

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри )

**Пояснювальна записка**

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ КОМПЛЕКСНИХ СИСТЕМНИХ  
ПІДХОДІВ ДО РОЗРОБКИ ВЕБЗАСТОСУНКІВ НА ПРИКЛАДІ  
ФОТОХОСТИНГУ  
RESEARCH AND IMPLEMENTATION OF COMPLEX SYSTEM  
APPROACHES TO THE DEVELOPMENT OF WEB APPLICATIONS BY THE  
EXAMPLE OF PHOTO HOSTING

Виконав: студент 2 курсу, групи КНТ-122м  
Спеціальності 121 Інженерія програмного  
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КЛОКОВ А.О.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**Національний університет «Запорізька політехніка»**  
 (повне найменування закладу вищої освіти)

Інститут, факультет КНТ  
 Кафедра програмних засобів  
 Ступінь вищої освіти магістр  
 Спеціальність 121 Інженерія програмного забезпечення  
 (код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення  
 (назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ПЗ, д.т.н, проф.  
Сергій СУББОТІН  
 “      ”                                  20   року

**З А В Д А Н Н Я**

**НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

КЛОКОВА Артема Олександровича  
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та реалізація комплексних системних підходів до розробки вебзастосунків на прикладі фотохостингу. Research and Implementation of Complex System Approaches to the Development of Web Applications by the Example of Photo Hosting

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна,  
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “14” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Життєвий цикл вебзастосунку. 2. Аналіз та постановка завдання. 3. Розробка інтерфейсу користувача. 4. Програмна реалізація вебзастосунку. 5. Тестування. 6. Керівництво користувача.

7. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

## 6. Консультанти розділів проєкту (роботи)

| Розділ              | ПРІЗВИЩЕ, ініціали та посада консультанта | Підпис, дата   |                           |
|---------------------|-------------------------------------------|----------------|---------------------------|
|                     |                                           | завдання видав | прийняв виконане завдання |
| 1-6 Основна частина | ГЛАДКОВА О.М., доцент                     |                |                           |
| Нормоконтроль       | БЄЛОВА А.В., асистент                     |                |                           |
|                     |                                           |                |                           |
|                     |                                           |                |                           |

7. Дата видачі завдання “08” вересня 2023 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проєкту (роботи)                                             | Строк виконання етапів проєкту ( роботи ) | Примітка     |
|-------|--------------------------------------------------------------------------------------|-------------------------------------------|--------------|
| 1     | Постановка завдання роботи                                                           | 1 тиждень                                 | Завдання, ТЗ |
| 2     | Планування виконання роботи                                                          | 3 тиждень                                 | Розділ 1     |
| 3     | Аналіз предметної області                                                            | 3-4 тижні                                 | Розділ 2     |
| 4     | Розробка інтерфейсу користувача                                                      | 5-6 тиждень                               | Розділ 3     |
| 5     | Розробка програми                                                                    | 7-8 тиждень                               | Розділ 4     |
| 6     | Тестування програми                                                                  | 9 тиждень                                 | Розділ 5     |
| 7     | Експериментальне дослідження програми                                                | 10 тиждень                                | Розділ 6     |
| 8     | Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування | 11 тиждень                                | Додатки      |
| 9     | Захист роботи                                                                        | 12 тиждень                                |              |
|       |                                                                                      |                                           |              |
|       |                                                                                      |                                           |              |
|       |                                                                                      |                                           |              |
|       |                                                                                      |                                           |              |

Студент(ка)

\_\_\_\_\_ Артем КЛОКОВ  
( підпис ) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

\_\_\_\_\_ Ольга ГЛАДКОВА  
( підпис ) (Ім'я ПРІЗВИЩЕ)

## РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:  
123с., 1 табл., 53 рис., 2 дод., 21 джерел.

ФОТОХОСТИНГ, ФОТО, WEB, TYPESCRIPT, JAVASCRIPT,  
ANGULAR, EXPRESS.JS.

Об'єкт дослідження – процес розробки вебзастосунку.

Предмет дослідження – комплексні системні підходи до розробки вебзастосунку.

Мета роботи полягає в дослідженні комплексних системних підходів до розробки вебзастосунків на прикладі фотохостингу, розгляд етапів та ролей у ході розробки.

Матеріали, методи та технічні засоби: принципи створення користувацького інтерфейсу, структурне, функціональне та об'єктно-орієнтоване програмування, мови програмування TypeScript та JavaScript, фреймворки Angular та Express.js, методи та засоби тестування програмного забезпечення, персональний комп'ютер з чіп-сетом Ryzen 5 4600p під управлінням операційної системи Microsoft Windows 11.

Результати. Реалізовано проєкт у вигляді вебзастосунку фотохостингу, призначеного для перегляду та публікації зображень й текстового супроводження, з описом кожного етапу розробки та експертних ролей.

Висновки. Досліджено повний цикл розробки вебзастосунків, виявлено та вивчено ролі учасників процесу розробки на прикладі реалізації вебзастосунку фотохостингу, призначеного для публікації та перегляду зображень з текстовим супроводом.

Галузь використання – процес розробки вебзастосунків.

## ABSTRACT

Explanatory note to the diploma qualifying work of the master: 123 pages, 1 tables, 53 figures, 2 appendixes, 21 sources.

PHOTO HOSTING, PHOTO, WEB, TYPESCRIPT, JAVASCRIPT, ANGULAR, EXPRESS.JS.

The object of the study is the web application development process.

The subject of research is complex system approaches to web application development.

The purpose of the work is to study complex system approaches to the development of web applications using the example of photo hosting, to consider the stages and roles during development.

Materials, methods and technical means: principles of creating a user interface, structural, functional and object-oriented programming, TypeScript and JavaScript programming language, Angular and Express.js frameworks, methods and means of software testing, a personal computer with Ryzen 5 4600h chipset running Microsoft Windows 11 operating system.

Results. The project was implemented in the form of a photo hosting web application designed for viewing and publishing images and text support, with a description of each stage of development and expert roles.

Conclusions. The full cycle of web application development was studied, the roles of participants in the development process were identified and studied using the example of the implementation of a photo hosting web application designed for publishing and viewing images with text.

Field of use – web application development process.

## ЗМІСТ

|                                                   |    |
|---------------------------------------------------|----|
|                                                   | С. |
| Перелік скорочень та умовних позначок .....       | 8  |
| Вступ.....                                        | 9  |
| 1 Життєвий цикл вебзастосунку .....               | 11 |
| 1.1 Етапи розробки вебзастосунку .....            | 11 |
| 1.2 Експертиза в розробці вебзастосунку .....     | 13 |
| 1.2.1 Бізнес аналітик та проектний менеджер ..... | 14 |
| 1.2.2 Вебдизайнер та юзабіліті-фахівець .....     | 16 |
| 1.2.3 Веброзробник та архітектор застосунку ..... | 17 |
| 1.2.4 Тестувальник .....                          | 18 |
| 1.3 Висновки за розділом .....                    | 19 |
| 2 Аналіз та постановка завдання .....             | 20 |
| 2.1 Аналіз предметної області .....               | 20 |
| 2.2 Огляд існуючих рішень .....                   | 20 |
| 2.2.1 Pinterest.....                              | 20 |
| 2.2.2 Instagram.....                              | 25 |
| 2.2.3 YouTube.....                                | 28 |
| 2.3 Функціональні вимоги до вебзастосунку .....   | 31 |
| 2.4 Прототипування вебзастосунку.....             | 33 |
| 2.5 Висновки за розділом .....                    | 35 |
| 3 Розробка інтерфейсу користувача .....           | 37 |
| 3.1 Рекомендації щодо інтерфейсу користувача..... | 37 |
| 3.2 Висновки за розділом .....                    | 39 |
| 4 Програмна реалізація вебзастосунку .....        | 40 |
| 4.1 Визначення архітектури застосунку .....       | 40 |
| 4.2 Обґрунтування засобів розробки.....           | 41 |

|                                                   |     |
|---------------------------------------------------|-----|
|                                                   | 7   |
| 4.2.1 Вибір середовища розробки.....              | 43  |
| 4.2.2 Вибір системи керування базою даних.....    | 44  |
| 4.2.3 Вибір мов програмування та фреймворків..... | 45  |
| 4.3 Опис структури застосунку.....                | 49  |
| 4.3.1 Користувацька частина.....                  | 49  |
| 4.3.2 Серверна частина .....                      | 51  |
| 4.4 Висновки за розділом .....                    | 53  |
| 5 Тестування .....                                | 54  |
| 5.1 Етапи тестування.....                         | 54  |
| 5.2 Висновки за розділом .....                    | 54  |
| 6 Керівництво користувача .....                   | 56  |
| 6.1 Призначення вебзастосунку.....                | 56  |
| 6.2 Умови використання вебзастосунку .....        | 56  |
| 6.3 Опис процесу використання застосунку .....    | 57  |
| 6.4 Висновки за розділом .....                    | 62  |
| Висновки .....                                    | 63  |
| Перелік джерел посилань .....                     | 65  |
| Додаток А Текст програми.....                     | 67  |
| Додаток Б Слайди презентації .....                | 117 |

## ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

|      |                                             |
|------|---------------------------------------------|
| API  | – Application Programming Interface;        |
| BE   | – Back-end, серверна частина застосунку;    |
| BA   | – Business Analytics;                       |
| CSS  | – Cascading Style Sheets;                   |
| FE   | – Front-end, клієнтська частина застосунку; |
| HTML | – Hypertext Markup Language;                |
| IDE  | – Integrated development environment;       |
| JS   | – JavaScript;                               |
| PM   | – Project Manager;                          |
| SPA  | – Single Page Application;                  |
| TS   | – TypeScript;                               |
| UI   | – User Interface;                           |
| UX   | – User Experience;                          |
| QA   | – Quality Assurance;                        |
| БД   | – база даних;                               |
| ПЗ   | – програмне забезпечення;                   |
| СКБД | – система керування базами даних.           |

## ВСТУП

В нашому світі кожна людина є споживачем певних товарів чи послуг, фізичних, культурних, інформаційних тощо. Ці товари чи послуги мають задовольняти потреби споживача.

Бізнес надає товари чи послуги, та має на меті задоволення потреб користувача, і в свою чергу стикається з низкою потреб, таких як прибутковість, конкурентоспроможність та актуальність.

В сучасному світі, для зв'язку між споживачем та бізнесом найбільшу актуальність має програмне забезпечення (ПЗ). В свою чергу, ПЗ допомагає споживачеві більш зручно та швидко отримати результат. Бізнес, завдяки ПЗ спрощує комунікацію з користувачем. Це відноситься не тільки до застосунків, що мають на меті реалізацію товарів чи послуг:

- онлайн магазини;
- освітні платформи;
- онлайн кінотеатри;
- рекламні сайти та соціальні мережі.

Іноді ПЗ може й бути певною послугою чи товаром:

- конструктори сайтів;
- інформаційні платформи;
- CRM-системи;
- ігри.

Вебзастосунки мають певний пріоритет перед звичайними застосунками і для користувача, і для бізнеса.

Користувач не повинен встановлювати низку застосунків, щоб мати доступ до продукції різних компаній. Все, що йому необхідно – браузер, який зараз встановлюється за замовченням у більшості розповсюджених операційних систем. Отже, якщо користувач користується застосунком не постійно, а в

окремих випадках, йому зручніше використовувати саме вебверсію. Також, цільова аудиторія має доступ до застосунку з різних платформ, що є неодмінною перевагою. Використання вебзастосунків також впливає на розмір вільного простора у сховищі даних на комп'ютері або портативному девайсі користувача.

Бізнесу, який потребує застосунку, у більшості випадків також вигідніше починати з вебсайту. По-перше – це можливість охопити більшу аудиторію за рахунок мультиплатформеності вебзастосунку. По-друге – на основі статистичних даних, що збираються при використанні вебзастосунку, можна зробити певні висновки і правки, застосувавши їх при розробці мобільних чи десктопних версій застосунків. Це стосується і дизайну, і функціоналу. Також, при розробці окремих застосунків після реалізації вебверсії, можна використовувати ті самі сховища даних, моделі баз даних, та API вже створеного сайту, що значно спрощує процес проектування та розробки.

## **1 ЖИТТЄВИЙ ЦИКЛ ВЕБЗАСТОСУНКУ**

Життєвий цикл будь яких застосунків поділяється на певну кількість етапів, але має саме циклічний характер. Циклічність зумовлена тим, що, крім безпосередньої розробки, програмні застосунки потребують постійної актуалізації технічної сторони, виправлення помилок, збільшення кількості функціональних можливостей, оптимізації. Над проектами застосунків зазвичай працює команда, що складається з певних ролей, де, представники кожної з них виконують конкретні задачі.

### **1.1 Етапи розробки вебзастосунку**

Загалом, життєвий цикл будь-якого програмного забезпечення можна описати за Agile-методологією (рис. 1.1), що ґрунтується на командній співпраці та постійному вдосконаленні. Можна виділити деяку кількість етапів в життєвому циклі програмного забезпечення:

- планування;
- проєктування ;
- розробка;
- тестування;
- розгортання;
- обслуговування.

Планування – початкова стадія існування програмного забезпечення. На цій стадії досліджується життєздатність та цінність проекту, можливість реалізації окремих його частин. В результаті, команда, що займається плануванням, має чітко визначену стратегію щодо програмного продукту, оцінку обсягу роботи та мінімальний життєздатний продукт, рішення, щодо ресурсів та технологій необхідних для реалізації проєкту.



Рисунок 1.1 – Життєвий цикл ПЗ

Етап проєктування націлений на створення макетів та прототипів, визначення архітектури та створення дизайнерами каркасів користувацького інтерфейсу. Результат – можливість візуалізації кінцевого продукту з інформацією щодо користувацького досвіду.

