, в 1988 році був винайдений протокол «IRC» - ретрансльований інтернет-чат – що дозволило спілкуватися онлайн, але справжня популярність для соціальних мереж прийшла в 1995 р., коли створили Classmates.com – першу соціальну мережу в сучасному розумінні.

В Україні популярність соціальних мереж почала рости у 2010-х роках і вже в 2016 році за даними рекламного кабінету, Facebook нараховував 5,4 млн користувачів, а в березні 2019 р. в тому ж кабінеті була цифра 19 млн користувачів. Звісно, не всі користувачі являються активними, є дублювання акаунтів та й так званих «ботів» вистачає, але в будь-якому разі цифра гігантська. За підрахунками Global Logic платформи мають іншу кількість користувачів (рис. 1.1).

Рисунок 1.1 – Найпопулярніші соціальні мережі – дослідження Global Logic

Шалений ріст в 350% лише за 3 роки, звісно був обумовлений блокуванням російських мереж. Через велику популярність цього засобу зв'язку, ним зацікавилися не лише медійні персони, а й політичні групи росії, через які почали вводити регулювання інтернет-провайдерів, які мали аудіо, текстові та відео файли, тобто країна агресор мала доступ до всієї бази навіть приватних листувань. Новини, які відображалися на цих платформах піддавалися жорстокій критиці і маніпуляціям, тому «указом Президента України № 133/2017 від 15 травня 2017 р. про введення і дію рішення РНБО»[2] всі ці платформи були заблоковані. Ще якийсь короткий час ці платформи зберігали свою монополію, але вже в червні, Facebook обігнав VK (рис. 1.2).

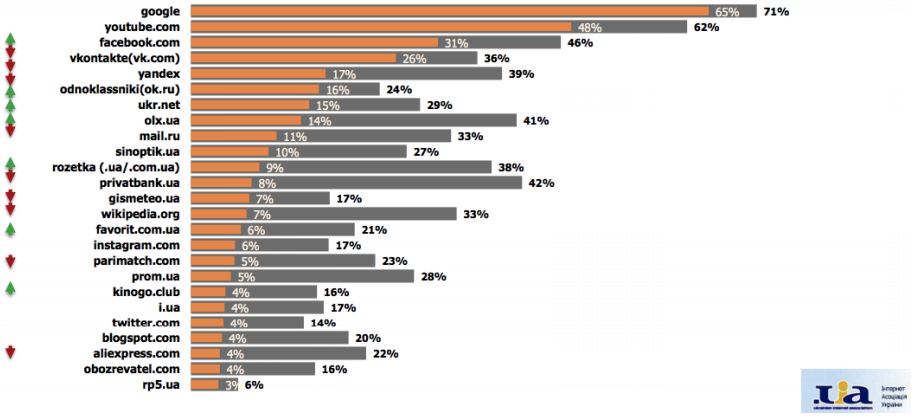


Рисунок 1.2 – Дослідження ранжування за середньодобовою нормою, проведене організацією Factum Group Ukraine

Втрата впливу призвела перенесення інформаційної війни на поле нових платформ, де фільтрування спаму та брехні проводилося краще, але не ідеально. На жаль зміна платформ майже не змінила спікерів, і більшість контент-мейкерів була з інших країн, на фоні їх популярності український споживач не бачив вітчизняних креаторів.

1.2 Створення платформи для українців

Україна, як суверенна країна, потребує власних соціальних мереж, які б

відповідали потребам українських користувачів та сприяли розвитку української спільноти в інтернеті, тому на основі переліченої інформації, було прийнято рішення про створення платформи, основні задачі якої:

- фільтрація при реєстрації, аудиторія має бути українською або проукраїнською;
- користувачу на платформі мають відображатися пости по категоріях, які він вибрав;
- українські контент-мейкери мають бути вище у стрічці рекомендацій;

Для досягнення успішного розв'язання поставлених задач, планується створити:

- тест, який потрібно буде пройти перед реєстрацію акаунту на платформі, він міститиме питання, відповісти на які можуть проукраїнськи мотивовані користувачі;
- при реєстрації, або при редагуванні інформації в профілі можна буде змінити категорії, які цікавлять;
- українські контент-мейкери будуть помічатися «прапорцем» і ця мітка буде мати найбільшу вагу при видачі стрічки користувачу.

1.3 Аналіз та дослідження доступних аналогів

Засоби для створення та поширення нових постів, присутні в цьому переліку платформ, включають такі соціальні мережі:

- Twitter;
- Instagram;
- Mastodon.

1.3.1 Twitter

Twitter - соцмережа, де можна обмінюватися короткими повідомленнями, також відомими як "твіти". Процес створення твіту один з найбільш мінімалістичних на популярних платформах (рис. 1.3).

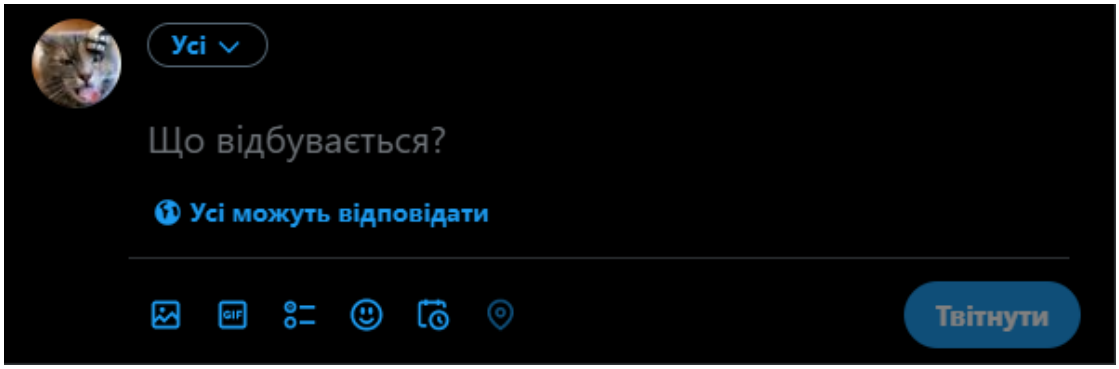


Рисунок 1.3 – Поле введення нового посту в Twitter

На головній сторінці Twitter є поле для введення повідомлення, в якому користувач може написати свій твіт. Максимальна довжина твіту до 280 символів, що дозволяє користувачам швидко поділитися своєю емоцією. Є можливість прикріплення медіа, геолокації, створення опитування та вибору emodji. Також є обмеження аудиторії на ту що може побачити твіт, і на ту що може відповідати на нього.

Після введення твіту, користувач може опублікувати його, натиснувши кнопку "Tweet". Можна також додати хештеги до свого твіту, щоб інші користувачі могли знайти його за певною темою.

Після публікації твіту він з'являється у стрічці новин (рис. 1.4) та може бути переглянутий іншими користувачами, які підписані на нього. Користувачі можуть також додавати коментарі до твіту, які відображаються під оригінальним постом.

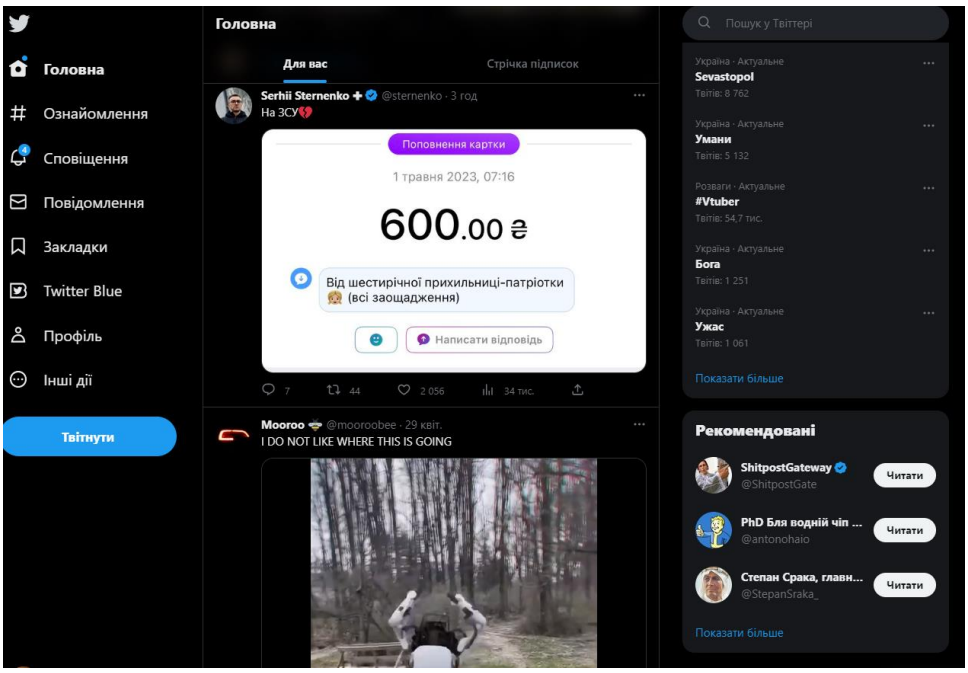


Рисунок 1.4 – Стрічка користувача з розміщеними постами в Twitter

Крім того, на головній сторінці є «Тренди для вас», де відображаються теги, які схожі на інтереси користувача. Ця функція дозволяє користувачам бути в курсі останніх подій та трендів у світі.

Переваги Twitter:

- простота в застосуванні, через зрозумілу форму введення та коротку форму передачі думки;
- можливість обмеження постів під різний тип аудиторії(всі, обрані);
- різні типи взаємодії з аудиторією: опитування, розміщення геолокації, фото.

Недоліки:

- передавати довгі статті доволі не зручно через обмеження в 280 символів;
- рекомендації просувають краще верифікованих користувачів (тих хто має підписку);
- розповсюдження фейків не блокується і захищається свободою слова.

1.3.2 Instagram

Instagram – соціальна мережа, в якій основний зміст постів - візуальний. Користувачі Instagram можуть ділитися фотографіями та відео, додавати фільтри та редагувати свої матеріали, а також переглядати та коментувати зміст інших користувачів.

Введення посту на Instagram є більш складним, ніж це реалізовано в Twitter, але й функціонал тут ширше. Щоб опублікувати новий пост, користувач має натиснути на значок "+" на головній сторінці Instagram. Потім він повинен вибрати фотографію або відео зі свого пристрою, це є обов'язковою умовою – пости без фото або відео в Instagram недоступні. Одразу після завантаження фото можна редагувати його: різними фільтрами або більш професійними налаштуваннями у вкладці корегування (рис 1.5).

Після того, як користувач закінчив редагування свого посту, він може додати опис до своєї фотографії або відео. Опис може бути достатньо довгим оскільки тут обмеження на довжину поста 2200 символів.

Також можна позначати об'єкти на фото через відмічення акаунтів з Instagram, це може бути якась компанія або інший користувач, який дозволив цю дію. Звісно, можна відмічати геолокація конкретного місця або ж країни, яка зв'язана з фото або самим постом. Для людей з порушенням зору можна описати текстом, щоб передати основну ідею фото. Ще є розширенні налаштування, в них можна вимкнути коментарі та показ лайків. Крім того, користувач може додати хештеги до свого опису, щоб інші користувачі мали змогу знайти його пост за певними темами(Рис 1.6).

Після того, як користувач опублікував свій пост, він з'являється у стрічці новин людей, які підписані на нього, вони можуть лайкати та коментувати цей пост.

Щодо відображення посту, Instagram має стрічку новин, в якій користувач може переглядати пости своїх друзів та інших користувачів, на яких він підписаний. При натисканні на пост, він відкривається в повноекранному режимі, де користувач може переглянути деталі та коментарі до посту.

Крім того, Instagram має функцію "Explore", яка дозволяє користувачам

переглядати пости, які вони ще не побачили.

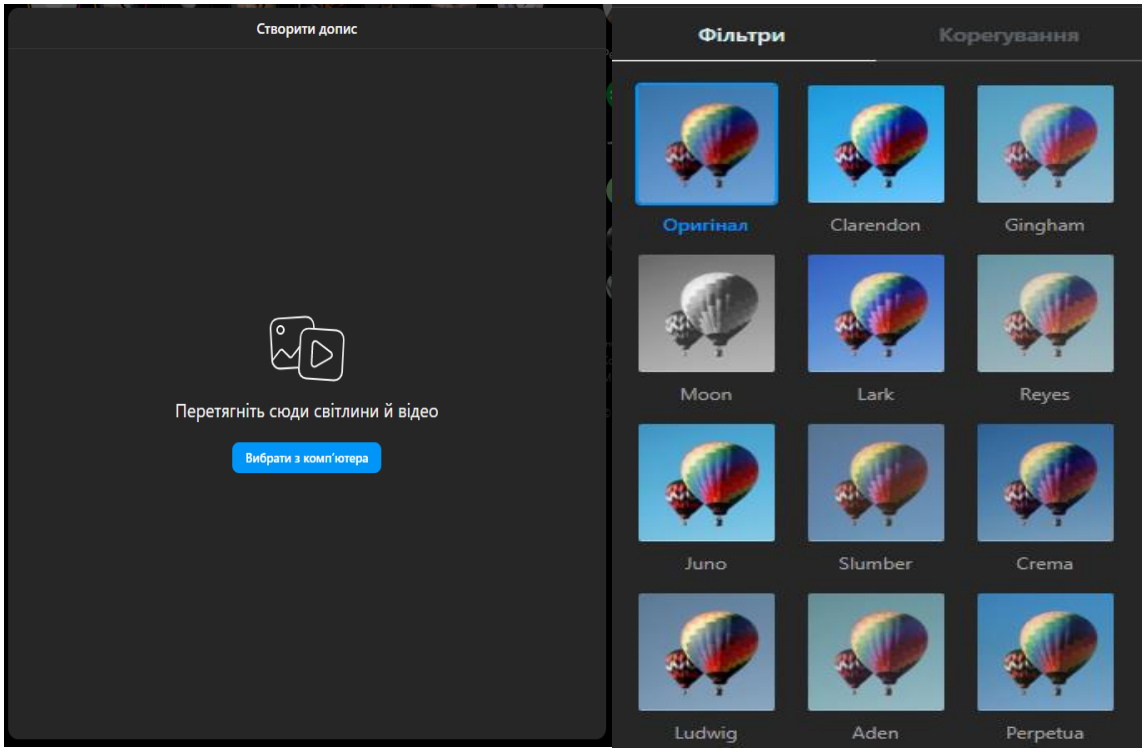


Рисунок 1.5 – Екрани завантаження фото та вибору фільтра в Instagram

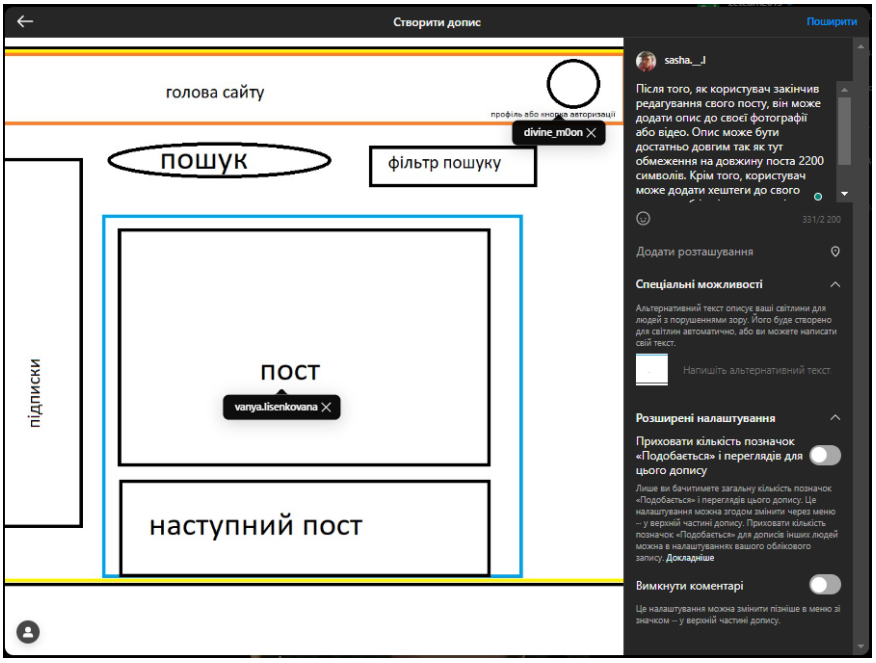


Рисунок 1.6 – Введення текстової частини посту відмічення користувачів та інші налаштування в Instagram

Переваги Instagram:

- можливість розміщення значно довших постів;
- пости вважаються більш серйознішими, бо створити гарне фото і цікавий текст до нього - важка задача;
- профіль візитна карта будь-якого активного користувача.

Недоліки:

- фото обов’язкова частина посту, що часто відвертає людей які підготували текстову частину, а фото ні;
- великий об’єм контенту, який забирає велику частину часу користувача;
- функціонально обмежена версія веб застосунку для використання на комп'ютері.

1.3.3 Mastodon

Mastodon - це соціальна мережа, яка працює на основі відкритого програмного забезпечення (Open Source Software). Вона була створена у 2016 році, з основною метою - створення децентралізованої альтернативи Twitter.

Однією з ключових відмінностей Mastodon від інших соціальних мереж є те, що вона працює на основі принципу децентралізації. Це означає, що мережа складається з багатьох серверів, кожен з яких може мати свої правила, налаштування та спільноти користувачів. Кожен користувач може обрати сервер, який найкраще відповідає його потребам та інтересам, і під'єднатися до мережі.

При реєстрації можна вибрати сервер, де правила і спільнота тебе влаштовує, але це не означає що правила можуть бути які завгодно, є перелік базових принципів необхідних для дотримання (рис 1.7).

Форма введення постів у Mastodon нагадує формат Twitter. Є вікно для вводу тексту посту (рис 1.8) з обмеженням в 500 символів, що є чимось середнім між Instagram і Twitter. Є також можливості: додавання опитування, обмеження відображення посту для різної аудиторії, вибору мови для посту та прикріплення тегів, але і є незвичне налаштування «попередження про вміст». Його часто використовують, якщо в пості є згадування політичних питань, обговорення віри,

показ контенту з обмеженням по віку, спойлерів до фільму і так далі.

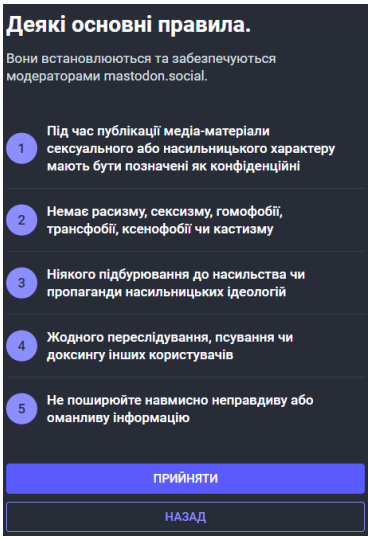


Рисунок 1.7 - Основні правила перед реєстрацією в Mastodon

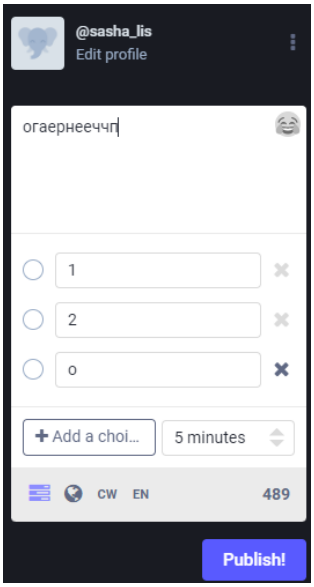


Рисунок 1.8 – Вікно введення нового посту в Mastodon

Після введення, пост з'являється у хронологічній стрічці новин підписників користувача (рис. 1.9), бо Mastodon не має власних рекомендацій, можуть бути певні відмінності на окремих серверах, але як правило їх немає. Mastodon не ставить за ціль забрати найбільше часу в користувача, тому рекомендації не є необхідністю, що треба користувач знайде сам.