За результатами проєктування починається безпосередньо розробка, тобто ітеративна послідовна реалізація окремих частин проєкту. Під час цього етапу задіяні всі можливі ролі. Команда, попередньо надавши окремим задачам пріоритет, на основі їх важливості та складності, поступово реалізують проєкт, до його робочої версії.

Після етапу розробки з'являється потреба в тому, щоб переконатися, чи працює застосунок належним чином, та чи відповідають його частини попередньо визначеним вимогам. Для цього тестувальники проводять низку заходів за різними методологіями тестування.

Наступний етап – розгортання. Команда розгортає проєкт в робочому середовищі, щоб провести фінальне тестування та налагодження застосунку. На цьому етапі ми маємо першу релізну версію застосунку, умови використання якого максимально наближені до реальних.

Останнім етапом є етап обслуговування, адже ПЗ має постійно оновлюватись, щоб залишатися актуальним, задовольняти потреби цільової аудиторії протягом довгострокового періоду та виконувати необхідні задачі бізнесу.

## **1.2 Експертиза в розробці вебзастосунку**

Експертиза – діяльність, метою якої є аналіз, дослідження, перевірка та оцінка об'єкту експертизи і підготовка обґрунтованих висновків для прийняття рішень щодо таких об'єктів.

Команда розробників програмного забезпечення складається не тільки з інженерів та дизайнерів. Щоб задовольнити потреби клієнта, в ролі якого виступає бізнес, потрібно пройти всі етапи розробки. Для прикладу: на першому етапі проектування активними ролями є бізнес аналітики, діяльність їх разом з проектним менеджером супроводжує увесь цикл його існування, а ось на стадії тестування більш за всіх беруть участь тестувальники. Задачі кожної з ролей різняться та мають окрему експертизу. Так, можна виділити наступний список ключових ролей:

- бізнес аналітик;
- проектний менеджер;

- вебдизайнер;
- юзабіліті фахівець;
- архітектор програмного забезпечення;
- веброзробник;
- тестувальник.

Але, все ж таки, деякі ролі можна об'єднати, адже вони виконують споріднені функції у розробці застосунку. Часто спеціалісти обіймають кілька посад на проекті, що зменшує кількість взаємодії між ролями, уникаючи непорозумінь в постановці вимог.

### **1.2.1 Бізнес аналітик та проектний менеджер**

Ролі бізнес аналітика або ВА (від Business Analyst) та проектного менеджера або РМ (від Project Manager) в розробці вебзастосунку є ключовими під час всього циклу існування застосунку.

Головна задача ВА на проекті – комунікація з замовником, аналіз його вимог та потреб цільової аудиторії кінцевого продукту, в якості якого виступає сам вебзастосунок.

РМ у своїй роботі має на меті адаптувати існуючі вимоги під строки та можливості команди розробників.

ВА та РМ займаються й постановкою задач, визначаючи складність, важливість та строки для кожної з них. На основі аналізу складності та важливості вибудовується пріоритети, за якими в подальшому працює команда.

Надважливою частиною роботи цих ролей є й декомпозиція проекту – рекурсивне розбиття його на окремі, менші за об'ємом, кроки. Мета декомпозиції полягає в поступовому зменшенні задачі до примітивних складових.

Бізнес аналіз проводиться за допомогою методів, описаних в професійній літературі. У приклад можна привести такі методи, що використовуються найчастіше:

- мозковий штурм;
- словник даних та глосарій;
- інтерв'ю;
- метрики та ключові показники продуктивності;
- відстеження проблем;
- сценарії та варіанти використання;
- практичні заняття щодо виявлення вимог;
- діаграми потоків даних;
- моделювання організації;
- моделювання процесів;
- моделювання даних;
- аналіз бізнес-правил;
- аналіз ухвалення рішень;
- аналіз документів;
- аналіз нефункціональних вимог.

Ці методи та підходи, зазвичай, використовуються в поєднанні один з одним або одночасно, або послідовно. Деякі з них тісно пов'язані, і результат проведення одного з методів є вхідними даними для використання іншого.

З плином часу ролі ВА та РМ, не змінюючи своєї значущості, стають все менш формалізованими. Якщо раніше проекти супроводжувались великою кількістю документації, що складалася за суворими вимогами та з високим рівнем формальності, то зараз робота ВА та РМ більш опирається на створення схем, таблиць та прототипів, а також прямою взаємодією з замовником та командою розробників.

### 1.2.2 Вебдизайнер та юзабіліті-фахівець

Вебдизайнер, або UI-дизайнер (від User Interface), займається візуальною частиною взаємодії користувача та застосунку, користуючись експертними знаннями в області стенографії, колористики та інших прикладних художніх наук, а також власним розумінням візуальної естетики.

Юзабіліті-фахівець, або UX-дизайнер (від User Experience), вивчає очікування користувача, його поведінку при взаємодії з застосунком. Цей фахівець має на меті знайти найпростіші та найзрозуміліші засоби і шляхи взаємодії з інтерфейсом, для досягнення цілей користувача.

Зв'язок цих ролей, а також їх значення в розробці складно переоцінити. Вебдизайнери, які, часто, одночасно ще займають посаду юзабіліті-фахівця, мають створити зручний для використання, інтуїтивно зрозумілий і неперевантажений інтерфейс, що має актуальний і доцільний вигляд, відповідає цілі, тематиці та вимогам замовника, а також можливостям команди веброзробників.

Цей інтерфейс будується за прототипами та схемами, що складають ВА та РМ. Результат має відповідати не тільки вимогам, що ставить замовник, а й професійним вимогам та принципам. Це стосується розташування елементів на сторінці, навігації між сторінками та меню, кольорової гама тощо (трошки розписати).

Дизайнер, як UI так UX, має у своєму професійному арсеналі достатню кількість інструментів, починаючи від не умовного олівця, закінчуючи програмним оточенням, створеним спеціально для задоволення потреб у розробці інтерфейсу користувача.

### 1.2.3 Веброзробник та архітектор застосунку

Безпосередньо програмною реалізацією вебзастосунку займається команда розробників, що включає в себе інженерів програмного забезпечення та архітекторів застосунку.

Архітектура застосунку створюється на різних рівнях, та являє собою комплекс необхідних системних заходів, спрямованих на побудову проекту. На найвищому рівні проектується клієнт-серверна архітектура. Окремого проектування потребують бази даних та їх взаємодія, а також безпосередньо структура застосунку, що описує взаємодію модулів, компонентів тощо.

Для клієнт-серверної архітектури існує набір патернів, кожен з яких має і переваги і недоліки, та визначається індивідуально під особливості конкретного проекту. Ці патерни будуть розглянуті в одному з наступних пунктів, з обґрунтуванням вибору для вебзастосунку, що розробляється.

Структура застосунку та баз даних проектується під функціональні можливості застосунку, з урахуванням обраних систем керування базами даних, мов програмування, застосованих фреймворків.

Роль архітектора вебзастосунку в команді розробників має займати спеціаліст з великим інженерним досвідом.

Інженери програмного забезпечення, або розробники вебзастосунку, умовно поділяються на front-end та back-end спеціалістів. Перші займаються клієнтською частиною вебзастосунку, а другі, відповідно – серверною. Це розділення умовне, адже із досвідом, спеціалісти, так би мовити, переростають напрямлення. Інженерів, що займаються, або можуть займатися обома напрямленнями називають full-stack розробниками.

Для front-end характерні навички роботи з HTML, CSS, та скриптової мови програмування, як Java Script, а також різноманітними фреймворками,

бібліотеками. Це дозволяє верстати вебсторінку та реалізовувати функціональність інтерфейсу користувача.

Back-end зосереджений на написанні логіки застосунку, зокрема, роботи з даними. Для написання серверної частини є багата варіативність вибору мов програмування зі списку мов загального призначення. Також, спеціалістам цього напрямлення необхідні знання з керування базами даних, реляційними або нереляційними, що залежить від особливостей проекту.

#### **1.2.4 Тестувальник**

Тестувальник, або QA інженер – фахівець, що займається тестуванням готового застосунку на наявність помилок (помилки ПЗ прийнято називати багами). Тестування проходить з різними аспектами застосунку, а спеціалісти використовують різні підходи та парадигми.

Взаємодія тестувальника з командою проекту достатньо формалізована. QA інженери отримують чіткі вимоги до застосунку, та, якщо знаходять помилки, перенаправляють їх до команди розробників шляхом формування звітів, також відомих, як багрепорти.

Загалом, зону відповідальності тестувальників можна подати в виді наступного списку:

- розробка сценаріїв тестування ПЗ;
- розробка та вдосконалення автотестів;
- проведення спеціальних тестів на виявлення помилок;
- контроль якості виконання тестів;
- документація знайдених помилок з описом та наданням вичерпної інформації спеціалісту, в роботі якого ці помилки знайдені.

### 1.3 Висновки за розділом

В цьому розділі було проаналізовано життєвий цикл програмного забезпечення за Agile-методологією, розглянуто особливості етапів існування застосунків, обґрунтовано необхідність в циклічності цих етапів у проєктах. Також були вивчені ролі учасників створення програмного забезпечення, досліджено використання експертизи цих ролей, їх відмінності та необхідність в співпраці.

Таким чином, програмне забезпечення має 6 основних етапів існування, серед них: планування, проєктування, розробка, тестування, розгортання та підтримка. Ці етапи є базово необхідними для створення актуального, корисного, відлагодженого програмного продукту. На кожному етапі є необхідність створення цільової команди, задля досягнення успіху проєкту в цілому.

Експертні ролі, що існують під час життєвого циклу застосунку мають власні особливості, але деякі з них мають споріднені або залежні між собою функції. Так можна виділити наступні ключові ролі: бізнес аналітик, проєктний менеджер, вебдизайнер, юзабіліті фахівець, архітектор програмного забезпечення, веброзробник, тестувальник.

Надалі, проєкт, наведений для прикладу у цій роботі, буде підпорядковуватись визначеній структурі етапів та ролей, для більшої наглядності. Однак, через виконання проєкту однією особою, формальність виконання рольових функцій буде знижена. Що не завадить зробити конкретні висновки щодо комплексних підходів до створення вебзастосунку.

## **2 АНАЛІЗ ТА ПОСТАНОВКА ЗАВДАННЯ**

### **2.1 Аналіз предметної області**

Проект, на прикладі якого проводиться дослідження – фотохостинг. Визначаючи ключові функції вебзастосунку, що знаходиться у розробці, стає зрозуміло, що крім функцій зберігання та перегляду картинок, необхідно розробити додатковий функціонал, щоб застосунок був актуальним. Перше, що приходить на думку, коли мова йде про актуалізацію – інтерактивність. Найкращим рішенням цієї проблеми є функціонал соціальної мережі, де користувач може отримати оцінку, відслідкувати метрики популярності та актуальності контенту. Контентом в даному випадку виступають саме фото та картинки. Користуючись досвідом соціальних мереж, розвинутих відеохостингів та платформ, можна прийти до висновку, що основною метрикою для відстеження вподобань будуть виступати лайки.

### **2.2 Огляд існуючих рішень**

Сучасні користувачі досить активні в соціальних мережах, і не віддають перевагу окремій, адже вони мають різне направлення, а досвід користувача значно відрізняється. Для огляду існуючих рішень, серед популярних застосунків соціальних мереж було обрано контенто напрямлені, які так чи інакше можна віднести до категорії хостингів візуального контенту.

#### **2.2.1 Pinterest**

Pinterest – яскравий приклад фотохостингу та соціального інтернет-сервісу, що розроблявся з 2009 року однойменною компанією, засновником якої

став Бен Зільберман, громадянин Сполучених Штатів Америки. Бета версія застосунку була запущена у 2010 році.

Після розгортання сторінки, неавторизованому користувачеві не відкривається основний функціонал сайту рис. 2.1 [1].



Рисунок 2.1 – Головна сторінка Pinterest для неавторизованого користувача

Але, шляхом деяких маніпуляцій, неавторизований користувач все таки може потрапити на сторінку з добірками та можливістю пошуку зображень за ключовими словами, кольором, автором тощо. рис. 2.2 [1].

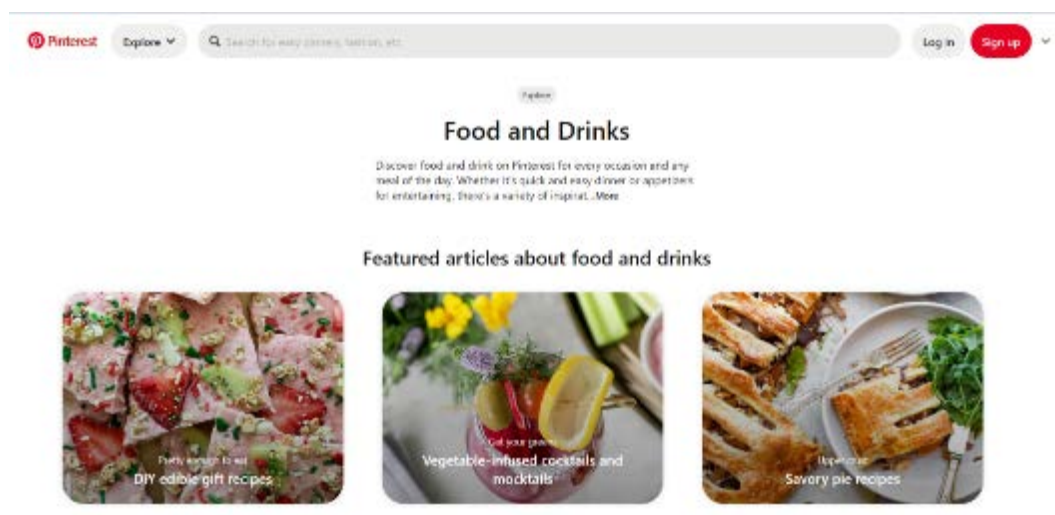


Рисунок 2.2 – Сторінка контенту Pinterest для неавторизованого користувача

Користувач має можливість реєстрації рис. 2.3 [1] та авторизації рис. 2.4 [1]. Під час авторизації і реєстрації існує можливість використання облікових записів Facebook або Google, для спрощення і пришвидшення цих процесів.

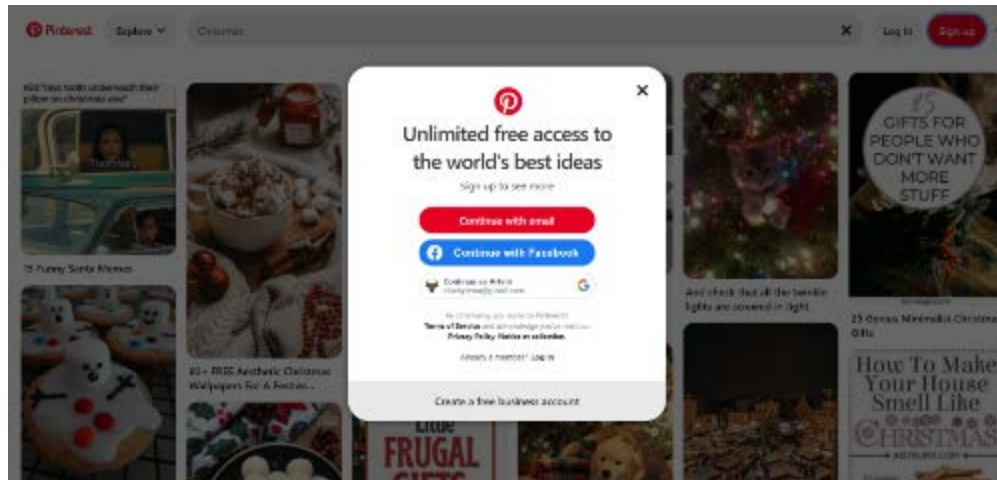


Рисунок 2.3 – Вікно реєстрації нового користувача

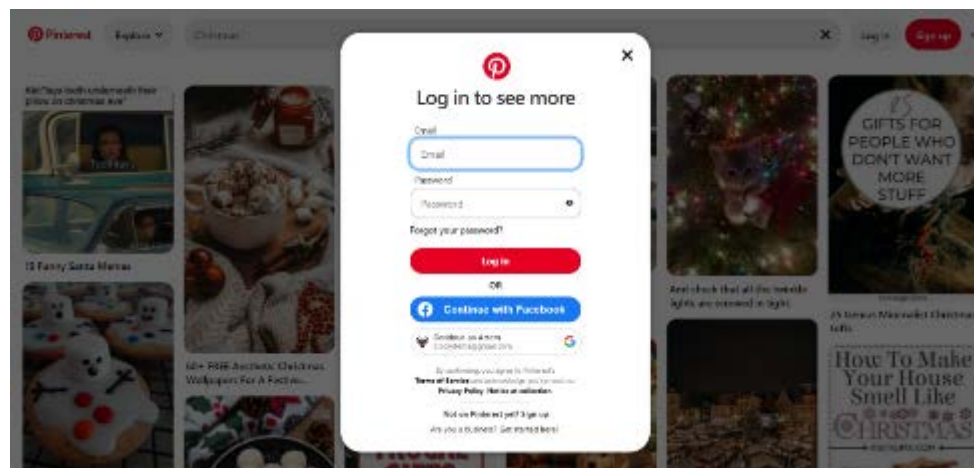


Рисунок 2.4 – Вікно авторизації існуючого користувача

Після авторизації, на головній сторінці сайту користувач може бачити добірку зображень на основі аналізу його історії пошуку зображень, вподобань, улюблених кольорів тощо. рис. 2.5 [1].

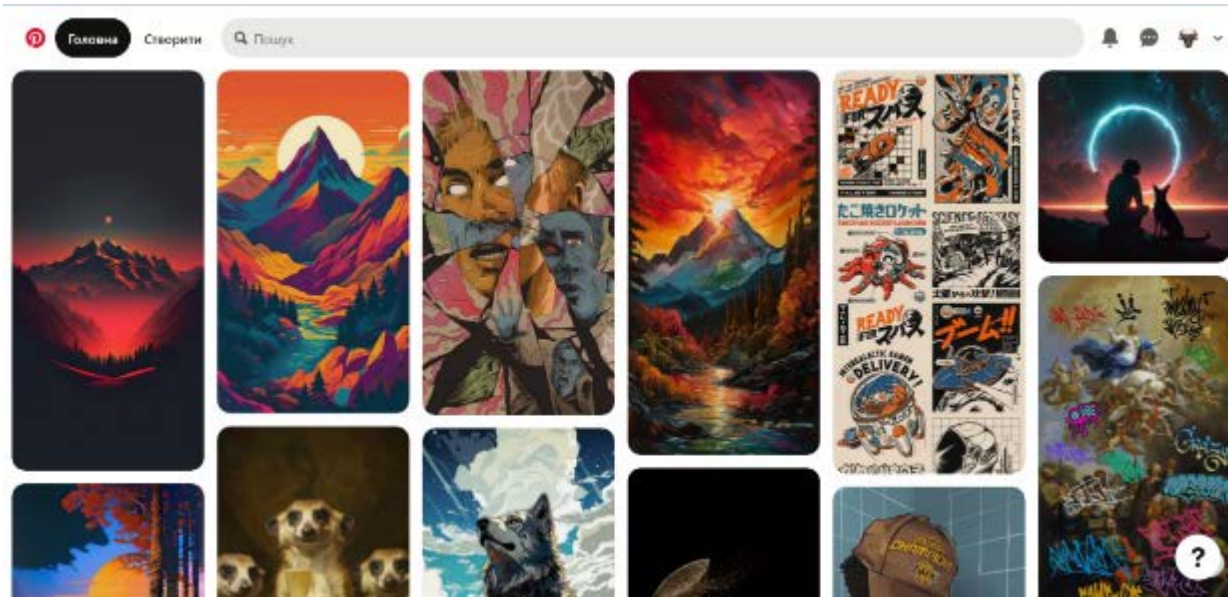


Рисунок 2.5 – Головна сторінка Pinterest для авторизації користувача

Будь-який користувач може здійснювати пошук рис. 2.6 [1]. Пошук відбувається за спеціальними алгоритмами та має дуже розвинуту внутрішню структуру. Так, зображення можна знайти за назвою, контекстом, кольоровою гамою, автором.

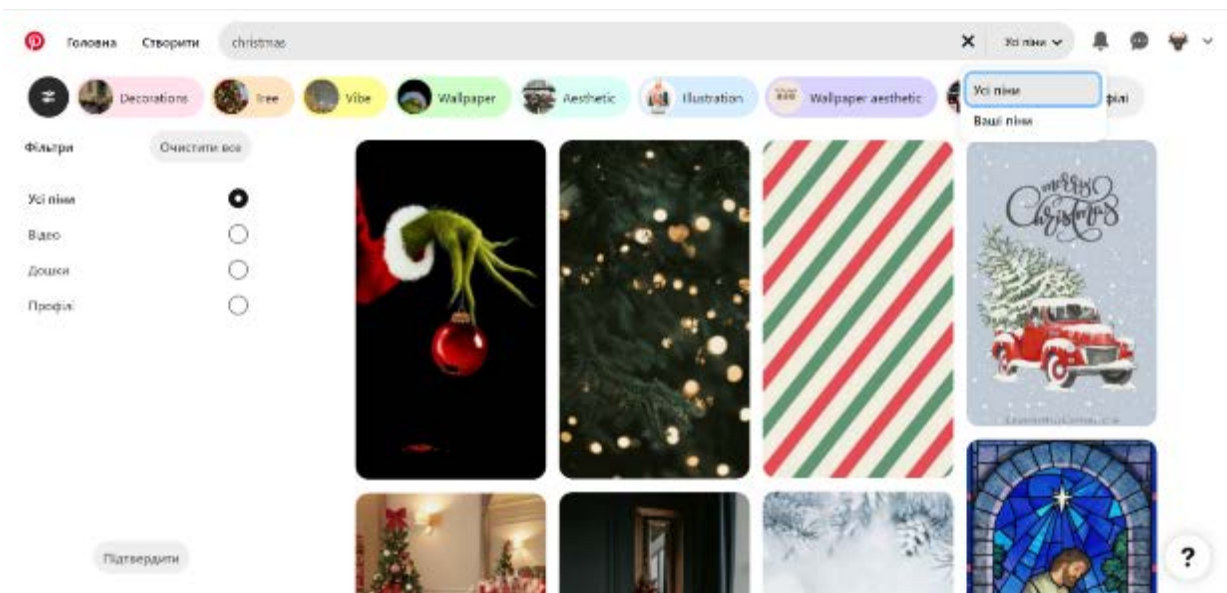


Рисунок 2.6 – Пошук та пошукові фільтри Pinterest

Авторизований користувач має низку переваг. Окрім індивідуальних добірок за вподобаннями, він може створювати власні добірки, додавати свої зображення, об'єднувати їх рис. 2.7 [1]. Авторизований користувач має можливість тонкого налаштування профілю рис. 2.8 [1], починаючи з інформації про себе, закінчуючи правилами рекомендацій та мовою сайту, вибір якої прив'язується саме до аккаунта.

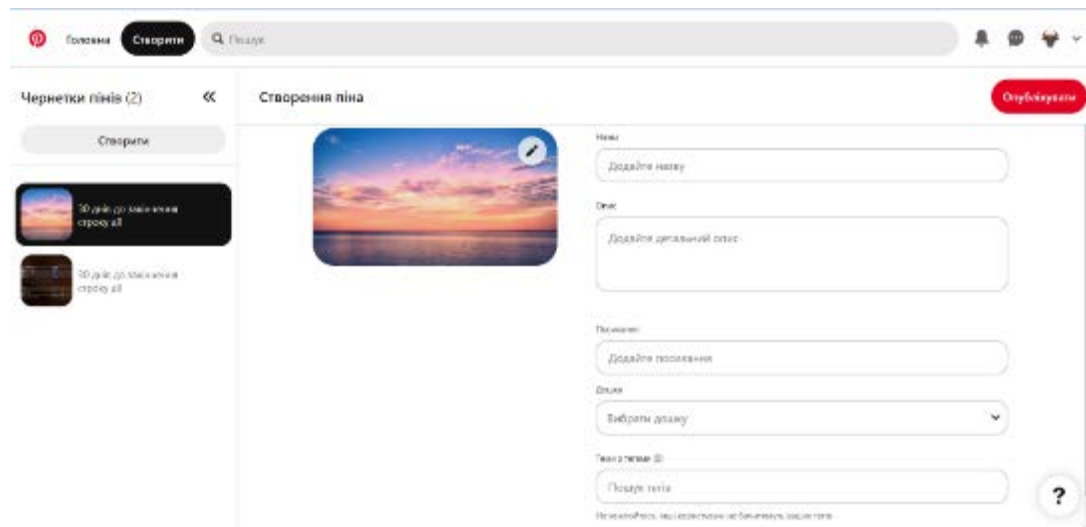


Рисунок 2.7 – Сторінка публікації свого контенту на Pinterest

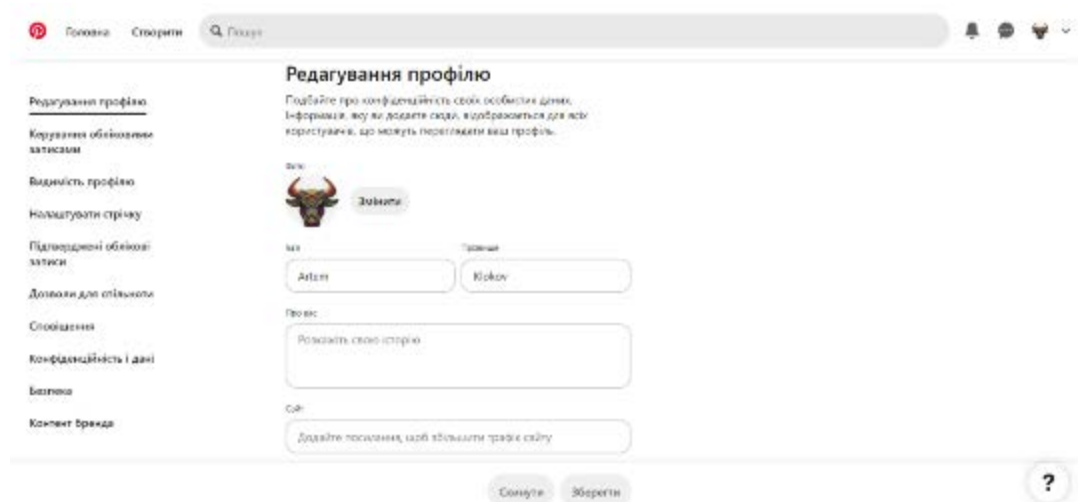


Рисунок 2.8 – Сторінка редагування інформації та зміни налаштувань власного профілю

### 2.2.2 Instagram

Instagram – популярна соціальна мережа, головною ідеєю якої є публікація та обмін світлинами між користувачами. Застосунок був створений американцями Кевіном Систром та Майком Крігером, для мобільної платформи IOS в 2010 році. Пізніше, застосунок вийшов на Android. А після купівлі прав на застосунок корпорацією Facebook в 2012 році, велася розробка веб версії. Зараз, Instagram є одним з флагманських проєктів компанії Meta Platforms.

Instagram вимагає від користувача обов'язкової авторизації. Авторизований користувач має низку можливостей, таких як перегляд рекомендацій рис. 2.9 [2], стрічки своїх підписок рис. 2.10 [2], особистого профілю рис. 2.11 [2], та профілів інших користувачів рис. 2.12 [2].