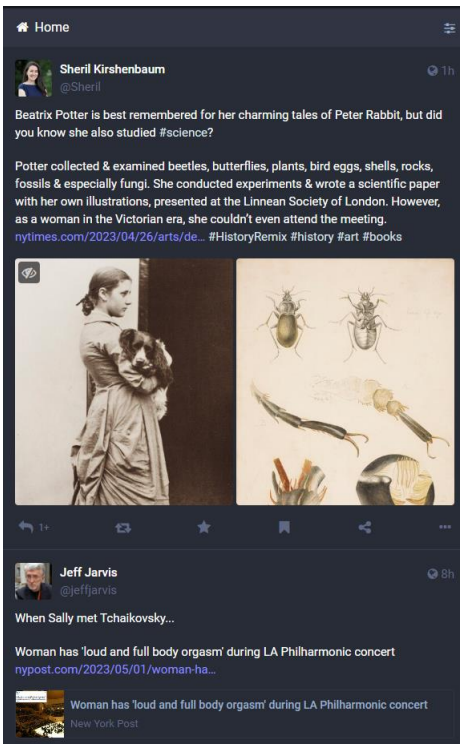


Рисунок 1.9 – Стрічка постів в Mastodon

Головними правилами відображення постів є:

- підписка на користувача який викладає пост;
- час викладення посту (відображення від новіших до старіших).

Переваги Mastodon:

- децентралізація, яка дає можливість кому завгодно створити свій сервер з унікальними правилами та модерацією;
- сервіс працює від пожертвувань, тому в ньому немає реклами від платформи;
- обмеження видимості посту на уразливий контент, спеціальним налаштуванням під час створення посту.

Недоліки:

- через відсутність бізнес-моделі, розвиток сервісу йде доволі повільно;
- невелика кількість символів для публікування постів що мають довгі роздуми або є науковими.

1.3.4 Результати аналізу аналогів

На основі підготовлених даних була створена порівняльна таблиця, де представленні основні особливості вебплатформ (табл. 1.1).

Таблиця 1.1 – Зведені результати аналізу характеристик вебзастосунків для створення та розповсюдження постів

Особливість	Instagram	Twitter	Mastodon
Вільна ліцензія на контент	-	-	+
Основний контент	Фото/відео	Текст	Текст
Довжина посту	2200 символів	280 символів	500 символів
Ефекти/фільтри	+	-	-
Хештеги	+	+	+
Безкоштовна верифікація	+/-	-	+
Домінація підписок над рекомендаціями	+	-	+
Приватність акаунту	+	+	+
Відсутність рекламаи	-	-	+
Функція обмеження контенту	Тільки через модерацію	Тільки через модерацію	+
Підтримка файлів GIF	-	+	+
Фільтрація аудиторії на проукраїнськість	-	-	-

Отже, майбутній веб-застосунок має мати в собі такі особливості:

- основний контент – текст;
- вільна ліцензія на контент;
- довжину посту – 500 символів;

- наявність хештегів;
- можливість безкоштовної верифікації акаунта;
- алгоритми рекомендацій засновані на підписках на користувачів та вибраних тегах;
- підтримку gif-файлів;
- фільтрація аудиторії на проукраїнськість;

.

2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Вибір середовища розробки

2.1.1 Visual Studio Code

VS Code, повністю відомий як Visual Studio Code, є одним з найпопулярніших та потужних середовищ розробки (IDE) на сьогодні (рис. 2.1). Він надає розробникам широкий набір функцій та інструментів, що сприяють зручному та продуктивному процесу написання коду.

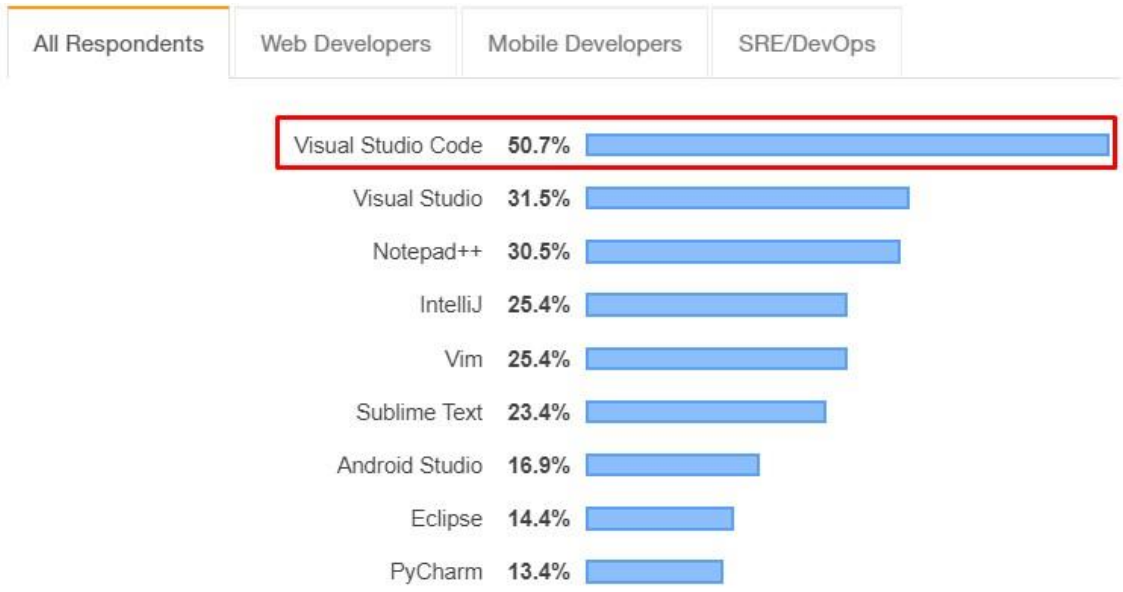


Рисунок 2.1 – Топ популярних IDE серед всіх опитаних на StackOverflow 2019 р.

VS Code розроблений компанією Microsoft та є Open Source проєктом та використовує ліцензію MIT, яка дозволяє використовувати, модифікувати та поширювати код. Він є кросплатформеним, що означає, що ви можете використовувати його на операційних системах Windows, macOS і Linux. Ця універсальність робить його популярним серед розробників з різних фахових напрямків та платформ.

Одна з ключових переваг VS Code полягає в його легкості та швидкодії. Він має мінімалістичний інтерфейс, який дозволяє зосередитись на кодуванні без

зайвих відволікань. Крім того, він відзначається високою продуктивністю завдяки вбудованому асинхронному виконанню операцій та підтримці багатопоточності. Це дозволяє швидко відгукатися на дії розробника та забезпечувати плавне редагування та перегляд коду навіть для великих проєктів.

Крім базового набору функцій, VS Code дозволяє розширювати свої можливості за допомогою розширень, які завантажуються через саму IDE в спеціальній вкладки «EXTENSIONS» (рис. 2.2). Завдяки активній спільноті розробників, ви зможете знайти розширення для практично будь-якої потреби, починаючи від розширеної підсвітки синтаксису та автодоповнення коду до інтеграції з різними фреймворками та інструментами. Це дозволяє налаштувати VS Code під ваші індивідуальні потреби та стиль розробки.

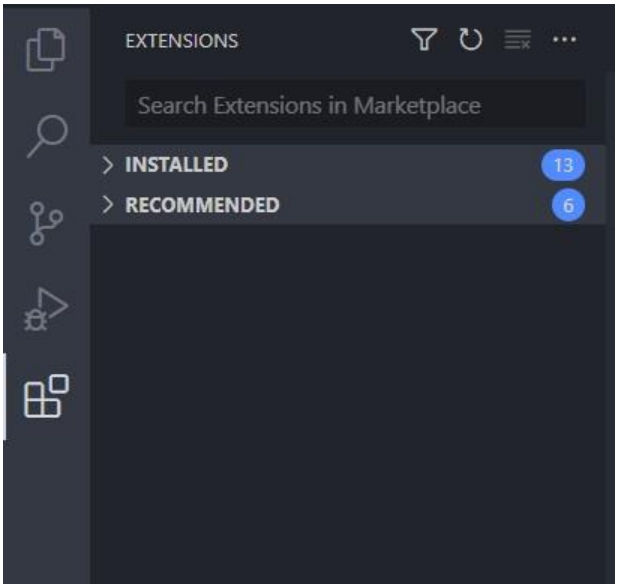


Рисунок 2.2 – Вкладка EXTENSONS в VS Code, з завантаженими розширеннями, рекомендаціями та магазином розширень.

Через прекрасне співвідношення ресурсів яких потребують й отриманої зручності, яка регулюється через кількість завантажених розширень, Visual Studio Code так цінується і серед програмістів з поганим «залізом», і серед програмістів які готові робити власну збірку розширень під свій проєкт.

2.1.2 Розширення для VS Code

VS Code має можливість модифікування, а також доповнення власного функціонала через використання різних розширень. В даному проєкті були необхідними (рис. 2.3):

- Auto close tag. Розширення допомагає швидше писати HTML-код через автоматично закриття тегів. При введенні тега який відкривається (наприклад , <p>, тощо), плагін автоматично додає тег який закривається з відповідною розстановкою курсора, що дозволяє ефективно працювати з розміткою та запобігає помилкам в синтаксисі.
- Auto rename tag. Допомагає в автоматичній заміни назви в парних тегах. Коли користувач змінює назву тега який відкривається, плагін автоматично оновлює відповідний тег який закривається, що допомагає уникнути невідповідностей у структурі HTML-коду та покращує зручність редагування.
- eCSStractor. Полегшує написання CSS-коду, через копіювання виділеного HTML-коду в якому виконує пошук вже введених назв класів для різних елементів та зберігає їх в буфері обміну. Замість того щоб змінювати 2 типи файлів, можна раз скопіювати і використовувати ці класи.
- htmltagwrap. Розширення htmltagwrap дозволяє зручно обгортати вибраний фрагмент HTML-коду в теги-обгортки. Це робить процес редагування та модифікації HTML-структури більш зручним та швидким, оскільки не потрібно вручну додавати обгортки навколо вибраного коду.

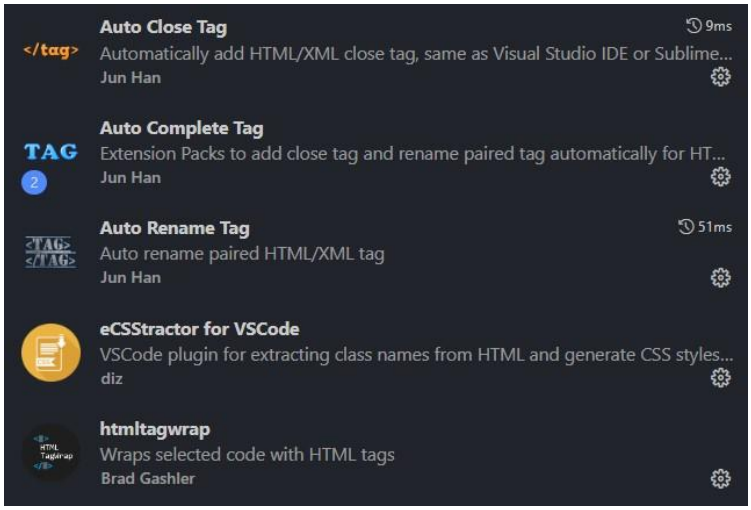


Рисунок 2.3 – Розширення VS Code використані в даному проєкті

2.2 Виріб засобів розробки клієнтської частини

2.2.1 JavaScript

JavaScript є однією з найпоширеніших мов програмування, яка використовується для розробки клієнтської сторони веб-додатків (рис 2.4). Ця мова програмування є динамічною та об'єктно-орієнтованою, це надає розробникам широкі можливості для створення функціональних інтерфейсів та взаємодії з користувачем.

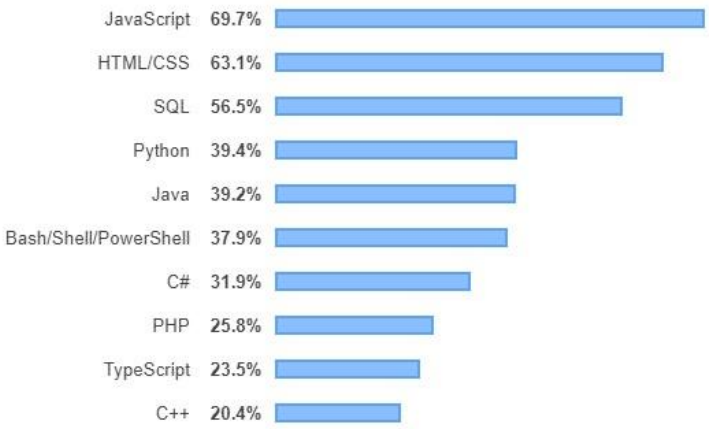


Рисунок 2.4 – Рейтинг популярності мови програмування серед опитаних на StackOverflow 2020 р.

У контексті клієнтської сторони, JavaScript використовується для динамічного створення та зміни елементів веб-сторінки. Також з його допомогою можна реагувати на дії користувача, обробляти події, валідувати дані та забезпечувати інтерактивність веб-додатків.

JavaScript має велику кількість вбудованих функцій і методів, що спрощують роботу з маніпуляцією DOM-дерева (рис 2.5), обробкою форм, валідацією даних, анімацією, взаємодією з сервером та іншими завданнями. Також існують багато сторонніх бібліотек і фреймворків, які розширюють можливості JavaScript та допомагають в розробці більш складних проєктів.

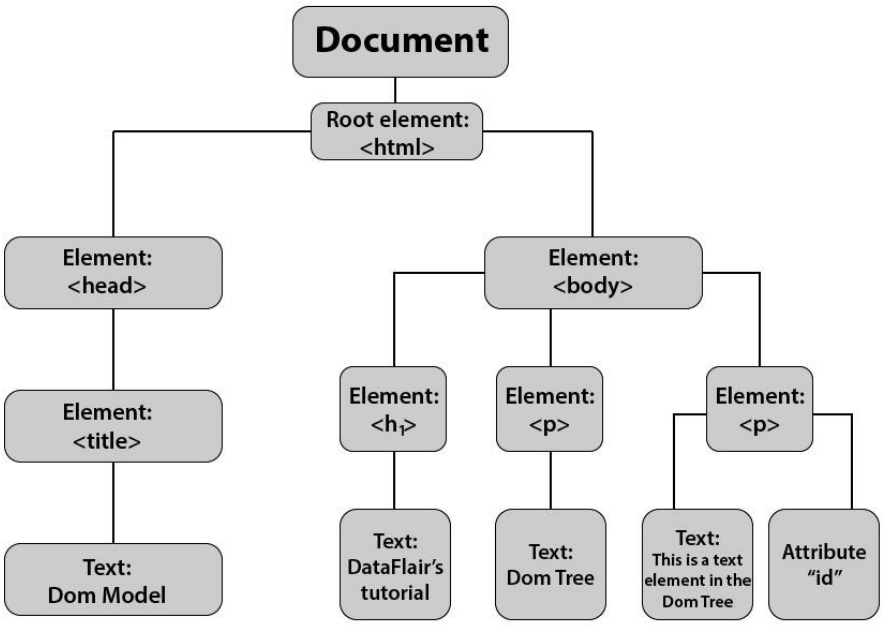


Рисунок 2.5 – Структура DOM-дерево

У даному проєкті JavaScript використовується для створення динамічного інтерфейсу, взаємодії з користувачем, обробки подій на клієнтській стороні, надсиланням запитів та роботою з відповідями серверної сторони. Використання JavaScript разом з іншими веб-технологіями, такими як HTML та CSS, дозволяє створювати привабливі, функціональні та інтерактивні веб-додатки.

2.2.2 HTML та CSS

HTML і CSS є основними мовами, використовуваними для розробки клієнтської сторони веб-додатків.

HTML використовується для створення структури та контенту веб-сторінок. Він визначає базові елементи сторінки заголовки, параграфи, таблиці, зображення та посилання. За допомогою HTML можна створювати різні типи вмісту і встановлювати їх ієрархічність та взаємозв'язок.

CSS використовується для стилізації та вигляду веб-сторінок. Він визначає зовнішній вигляд елементів, таких як кольори, шрифти, розташування, розміри та анімацію. CSS дозволяє відокремлювати вигляд веб-сторінки від її структури, що робить код більш організованим і зручним для редагування та підтримки.

У даному проєкті HTML та CSS використовуються для створення структури та оформлення декількох веб-сторінок, що відображаються на клієнтському браузері.

2.2.3 Gulp та інші модулі для Node.js

Gulp є одним з найпопулярніших інструментів для автоматизації рутинних задач у розробці веб-додатків. Він працює на основі Node.js і використовується для автоматизації збірки проєкту. За допомогою нього можна виконати компіляцію, об’єднання, мінімізацію файлів коду та зображень, а також відображати в браузері зміни в коді. За допомогою Gulp можна значно збільшити продуктивність розробки та знизити час, необхідний для виконання повсякденних завдань.

Основним модулем для розробки клієнтської сторони є browser-sync. Це інструмент, який надає можливість відображення тих самих змін на веб-сторінках без необхідності вручну оновлювати сторінку. Browser-sync автоматично оновлює браузер, коли відбуваються зміни в коді, що дозволяє розробникам бачити результати змін миттєво, без ручного запуску готової сторінки.

Одним з найзручніших модулів для полегшення створення HTML-сторінок є gulp-file-include. Він дозволяє розділити HTML-сторінку на базові повторюванні елементи, які наявні на всіх сторінках проєкту та на унікальну частину, яка з’являється тільки на конкретній сторінці. Це схоже на функцію куди виноситься повторюваний код і викликається в тих частинах проєкту де це потрібно.

В файлі з унікальною частиною виконується виклик файлів в яких зберігаються шматки повторюваного коду та при потребі вносяться необхідні параметри для даної сторінки, такі як title (рис. 2.6).

```
<!DOCTYPE html>
<html lang="en">
    @@include('html/head.html',{
        "title":"Авторизація"
    })
    <body>
        @@include('html/header.html',{})
        <main>···
        </main>
        @@include('html/footer.html',{})
        <script type="module" src="./js/tasks/login.js"></script>
    </body>
</html>
```

Рисунок 2.6 – Файл до виконання збірки проєкту через Gulp

Вже після виклику Gulp проходить збір проєкту і вставлення потрібних шматків коду на сторінку (рис 2.7).

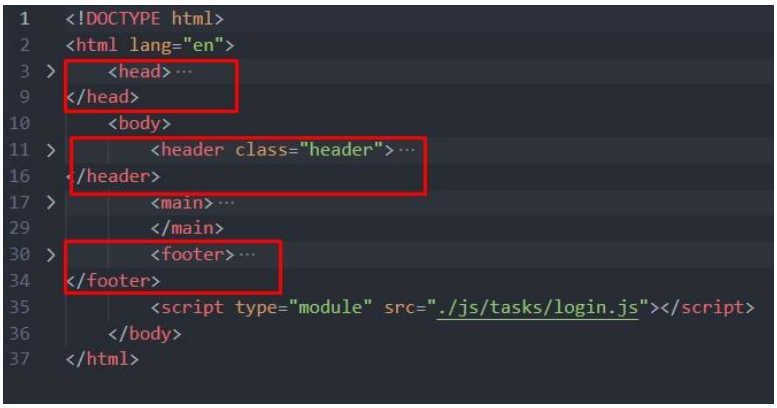


Рисунок 2.7 – Файл після виконання збірки проєкту через Gulp

З використанням Gulp та інших модулів для Node.js розробники можуть ефективно автоматизувати процеси розробки, зменшити кількість рутинних завдань та покращити продуктивність.

2.3 Серверна сторона

2.3.1 Node.js

Node.js є платформою, яка дозволяє виконувати JavaScript-код на серверній частині. Технологія базується на двигуні V8 (рис. 2.8), розробленому компанією Google для браузера Chrome. Однак, в контексті серверної розробки, Node.js забезпечує можливість виконання JavaScript на сервері з великою продуктивністю і швидкодією.

Ключовою перевагою Node.js є те, що він дозволяє розробникам використовувати одну мову програмування - JavaScript - як на клієнтській, так і на серверній стороні. Це дає змогу не вчити нову технологію з нуля, а просто ознайомитися з меншою кількістю розбіжностей в використанні JS, як мову для написання серверної сторони.

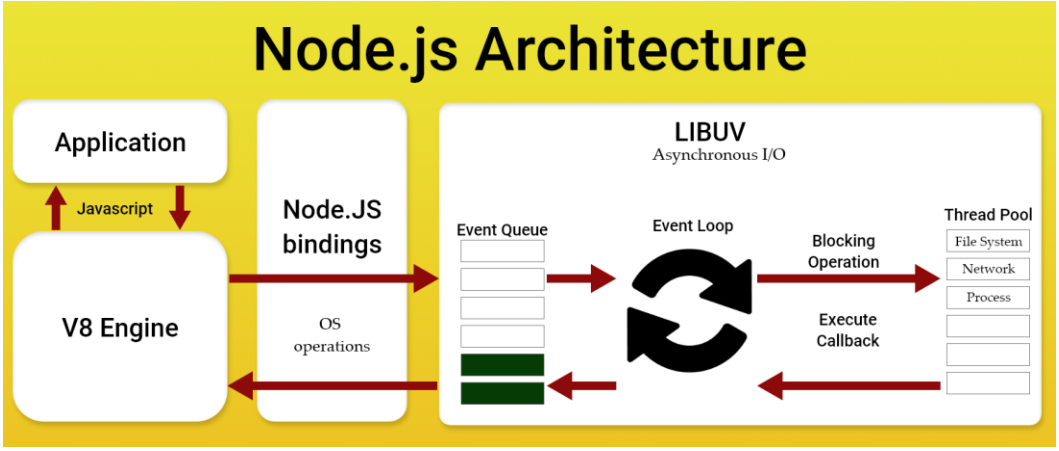


Рисунок 2.8 – Архітектура Node.js з використанням V8

Node.js також відомий своєю архітектурою яка не блокується, що дозволяє обробляти багато запитів одночасно без блокування інших операцій. Це дозволяє побудувати масштабовані та ефективні серверні додатки, які можуть витримувати великі навантаження.

Крім того, Node.js має велику кількість пакетів та модулів, доступних через платформу Node Package Manage. Це дозволяє розробникам використовувати готові рішення для різних задач, таких як маршрутизація, обробка запитів HTTP, робота з базами даних та багато іншого.

2.3.2 Express.js та інші модулі

Express.js один з найпопулярніших фреймворків для побудови серверних додатків з використанням Node.js. Він надає простий, короткий та зрозумілий спосіб створення веб-серверів та обробки маршрутів. По кожному маршруту можна виконувати свої обробники запитів (handlers), такі як: внесення інформації в БД, відправка даних про акаунт та оновлення даних.

Крім цього, є можливість використовувати проміжні шари (middleware), які повинні бути виконані до виконання самого обробника. Наприклад треба перевірити чи авторизований користувач до внесення інформацію про пост.

Nodemon інструмент схожий на browser-sync, але він перезапускає серверний застосунок при зміні файлів в проєкті. Це дозволяє розробникам зосередитися на

написанні коду без необхідності кожного разу ручно перезапускати сервер.

Mongoose є об'єктно-документною бібліотекою моделювання для бази даних MongoDB. Вона надає зручний інтерфейс для роботи з базою даних MongoDB, дозволяючи визначати схеми, виконувати запити та взаємодіяти з колекціями даних. Mongoose спрощує роботу з базою даних та забезпечує більш зрозуміле та структуроване програмування.

Важливим етапом при створені акаунту на будь-якій платформі є шифрування паролю, щоб навіть при викрадені даних користувача, зловмисники не мали доступу до акаунту. Bcrypt.js є цим модулем для Node.js, який дозволяє хешувати паролі. Він надає безпечний спосіб зберігання паролів у базі даних, шифруючи їх перед збереженням. Bcrypt.js використовує сильний хеш-алгоритм для забезпечення безпеки паролів та запобігання їх небажаному доступу.

Jsonwebtoken є модулем для створення та перевірки JSON Web Tokens (JWT) в Node.js. JWT є механізмом автентифікації та передачі даних, що заснований на JSON форматі. Використовуючи Jsonwebtoken, розробники можуть створювати та перевіряти JWT для забезпечення безпеки та автентифікації в своїх серверних додатках.

Ці та інші модулі разом з Express.js допомагають розробникам створювати потужні та безпечні серверні додатки. Вони надають готові рішення для роботи з базою даних, хешування паролів та автентифікації, що спрощує розробку та забезпечує надійність серверних додатків.

2.3.3 MongoDB

MongoDB є документоорієнтованою NoSQL системою управління базами даних (рис. 2.11). В прикладі можна побачити назву бази даних “dut-diploma” та документи в яких розташовані дані в залежності від назви колекції.

Вона забезпечує гнучкість та швидкодію для зберігання та обробки великих обсягів даних. MongoDB використовує JSON-подібну структуру даних (рис. 2.12), що дозволяє зручно зберігати та маніпулювати документами.

Рисунок 2.11 – Взаємодія з базою даних MongoDB

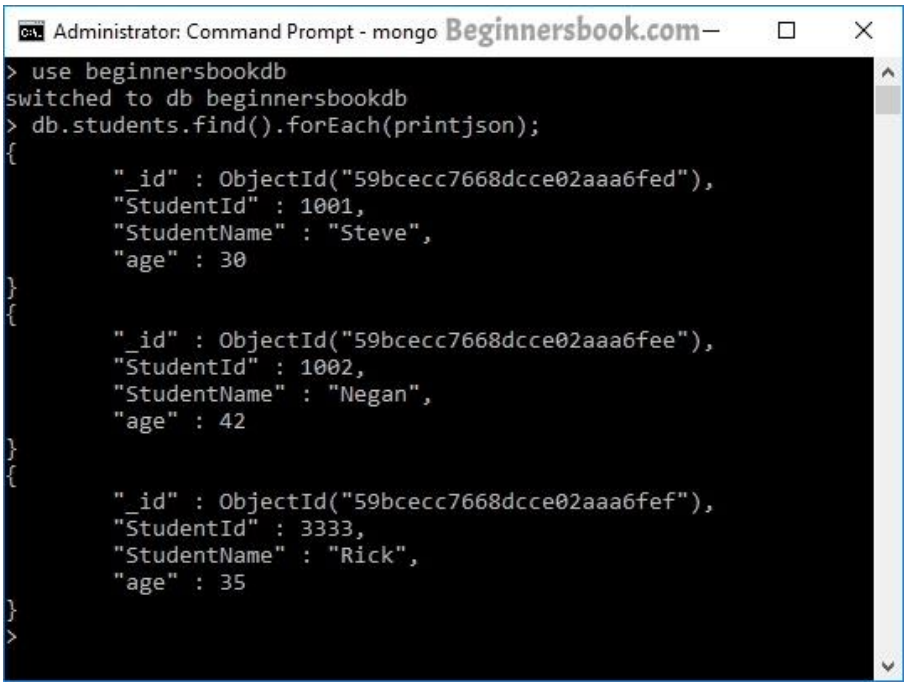


Рисунок 2.12 – JSON-структура даних в MongoDB

Це також має плюс для тих працював з об’єктами в Javascript і не має вивчати структуру роботи базою даних. Вона також підтримує горизонтальне масштабування, що дозволяє розподіляти навантаження між серверами та підвищувати продуктивність системи. MongoDB використовується в багатьох сучасних веб-додатках для зберігання та обробки даних з високою швидкодією та масштабованістю.

.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Розробка архітектури системи

Клієнтська сторона:

– відповідає за відображення інтерфейсу користувача та в залежності від того чи пройшла авторизація чи ні, має відображати елементи доступні користувачу;

– включає різні сторінки, такі як: сторінка входу в акаунт, сторінка реєстрація акаунту, сторінка перегляду постів, сторінка перегляду конкретного посту, сторінка користувача та сторінка проходження тесту;

– включає в себе відображення таких компонентів: кнопка перегляду власного акаунту, кнопка виходу з нього, кнопка редагування даних акаунту, кнопка підписки/відписки на акаунт. при виконанні різних умов;

– обмежує пересування користувача по сторінкам сайту, якщо користувач не авторизований;

– виконує валідацію даних перед відправкою на сервер;

– взаємодіє з серверною стороною шляхом відправки запитів та відображає дані отримані від нього.

Серверна сторона:

– обробляє запити від клієнтської частини та надає відповіді про успішне виконання запиту або видає помилки які виникли при виконанні;

– взаємодіє з базою даних, створює підключення, додає дані та оновлює їх;

– зберігає фото постів, якщо вони були надіслані, та змінює їх назву задля захисту від дублікатів.

База даних:

– зберігає дані про користувача та пости;

– перевіряє дані доступу та забезпечує доступ до даних серверній частині;

– виконує запити на отримання, додавання та зміну наявних даних у базі.

3.2 Розробка модулів авторизації та реєстрації

Процес авторизації логічно розбивається на 3 етапи:

- реєстрація нового користувача;
- вхід в акаунт користувача який пройшов реєстрацію;
- перевірка авторизаційних даних при переміщенні по сторінках.

Реєстрація на стороні клієнта відбувається введення даних таких як username, пошта, пароль та проставлення прапорців на питаннях (рис. 3.1).

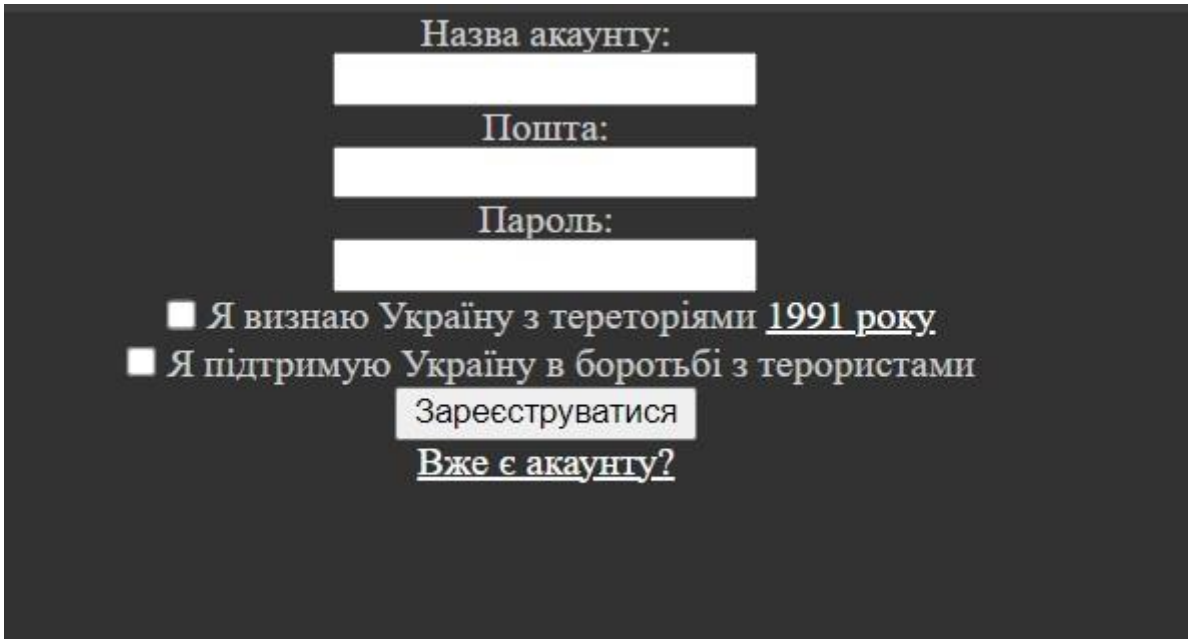


Рисунок 3.1 – Форма реєстрації

При введенні всіх даних користувач відправляє дані на сервер і вже тут відбувається робота з ними:

- відбувається перевірка чи вільний цей username, так як він має бути унікальним для кожного акаунту;
- відбувається робота з паролем користувача, за допомогою описаного bcryptjs відбувається процес шифрування і в результаті отримується довге значення;
- дані про користувача вносяться у базу даних;

- відбуваться створення авторизаційного токена через застосування секретного ключа, який діятиме 30 днів, в ньому зберігається id акаунту.

- відправляється відповідь на клієнтську частину з даними нового акаунту, авторизаційним токеном.

На стороні клієнта зберігається в localStorage значення токена (рис. 3.2) та відбувається перенесення на сторінку з постами.

Рисунок 3.2 – Збереження токена в localStorage

Вхід в акаунт відбувається на стороні клієнта: вводяться дані username та паролю(рис. 3.3).

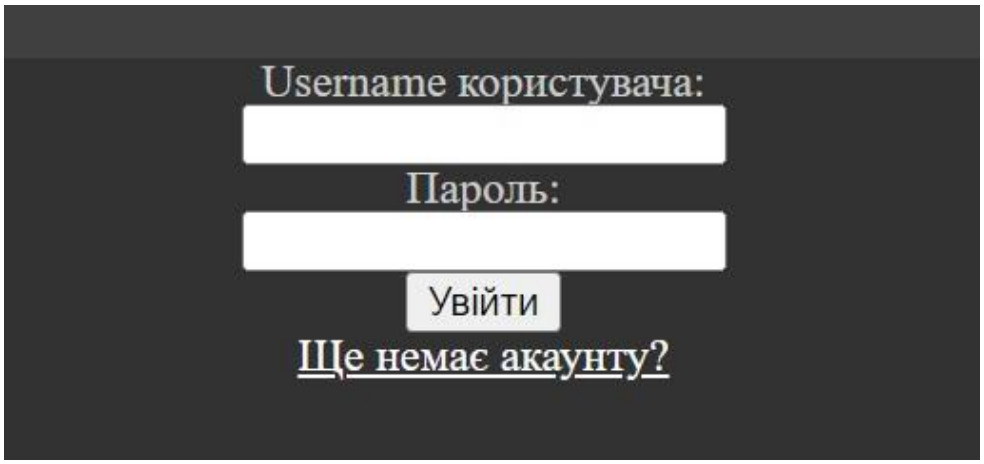


Рисунок 3.3 – Форма входу в акаунт

На сервері:

- відбувається запит в базу даних, де відбувається пошук користувача з таким username, якщо його немає відправляється повідомлення: “username не вірний”;

- потім відбувається перевірка паролю через функцію модуля bcryptjs якщо пароль не вірний, відправляється відповідь: “Пароль не вірний”;

- відбувається створення авторизаційного токена;
- створюється відповідь з даними акаунту та авторизаційним токеном.

На стороні клієнта зберігається в localStorage значення токена та відбувається перенесення на сторінку з постами.

3.3 Перевірка авторизаційних даних

При переміщенню по сторінкам сайту чи при оновленні сторінки, задля збереженні доступу до акаунту, потрібно перевіряти чи авторизований користувач, тому на клієнтській стороні відбувається запит до серверної частини з вкладеним значенням авторизаційного токена. На стороні сервера:

- відбувається верифікація токена за допомогою того ж ключа який використовувався при його створенні;
- з верифікованого значення виносять id-акаунту;
- йде запит в базу даних про наявність такого акаунту і повертається відповідь з помилкою при його відсутності;
- створюється нове значення авторизаційного токену;
- відправляється відповідь з даними акаунту та токеном;

При успішній перевірці дані токена оновлюється, а при отримані будь-якої помилки відкривається сторінка входу.

3.4 Розробка сторінки постів

Сторінка постів розбита на 2 частини:

- створення нового посту у верхній частині сторінки;
- відображенням постів розташованим нижче.

На клієнтській стороні відображається форма введення нового посту з обов’язковими значеннями тексту посту та тегів посту, також можна завантажити фото посту (рис 3.4).

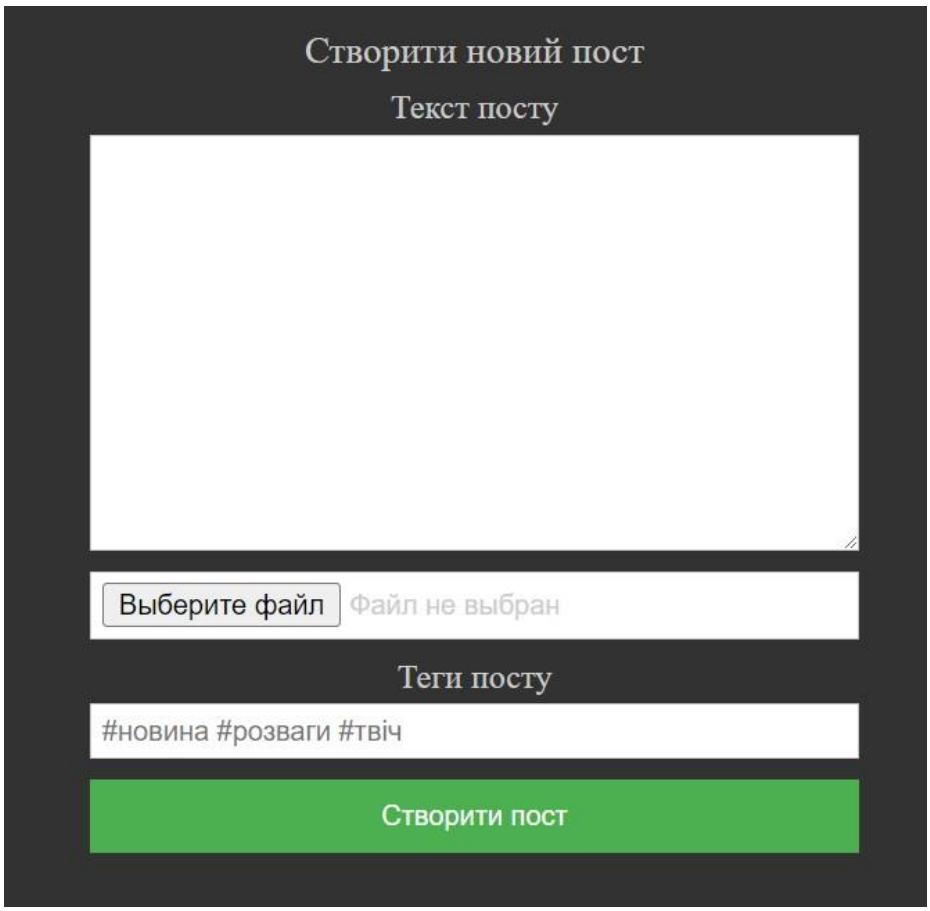


Рисунок 3.4 – Форма створення нового посту

Після підтвердження виконується запит на серверну частину з значеннями форми та токеном авторизації, та відбуваються такі дії:

- з токена виділяється значення id-акаунту;
- відбувається запит до бази даних по знаходженню користувача по такому id, відповідь записується в змінну;
- при наявності фотографії відбувається змінна назви та збереження фото на сервері, а назва фото з даними отриманими з форми зберігається в базі даних;
- якщо фото відсутнє, то базі даних зберігаються дані форми без значення назви фото.

Відбувається оновлення сторінки і очищення полів форми.

Відображення постів відбувається на клієнтській частині: відбувається запит на отримання постів. Серверна частина:

- виділяє з токена id-акаунту;

- відбувається запит в базу даних з пошуком користувача по отриманому id, відповідь зберігається;

- відправляється запит в БД з пошуком постів в яких значення автора або тегів співпадає зі списком підписок або списком тегів користувача, потім відбувається фільтрація по давності;

- відповідь зі знайденими постами відправляється на сторону клієнта.

Клієнтська сторона фільтрує список по тому чи верифікований клієнт чи ні. Потім перебирає дані з внесенням їх у шаблон та відображає їх (рис. 3.5).

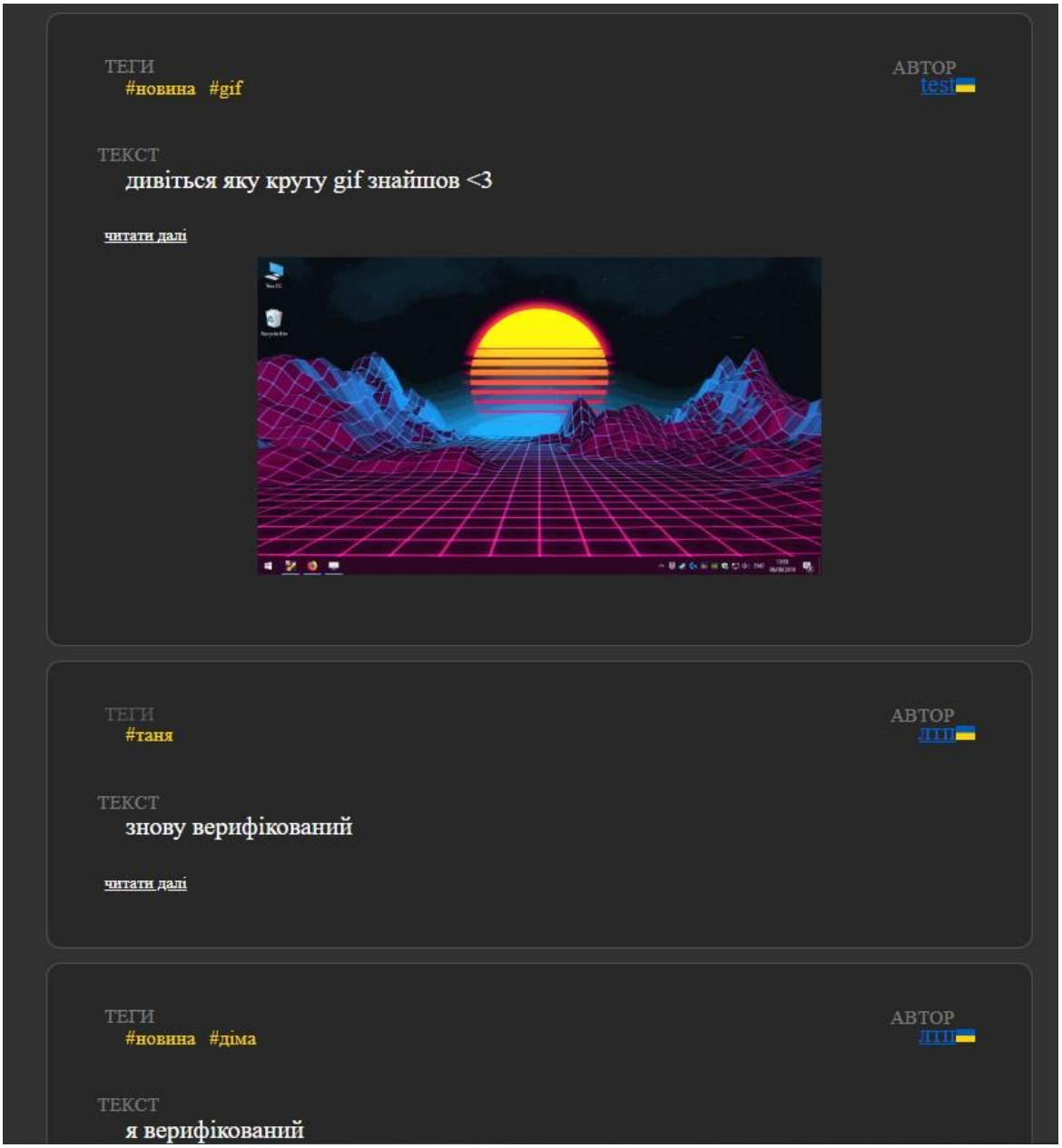


Рисунок 3.5 – Відображення постів

3.5 Розробка засобів роботи з акаунтом

Перегляд акаунту розділений на 2 частини:

- відображення інформації акаунту та постів;
- взаємодія з акаунтом.