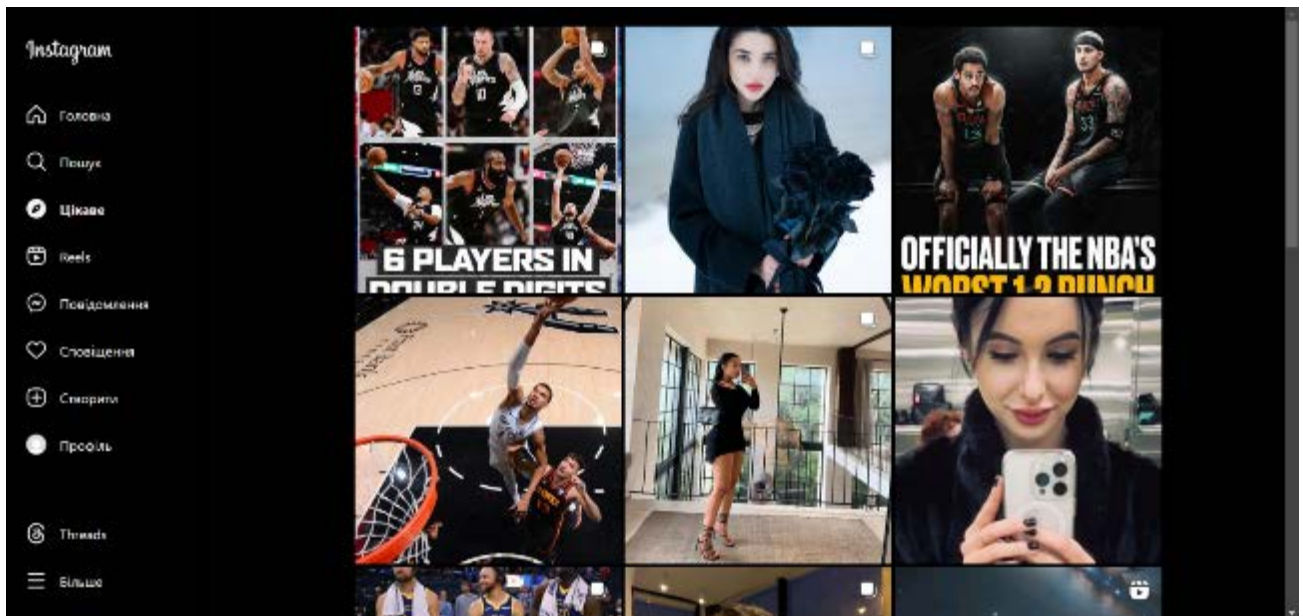


Рисунок 2.9 – Рекомендації Instagram



Під постами можна ставити лайки, додавати їх в обране та залишати коментарі. рис. 2.13 [2].



Рисунок 2.13 – Пост з відмітками та коментарями

У власному профілі можна керувати підписками та підписниками, обирати видимі фотографії чи пости, додавати динамічні Stories рис. , що зникають протягом доби, редагувати інформацію профілю, змінювати налаштування.

Основною функцією Instagram є можливість створювати власні пости з фото 2.14 [2]. При цьому, створюючи пост, до фотографії можна додати допис, відмітити інших зареєстрованих користувачів на фото, накласти фільтри на зображення, об'єднати декілька зображень у карусель.

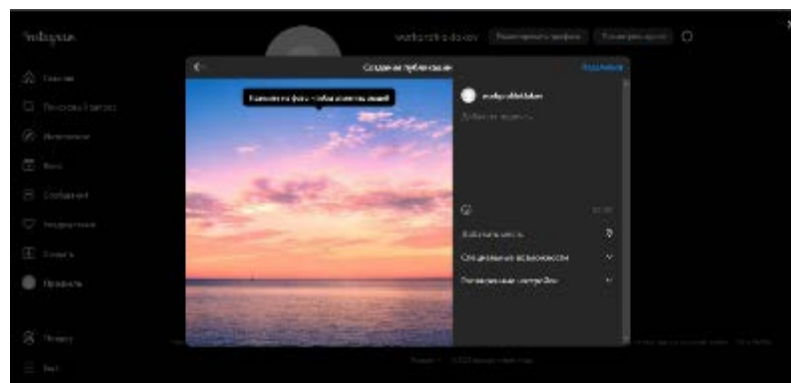


Рисунок 2.14 – Створення власного поста в Instagram

### 2.2.3 YouTube

YouTube – це відеохостинг, на відміну від інших наведених прикладів, але він найкраще показує можливість поєднати функції хостингу та соціальною мережі. За допомогою цього сервісу користувач має можливість завантажувати, дивитися, оцінювати, коментувати відео. Застосунок створено в Сполучених Штатах Стівом Ченом, Чадом Хьордлі, Джаведом Карімом в 2005 році. А вже в 2006 році застосунок куплений компанією Google.

Неавторизованого користувача застосунок YouTube зустрічає сторінкою з рекомендаціями щодо актуальних відео та відомих авторів рис. 2.15 [3]. Ці рекомендації називають тренди YouTube, вони формуються з інтересів аудиторії певного регіону, до якого належить користувач. Авторизований користувач має в цьому списку ще й індивідуальні рекомендації, підібрані за його інтересами, історією переглядів та пошуку.

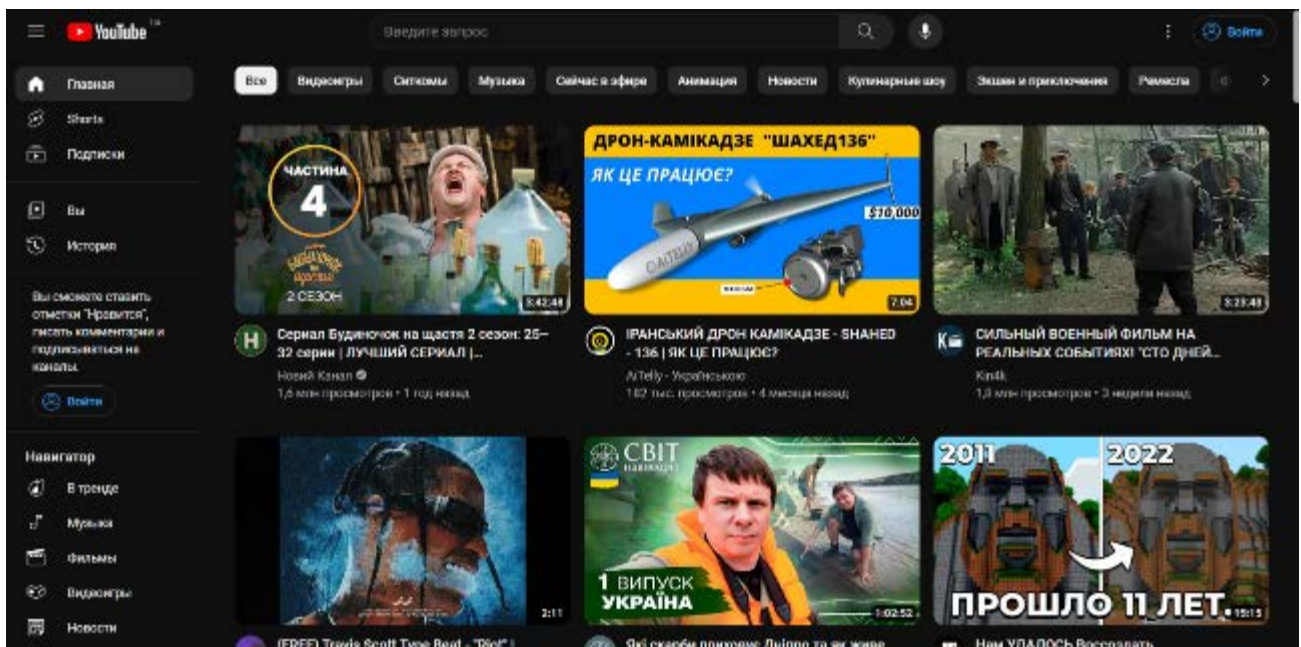


Рисунок 2.15 – Сторінка YouTube з регіональними рекомендаціями для неавторизованого користувача

Як споживач контенту, авторизований та неавторизований користувач мають майже рівні можливості. Відмінності полягають тільки в можливості написання коментарів, оцінки відео, підписки на окремі канали.

Користувачі мають можливість дивитися відео з рекомендацій, користуватися пошуком за автором або назвою відео рис. 2.16 [3], переглядати окремі канали з відео рис. 2.17 [3].

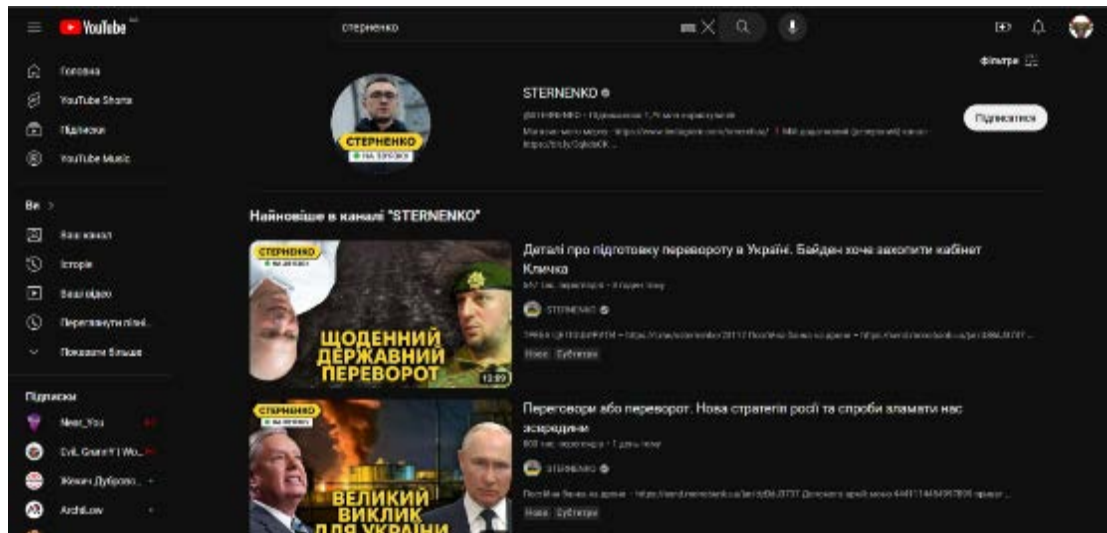


Рисунок 2.16 – Пошуковий запит в YouTube



Рисунок 2.17 – Сторінка каналу STERNENKO на YouTube

Канали – це і є облікові записи, або особисті сторінки авторизованих користувачів. Використовуються канали для публікації відео. Застосунок YouTube має спеціальні інструменти для керування каналами та саме публікації контенту 2.18 [3].

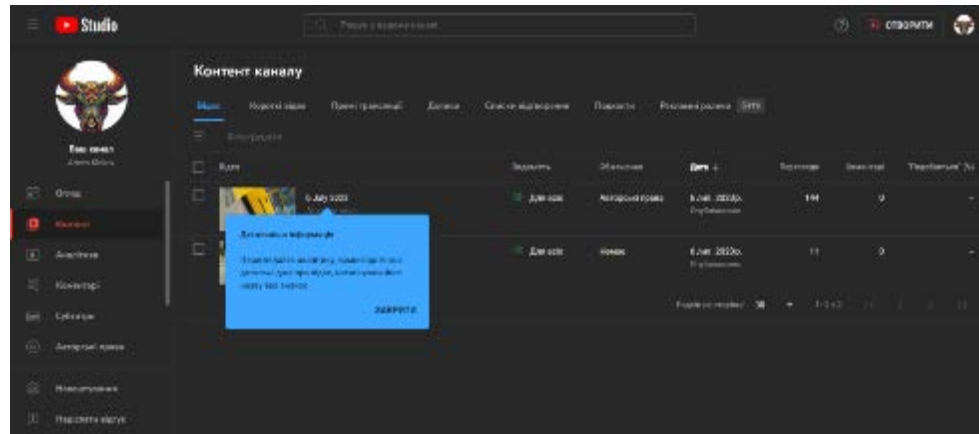


Рисунок 2.18 – Інструмент для публікації та адміністрування відео YouTube Studio

Авторизація рис. 2.19 [3] на YouTube відбувається тільки за допомогою вже існуючого облікового запису Google. Окремої реєстрації на YouTube не існує. Це зумовлено належністю цього сервісу до переліку сервісів компанії Google.

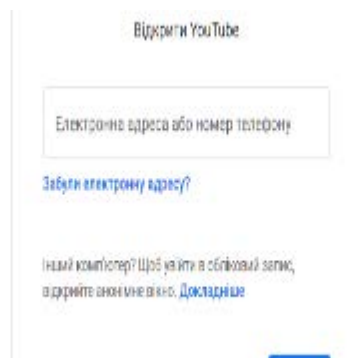


Рисунок 2.19 – Сторінка авторизації в YouTube та інших сервісах компанії Google

Сучасний YouTube зберігає відео користувачів, умовно поділяючи їх на формати. Так, цей сервіс, крім зберігання звичайних відео, виділяє в окремий формат YouTube Shorts короткі ролики у вертикальній орієнтації рис. 2.20 [3].