Клієнтська частина відправляє запит на отримання інформації про цей акаунт на сервері:

- отримуємо значення пошукового рядка де внесена id-акаунту інформацію про який необхідно знайти;
- отримуємо значення з БД про акаунт який шукаємо та про акаунту з якого надійшов запит;
- отримуємо значення постів для шуканого акаунту з фільтрацією їх по часу створення;
- відправляємо відповідь в якій: значення шуканого акаунту, значення акаунту з якого отримали запит та список постів.

Клієнтська частина вносить інформацію про акаунт та пости в шаблон (рис. 3.6).

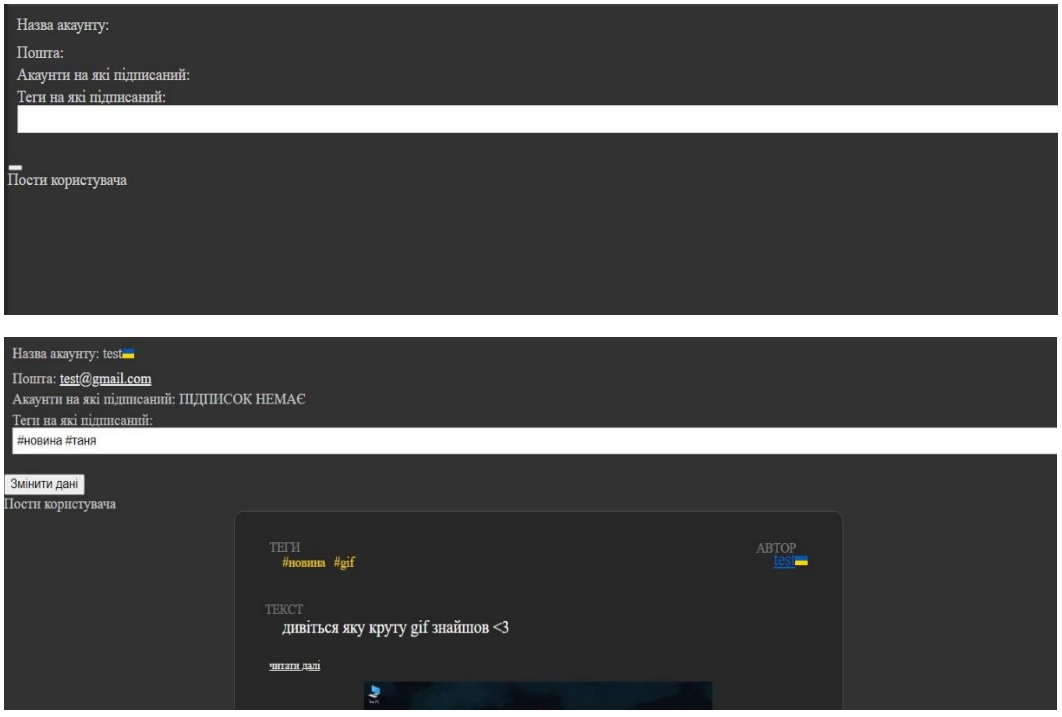


Рисунок 3.6 – Форма до і після підтягнення даних

Після отриманої відповіді про шуканий акаунт та акаунт з якого отримали запит, відбувається перевірка чи це один й той самий акаунт, чи різні. Залежно від цього створюється або кнопка редагування акаунту чи кнопка підписки/відписки на акаунт.

Редагування акаунту. Натиснення на цю кнопку дозволяє змінювати рядок в якому відображаються наявні теги. Після внесення необхідних змін та натиснення кнопки підтвердити відбувається обробка тегів та запит до сервера і в ньому надсилається запит в БД одночасно з пошуком та заміною поля тегів користувача. Після цього сторінка оновлюється.

Підписка. Після натиснення на кнопку одразу відбувається запит до серверної частини і там:

– отримується інформація з БД про користувача який надіслав запит;

– перевіряється чи в списку підписок користувача не наявний користувач на якого підписуємося;

– додаємо інформацію про користувача на якого підписуємося і надсилаємо запит до БД з оновленим списком;

– відправляється відповідь про результат.

Після виводу результату сторінка оновлюється.

Відписка. По натисненню на кнопку надсилається запит до сервера і там відбуваються такі дії:

- отримується інформація з БД про користувача який надіслав запит;
- фільтруємо список підписок користувача і отримуємо список в якому точно немає користувача від якого відписуємося;
- перевіряємо чи змінилась довжина списку підписок та відфільтрований список, якщо він став менше, то надсилаємо запит до БД з оновленим списком;
- відправляється відповідь про результат.

Після виводу результату сторінка оновлюється.

3.6 Розробка засобів для перегляду посту

В кожному посту крім всієї внесеної інформаціє є посилання “читати далі”, якщо натиснути на нього, то відкриється нове вікно з id-поста в пошуковому рядку і одразу відбудеться запит до сервера. На серверній частині буде виконано наступне:

- отримується значення id-посту зі значення рядка пошуку;
- відправляється запит до БД і отримується значення цього посту;
- відправляється відповідь на серверну частину.

Потім відбувається внесення значень посту в шаблон і виведення інформації про нього на сторінці (рис. 3.7).

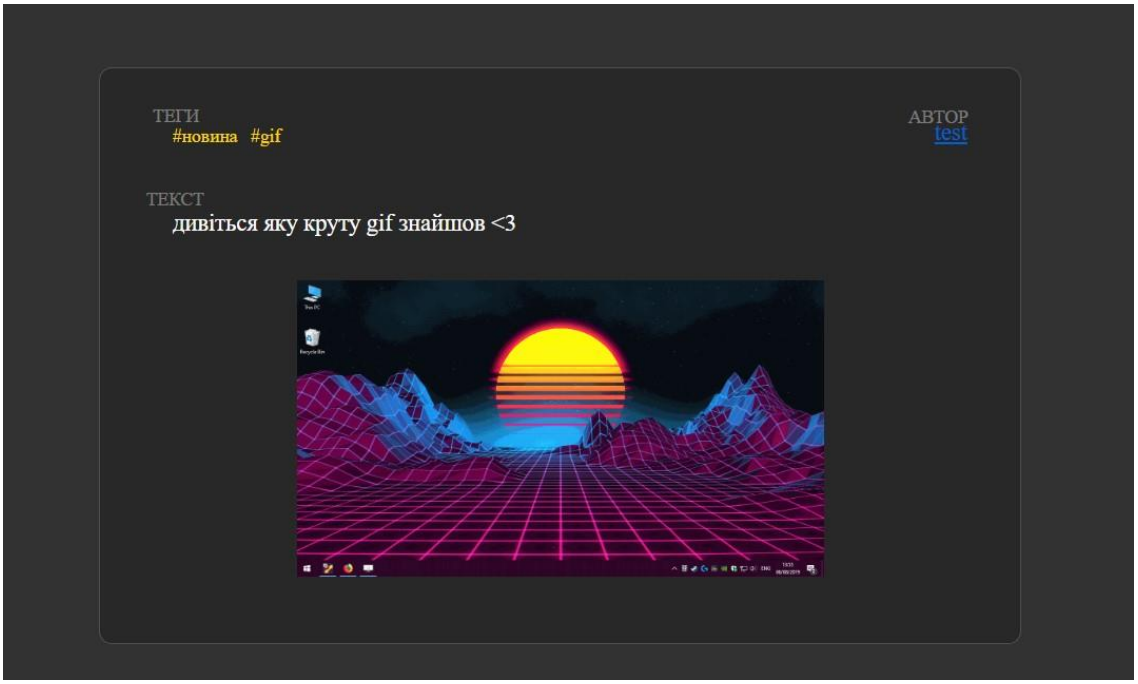


Рисунок 3.7 – Відображення конкретного посту

3.7 Організація тесту на верифікацію

На сторінці відображаються форма тесту з номером питання, питанням та можливими відповідями (рис. 3.8).

Рисунок 3.8 – Форма тесту на верифікацію

Після натиснення на відповідь у список відповідей вноситься індекс відповіді і виводиться інше питання. По закінченню питань відбувається перевірка відповідей і якщо на всі запитання дані правильні відповіді, відбувається запит на сервер, де відбуваються такі дії:

- отримуємо id-користувача з токену авторизації;
- змінюємо значення поля яке відповідає за верифікацію на true.

3.8 Розробка бази даних

База даних має два документи для збереження постів і інформації про користувачів. Документ "posts" (рис. 3.9) містить наступні поля:

- id: унікальний ідентифікатор поста;
- text: текстовий зміст поста;
- tags: теги, які вказують на тему або категорію поста;
- imgUrl: URL-адреса зображення, пов'язаного з постом;
- author: ім'я автора поста;
- authorId: унікальний ідентифікатор автора поста;
- authorIsVerify: позначка, що підтверджує, що автор є перевіреним користувачем;
- createdAt: дата і час створення поста;

- updatedAt: дата і час останнього оновлення поста.

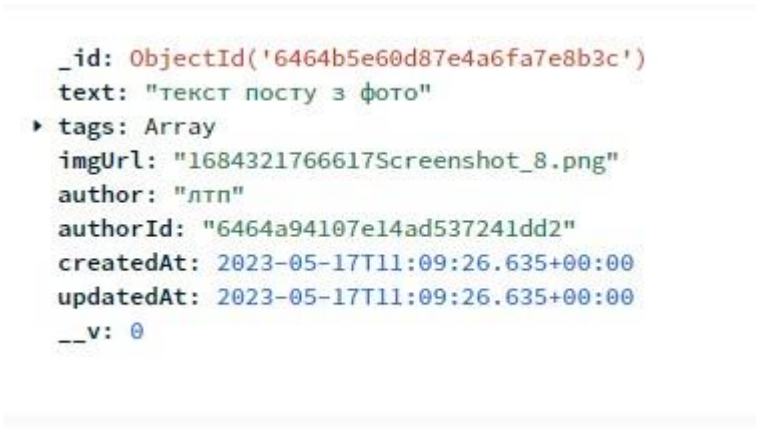


Рисунок 3.9 – Структура об’єкта в документі posts

Документ "users" (рис. 3.10) містить наступні поля:

- username: ім'я користувача або псевдонім;
- password: захешований пароль користувача;
- email: адреса електронної пошти користувача;
- tags: теги, що відображають інтереси або вподобання користувача;
- subscribeTo: перелік користувачів, на яких користувач підписаний;
- createdAt: дата і час створення облікового запису користувача;
- updatedAt: дата і час останнього оновлення інформації про користувача.

Рисунок 3.10 – Структура об’єкта в документі users

4 ТЕСТУВАННЯ СИСТЕМИ

Для тестування системи було розроблено ряд тест-кейсів, що призначені для перевірки правильності реалізації функціональних вимог до системи.

В таблиці 4.1 наведено тест-кейс, який дозволяє перевірити функцію «Реєстрація користувача». Передумовою для виконання цієї операції є те, що ім’я користувача, з яким він реєструється, не повинно бути зайнятим. Результати роботи функції торкаються клієнтської частини (сповіщення про реєстрації, перехід на сторінку постів) та роботи з базою даних (внесення записів в БД). Очікувані результати, наведені в тест-кейсі, співпадають з реальними результатами, що представлено на рис. 4.1-4.4.

Таблиця 4.1 - Тест кейс 1. Реєстрація користувача

Перед умови	Username має бути не зайнятий
Кроки	1. Відкрити сторінку реєстрації 2. Заповнення полів реєстраційної форми 3. Відмічення чекбоків 4. Натиснення кнопки “Зареєструватися”
Дані на вхід	Username: dut Пошта: dut@gmail.com Пароль: dut Прапорець1: true Прапорець2: true
Очікуваний результат	Сповіщення про успішну реєстрацію Створення в БД нового користувача Отримання jwt токена в localStorage Перехід на сторінку з постами

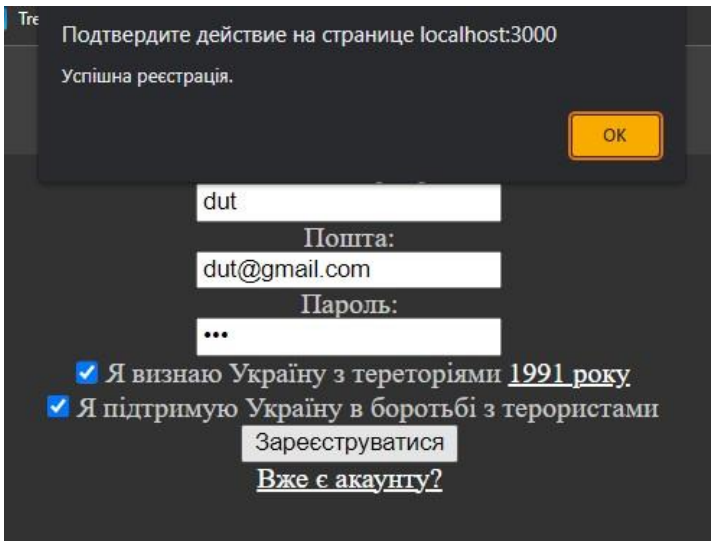


Рисунок 4.1 – Сповіщення про успішну реєстрацію [Тест кейс 1]

Рисунок 4.2 - Створення в БД нового користувача [Тест кейс 1]

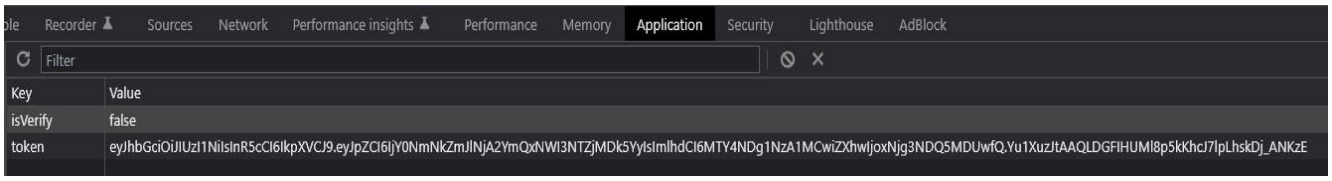


Рисунок 4.3 - Отримання jwt токена в localStorage [Тест кейс 1]

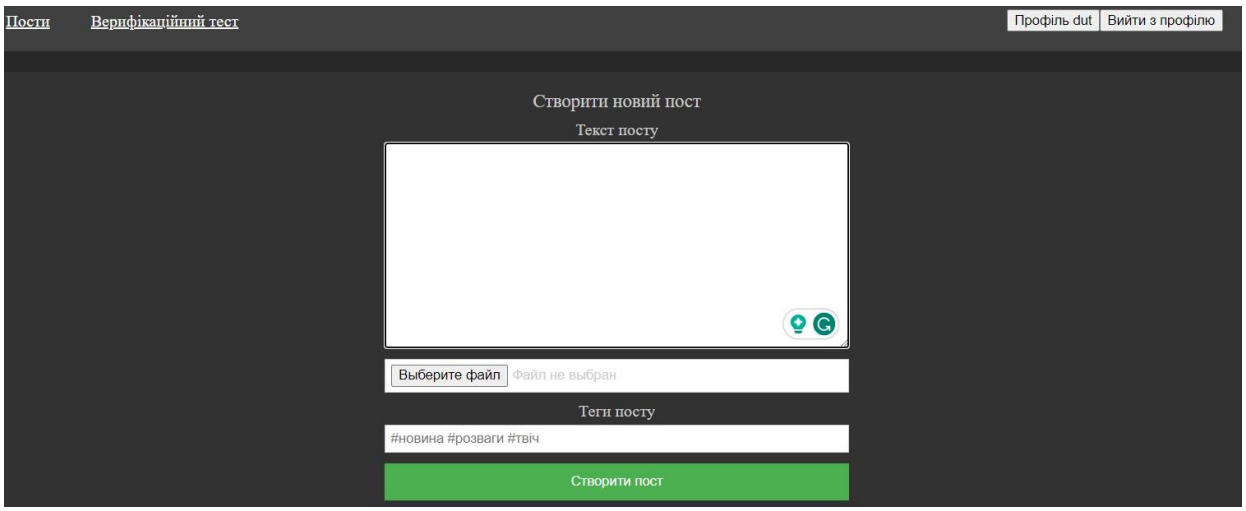


Рисунок 4.4 - Перехід на сторінку з постами [Тест кейс 1]

В таблиці 4.2 наведено тест-кейс, який дозволяє перевірити функцію «Зміна тегів користувача». Передумовою для виконання цієї операції є те, що користувач має бути авторизований. Результати роботи функції торкаються клієнтської частини (сповіщення про різні помилки або про успішну операцію) та роботи з базою даних (внесення записів в БД). Очікувані результати, наведені в тест-кейсі, співпадають з реальними результатами, що представлено на рис. 4.5-4.7.

Таблиця 4.2 - Тест кейс 2. Зміна тегів користувача

Перед умови	Користувач має бути авторизований
Кроки	1. Перехід на сторінку профіля через кнопку “Профіль (username)” 2. Натиснення кнопки “Змінити дані” 3. Внесення нового списку тегів 4. Підтвердження даних
Дані на вхід	Теги: таня gif
Очікуваний результат	Виведення повідомлення про успішну операцію Зміна даних про теги користувача у БД Відображення нових тегів на сторінці профіля

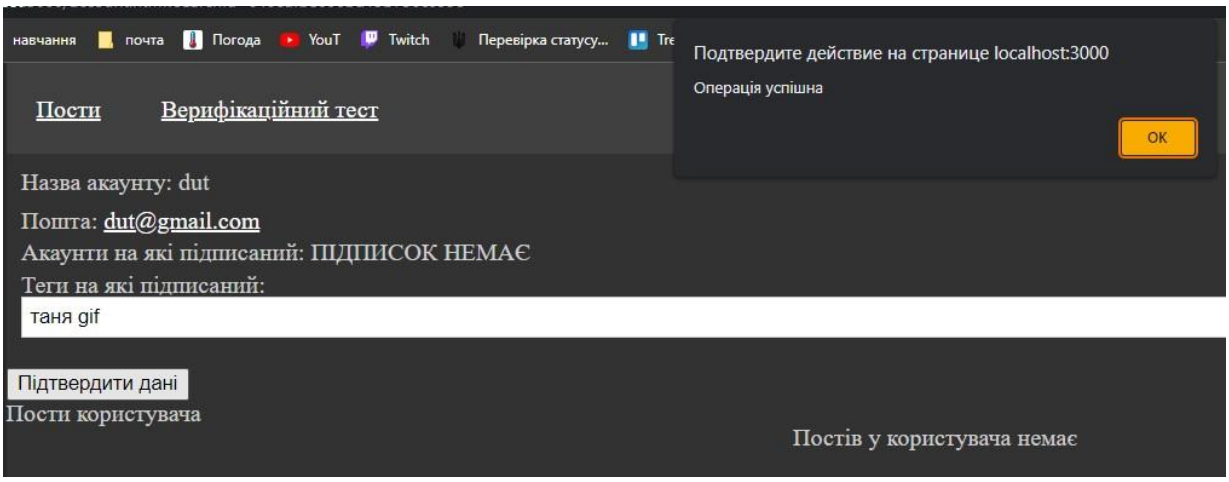


Рисунок 4.5 - Виведення повідомлення про успішну операцію [Тест кейс 2]

Рисунок 4.6 - Зміна даних про теги користувача у БД [Тест кейс 2]

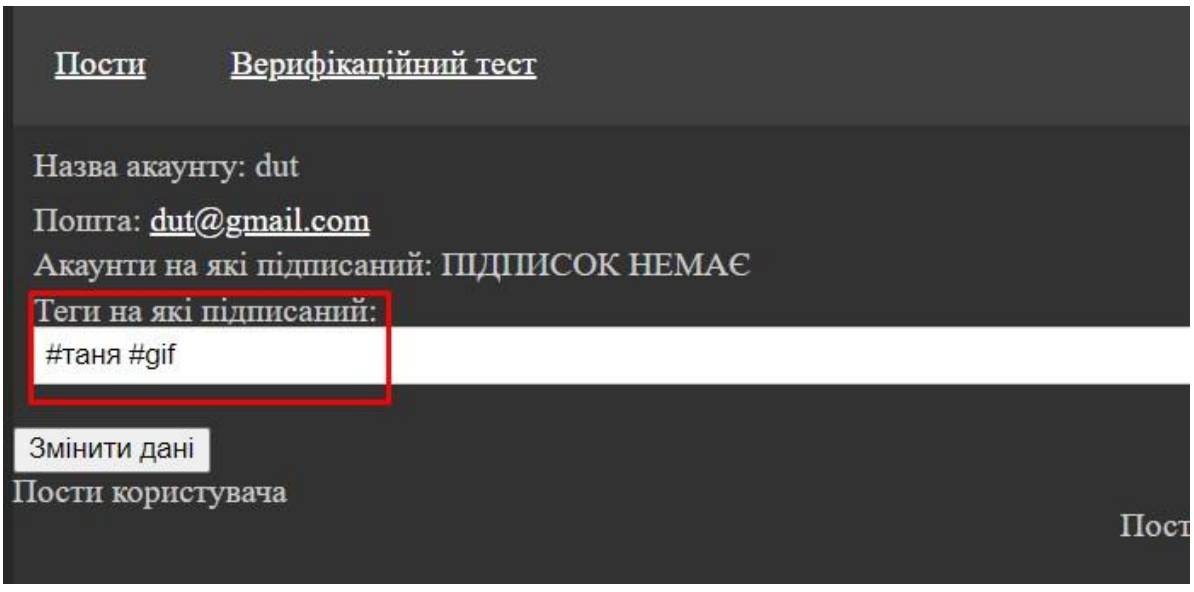


Рисунок 4.7 - Відображення нових тегів на сторінці профіля [Тест кейс 2]

В таблиці 4.3 наведено тест-кейс, який дозволяє перевірити функцію «Відображення постів по тегам». Передумовою для виконання цієї операції є те, що користувач має бути авторизований та мати введені теги у відповідності до тегів які його цікавлять. Результати роботи функції торкаються клієнтської частини (сповіщення про різні помилки або успішне відображення постів). Очікувані результати, наведені в тест-кейсі, співпадають з реальними результатами, що представлено на рис. 4.8-4.10.

Таблиця 4.3 - Тест кейс 3. Перегляд постів по введеним тегам

Перед умови	Користувач має бути авторизованим та мати введенні теги постів в акаунті
Кроки	1. Перехід на сторінку Пости
Очікуваний результат	Відображення постів які мають теги такі ж як і користувач

Рисунок 4.8 – Значення поля тегів користувача в БД [Тест кейс 3]

```
_id: ObjectId('646741b4d072be0efb662135')
text: "я Таня. і я хочу буди зверху в стрічці"
▸ tags: Array
imgUrl: ""
author: "лтп"
authorId: "6464a94107e14ad537241dd2"
authorIsVerify: false
createdAt: 2023-05-19T09:30:28.871+00:00
updatedAt: 2023-05-19T09:30:28.871+00:00
__v: 0
```

```
_id: ObjectId('6468d33504c46243c4d40fcd')
text: "дивіться яку круту gif знайшов <3"
▸ tags: Array
imgUrl: "1684591413325anim.gif"
author: "test"
authorId: "64674634c2ff79f38c8e7ba4"
authorIsVerify: true
createdAt: 2023-05-20T14:03:33.336+00:00
updatedAt: 2023-05-20T14:03:33.336+00:00
__v: 0
```


Рисунок 4.9 – Наявні пости по тегам [Тест кейс 3]

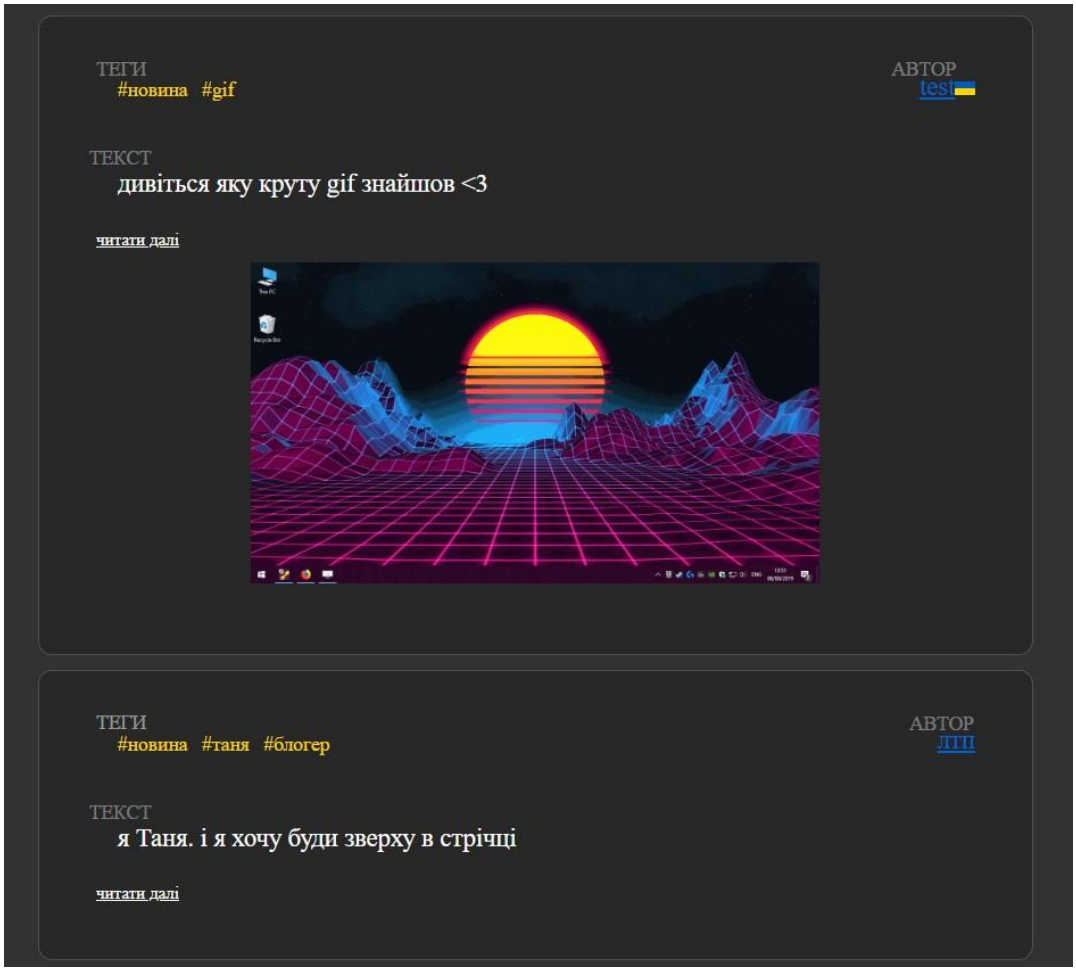


Рисунок 4.10 – Пости які відобразилися [Тест кейс 3]

В таблиці 4.4 наведено тест-кейс, який дозволяє перевірити функцію «Відображення постів від верифікованих акаунтів вище у стрічці». Передумовою для виконання цієї операції є те, що користувач має бути авторизований та мати введені теги у відповідності до тегів які його цікавлять. Результати роботи функції торкаються клієнтської частини (пости верифікованих користувачів мають український прапорець біля помітки автора, а також відображаються вище у списку постів). Очікувані результати, наведені в тест-кейсі, співпадають з реальними результатами, що представлено на рис. 4.11-4.12.

Таблиця 4.4 - Тест кейс 4. Перегляд постів верифікованих користувачів

Перед умови	Користувач має бути авторизованим та мати введенні теги постів в акаунті.
Кроки	1. Перехід на сторінку Пости
Очікуваний результат	У верифікованих користувачів біля username автора посту повинен бути прапор Пости верифікованих користувачів мають відображатися поверх інших

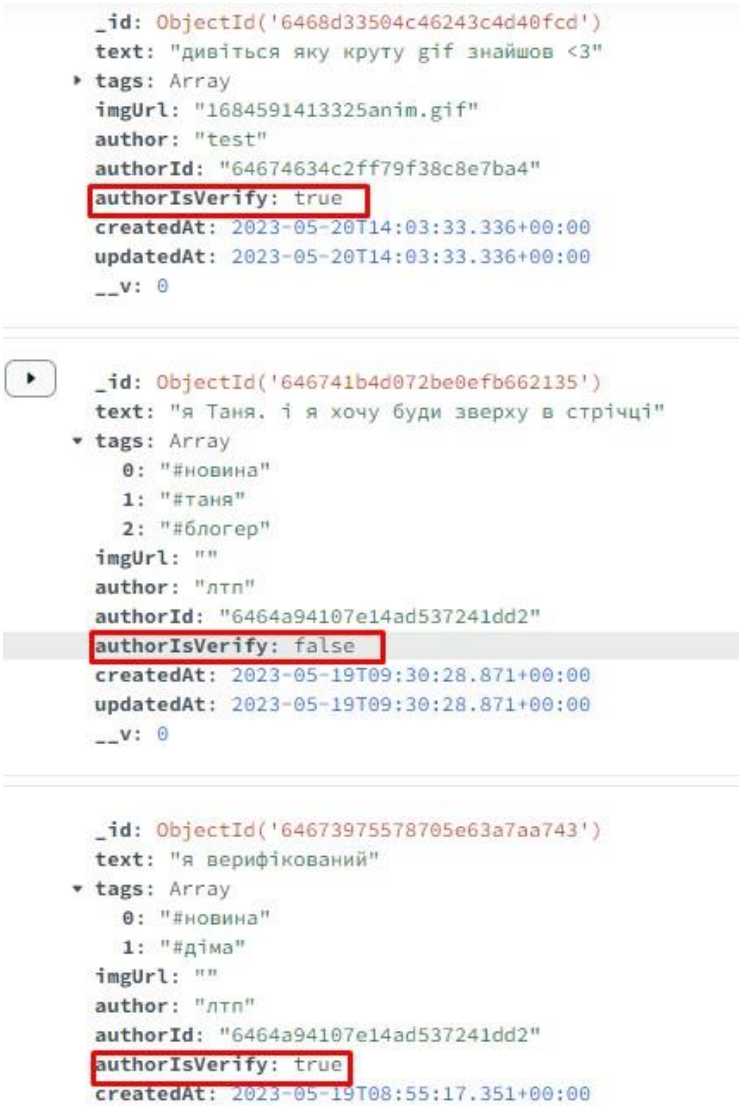


Рисунок 4.11 – Відфільтрований по дані список постів [Тест кейс 4]

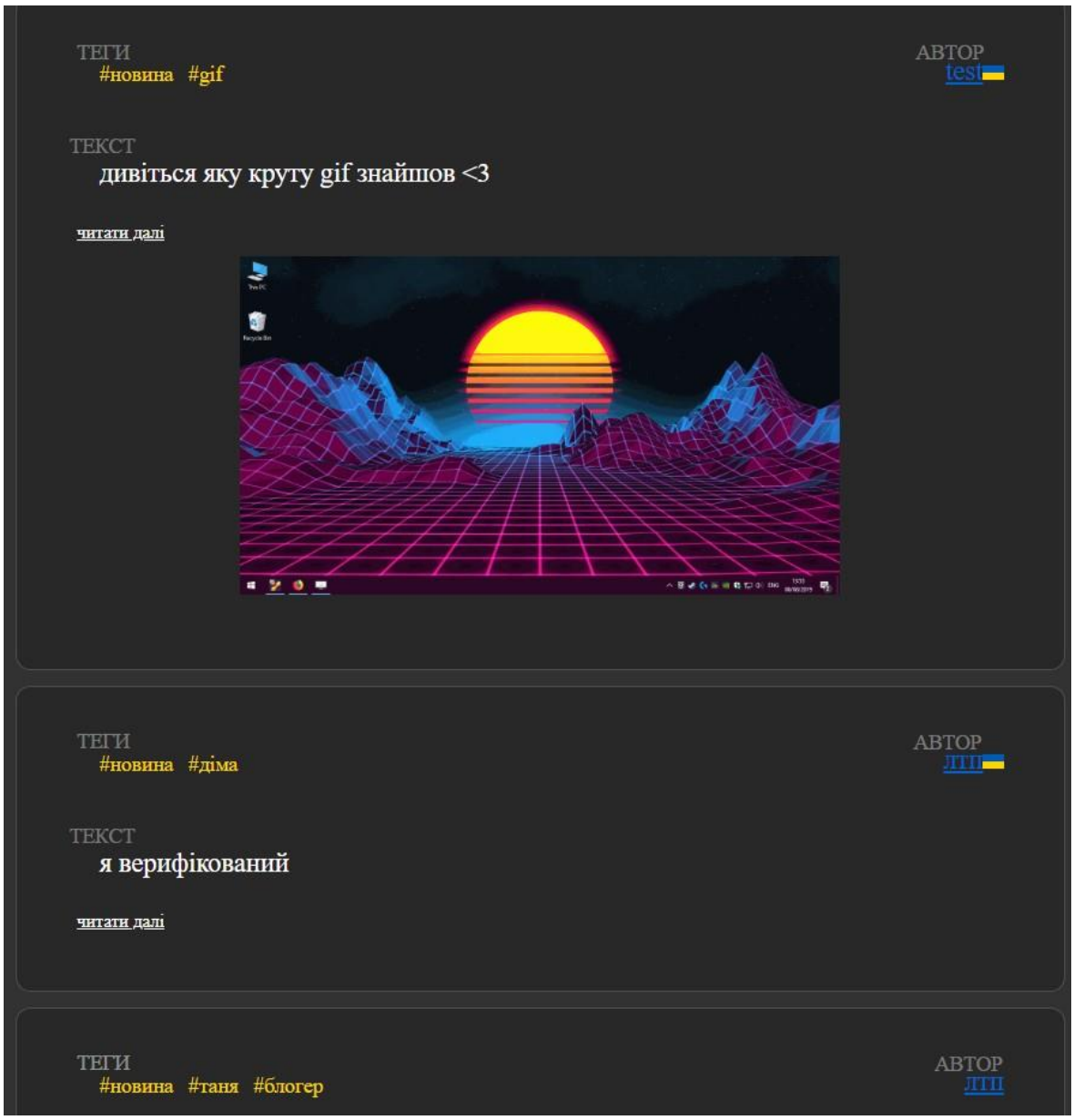


Рисунок 4.12 – Пости верифікованих авторів відображаються вище у стрічці [Тест кейс 4]

В таблиці 4.5 наведено тест-кейс, який дозволяє перевірити функцію «Підписки на користувача». Передумовою для виконання цієї операції є те, що користувач має не підписаним на користувача. Результати роботи функції торкаються клієнтської частини (сповіщення про різні помилки або про успішну операцію) та роботи з базою даних (внесення записів в БД). Очікувані результати, наведені в тест-кейсі, співпадають з реальними результатами, що представлено на рис. 4.13-4.15.

Таблиця 4.5 - Тест кейс 5. Підписка на користувача

Перед умови	Користувач не має бути підписаним на користувача на якого бажає підписатися
Кроки	1. Перехід на сторінку користувача 2. Натиснення кнопки “Підписатися”
Очікуваний результат	Повідомлення про успішну операцію Додавання бажаного користувача в список підписок в БД Оновлення сторінки та відображення кнопки “Відписатися”

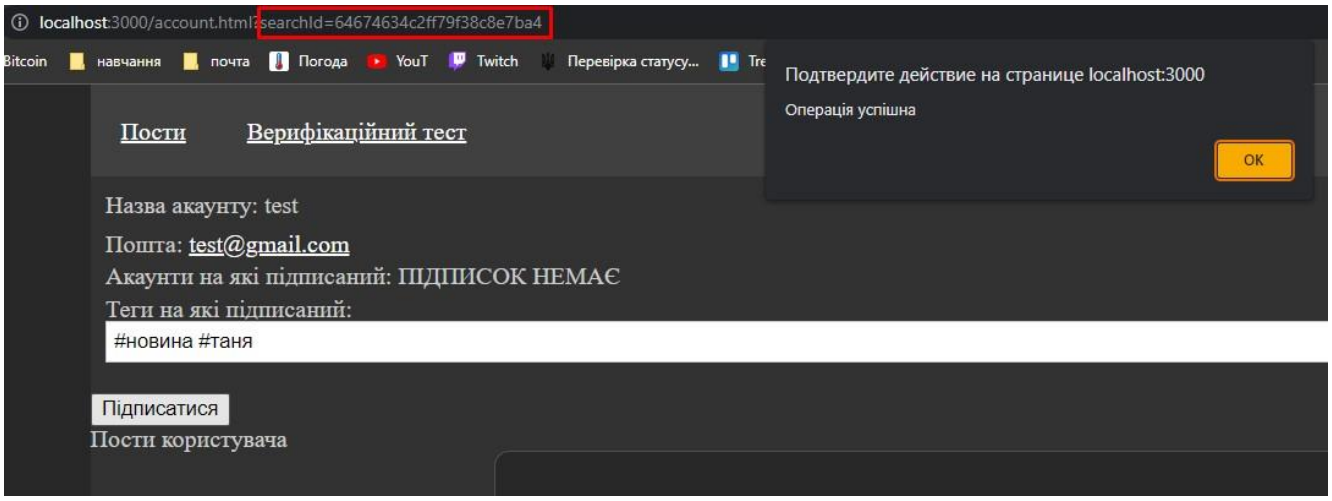


Рисунок 4.13 – Повідомлення про успішну операцію [Тест кейс 5]

Рисунок 4.14 – Додавання бажаного користувача в список підписок в БД [Тест кейс 5]

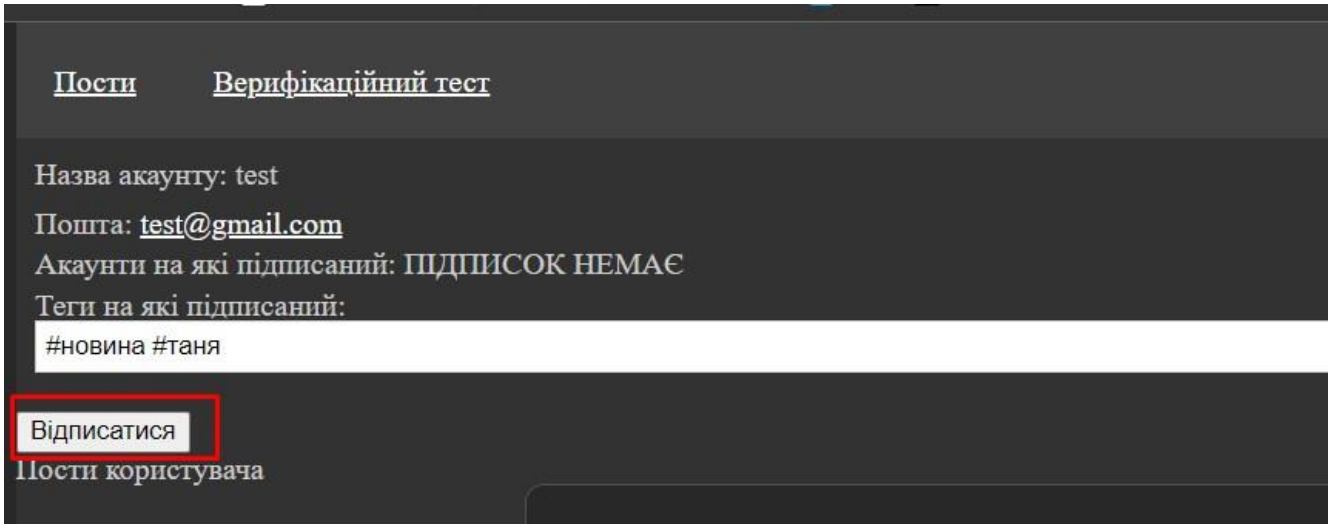


Рисунок 4.15 – Оновлення сторінки та відображення кнопки “Відписатися” [Тест кейс 5]

.