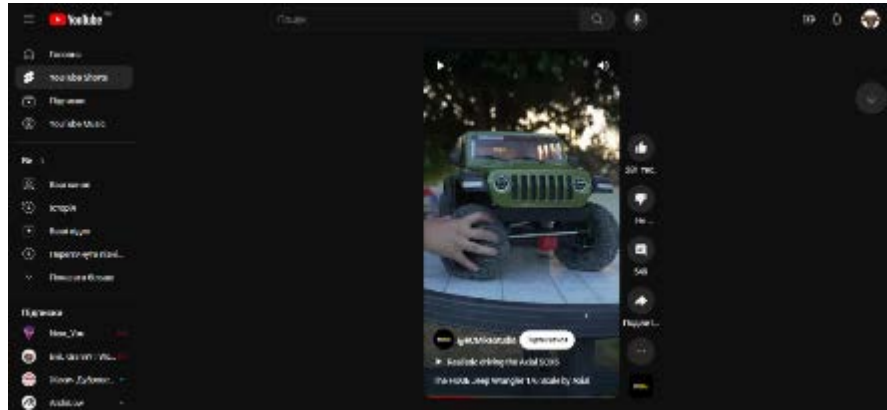


Рисунок 2.20 – YouTube Shorts

Також, YouTube, крім відеохостингу, є і стримінговою платформою. Тобто, користувачі транслювати зображення в реальному часі.

### 2.3 Функціональні вимоги до вебзастосунку

Проаналізувавши предметну область, на основі розглянутих існуючих рішень, висувуються мінімально необхідні функціональні вимоги, що забезпечать життєздатність проєкту. Отже, маємо наступний перелік функцій:

- реєстрація нового облікового запису користувача;
- авторизація користувача у існуючий обліковий запис;
- відображення на головному екрані фотографій, що були завантажені іншими користувачами;
- встановлення міток на зображення, що сподобались;
- відображення зображень із мітками вподобання;
- видалення раніше встановлених міток вподобання;

- пошук фотографій за ключовими словами або автором;
- завантаження власних фотографій із можливістю залишити допис і ключові слова;
- відображення власної сторінки користувача із завантаженими раніше фотографіями;
- відображення сторінок інших авторів із завантаженими ними фотографіями;
- фільтрація фотографій за датою створення.

Функціональні можливості застосунку напряду залежать від статусу користувача, а саме – чи авторизовано обліковий запис користувача рис. 2.21.

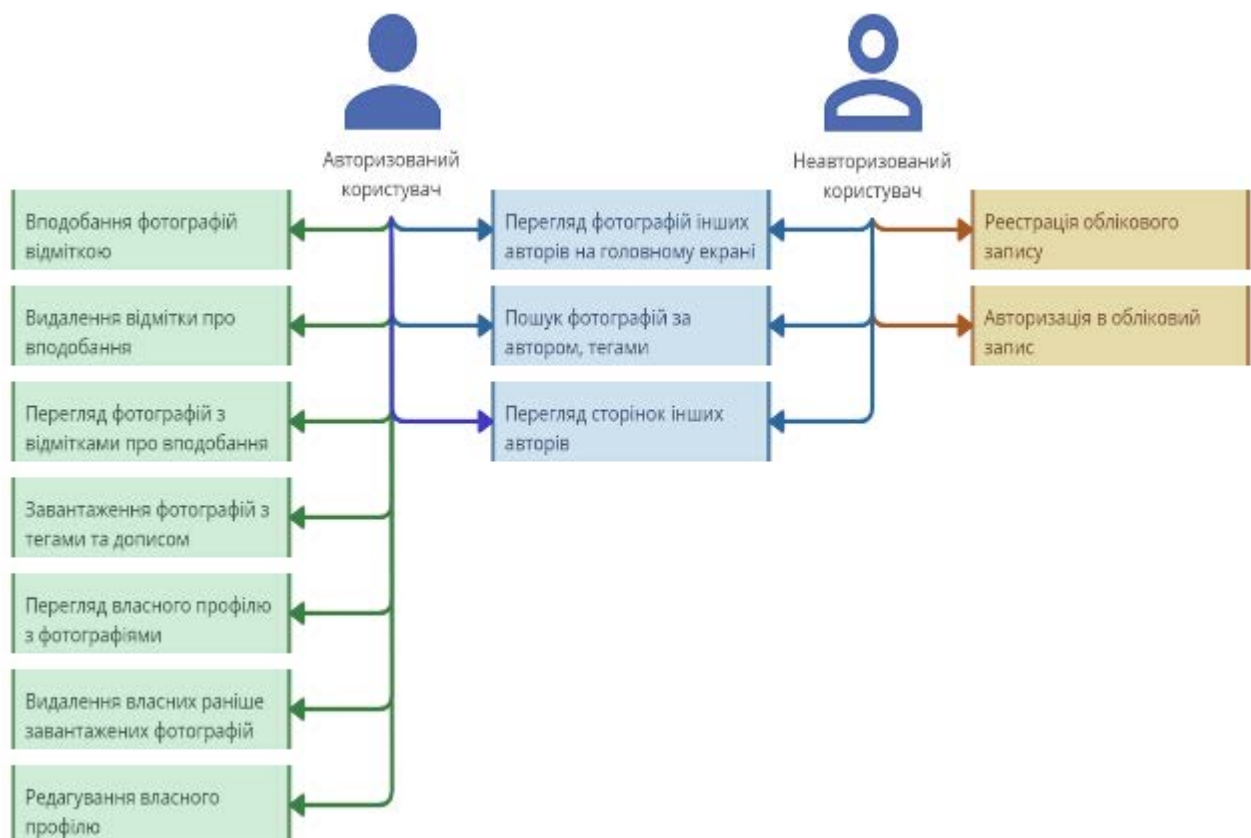


Рисунок 2.21 – Діаграма прецедентів користувача в залежності від його статусу

Функції споживача контенту, такі як: перегляд фотографій на головній сторінці, пошук фотографій, перегляд сторінок з фотографіями зареєстрованих користувачів –доступні кожному. А ось доступ до функціоналу умовних експерта та автора, серед якого: можливість оцінювати фотографії інших авторів, завантажувати власні фото, писати до них допис – є тільки в авторизованих користувачів. Звичайно, функції реєстрації та авторизації необхідні навпаки, тільки неавторизованим в свій обліковий запис.

## **2.4 Прототипування вебзастосунку**

Користуючись фотохостингом у вигляді вебзастосунку, користувач фокусується на споживанні візуального контенту, тобто зображень. З цією думкою, необхідно розробити застосунок, де в центрі уваги будуть фотографії.

Отже, планується розробка SPA застосунку [10]. Сторінка, канонічно, буде складатися з 3 частин: хедер, тіло, футер.

Хедер буде розміщувати у собі основну навігацію, та логотип у лівому верхньому куті. Він має мати два різні стани рис. 2.22, для авторизованого та неавторизованого користувача відповідно. Звертаючи увагу на функціонал, для неавторизованого користувача хедер буде мати елементи, розташовані один за одним в ряд:

- логотип, що одночасно виконує функцію кнопки повернення на домашню сторінку;
- поле введення для пошуку зображень за тегами або автором;
- кнопка авторизації в обліковий запис;
- кнопка реєстрації нового облікового запису.

Авторизований користувач, замість кнопок реєстрації та авторизації буде мати іконку вподобання, для переходу до зображень зі встановленою міткою

вподобання, та іконку власного профілю, для переходу на сторінку профілю або до його редагування за допомогою списку, що розгортається.



Рисунок 2.22 – Схема розташування елементів у хедері

Тіло буде складатися з двох блоків рис. 2.23. Верхній блок з’являється при переході на власний профіль, або профіль іншого користувача. У собі він містить аватар відповідного профілю, ім’я власника цього профілю, та надану ним інформацію. Нижній блок – блок з фотографіями, що завжди відображається, але змінює своє наповнення залежно від сценарію використання застосунку. Тобто, стандартно, у цьому полі відображатимуться останні додані фотографії, коли користувач переходить на сторінку профіля певного автора – зображення, завантажені цим автором, на сторінці власного профілю – власні завантаженні фотографії. Також цей блок буде змінювати своє наповнення в результаті використання пошуку.

Футер, зазвичай, містить у собі не дуже багато інформації, серед якої можуть бути контакти служби підтримки, основні посилання, інформація про компанію тощо. Це найменший блок, з видимістю якого можна експериментувати. Наприклад, футер може відображатися користувачу тільки при наведенні в виділену зону знизу сторінки, якщо він займає більше 10% сторінки. Або, якщо футер не несе на собі функціонального навантаження, посилань або контактної інформації, він може відображатися тільки після

прокрутки основного блоку. Серед цих варіантів, середній – стале відображення футеру у низу екрану без залежності від розмірів основного блоку.

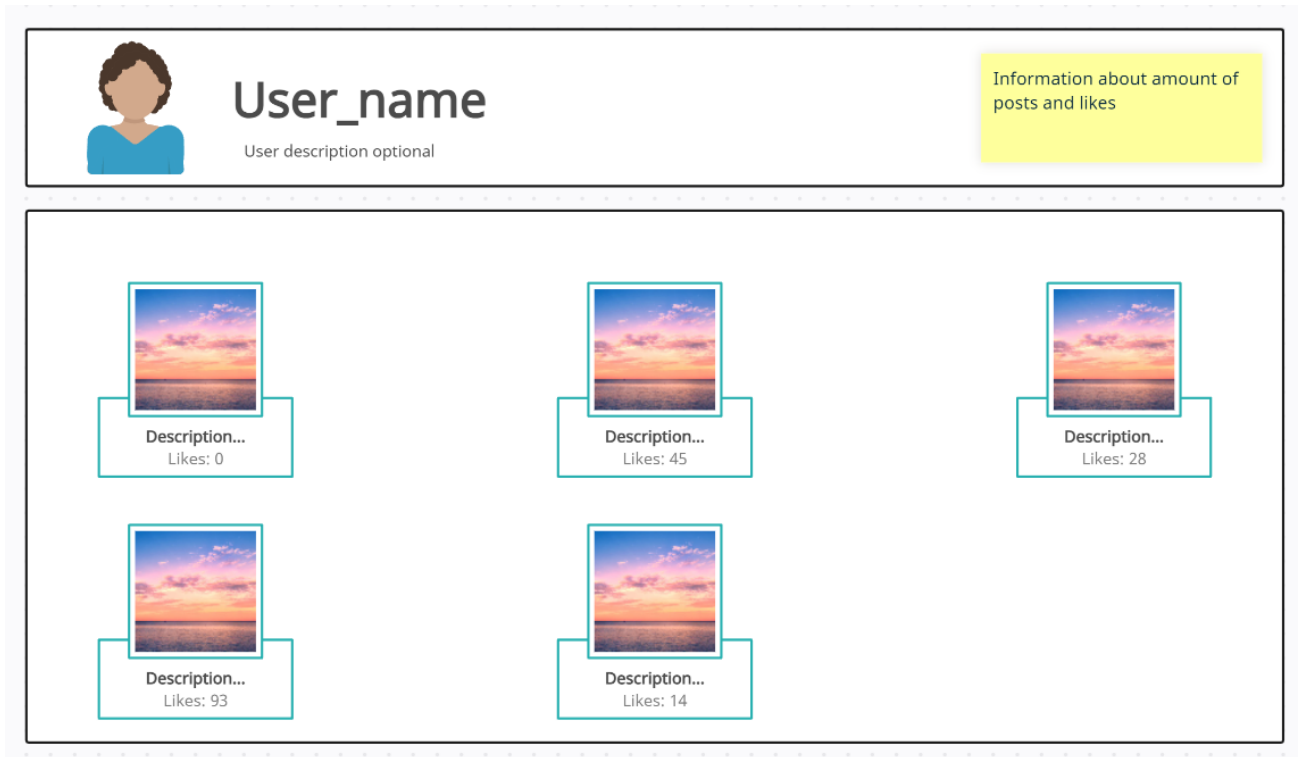


Рисунок 2.23 – схема розташування блоків та елементів у тілі сайту

## 2.5 Висновки за розділом

У цьому розділі під час аналізу предметної області було виявлено низку засобів задля актуалізації ідеї створення фотохостингу.

Розглянуто існуючі застосунки, що задовольняють схожі потреби користувачів. Pinterest виявився найкращім прикладом фотохостингу з елементами соціальної мережі, де найбільше уваги приділено алгоритмам пошуку та рекомендацій щодо візуального контенту. Instagram задовольняє потреби саме соціалізації за допомогою контенту у вигляді зображень, тобто є більше соціальною мережею, побудованою на візуальному контенті. You-Tube – яскравий приклад поєднання хостингу візуального контенту та соціальної

мережі, побудованої саме навколо контенту, але, цей приклад стосується відео контенту, що відрізняє цей приклад застосунку від цільових.

На базі здійсненого аналізу було створено мінімальний список функціональних вимог, що забезпечують життєздатність застосунку. Серед них: реєстрація, авторизація, перегляд контенту, оцінка контенту, створення власного контенту, функція пошуку.

Результатами проведеного аналізу та висунутих мінімально необхідних вимог став прототип користувацького інтерфейсу, що дає можливість мати доступ до всіх необхідних засобів, які допомагають користувачу досягнути цілей використання застосунку.

## 3 РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА

### 3.1 Рекомендації щодо інтерфейсу користувача

Інтерфейс користувача має забезпечувати доступність необхідного функціоналу інтуїтивно зрозумілим шляхом. Зручність інтерфейсу [7] також зумовлена кількістю дій, що необхідно виконати користувачу задля досягнення своїх цілей. Судячи з цього, більшість елементів керування мають приводити користувача до цілі за мінімальну кількість кроків. При цьому, інтерфейс не має бути перевантаженим, елементи керування не повинен відволікати увагу від цільового наповнення сайту.

Приймаючи до уваги описані принципи [7], можна зробити наступні висновки:

- цільове наповнення має займати основну частину, отже, тіло сайту буде складатися зі списку постів у вигляді плиток, основну частину плитки займає зображення або фотографія;
- картка профілю авторизованого користувача, або картка профілю окремого автора має з'являтися на сторінці тільки за запитом, і зникати, якщо в його відображенні немає потреби;
- якщо картка профілю користувача з'являється тільки за запитом, необхідно сконцентрувати увагу користувача саме на ній, відображуючи інформацію профілю у тілі сайту вище, ніж пости та по центру в горизонтальній площині;
- елементи керування для авторизованого користувача, такі як кнопки створення постів, виходу з профілю, відображення постів з відміткою лайк, слід сховати у вертикальне випадаюче меню;
- перехід на власний профіль за запитом слід помістити в клікабельний аватар профілю.