ВИСНОВКИ

1. Під час аналізу задач були визначені функціональні нефункціональні вимоги, що дало чітке розуміння потреб web-платформи.
2. Аналіз доступних аналогів показав ряд особливостей відсутніх на одній платформі, таких як фільтрація аудиторії за рахунок проходження тесту та згода з проукраїнськими пунктами при реєстрації, що підкреслило унікальність розробки.
3. Проведено розробку серверної частини, яка включала створення бази даних та реалізації необхідного функціоналу.
4. Проведено розробку клієнської сторони, включаючи реалізацію інтерфейсу користувача та взаємодії з сервером.
5. Проведене тестування для перевірки його функціональності, надійності зі створенням таблиць тест кейсів, що в свою чергу підтвердило роботоспроможність системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Протокол Internet Relay Chat [Електронний ресурс] // RFC Editor. – 2000. – Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc2810.html.
2. Блокування російських інтернет-сервісів в Україні [Електронний ресурс] // Вікіпедія. – 2023. – Режим доступу до ресурсу: http://surl.li/gxpxu.
3. Документація MongoDB [Електронний ресурс] // MongoDB. – 2023. – Режим доступу до ресурсу: https://www.mongodb.com/docs.
4. Навчальний підручник JavaScript [Електронний ресурс] // JAVASCRIPT.INFO. – 2023. – Режим доступу до ресурсу: https://uk.javascript.info.
5. Twitter. [Електронний ресурс] – Режим доступу: https://twitter.com/.
6. Instagram. [Електронний ресурс] – Режим доступу: https://www.instagram.com.
7. Mastodon. [Електронний ресурс] – Режим доступу: https://mastodon.social/explore.
8. Visual Studio Code. [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/.
9. Gulp. [Електронний ресурс] – Режим доступу: https://gulpjs.com/.
10. Довідник по HTML тегам [Електронний ресурс] – Режим доступу: https://css.in.ua/html/tags.
11. Node.js. [Електронний ресурс] – Режим доступу: https://nodejs.org/uk.
12. MongoDB. [Електронний ресурс] – Режим доступу: https://www.mongodb.com/
13. Expressjs. [Електронний ресурс] – Режим доступу: https://expressjs.com/.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-платформи для українських контент-мейкерів мовою JavaScript

Виконав студент 4 курсу
групи ПД-43
Лисенко Олександр Анатолійович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи -** спрощення процесу формування рекомендацій постів українських контент-мейкерів для цільової аудиторії за рахунок використання web-платформи, розробленої мовою JavaScript.
- **Об'єкт дослідження -** формування рекомендацій постів українських контент-мейкерів для цільової аудиторії.
- **Предмет дослідження -** програмне забезпечення для автоматизованого формування рекомендацій постів українських контент-мейкерів на основі характеристик цільової аудиторії.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Аналіз задач які повинні виконуватися на web-платформі.
2. Аналіз та дослідження доступних аналогів.
3. Розробка серверної сторони.
4. Розробка клієнтської сторони.
5. Проведення тестування web-платформи.

3

АНАЛІЗ АНАЛОГІВ

Особливість	**Instagram**	**Twitter**	**Mastodon**	**Розроблений проєкт**
Вільна ліцензія на контент	-	-	+	+
Основний контент	Фото/Відео	Текст	Текст	Текст
Довжина посту	2200 символів	280 символів	500 символів	500 символів
Хештеги	+	+	+	+
Безкоштовна верифікація	-	-	+	+
Домінація підписок над рекомендаціями	+	-	+	+
Підтримка gif-файлів	-	+	+	+
Фільтрація аудиторії на проукраїнськість	-	-	-	+

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні:

1. Реєстрація та авторизація користувача.
2. Введення та зміна параметрів для відображення постів.
3. Відображення постів за введеними параметрами.
4. Створення постів.
5. Проходження верифікаційного тесту.

Нефункціональні:

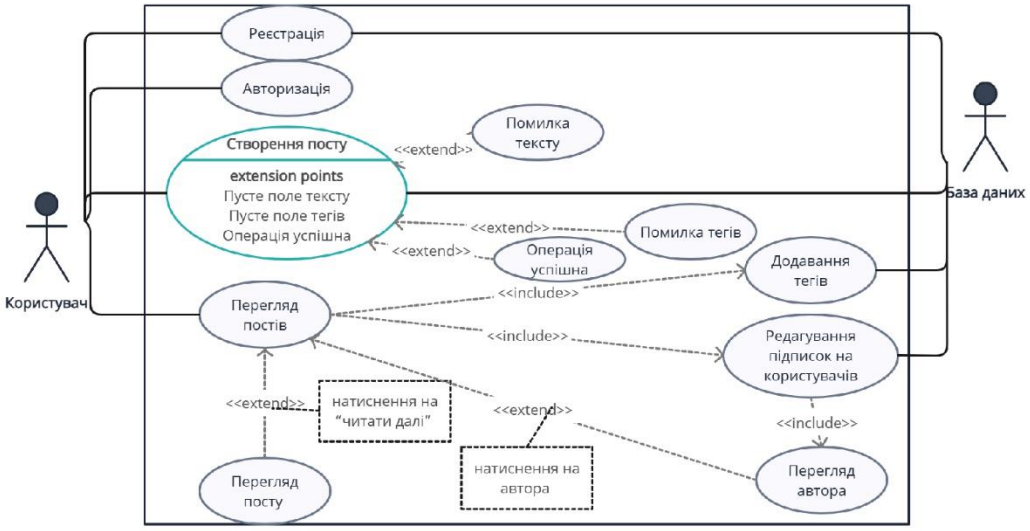
1. Безпечне збереження паролів.
2. Свобода від модерації. Користувачі самі вибирають за ким слідкувати.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

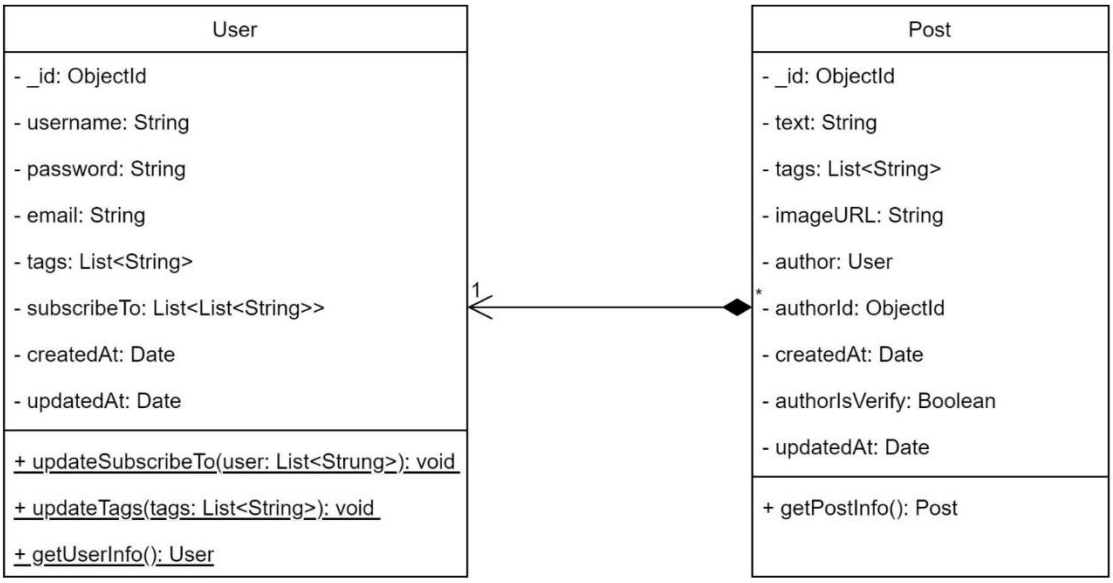
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



7

ДІАГРАМА КЛАСІВ

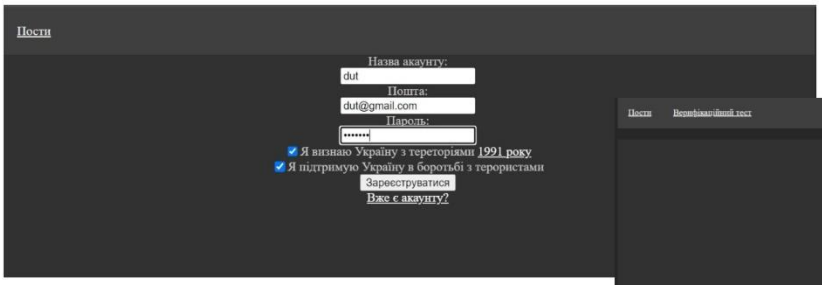


8

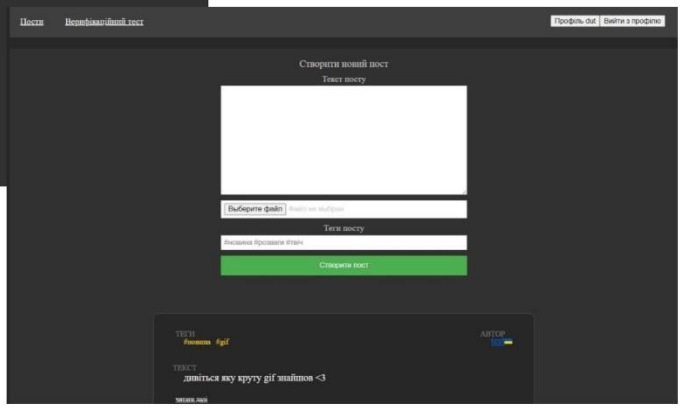
КАРТА САЙТУ

9

ЕКРАННІ ФОРМИ



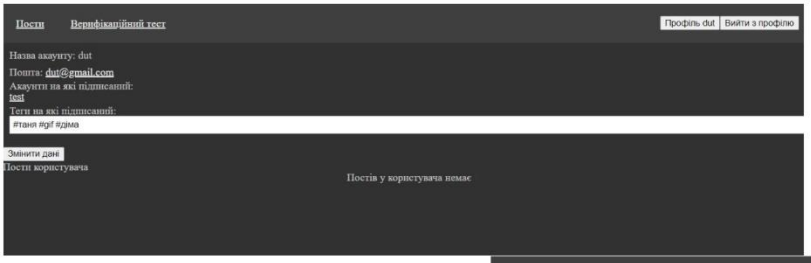
Сторінка реєстрації



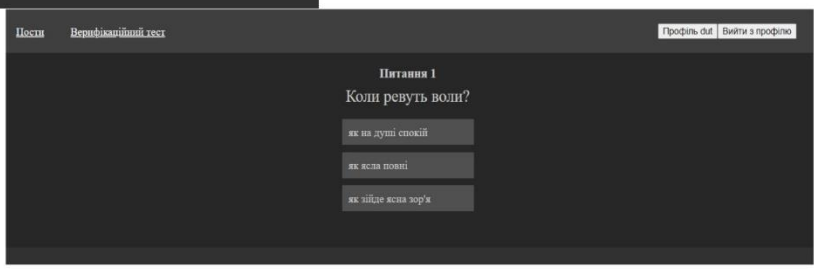
Сторінка створення поста та відображення постів

10

ЕКРАННІ ФОРМИ



Сторінка профілю



Сторінка верифікаційного тесту

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Лисенко О.А. Розробка структури бази даних для системи формування рекомендацій постів українських контент-мейкерів / Золотухіна О. А., Лисенко О.А. // Перший крок у науку: Конотопські наукові студії - 2023, 20 травня 2023р., Класичний фаховий коледж СумДУ, м. Конотоп: СумДУ 2023. Подано до друку.
2. Лисенко О.А. Мова програмування для написання web-платформи для українських контент-мейкерів / Золотухіна О. А., Лисенко О.А. // Перший крок у науку: Конотопські наукові студії - 2023, 20 травня 2023р., Класичний фаховий коледж СумДУ, м. Конотоп: СумДУ 2023. Подано до друку.

12

ВИСНОВКИ

1. Під час аналізу задач були визначені функціональні нефункціональні вимоги.
2. Аналіз доступних аналогів показав ряд особливостей відсутніх на одній платформі.
3. Проведено розробку серверної частини, яка включала створення бази даних та реалізації необхідного функціоналу.
4. Проведено розробку клієнської сторони, включаючи реалізацію інтерфейсу користувача та взаємодії з сервером.
5. Проведене тестування для перевірки його функціональності, надійності зі створенням таблиць тест кейсів.

13

ДЯКУЮ ЗА УВАГУ!

.

ДОДАТОК Б

Серверна частина

pro/server/middleware/checkAuth.js

```
import jsonwebtoken from "jsonwebtoken";

export const checkAuth = (req, res, next) => {
   const authToken = (req.headers.authorization
|| '').replace(/Bearer\s*/, '');

   if(authToken === 'null'){
      return res.json({message: "Ви не
авторизовані."});
   }else{
      try {
         const verified =
jsonwebtoken.verify(authToken,
process.env.JWT_KEY);
         req.accId = verified.id;
         next();
      } catch (error) {
         return res.json({message: `Виникла
помилка: ${error}`});
      }
   }
}
```

pro/server/models/Post.js

```
import mongoose from 'mongoose';

const Post = new mongoose.Schema(
   {
      text: {type: String, required: true},
      tags: {type: Array, required: true},
      imgUrl: {type: String, default: ''},
      author: {type: String},
      authorId: {type: String},
      authorIsVerify: {type: Boolean}
   },
   {timestamps: true},
);

export default mongoose.model('Post', Post);
```

pro/server/models/User.js

```
import mongoose from "mongoose";

const User = new mongoose.Schema(
   {
      username: {
         type: String,
         required: true,
         unique: true,
      },
      password: {
         type: String,
         required: true,
      },
      email: {
         type: String,
         required: true,
      },
      isVerify: {
         type: Boolean,
         default: false
      },
      tags: {
         type: Array,
         default: [],
      },
      subscribedTo: {
         type: Array,
         default: []
      },
   },
   {timestamps: true},
)

export default mongoose.model('User', User);
```

pro/server/routes/account.js

```
import {Router} from "express";
import { checkAuth } from
"../middleware/checkAuth.js";

import User from "../models/User.js";

const router = new Router();

router.post('/subTo', checkAuth, async (req, res)
=> {
   try {
      const account = await
User.findById(req.accId);
      const subList = account.subscribedTo;
```

```
    const subTo = await User.findById(req.body.subTo);

    if(subList.includes(subTo)){
      return res.json({message: 'Ви вже підписані на цього користувача'})
    }

    await User.findByIdAndUpdate(account._id, {
      $push: {subscribedTo: [subTo._id, subTo.username]}
    });

    res.json({message: 'Операція успішна', actionSuccess: true});
  } catch (error) {
    return res.json({message: 'Щось пішло не так', actionSuccess: false});
  }
});

router.post('/unsubFrom', checkAuth, async (req, res) => {
  try {
    const account = await User.findById(req.accId);
    const subList = account.subscribedTo;
    const unsubFrom = await User.findById(req.body.unsubfrom);

    const changedSubList = subList.filter( ([id, _]) => id.toString() !== unsubFrom._id.toString());

    if (changedSubList.length === subList.length) {
      return res.json({message: 'Ви не підписані на цього користувача'});
    }

    await User.findByIdAndUpdate(account._id, {subscribedTo: changedSubList});
    res.json({message: 'Операція успішна', actionSuccess: true});
  } catch (error) {
    return res.json({message: 'Щось пішло не так', actionSuccess: false});
  }
});

router.post('/newtags', checkAuth, async (req, res) => {
  try {
    await User.findByIdAndUpdate(req.accId, {tags: req.body.tags});
    res.json({message: 'Операція успішна', actionSuccess: true});
  } catch (error) {
    return res.json({message: 'Щось пішло не так.', actionSuccess: false});
  }
});

router.post('/verify', checkAuth, async(req, res) => {
  try {
    await User.findByIdAndUpdate(req.accId, {isVerify: true});
    res.json({message: 'Операція успішна'});
  } catch (error) {
    return res.json({message: 'Щось пішло не так.'});
  }
});


export default router;
```

pro/server/routes/auth.js

```
import { Router } from "express";
import User from "../models/User.js";
import bcrypt from "bcryptjs";
import jsonwebtoken from "jsonwebtoken";

import {checkAuth} from '../middleware/checkAuth.js';

const router = new Router();

//Registratio
router.post('/reg', async (req, res) => {
  try {
    const { username, password, email } = req.body;

    const isUsed = await User.findOne({username});

    if(isUsed){
      return res.json({
        message: "Нажаль, цей username вже зайнятий."
```

```
        })
    }

    const severity = bcrypt.genSaltSync(7);
    const hashedPassword = bcrypt.hashSync(password, severity);

    const newAccount = new User({
        username,
        email,
        password: hashedPassword
    })

    const authToken = jsonwebtoken.sign(
        {id: newAccount._id},
        process.env.JWT_KEY,
        {expiresIn: "30d"},
    );

    await newAccount.save()
    res.json({
        authToken,
        newAccount,
        message: 'Успішна реєстрація.'
    })

  } catch (error) {
    res.status(500).json({
        message: "Помилка на сервері",
        error: error.message
    });
  }
})

//Login
router.post('/login', async (req, res) => {
  try {
    const {username, password} = req.body;

    const account = await User.findOne({username});
    if(!account){
        return res.json({message: "Невірний username."});
    }

    const isCorPassword = await bcrypt.compare(password, account.password);
    if(!isCorPassword){
        return res.json({message: "Невірний пароль."});
    }

    const authToken = jsonwebtoken.sign(
        {id: account._id},
        process.env.JWT_KEY,
        {expiresIn: "30d"},
    );

    res.json({
        authToken,
        account,
        message: 'Авторизація успішна.'
    })
  } catch (error) {
    console.log(error);
  }
})

//Get token
router.get('/me', checkAuth, async (req, res) => {
  try {
    const account = await User.findById(req.accId);
    if(!account){
        return res.json({message: "Невірний логін."});
    }
    const authToken = jsonwebtoken.sign(
        {id: account._id},
        process.env.JWT_KEY,
        {expiresIn: "30d"},
    );

    res.json({
        authToken,
        account,
    })
  } catch (error) {
    res.json({message: "Проблеми з доступом."});
  }
})

export default router;
```

pro/server/routes/post.js

```
import {Router} from "express";
import path, {dirname} from 'path';
import {fileURLToPath} from 'url';
import { checkAuth } from "../middleware/checkAuth.js";
```

```
import Post from "../models/Post.js";
import User from "../models/User.js";

const router = new Router();

//create post
router.post('/newpost', checkAuth, async (req, res) => {
   try {
      const {text, tags} = req.body;
      const files = req.files;
      const account = await User.findById(req.accId);
      const splitTags = tags.split(' ');

      if(text.length > 500){
         return res.json({message: "Текст немає бути довший за 500 символів."})
      }

      if(files){
         //змінюємо ім'я фото, шукаємо поточне розташування та переносимо фото в папку з фото
         const imageName = Date.now().toString() + files.image.name;
         const currentPath = dirname(fileURLToPath(import.meta.url));
         files.image.mv(path.join(currentPath, '..', 'images', imageName));

         //Пост з картинкою
         const newPostImg = new Post(
            {
               text,
               tags: splitTags,
               imgUrl: imageName,
               author: account.username,
               authorId: account._id,
               authorIsVerify: account.isVerify
            }
         );
         await newPostImg.save();
         res.json({newPostImg});
      }else{
         const newPost = new Post(
            {
               text,
               tags: splitTags,
               author: account.username,
               authorId: account._id,
               authorIsVerify: account.isVerify
            }
         );
         await newPost.save();
         res.json({newPost});
      }
   } catch (error) {
      return res.json({message: 'Щось пішло не так.'})
   }
});

//get posts by recomendation
router.get('/posts', checkAuth, async (req, res) => {
   try {
      const account = await User.findById(req.accId);
      const {tags, subscribedTo} = account;
      const idSubscribedTo = subscribedTo.map(([id, _]) => id);

      let posts = await Post
      .find({$or: [ {author: { $in: idSubscribedTo }}, {tags: { $in: tags }}]})
      .sort({ createdAt: -1 })
      .limit(50);

      res.json({posts});
   } catch (error) {
      return res.json({message: 'Щось пішло не так.'});
   }
})

//get posts by accId
router.get('/account', checkAuth, async (req, res) => {
   try {
      const searchId = req.query.searchId;
      const searchAcc = await User.findById(searchId);
      const account = await User.findById(req.accId);


      let posts = await Post
      .find({authorId: searchId })
      .sort({ createdAt: -1 })
      .limit(50);

      res.json({posts, account, searchAcc});
   } catch (error) {
```

```
        return res.json({message: 'Щось пішло не так.'});
    }
})

//get post by id
router.get('/post', checkAuth, async (req, res) => {
    try {
        const postId = req.query.searchId;
        const post = await Post.findById(postId);

        res.json({post});
    } catch (error) {
        return res.json({message: "Щось пішло не так."})
    }
});

export default router;
```

pro/server/.env

```
PORT=4444
DB_USER=sasha
DB_PASSWORD=sasha123
DB_NAME=dut-diploma
JWT_KEY=fesdjvnfgvdfgbvdv32rnjserdgv4n3rkfdj
```

pro/server/index.js

```
import express from "express";
import mongoose from "mongoose";
import dotenv from "dotenv";
import cors from "cors";
import fileUpload from 'express-fileupload';

const app = express();

//.env
dotenv.config();
const PORT = process.env.PORT;
const DB_USER = process.env.DB_USER;
const DB_PASSWORD = process.env.DB_PASSWORD;
const DB_NAME = process.env.DB_NAME;

//middleware
app.use(cors());
app.use(fileUpload())
app.use(express.json());
app.use(express.static('images'))

//імпортуємо routers
import auth from './routes/auth.js';
import post from './routes/post.js';
import account from './routes/account.js';

//router
app.use('/auth', auth);
app.use('/post', post);
app.use('/account', account);

async function start() {
  try {
    await mongoose.connect(`mongodb+srv://${DB_USER}:${DB_PASSWORD}@cluster0.xpi1r1m.mongodb.net/${DB_NAME}`);
    app.listen(PORT, () => console.log(`Сервер стартував: ${PORT}`));
  } catch (error) {
    console.log(error);
  }
}

start();
```

pro/server/package.json

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "express-fileupload": "^1.4.0",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.1.0",
    "multer": "^1.4.5-lts.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