Така концепція застосунку дозволяє зробити його інтерфейс мінімалістичним і не перевантаженим елементами управління, що концентрує увагу користувача на сприйнятті візуального контенту.

Отже, відредагований концепт інтерфейсу застосунку має виглядати як на рис. 3.1 для авторизованого користувача, та на рис. 3.2 для неавторизованого.

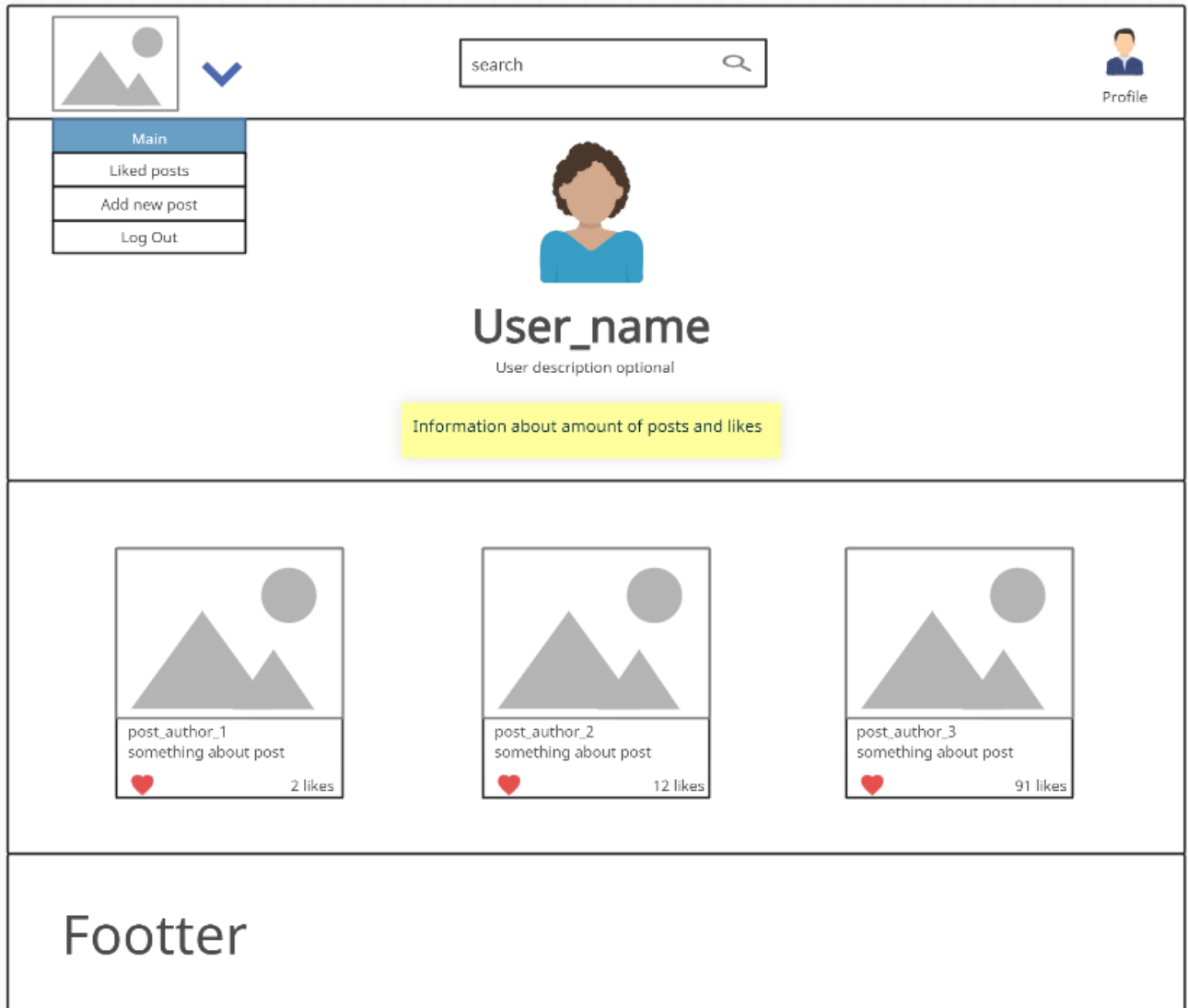


Рисунок 3.1 – Приклад інтерфейсу авторизованого користувача

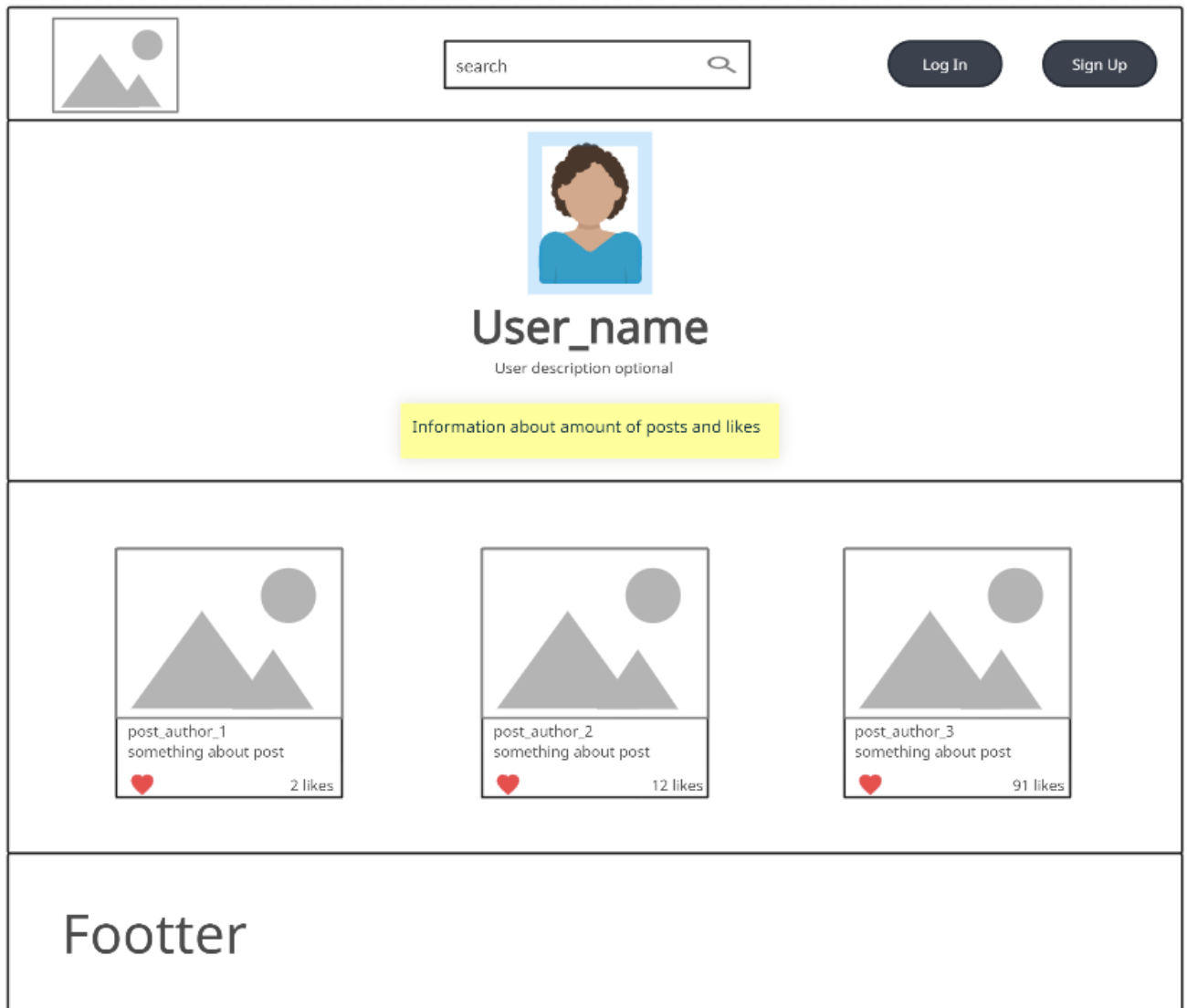


Рисунок 3.2 – Приклад інтерфейсу неавторизованого користувача

### 3.2 Висновки за розділом

В цьому розділі були прийняті додаткові правки до концепту застосунку щодо інтерфейсу користувача, розташування елементів керування. З’ясовані акценти при основних сценаріях використання застосунку. Відредаговані концептуальні приклади користувацького інтерфейсу.

## 4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

### 4.1 Визначення архітектури застосунку

Вебзастосунки поділяються на статичні та динамічні. Статичними вебсайтами прийнято рахувати додатки, які в своїй структурі не містять серверної частини. При такій концепції, HTML, CSS, JS файли зберігаються на хостингу, та містять у собі обмежений функціонал, що ніяк не взаємодіє з базами даних. Використовують такий підхід при створенні сайтів-візитівок, маленьких вебзастосунків з простою логікою. Як приклад: онлайн калькулятор, онлайн годинник, власний блог тощо.

У реалізації ідеї створення фотохостингу з елементами соціальної мережі, нам необхідно створювати динамічний сайт, що має ускладнену логіку, виконує користувацькі запити, маніпулює даними, використовуючи системи керування базами даних. Такі сайти використовують певний набір окремих функціональних і нефункціональних компонентів, так би мовити, мають декілька слоїв. Кількість таких компонентів залежить від складності застосунку та рівню декомпозиції.

Проаналізувавши складність, вимоги та способи взаємодії, найбільш підходящою для реалізації проєкта фотохостингу є клієнт-серверна архітектура. Ця модель припускає, що додаток складається з двох фундаментальних компонентів: клієнта та сервера. В ролі клієнта виступає веб-браузер, який відповідає за реєстрацію активності користувача, збирання введених даних та відображення користувацького інтерфейсу. Сервер, в свою чергу, обробляє отриману інформацію, виконуючи запити користувача, організує доступ та маніпуляцію даними, виконує бізнес-логіку.

Під час розробки за цією моделлю окремо розробляються два ключові компоненти:

- клієнтська частина або front-end – реалізує інтерфейс користувача, для взаємодії з застосунком, кнопками, посиланнями, полями вводу, текстом тощо

(зазвичай складається з HTML, CSS, JS(TS) файлів, що використовуються для взаємодії через інтерактивні елементи та забезпечують користувацький досвід);

– серверна частина або back-end – обробляє запити користувача, маніпулює даними, виконує бізнес логіку (для реалізації цієї частини використовується широкий список мов програмування, технологій для керування даними тощо).

Також, клієнт-серверна архітектура поділяється на рівні. Багаторівнева система рис. 4.1 дозволяє відокремити не тільки рівні клієнтської частини та серверної, а й винести на окремий рівень даних. Таким чином, у багаторівневій системі є винесену на окремий сервер базу даних, що керується спеціальною системою керування баз даних (СКБД). Причини відокремлення третього шару з базами даних – їх великий об’єм. Використання багаторівневої архітектури дозволяє знизити навантаження на основний сервер.

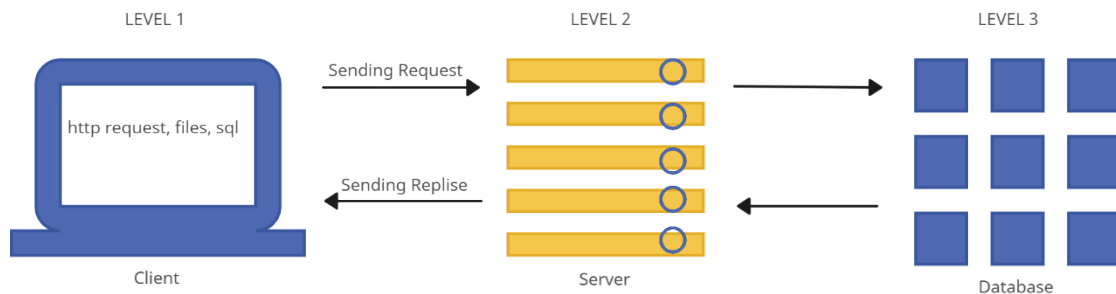


Рисунок 4.1 - Модель багаторівневої серверної архітектури

## 4.2 Обґрунтування засобів розробки

Існує дуже багато засобів розробки, програмісти обирають середовище розробки, мови програмування, фреймворки та бібліотеки, системи керування базами даних таким чином, щоб задовільнити потреби у реалізації задач. Таким чином, головним критерієм в виборі засобів розробки грають зручність, наявність

зрозумілої документації, специфічність та складність завдання, наявність необхідних функціональних можливостей. Іноді, не останню роль грає і звички та досвід розробника при роботі з певними засобами.

Спираючись на це, при виборі середовища розробки, мови програмування та інших засобів, програміст може поступово звужувати свій вибір на користь конкретного інструменту або підходу. Модель вибору засобів складається з декількох шарів критеріїв рис. 4.2. Найбільший пріоритет має критерій відповідності можливостей засобу розробки до вимог кінцевого продукту. Тобто, чи можливо виконати завдання за допомогою цього стеку, середовища. Наступним критерієм є доцільність та зручність. Чим простіше реалізація програмного продукту з певним інструментом – тим менше ресурсу та часу потрібно витратити розробнику. Останній шар – досвід розробника. Тобто, серед тих рішень, що залишилися після перших двох етапів вибору засобів розробки, програміст обирає те рішення, яке йому більше здається зрозумілішим, чи, з яким в нього більший досвід користування. Це значно облегшує задачу, та пришвидшує процес розробки.