Клієнтська частина

pro/client/dist/css/style.min.css

```
.profile {
  padding: 10px;
}

.profile-username {
  margin-bottom: 10px;
}

.profile-info {
  margin-bottom: 10px;
}

.profile-info-email {
  margin-bottom: 5px;
}

.profile-info-subscribedTo {
  list-style: none;
  padding: 0;
  margin: 0;
  margin-bottom: 5px;
}

.profile-info-subscribedTo li {
  margin-bottom: 5px;
}

label {
  display: block;
  margin-bottom: 5px;
}

#profile-info-tags {
  width: 100%;
  padding: 5px;
  border: 1px solid #ccc;
  background-color: #fff;
}

/* ПОСТИ */

.posts-list {
  width: 80%;
  margin: 0 auto;
}

.post-block {
  display: flex;
  flex-direction: column;
  background-color: #f0f0f0;
  padding: 10px;
  margin-bottom: 10px;
}

.post-details {
  align-items: center;
  margin-bottom: 10px;
}

.flag {
  width: 14px;
}
html,
body,
div,
span,
applet,
object,
iframe,
h1,
h2,
h3,
h4,
h5,
h6,
p,
blockquote,
pre,
a,
abbr,
acronym,
address,
big,
cite,
code,
del,
dfn,
em,
img,
ins,
kbd,
q,
s,
samp,
small,
strike,
strong,
```

```
sub,
sup,
tt,
var,
b,
u,
i,
center,
dl,
dt,
dd,
ol,
ul,
li,
fieldset,
form,
label,
legend,
table,
caption,
tbody,
tfoot,
thead,
tr,
th,
td,
article,
aside,
canvas,
details,
embed,
figure,
figcaption,
footer,
header,
hgroup,
menu,
nav,
output,
ruby,
section,
summary,
time,
mark,
audio,
video {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}

/* HTML5 display-role reset for older browsers
*/

article,
aside,
details,
figcaption,
figure,
footer,
header,
hgroup,
menu,
nav,
section {
  display: block;
}

body {
  line-height: 1;
}

ol,
ul {
  list-style: none;
}

blockquote,
q {
  quotes: none;
}

blockquote:before,
blockquote:after,
q:before,
q:after {
  content: '';
  content: none;
}

table {
  border-collapse: collapse;
  border-spacing: 0;
}

a {
  color: #fff;
}

* {
  box-sizing: border-box;
```

```
}

body {
  background-color: #282828;
  color: #cdcdcd;
}

.flex {
  display: flex;
  align-items: center;
}

.header {
  margin: 0 auto;
  padding: 0 20px;
  background-color: #404040;
  width: 80vw;
  height: 60px;
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.navbar {
  display: flex;
  gap: 40px;
}

.account-management {
  width: 215px;
}

main {
  margin: 0 auto;
  width: 80vw;
  min-height: 100vh;
  background-color: #323232;
}

form {
  margin: 0 auto;
  display: flex;
  flex-direction: column;
  width: 40%;
  align-items: center;
}

footer {
  padding: 30px;
}

.copyright {
  text-align: center;
  font-size: 14px;
}
.create-post {
  max-width: 500px;
  margin: 20px auto 50px;
  padding: 20px;
}

.create-post p {
  font-size: 18px;
  margin-bottom: 10px;
}

.create-post label {
  display: block;
  margin-bottom: 5px;
}

.create-post textarea,
.create-post input[type="text"],
.create-post input[type="file"] {
  width: 100%;
  padding: 5px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  background-color: #fff;
}

.create-post textarea {
  height: 200px;
  resize: vertical;
}

.create-post button[type="submit"] {
  display: block;
  width: 100%;
  padding: 10px;
  background-color: #4caf50;
  color: #fff;
  border: none;
  cursor: pointer;
}

.create-post button[type="submit"]:hover {
  background-color: #45a049;
}

.posts-list {
  display: flex;
  flex-direction: column;
  align-items: center;
```

```
}

.post-block {
  padding: 40px;
  width: 700px;
  display: flex;
  flex-direction: column;
  border: 1px solid #ffffff30;
  border-radius: 10px;
  background-color: #282828;
}

.post-text {
  margin: 20px 0;
  padding: 15px 0 0 15px;
  font-size: 18px;
  line-height: 1.5;
  color: #fff;
  position: relative;
}

.post-text::before {
  content: 'ТЕКСТ';
  color: hsla(0, 0%, 80%, 0.5);
  font-size: 14px;
  position: absolute;
  top: 0;
  /* встановлюємо відстань зверху від .post-author */
  left: -5px;
  /* встановлюємо відстань зліва від .post-author */
}

.post-image {
  width: 400px;
}

.post-author {
  color: rgb(0, 99, 211);
  font-size: 18px;
  margin-left: 15px;
  cursor: pointer;
  position: relative;
}

.post-author::before {
  content: 'АВТОР';
  color: hsla(0, 0%, 80%, 0.5);
  font-size: 14px;
  position: absolute;
  top: -10px;
  /* встановлюємо відстань зверху від .post-author */
  left: -20px;
  /* встановлюємо відстань зліва від .post-author */
}

.post-tags {
  display: flex;
  padding-top: 5px;
  padding-left: 15px;
  font-size: 14px;
  position: relative;
  list-style-type: none;
  color: #ccc;
}

.post-tags li:not(:first-child) {
  margin-left: 10px;
}

.post-tags-tag {
  font-size: 14px;
  color: rgba(255, 215, 0, 1);
}

.post-tags-tag:first-child::before {
  content: 'ТЕГИ';
  color: hsla(0, 0%, 80%, 0.3);
  font-size: 14px;
  position: absolute;
  top: -10px;
  left: 0;
}

.post-image {
  display: block;
  margin: 10px auto;
  max-width: 100%;
}

.post-details {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
}

.post-link {
  font-size: 12px;
}

.post {
```

```
  display: flex;
  width: max-content;
  margin: 0 auto;
  padding-top: 50px;
}
#quiz {
  background-color: #282828;
  padding: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
}

#quiz-number {
  font-size: 18px;
  font-weight: bold;
  margin-bottom: 10px;
}

#quiz-question {
  font-size: 24px;
  margin-bottom: 20px;
}

#quiz-answer {
  margin-bottom: 20px;
}

#quiz-answer-list {
  list-style: none;
  padding: 0;
  margin: 0;
}

#quiz-answer-list li {
  font-size: 16px;
  width: 200px;
  padding: 10px;
  background-color: #505050;
  margin-bottom: 10px;
  cursor: pointer;
}

#quiz-answer-list li:hover {
  background-color: #404040;
  transition: .5s ease;
}
```

pro/client/dist/files/config.js

```
export const URL = 'http://localhost:4444';
export const clientURL = "http://localhost:3000";
```

pro/client/dist/img/ Flag_of_Ukraine.svg

```
<svg xmlns="http://www.w3.org/2000/svg" width="1200" height="800">
<rect width="1200" height="800" fill="#005BBB"/>
<rect width="1200" height="400" y="400" fill="#FFD500"/>
</svg>
```

pro/client/dist/js/tasks/account.js

```
import {URL} from '../../files/config.js';

const urlParams = new URLSearchParams(window.location.search);
const searchId = urlParams.get('searchId');

async function handleSubUnsub(searchAccId, isSubscribed) {
   const url = isSubscribed ? `${URL}/account/unsubfrom` : `${URL}/account/subTo`;
   const payload = isSubscribed ? { unsubfrom: searchAccId } : { subTo: searchAccId };

   try {
      const res = await axios.post(url, payload, {
         headers: {
            Authorization: `Bearer ${localStorage.getItem('token')}`,
         },
      });

      const response = res.data;
      alert(response.message);

      if (response.actionSuccess) {
         location.reload();
      }
   } catch (error) {
      alert(error);
   }
}

axios.get(
`${URL}/post/account?searchId=${searchId}`,
{ headers: {Authorization: `Bearer ${localStorage.getItem('token')}`} })
.then(res => {
   const response = res.data;
   const posts = response.posts;
   const account = response.account;
```

```
    const searchAcc = response.searchAcc;

    const fieldUsername = document.querySelector('#profile-username');
    const fieldEmail = document.querySelector('#profile-info-email')
    const listSubscribedTo = document.querySelector('#profile-info-subscribedTo');
    const fieldTags = document.querySelector('#profile-info-tags');

    //якщо вланик акаунту: надання функцыоналу змін акаунту
    //якщо не вланик акаунту:надання можливості підписатися
    const actionBtt = document.querySelector('#action-button');

    if(account._id === searchAcc._id) {
        actionBtt.classList.add('edit');
        actionBtt.innerHTML = "Змінити дані";

        actionBtt.addEventListener('click', () => {
            fieldTags.readOnly = false;

            actionBtt.innerHTML = "Підтвердити дані";
            actionBtt.addEventListener('click', async (event) => {
                const noVerifyNewTags = fieldTags.value;
                const newTags = (noVerifyNewTags.length > 0) ? noVerifyNewTags.trim().split(/\s+/)
                .map(tag => {
                    if(tag[0] !== '#'){
                        return '#' + tag;
                    }
                    return tag;
                }) : '';

                const answer = await axios.post(`${URL}/account/newtags`,
                    {tags: newTags},
                    { headers: {Authorization: `Bearer ${localStorage.getItem('token')}`} }
                );

                alert(answer.data.message)
                location.reload();
            })
        })
    }else{
        const isSubscribed = account.subscribedTo.some(([id, _]) => id === searchAcc._id);

        if (isSubscribed) {
            actionBtt.classList.add('unsubscribe');
            actionBtt.innerHTML = "Відписатися";

            actionBtt.addEventListener('click', async (event) =>
                await handleSubUnsub(searchAcc._id, isSubscribed)
            );
        } else {
            actionBtt.classList.add('subscribe');
            actionBtt.innerHTML = "Підписатися";

            actionBtt.addEventListener('click', async (event) =>
                await handleSubUnsub(searchAcc._id, isSubscribed)
            );
        }
    }


    //внесення дати про акаунт
    const flag = (localStorage.getItem('isVerify') === "true") ? '<img class="flag" src="../../img/Flag_of_Ukraine.svg">' : '';
    fieldUsername.innerHTML += searchAcc.username + flag;
    fieldEmail.innerHTML += `<a href="mailto:${searchAcc.email}" target="_blank">${searchAcc.email}</a>`;
    const subscribedToHtml = searchAcc.subscribedTo.map(([id, username]) => `<li><a href="account.html?searchId=${id}">${username}</a></li>`);
    listSubscribedTo.innerHTML += (searchAcc.subscribedTo.length > 0) ? subscribedToHtml.join('\n') : "ПІДПИСОК НЕМАЄ";
    fieldTags.value = (searchAcc.tags.length > 0) ? searchAcc.tags.join(' ') : "ТЕГІВ НЕМАЄ";

    //вивід постів акаунт
    const postsHtml = posts.map((post) => {
```

```
    const tags = post.tags.map(tag => {
      return `<li><p class="post-tags-tag">${tag}</p></li>`;
    });

    const flagAuthor = (post.authorIsVerify) ? '<img class="flag" src="../../img/Flag_of_Ukraine.svg">' : '';

    const text = (post.text.length > 200) ? post.text.slice(0, 200) + '...' : post.text;

    const imgHtml = (post.imgUrl !== '') ? `<img class="post-image" src="${URL}/${post.imgUrl}" >` : '';

    return `<article class="post-block">
      <div class='post-details'>
        <ul class="post-tags">${tags.join('\n')}</ul>
        <a class="post-author" href="account.html?searchId=${post.authorId}">${post.author}${flagAuthor}</a>
      </div>
      <p class="post-text">${text}</p>
      <a href='postId.html?searchId=${post._id}' class='post-link'>читати далі</a>
      ${imgHtml}
    </article>`
  });

  const postsList = document.querySelector('#posts-list');
  postsList.innerHTML += (postsHtml.length > 0) ? postsHtml.join('\n') : "<p>Постів у користувача немає</p>";
})
.catch(eror => {
  alert(eror);
})
```

pro/client/dist/js/tasks/login.js

```
import {URL} from '../../files/config.js';
const formLogin = document.querySelector('#form-login');

formLogin.addEventListener('submit', async (event) => {
  event.preventDefault();
  const username = formLogin.elements.username.value;
  const password = formLogin.elements.password.value;

  await axios.post(`${URL}/auth/login`, { username, password })
  .then(response => {
    alert(response.data.message);
    if(response.data.account){
      localStorage.setItem('token', response.data.authToken);


      formLogin.elements.username.value = '';
      formLogin.elements.password.value = '';

      window.location.href = "post.html";
    }
  })
  .catch(error => alert(error))

});
```

pro/client/dist/js/tasks/post.js

```
import {URL} from '../../files/config.js';

const formCreatePost = document.querySelector('#create-post');

//create newpost
formCreatePost.addEventListener('submit', async (event) =>{
  event.preventDefault();

  const text = formCreatePost.elements["new-post-text"].value;
  const image = formCreatePost.elements["new-post-image"].files[0] || false;
  const noVerifyTags = formCreatePost.elements["new-post-tags"].value;

  // обробка тегів
  const tags = noVerifyTags.trim().split(/\s+/).map(tag => {
    // if(tag[0] === ' ') return ''; перевірка на пустоту
    return (tag[0] !== '#') ? '#' + tag : tag;
  }).join(' ');

  console.log(tags)
```

```
    let postData = new FormData();
    postData.append('text', text);
    postData.append('image', image);
    postData.append('tags', tags);

    axios.post(`${URL}/post/newpost`, postData, { headers: { Authorization: `Bearer ${localStorage.getItem('token')}` } })
    .then(res => {

        formCreatePost.elements["new-post-text"].value = '';
        formCreatePost.elements["new-post-image"].value = '';
        formCreatePost.elements["new-post-tags"].value = '';

        location.reload();
    })
    .catch(eror => {
        alert(eror);
    })
})

//get posts
await axios.get(`${URL}/post/posts`, { headers: {Authorization: `Bearer ${localStorage.getItem('token')}`} })
.then(res => {
    const posts = res.data.posts;
    //надаємо топ видачі верифікованим акаунтам
    posts.sort((a, b) => b.authorIsVerify - a.authorIsVerify);

    const postsList = document.querySelector('#posts-list');

    if(posts){
        const postsHtml = posts.map((post) => {
            const tagsHtml = post.tags.map(tag => {
                return `<li><p class="post-tags-tag">${tag}</p></li>`;
            });

            const flagAuthor = (post.authorIsVerify) ? '<img class="flag" src="../../img/Flag_of_Ukraine.svg">' : '';

            const text = (post.text.length > 200) ? post.text.slice(0, 200) + '...' : post.text;

            const imgHtml = (post.imgUrl !== '') ? `<img class="post-image" src="${URL}/${post.imgUrl}" >` : '';

            return `<article class="post-block">
                <div class='post-details'>
                    <ul class="post-tags">${tagsHtml.join('\n')}</ul>
                    <a class="post-author" href="account.html?searchId=${post.authorId}">${post.author}${flagAuthor}</a>
                </div>
                <p class="post-text">${text}</p>
                <a href='postId.html?searchId=${post._id}' class='post-link'>читати далі</a>
                ${imgHtml}
            </article>`
        });
        postsList.innerHTML += postsHtml.join('\n');
    }else{
        postsList.innerHTML = 'Постів по вашим тегам і підпискам немає';
    }
})
.catch(eror => {
    alert(`Сталася помилка ${eror}`);
});
```

pro/client/dist/js/tasks/account.js

pro/client/dist/js/tasks/account.js

pro/client/dist/js/tasks/postId.js

```
import {URL} from '../../files/config.js';

const urlParams = new URLSearchParams(window.location.search);
const searchId = urlParams.get('searchId');

axios.get(
`${URL}/post/post?searchId=${searchId}`, { headers: {Authorization: `Bearer ${localStorage.getItem('token')}`} })
.then(res => {
    const response = res.data;
```

```
    const post = response.post;

    console.log(post);
    //вивід посту
    const tags = post.tags.map(tag => {
        return `<li><p class="post-tags-tag">${tag}</p></li>`;
    });

    const text = post.text;

    const imgHtml = (post.imgUrl !== '') ? `<img class="post-image image-full" src="${URL}/${post.imgUrl}" >` : '';
    const postHtml =
    `<article class="post-block">
        <div class='post-details'>
            <ul class="post-tags">${tags.join('\n')}</ul>
            <a class="post-author" href="account.html?searchId=${post.authorId}">${post.author}</a>
        </div>
        <p class="post-text">${text}</p>
        ${imgHtml}
    </article>`;

    const postDiv = document.querySelector('.post');
    postDiv.innerHTML = postHtml;
})
.catch(eror => {
    alert(eror);
})
```

pro/client/dist/js/tasks/quiz.js

```
import {URL} from '../../files/config.js';

const questions = [
    {title: "Коли ревуть воли?", answers: ["як на душі спокій", "як ясла повні", "як зійде ясна зор'я"], correct: 1},
    {title: "Чій Крим?", answers: ["український", "наш", "ваш"], correct: 0},
    {title: "Які в нас олені-олені?", answers: ["чудні та дикі", "вільні як птахи", "не бриті і не голені"], correct: 2},
    {title: "Що роблять мертві бджоли?", answers: ["йдуть в останню путь", "відносно не шумлять", "не гудуть"], correct: 2},
    {title: "Слово яке відповідає офіційно-діловому стилю.", answers: ["заява", "Бог", "волейбол"], correct: 0},
];

let answers = [];
let currentQuestion = 0;

//вивід інформації
const quiz = document.querySelector('#quiz');
const questionNumberField = document.querySelector("#quiz-number");
const questionField = document.querySelector("#quiz-question");
const answerList = document.querySelector('#quiz-answer-list');

//обробка відповідей
async function handleAnswerClick(answer){
    answers.push(answer)
    console.log(answers);

    currentQuestion++;
    if(currentQuestion === questions.length){
        quiz.innerHTML = '';

        let result = 0;
        answers.forEach((answer, index) => {
            if(answer === questions[index].correct) result++;
        })

        if(result === questions.length){
            await axios.post(`${URL}/account/verify`, {result}, { headers: {Authorization: `Bearer ${localStorage.getItem('token')}`}})
            .then(res => {
                alert(res.data.message);
            })
            .catch(error => {
                alert(error);
            })
        }else{
            alert('Тест провалено');
        }

    }else{
        displayQuestion(currentQuestion);
    }
}
```

```
// Виведення запитання і варіантів відповідей
function displayQuestion(questionIndex) {
  const question = questions[questionIndex];
  questionNumberField.textContent = `Питання ${questionIndex + 1}`;
  questionField.textContent = question.title;

  answerList.innerHTML = '';

  question.answers.forEach((answer, index) => {
    const li = document.createElement('li');
    li.textContent = answer;
    li.addEventListener('click', () => {handleAnswerClick(index)});

    answerList.appendChild(li);
  });
}

if(localStorage.getItem('isVerify') === 'true' ){
  quiz.innerHTML = 'Ви вже пройшли цей тест';
}else{
  displayQuestion(currentQuestion);
}
```

pro/client/dist/js/tasks/register.js

```
import {URL} from '../../files/config.js';
const formRegister = document.querySelector('#form-register');

formRegister.addEventListener('submit', async (event) => {
  event.preventDefault();
  const username = formRegister.elements.username.value;
  const password = formRegister.elements.password.value;
  const email = formRegister.elements.email.value;

  await axios.post(`${URL}/auth/reg`, {username, password, email})
  .then(response => {
    alert(response.data.message);
    if(response.data.newAccount){
      localStorage.setItem('token', response.data.authToken);
      formRegister.elements.username.value = '';
      formRegister.elements.password.value = '';
      formRegister.elements.email.value = '';

      window.location.href = "post.html";
    }
  })
  .catch(error => alert(error))
});
```

pro/client/dist/js/app.min.js

```
/*
 * ATTENTION: The "eval" devtool has been used (maybe by default in mode: "development").
 * This devtool is neither made for production nor for readable output files.
 * It uses "eval()" calls to create a separate source file in the browser devtools.
 * If you are trying to read the output file, select a different devtool (https://webpack.js.org/configuration/devtool/)
 * or disable the default devtool with "devtool: false".
 * If you are looking for production-ready output files, see mode: "production" (https://webpack.js.org/configuration/mode/).
 */
/******/ (() => { // webpackBootstrap
/******/ 	"use strict";
/******/ 	var __webpack_modules__ = ({

/***/ "./src/files/config.js":
/*!*****************************!*\
  !*** ./src/files/config.js ***!
  \*****************************/
/***/
((__unused_webpack___webpack_module__, __webpack_exports__, __webpack_require__) => {

eval("__webpack_require__.r(__webpack_exports__);\n/* harmony export */ __webpack_require__.d(__webpack_exports__, {\n/* harmony export */   \"URL\": () => (/* binding */ URL),\n/* harmony export */ \"clientURL\": () => (/* binding */ clientURL)\n/* harmony export */ });\nconst URL = 'http://localhost:4444';\r\nconst
```

```
clientURL = \"http://localhost:3000\";\n\n//# sourceURL=webpack://project/./src/files/config.js?");

/***/ }),

/***/ "./src/js/app.js":
/*!***********************!*\
  !*** ./src/js/app.js ***!
  \***********************/
/***/ ((__webpack_module__, __webpack_exports__, __webpack_require__) => {

eval("__webpack_require__.a(__webpack_module__, async (__webpack_handle_async_dependencies__, __webpack_async_result__) => { try {\n__webpack_require__.r(__webpack_exports__);\n/* harmony import */ var _modules_functions_js__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*! ./modules/functions.js */ \"./src/js/modules/functions.js\");\n/* harmony import */ var _files_config_js__WEBPACK_IMPORTED_MODULE_1__ = __webpack_require__(/*! ../files/config.js */ \"./src/files/config.js\");\n\r\n\r\n\r\n\r\n_modules_functions_js__WEBPACK_IMPORTED_MODULE_0__.isWebp();\r\n\r\n\r\nconst token = localStorage.getItem('token');\r\n//аунтифікація\r\nawait axios.get(`${_files_config_js__WEBPACK_IMPORTED_MODULE_1__.URL}/auth/me`, { headers: {Authorization: `Bearer ${token}`} })\r\n.then(res => {\r\n const response = res.data;// якщо є токен то вхід успішний\r\n\r\n\r\n if(!response.authToken){\r\n if(!['/index.html', '/register.html'].includes(window.location.pathname)) {\r\n window.location.href = 'index.html';\r\n }\r\n }else{\r\n localStorage.setItem('token', response.authToken);\r\n localStorage.setItem('isVerify', response.account.isVerify);\r\n\r\n\r\n //створюмо кнопку виходу\r\n const exitBtt = document.createElement('button');\r\n const accountBtt = document.createElement('button');\r\n\r\n\r\n accountBtt.textContent = `Профіль ${response.account.username}`;\r\n exitBtt.textContent = `Вийти з профілю`;\r\n\r\n accountBtt.addEventListener('click', () => {\r\n window.location.href = `account.html?searchId=${response.account._id}`;\r\n });\r\n\r\n\r\n exitBtt.addEventListener('click', () => {\r\n localStorage.removeItem('token');\r\n localStorage.removeItem('isVerify');\r\n window.location.href = 'index.html';\r\n });\r\n\r\n\r\n const header = document.querySelector('header');\r\n const div = document.querySelector('#account-management');\r\n\r\n\r\n div.appendChild(accountBtt);\r\n div.appendChild(exitBtt);\r\n header.appendChild(div);\r\n }\r\n})\r\n.catch(error => {\r\n console.error(error);\r\n});\r\n\r\n\r\n//якщо не верифікований, то видати верифікаційний тест в header\r\nif(localStorage.getItem('isVerify') === 'false'){\r\n const navHeader = document.querySelector('#navbar');\r\n \r\n const quiz = document.createElement(\"a\");\r\n quiz.setAttribute(\"href\", \"quiz.html\");\r\n quiz.textContent = 'Верифікаційний тест';\r\n\r\n\r\n navHeader.appendChild(quiz);\r\n}\n__webpack_async_result__();\n} catch(e) { __webpack_async_result__(e); } }, 1);\n\n//# sourceURL=webpack://project/./src/js/app.js?");

/***/ }),

/***/ "./src/js/modules/functions.js":
/*!*************************************!*\
  !*** ./src/js/modules/functions.js ***!
  \*************************************/
/***/ ((__unused_webpack___webpack_module__, __webpack_exports__, __webpack_require__) => {

eval("__webpack_require__.r(__webpack_exports__);\n/* harmony export */ __webpack_require__.d(__webpack_exports__
```

```
, {\n/* harmony export */ \"isWebp\": () => (/* binding */ isWebp)\n/* harmony export */ });\nfunction isWebp() {\r\n // Перевырка пыдтримки webp\r\n function testWebP(callback) {\r\n let webP = new Image();\r\n webP.onload = webP.onerror = function () {\r\n callback(webP.height == 2);\r\n };\r\n webP.src = \"\";\r\n }\r\n // Додавання до html класу _webp чи _no-webp \r\n testWebP((support) => {\r\n let className = (support) ? 'webp' : 'no-webp';\r\n document.documentElement.classList.add(className);\r\n });\r\n}\n\n//# sourceURL=webpack://project/./src/js/modules/functions.js?");

/***/ })

/******/ });
/************************************************************************/
/******/ // The module cache
/******/ var __webpack_module_cache__ = {};
/******/
/******/ // The require function
/******/ function __webpack_require__(moduleId) {
/******/ // Check if module is in cache
/******/ var cachedModule = __webpack_module_cache__[moduleId];
/******/ if (cachedModule !== undefined) {
/******/ return cachedModule.exports;
/******/ }
/******/ // Create a new module (and put it into the cache)
/******/ var module = __webpack_module_cache__[moduleId] = {
/******/ // no module.id needed
/******/ // no module.loaded needed
/******/ exports: {}
/******/ };
/******/
/******/ // Execute the module function
/******/ __webpack_modules__[moduleId](module, module.exports, __webpack_require__);
/******/
/******/ // Return the exports of the module
/******/ return module.exports;
/******/ }
/******/
/************************************************************************/
/******/ /* webpack/runtime/async module */
/******/ (() => {
/******/ var webpackQueues = typeof Symbol === "function" ? Symbol("webpack queues") : "__webpack_queues__";
/******/ var webpackExports = typeof Symbol === "function" ? Symbol("webpack exports") : "__webpack_exports__";
/******/ var webpackError = typeof Symbol === "function" ? Symbol("webpack error") : "__webpack_error__";
/******/ var resolveQueue = (queue) => {
/******/ if(queue && !queue.d) {
/******/ queue.d = 1;
/******/ queue.forEach((fn) => (fn.r--));
/******/ queue.forEach((fn) => (fn.r-- ? fn.r++ : fn()));
/******/ }
/******/ }
/******/ var wrapDeps = (deps) => (deps.map((dep) => {
/******/ if(dep !== null && typeof dep === "object") {
/******/ if(dep[webpackQueues]) return dep;
/******/ if(dep.then) {
/******/ var queue = [];
```

```
/******/
	queue.d = 0;
/******/
	dep.then((r) => {
/******/
	obj[webpackExports] = r;
/******/
	resolveQueue(queue);
/******/ 	},
(e) => {
/******/
	obj[webpackError] = e;
/******/
	resolveQueue(queue);
/******/ 	});
/******/ 	var
obj = {};
/******/
	obj[webpackQueues] = (fn) =>
(fn(queue));
/******/
	return obj;
/******/ 	}
/******/ }
/******/ var ret = {};
/******/
	ret[webpackQueues] = x => {};
/******/
	ret[webpackExports] = dep;
/******/ return ret;
/******/ }));
/******/ __webpack_require__.a
= (module, body, hasAwait) => {
/******/ var queue;
/******/ hasAwait &&
((queue = []).d = 1);
/******/ var depQueues =
new Set();
/******/ var exports =
module.exports;
/******/ var currentDeps;
/******/ var outerResolve;
/******/ var reject;
/******/ var promise = new
Promise((resolve, rej) => {
/******/ reject =
rej;
/******/
	outerResolve = resolve;
/******/ });
/******/
	promise[webpackExports] = exports;
/******/
	promise[webpackQueues] = (fn) =>
(queue && fn(queue), depQueues.forEach(fn),
promise["catch"](x => {}));
/******/ module.exports =
promise;
/******/ body((deps) => {
/******/
	currentDeps = wrapDeps(deps);
/******/ var fn;
/******/ var
getResult = () => (currentDeps.map((d) => {
/******/
	if(d[webpackError]) throw
d[webpackError];
/******/
	return d[webpackExports];
/******/ }))
/******/ var
promise = new Promise((resolve) => {
/******/ fn
= () => (resolve(getResult));
/******/ fn.r
= 0;
/******/ var
fnQueue = (q) => (q !== queue &&
!depQueues.has(q) && (depQueues.add(q), q
&& !q.d && (fn.r++, q.push(fn))));
/******/
	currentDeps.map((dep) =>
(dep[webpackQueues](fnQueue)));
/******/ });
/******/ return fn.r
? promise : getResult();
/******/ }, (err) => ((err ?
reject(promise[webpackError] = err) :
outerResolve(exports)), resolveQueue(queue)));
/******/ queue &&
(queue.d = 0);
/******/ };
/******/ })();
/******/
/******/ /* webpack/runtime/define
property getters */
/******/ (() => {
/******/ // define getter functions
for harmony exports
/******/ __webpack_require__.d
= (exports, definition) => {
/******/ for(var key in
definition) {
```

```
/******/
        if(__webpack_require__.o(definition, key) && !__webpack_require__.o(exports, key)) {
/******/
        Object.defineProperty(exports, key, { enumerable: true, get: definition[key] });
/******/                                  }
/******/                          }
/******/                  };
/******/         })();
/******/
/******/         /* webpack/runtime/hasOwnProperty shorthand */
/******/         (() => {
/******/                  __webpack_require__.o = (obj, prop) => (Object.prototype.hasOwnProperty.call(obj, prop))
/******/         })();
/******/
/******/         /* webpack/runtime/make namespace object */
/******/         (() => {
/******/                  // define __esModule on exports
/******/                  __webpack_require__.r = (exports) => {
/******/                          if(typeof Symbol !== 'undefined' && Symbol.toStringTag) {
/******/
        Object.defineProperty(exports, Symbol.toStringTag, { value: 'Module' });
/******/                          }
/******/
        Object.defineProperty(exports, '__esModule', { value: true });
/******/                  };
/******/         })();
/******/
/************************************************************************/
/******/
/******/         // startup
/******/         // Load entry module and return exports
/******/         // This entry module can't be inlined because the eval devtool is used.
/******/         var __webpack_exports__ = __webpack_require__("./src/js/app.js");
/******/
/******/ })()
;
```

pro/client/dist/account.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/style.min.css">
  <title>Акаунт</title>
</head>
  <body>
    <header class="header">
  <nav id="navbar" class="navbar">
    <a href="post.html">Пости</a>
  </nav>
  <div id="account-management" class="account-management"></div>
</header>
      <main>
        <div class="profile">
          <div class="profile-username" id="profile-username">
            Назва акаунту:
          </div>
          <div class="profile-info">
            <p class="profile-info-email" id="profile-info-email">Пошта: </p>

            <ul class="profile-info-subscribedTo" id="profile-info-subscribedTo">
              Акаунти на які підписаний:
            </ul>

            <label>
              Теги на які підписаний:
              <input id="profile-info-tags" type="text" readonly>
            </label>
          </div>
        </div>

        <button class='action-button' id="action-button"></button>

        <p>Пости користувача</p>
```

```
        <ul class="posts-list" id="posts-list">

        </ul>
      </main>
      <footer>
   <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
   <script src="js/app.min.js"></script>
   <p class="copyright">&amp;#169; Всі права захищені 2023</p>
</footer>
      <script type="module" src="./js/tasks/account.js"></script>
   </body>
</html>
```

pro/client/dist/index.html

```
<!DOCTYPE html>
<html lang="en">
   <head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="css/style.min.css">
   <title>Авторизація</title>
</head>
   <body>
      <header class="header">
   <nav id="navbar" class="navbar">
      <a href="post.html">Пости</a>
   </nav>
   <div id="account-management" class="account-management"></div>
</header>
      <main>
                        <form id="form-login">
                              <label for="username">Username користувача:</label>
                              <input type="text" id="username" name="username" required>

                              <label for="password">Пароль:</label>
                              <input type="password" id="password" name="password" required>

                              <button type="submit">Увійти</button>
            <a href="./register.html">Ще немає акаунту?</a>
                        </form>
      </main>
      <footer>
   <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
   <script src="js/app.min.js"></script>
   <p class="copyright">&amp;#169; Всі права захищені 2023</p>
</footer>
      <script type="module" src="./js/tasks/login.js"></script>
   </body>
</html>
```

pro/client/dist/post.html

```
<!DOCTYPE html>
<html lang="en">
   <head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="css/style.min.css">
   <title>Пости</title>
</head>
   <body>
      <header class="header">
   <nav id="navbar" class="navbar">
      <a href="post.html">Пости</a>
   </nav>
   <div id="account-management" class="account-management"></div>
</header>
      <main>
         <form class="create-post" id="create-post">
            <p>Створити новий пост</p>

            <label for="new-post-text">Текст посту</label>
```

```
            <textarea name="new-post-text" id="new-post-text" required></textarea>

            <input type="file" name="new-post-image"
            accept="image/jpeg, image/png, image/gif">

            <label for="new-post-tags">Теги посту</label>
            <input id="new-post-tags" type="text" name="new-post-tags"
            placeholder="#новина #розваги #твіч" required>

            <button type="submit">Створити пост</button>
        </form>

        <ul class="posts-list" id="posts-list"></ul>
    </main>
    <footer>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script src="js/app.min.js"></script>
  <p class="copyright">&#169; Всі права захищені 2023</p>
</footer>
    <script type="module" src="./js/tasks/post.js"></script>
  </body>
</html>
```

pro/client/dist/postId.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/style.min.css">
  <title>Пост</title>
</head>
  <body>
    <header class="header">
  <nav id="navbar" class="navbar">
    <a href="post.html">Пости</a>
  </nav>
  <div id="account-management" class="account-management"></div>
</header>
    <main>
      <div class="post">

      </div>
    </main>
    <footer>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script src="js/app.min.js"></script>
  <p class="copyright">&#169; Всі права захищені 2023</p>
</footer>
    <script type="module" src="./js/tasks/postId.js"></script>
  </body>
</html>
```

pro/client/dist/quiz.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/style.min.css">
  <title>Тест</title>
</head>
  <body>
    <header class="header">
  <nav id="navbar" class="navbar">
    <a href="post.html">Пости</a>
  </nav>
  <div id="account-management" class="account-management"></div>
</header>
    <main>
      <div id="quiz">
        <div id="quiz-number"></div>
        <div id="quiz-question"></div>
        <div id="quiz-answer">
          <ul id="quiz-answer-list">
```

```
                <li id="quiz-answer-list-answer"></li>
            </ul>
          </div>
        </div>
      </main>
      <footer>
   <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
   <script src="js/app.min.js"></script>
   <p class="copyright">&amp;#169; Всі права захищені 2023</p>
</footer>
      <script type="module" src="./js/tasks/quiz.js"></script>
   </body>
</html>
```

pro/client/dist/register.html

```
<!DOCTYPE html>
<html lang="en">
   <head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="css/style.min.css">
   <title>Реєстрація</title>
</head>
   <body>
      <header class="header">
   <nav id="navbar" class="navbar">
      <a href="post.html">Пости</a>
   </nav>
   <div id="account-management" class="account-management"></div>
</header>
      <main>
                        <form id="form-register">
                            <label for="username">Назва акаунту:</label>
                            <input type="text" id="username" name="username" required>

            <label for="email">Пошта:</label>
                            <input type="email" id="email" class="email" name="email"required>

            <label for="password">Пароль:</label>
                            <input type="password" id="password" name="password" required>

            <div class="territory-approve flex">
               <input id="territory" type="checkbox" required>
               <label for="territory">Я визнаю Україну з тереторіями <a href="http://surl.li/gys" target="_blank">1991 року</a></label>
            </div>
            <div class="support-approve flex">
               <input id="support" type="checkbox" required>
               <label for="support">Я підтримую Україну в боротьбі з терористами</label>
            </div>
                            <button type="submit">Зареєструватися</button>
            <a href="./index.html">Вже є акаунту?</a>
                        </form>
      </main>
      <footer>
   <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
   <script src="js/app.min.js"></script>
   <p class="copyright">&amp;#169; Всі права захищені 2023</p>
</footer>
      <script type="module" src="./js/tasks/register.js"></script>
   </body>
</html>
```

pro/client/dist/gulp/config/path.js

```
import * as nodePath from 'path'
const rootFolder = nodePath.basename(nodePath.resolve());

const buildFolder = `./dist`;
const srcFolder = `./src`;
```

```
export const path = {
   build: {
      js: `${buildFolder}/js/tasks`,
      mainJs: `${buildFolder}/js/`,
      images: `${buildFolder}/img/`,
      css: `${buildFolder}/css/`,
      html: `${buildFolder}/`,
      files: `${buildFolder}/files`
   },
   src: {
      images:
`${srcFolder}/img/**/*.{jpg,jpeg,png,gif,webp
}`,
      svg: `${srcFolder}/img/**/*.svg`,
      mainJs: `${srcFolder}/js/*.js`,
      js: `${srcFolder}/js/tasks/*.js`,
      css: `${srcFolder}/css/**/*.css`,
      html: `${srcFolder}/*.html`,
      files: `${srcFolder}/files/**/*.*`
   },
   watch: {
      images:
`${srcFolder}/img/**/*.{jpg,jpeg,png,gif,webp
,svg,ico}`,
      js: `${srcFolder}/js/**/*.js`,
      css: `${srcFolder}/css/**/*.css`,
      html: `${srcFolder}/**/*.html`,
      files: `${srcFolder}/files/**/*.*`
   },
   clean: buildFolder,
   buildFolder: buildFolder,
   srcFolder: srcFolder,
   rootFolder: rootFolder,
   ftp: `diplomadut.zzz.com.ua`
}
```

pro/client/dist/gulp/config/plugins.js

```
import replace from "gulp-replace"
import browsersync from "browser-sync"
import newer from "gulp-newer";
import ifPlugin from "gulp-if";

export const plugins = {
   replace: replace,
   browsersync: browsersync,
   newer: newer,
   if: ifPlugin
}
```