Рисунок 4.2 – Загальні критерії вибору засобів розробки

### 4.2.1 Вибір середовища розробки

Середовище розробки впливає на швидкість реалізації проекту, якість коду та комфорт інженерів. Для того щоб налаштувати середовище розробки, ми маємо обрати потужне та зручне IDE (інтегроване середовище розробки), що підтримує необхідні мови програмування та фреймворки (включає в себе функції підсвічування синтаксису, інтелектуального доповнення коду), дає можливість використовувати розширення для редагування коду, використання систем контролю версій та інтеграції платформ для суспільної роботи над проектом, мати вбудовану консоль та засоби збирання і пакетування коду. Серед таких IDE, що можуть задовільнити потреби у розробці вебпроєкту фотохостингу, слід виділити Visual Studio Code та WebStorm, для порівняння яких наведена таблиця 4.1.

Після порівняння та аналізу, для реалізації вебзастосунку було обрано інтегроване середовище Visual Studio Code [11] – легке та зрозуміле безкоштовне IDE, розроблене компанією Microsoft, що має вбудований Git [19] та дозволяє користуватися великою кількістю розширень.

Таблиця 4.1 – Порівняння середовищ розробки

| Назва              | Переваги                                                                                                                                                 | Недоліки                                                                                                                                             |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Visual Studio Code | Безоплатність використання<br>Підтримка великої кількості розширень<br>Зручність в розгортанні декількох проєктів одночасно<br>Широка аудиторія продукту | Ресурсоємність розширень<br>пригальмовує швидкість роботи з проєктами<br>Мала кількість вбудованих функцій<br>Залежність багатьох функцій від мережі |

Продовження таблиці 4.1

| Назва    | Переваги                                                                                                                                                                          | Недоліки                                                                                                                                                                                                                                             |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WebStorm | <p>Досить швидкий</p> <p>Має велику кількість вбудованих функцій у тому числі перевірки коду, що набагато спрощує написання коду</p> <p>Має можливість встановлення розширень</p> | <p>Комерційне програмне забезпечення, потребує оплати для використання</p> <p>Перевантаження вбудованими функціями, які не будуть використані</p> <p>Обмежена підтримка мов програмування</p> <p>Складність використання при відсутності досвіду</p> |

Недолік у вигляді малої кількості вбудованих функцій компенсує наявність великого різноманіття доступних розширень, а найвпливовішою перевагою цього рішення є те, що Visual Studio Code – безкоштовний продукт, від чого поріг входження досить низький, і у користувачів є значно більше часу на звикання до продукту, на відміну від WebStorm, де пробний період є обмеженим.

#### 4.2.2 Вибір системи керування базою даних

У якості системи керування базою даних (БД) було обрано MongoDB [18]. MongoDB – система керування БД, що працює за концепцією NoSQL [22], тобто використовує структури даних замість реляційної моделі зберігання даних. Цю СКБД було обрано через можливість створювати документ на основі моделей прямо в програмному коді. Це, в поєднанні з NoSQL концепцією, дає можливість швидкого масштабування та гнучкості БД, що, в свою чергу, дозволяє змінювати та розширювати можливості застосунку у подальшому. Важливу роль у виборі на

користь MongoDB зіграла вбудована можливість зберігати дані на віддаленому сервері.

### 4.2.3 Вибір мов програмування та фреймворків

Вибір мов програмування і фреймворків тісно зв'язаний, адже фреймворки – еволюційний розвиток мов програмування, націлений на збільшення зручності та розширення базового функціоналу. Фреймворки базуються на мові або стеку, що неодмінно впливає на вибір розробника. Саме тому, при виборі мов програмування, ми будемо розглядати їх у зв'язці з фреймворками. Ще одним необхідним уточненням буде те, що архітектура застосунку має front-end та back-end складові, тому, вибір мов програмування та фреймворків для кожної складової буде обиратися окремо.

Для того, щоб обрати мову програмування та фреймворк не достатньо таблиці переваг та недоліків. Для більш репрезентативної картини, слід створити умовний графік. Осями для графіку рис. 4.3 будуть обрані основні критерії при виборі засобів розробки: енергоефективність (зручність та простота використання) та потужність технології (можливості розробки при використанні технології).

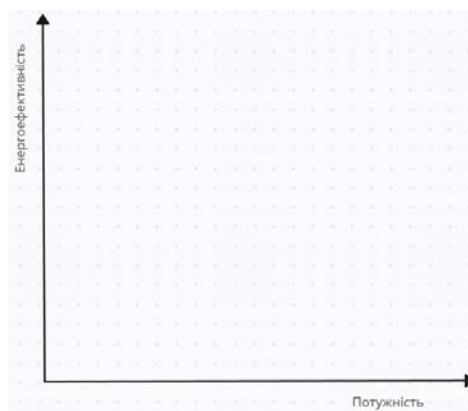


Рисунок 4.3 – Осі графіку вибору мов програмування та фреймворків

Слід зазначити, що існує більше чинників, що впливають на застосунок. Також, для реалізації клієнтської частини, велику роль буде грати стек.

На графіку рис. 4.4 зазначені лінії границь, паралельні осям координат. Ліворуч від червоної лінії – рішення, що не задовольняють розробника за своїми технологічними можливостями, праворуч – прийнятні рішення. Зверху від помаранчевої лінії – найпростіші в своєму використанні технології, а знизу – неприйнятно складні. Розташування цих ліній умовне, адже ми не маємо точних метрик ні для складності, ні для потужності.

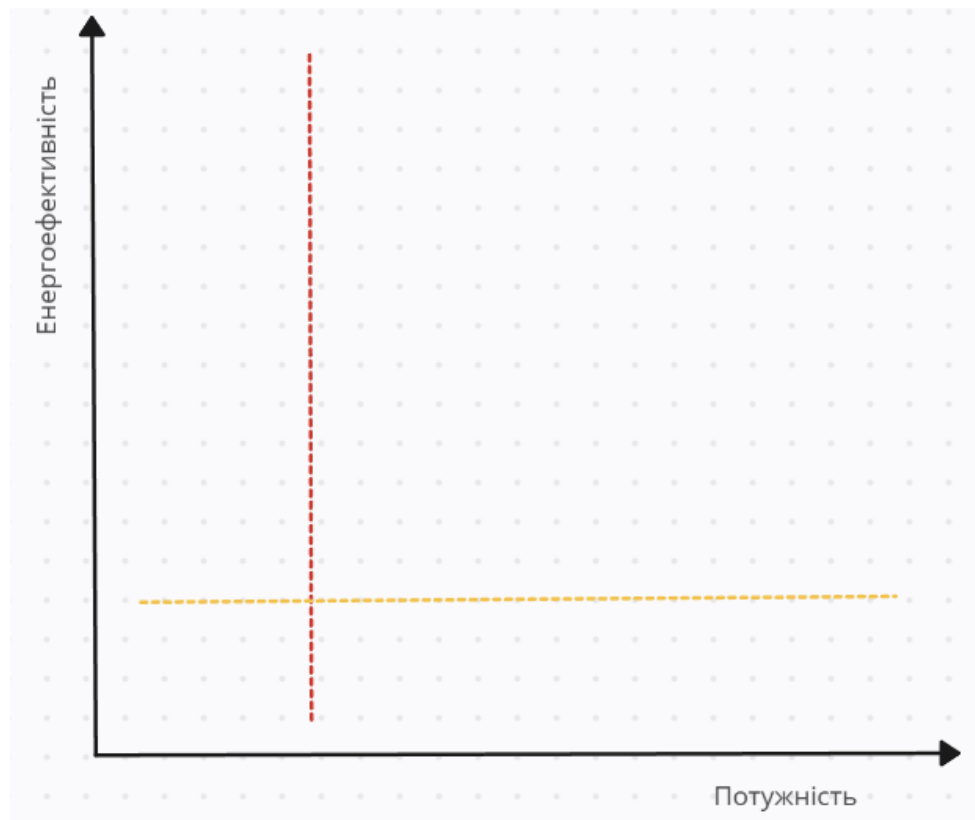


Рисунок 4.4 – Границі прийнятних рішень на графіку вибору мов програмування та фреймворків

Таким чином, маємо чотири умовні зони на графіку. Де ліва верхня зона – легкі в використанні, але не потужні засоби розробки. Ліва нижня – потребуючі велику кількість ресурсу, не потужні за своїми можливостями мови та

фреймворки. Права нижня – потужні, але досить складні в використанні рішення. Права верхня ж частина містить ті рішення, щодо мов програмування або фреймворків, що досить потужні, та, водночас, в міру складні за своїм використанням. Отже кінцевий вибір слід робити з тих рішень, що потрапили в останню описану область, та знаходяться значно правіше від інших. При такій методиці, проекція мітки певної мови чи фреймворку, де доступні всі необхідні можливості, а також, з яким у програміста більше досвіду (він буде вважати його легшим, що перенесе його вище на графіку), буде знаходитися вище проекцій інших точок рис. 4.5.

Фреймворки [6] будуть відображатися на графіку з прив'язкою до мов програмування, і, як правило, знаходяться вище та правіше на графіку від самої мови, зі зрозумілих причин.

На такому графіку відобразимо мови та фреймворки, для створення back-end рис. 4.5.

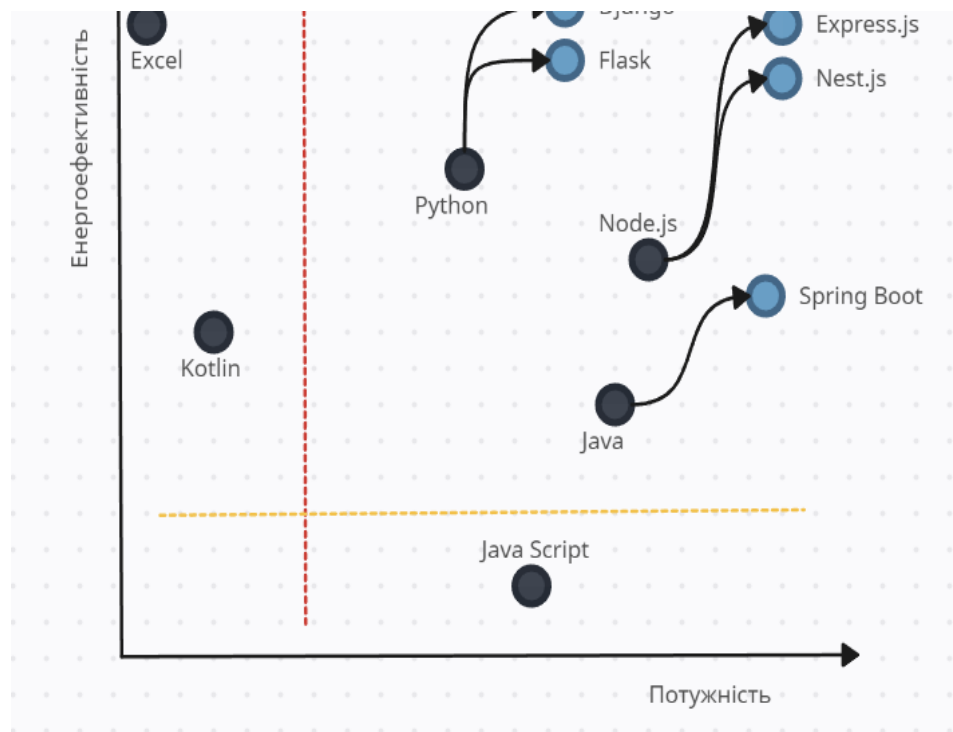


Рисунок 4.5 – Графік вибору мов програмування для back-end

Таким чином, суб'єктивно найкращим варіантом для програмної реалізації серверної частини обраного проєкту є Express.js [17], як фреймворк Node.js. Є декілька чинників, що вплинули на це:

- простота використання та наявність зрозумілої документації;
- розповсюдженість, а отже, велика кількість інформації на форумах і сторонніх ресурсів;
- потужність використання цієї технології для back-end;
- наявність досвіду роботи з фреймворком.

Графік мови програмування для front-end рис. 4.6 буде виглядати інакше.

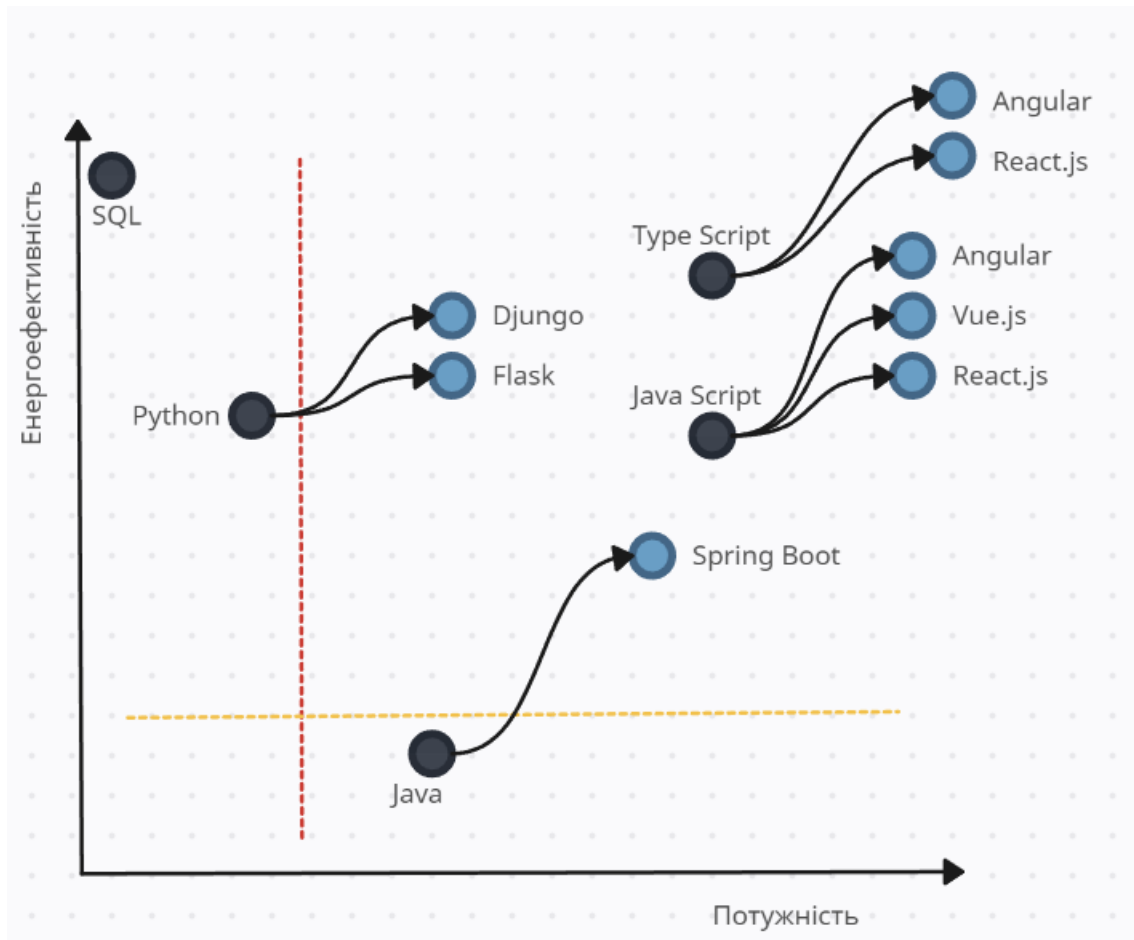


Рисунок 4.6 – Графік вибору мов програмування для front-end

У стеку, разом з мовою розмітки HTML [4] та каскадною таблицею CSS [5], згідно з поданим графіком, обрано Angular [15] та мову TypeScript. Ось ключові переваги такого вибору:

- реактивність за допомогою використання вбудованої бібліотеки NgRx Store [21];
- підтримка компонентної структури, що дозволяє розбити застосунок на окремі складові;
- статична типізація мови TypeScript дозволяє виключити помилки у роботі застосунку, що зв'язані з даними ще в процесі розробки;
- наявність доступної та зрозумілої документації, яка відповідає вимогам компанії Google;
- розповсюдженість, завдяки якій є велика кількість інформації від спільноти розробників на форумах та сторонніх ресурсах;
- розповсюджена екосистема, яка дає можливість використовувати для розробки різні інструменти та розширення, серед яких є bootstrap 5 [16], який і було використано під час розробки.

### **4.3 Опис структури застосунку**

Застосунок фотохостингу має серверну та користувацьку частину, що було оголошено під час визначення архітектури. Ці частини різні за своєю структурою, та мають низку особливостей згідно з обраними мовами програмування і фреймворками.

#### **4.3.1 Користувацька частина**

Користувацька частина або front-end застосунку рис 4.7 складається з:

- components (компонентів) – розмітка та логіка окремих елементів інтерфейсу користувача;
- interfaces (інтерфейсів) – класи, відповідні тим, що зберігаються у БД;
- services (сервісів) – окремі елементи логіки застосунку, спрямованих на взаємодію із серверною частиною;
- store (сховища) – локальне сховище, в якому реалізована реактивність за допомогою бібліотеки `ngrx`.

Слід зазначити, що в застосунку використовуються `standalone` компоненти, що дозволяє вилучити зі структури застосунку модулі. В застосунку існують наступні компоненти:

- header – компонент, що описує розмітку, стилі та логіку хедеру сайту;
- footer – компонент, що описує розмітку, стилі та логіку футеру сайту;
- home – компонент, що описує розмітку, стилі та логіку домашнього екрану застосунку;
- post-list – компонент, що виступає в ролі контейнера для масиву постів;
- post-card – компонент, що описує розмітку, стилі та логіку кожного окремого посту;
- user-profile – компонент, що описує розмітку, стилі та логіку профілю обраного користувача;
- sign-form – компонент модального вікна форми реєстрації;
- log-form – компонент модального вікна форми авторизації;
- create-form – компонент модального вікна форми створення нового поста;
- app-component – основний компонент застосунку.

В застосунку передбачено існування інтерфейсів `user` і `post`, що знаходяться в файлах `user.interface.ts` та `post.interface.ts` відповідно.

Сервіси описані в файлах `user.service.ts` та `post.service.ts`, назви яких мають співпадіння з назвами інтерфейсів, за виключенням окремо винесеного сервісу аутентифікації в файлі `auth.service.ts`.

Сховище має у собі наступні файли:

- `auth.reducer.ts` – змінює стан користувача в застосунку;
- `modal.actions.ts` – відстежує дії з модальними вікнами;
- `modal.reducer.ts` – змінює стан для модальних вікон.

Керуючим файлом виступає файл `main.ts`, а ініціює відображення сторінки файл `index.html`.

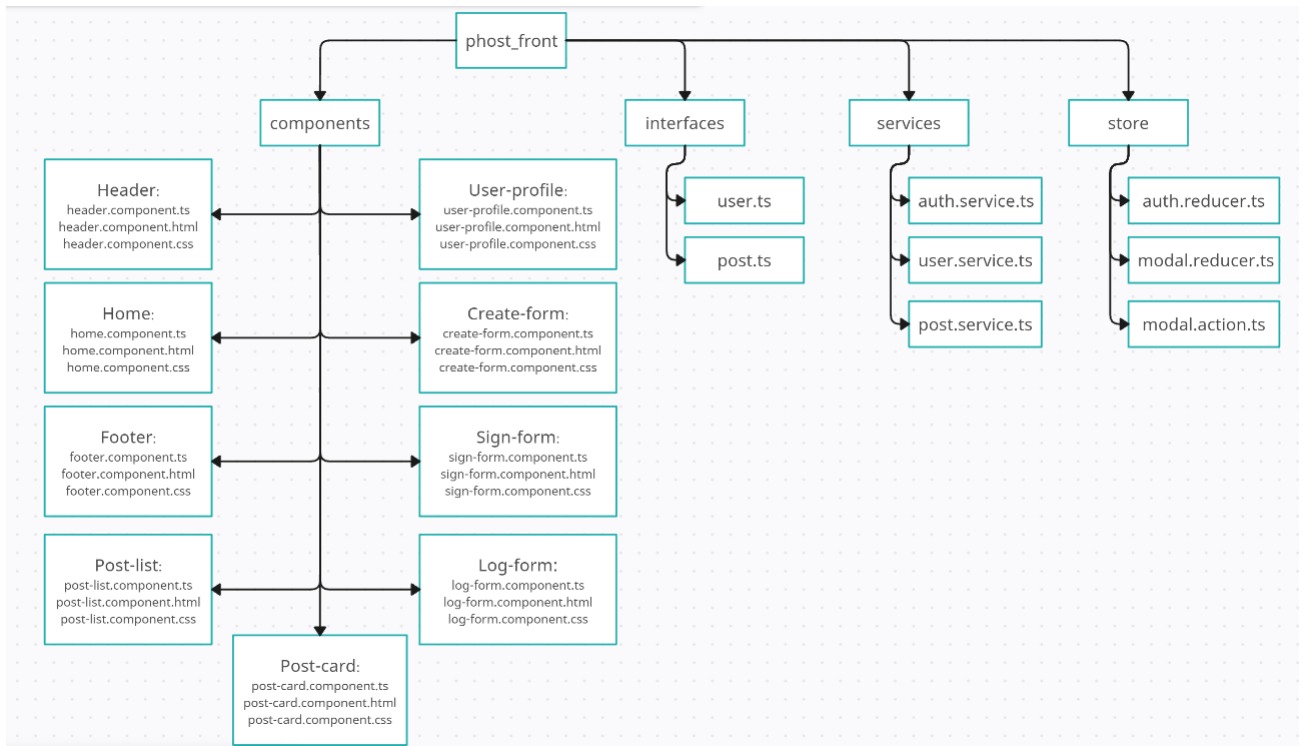


Рисунок 4.7 - Структура front-end

### 4.3.2 Серверна частина

Серверна частина або back-end застосунку рис. 4.8 поділяється на:

- `models` (моделі) – сутності, описані в коді, структура яких відповідає до моделей БД;

- `modules` (модулі) – багатоскладові частини з основним функціональним навантаженням;

- `middleware` (мідлвар) – файли, що описують логіку поведінки застосунку з авторизованим та неавторизованим користувачем.

Модулі включають в себе наступні складові:

- `routers` (маршрутизатори) – регулюють організацію шляхів у застосунку, в них описані необхідні HTML запити (POST, GET, PUT, DELETE);

- `services` (сервіси) – описують бізнес-логіку застосунку;

- `controllers` (контролери) – обробляють запити, викликаючи необхідні функції сервісів;

- `validation schemes` (схеми валідації) – відповідають за перевірку правильності отриманих даних з клієнтської частини перед їх обробкою.

Застосунок має дві моделі описані в файлах `user.model.js` та `post.model.ts`, що описують класи `Post` та `User` відповідно.

Модулі є однойменними до моделей, `post` модуль включає в себе файли: `post.service.js`, `post.controller.js`, `post.router.js`, а `user` модуль складається з файлів: `user.validation.scheme.js`, `user.service.js`, `user.controller.js`, `user.router.js`, `user.validation.scheme.js`.

Через особливості застосунку було прийнято рішення використовувати 2 мідлвари, що описані в файлах `auth.middleware.js` та `non-auth.middleware.js`.

Керуючим файлом back-end є `index.js`.

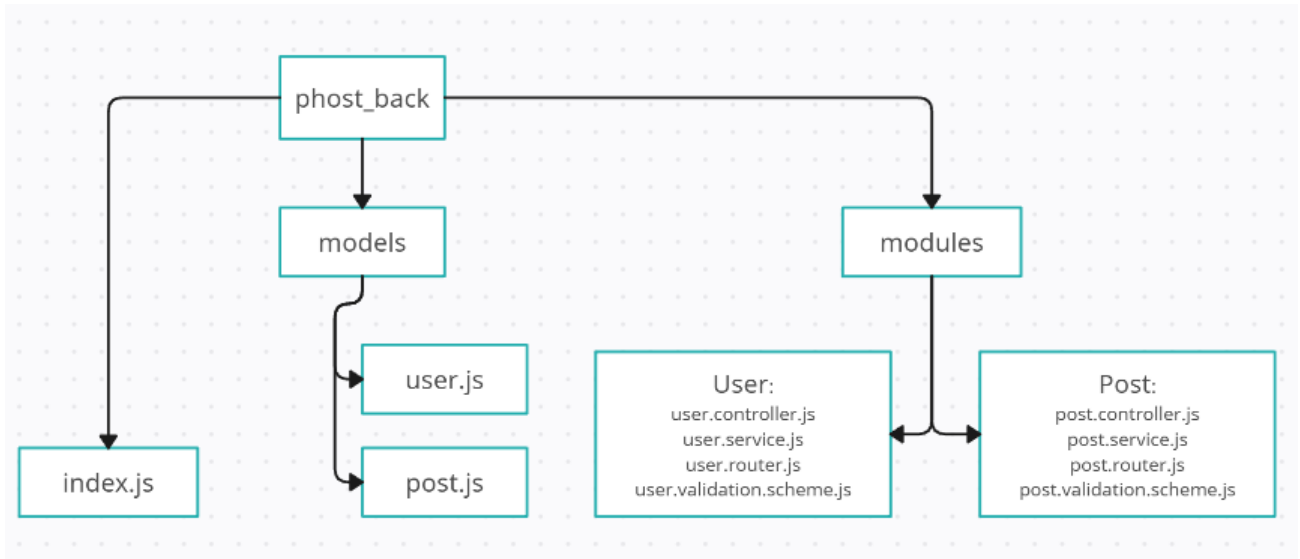


Рисунок 4.8 – Структура back-end

#### 4.4 Висновки за розділом

В цьому розділі було визначено архітектуру застосунку на основі аналізу складності, функціональних вимог та способів взаємодії користувача з застосунком. Таким чином було обрано використовувати багаторівневу клієнт-серверну архітектуру для вебзастосунку.

Аргументовано вибір засобів розробки серед яких, в якості IDE – Visual Studio Code, система контролю версій – Git з інтегрованою в середовище платформою GitHub. Система керування базами даних – MongoDB зі зберіганням даних на віддаленому сервері. Для реалізації серверної частини було обрано Express.js фреймворк для Node.js. Для реалізації клієнтської частини обрано фреймворки Angular та Bootstrap 5 з наступним набором мов програмування, стилів та розмітки: TypeScript, CSS, HTML.

Також було описано структуру вебзастосунку, описано складові клієнтської та серверної частини, визначено зв'язку між ними. Описано функціональне навантаження створених файлів з програмним кодом у всіх складових застосунку.

## 5 ТЕСТУВАННЯ

### 5.1 Етапи тестування

Процес тестування складається з низки аспектів, що спрямовані на різнопланову перевірку застосунку, для забезпечення відповідності до вимог, безпеці та безпомилкової роботи. Процес тестування поділяється на декілька етапів:

- аналіз вимог до застосунку для визначення цілей тестування;
- розробка плану тестування з визначенням необхідних областей та методів тестування;
- створення тестових сценаріїв для тестування різних аспектів програмного забезпечення, з урахуванням його особливостей;
- виконання тестування шляхом процесів ручних або автоматизованих перевірок, що включають в себе тест дизайну, коректності роботи інтерфейсу, функціональне тестування;
- тестування безпеки, сумісності та продуктивності;
- документація виявлених проблем та помилок та передача їх до команди розробників;
- підтримка протестованого програмного забезпечення шляхом подання звітів та рекомендацій команді розробників.

Процес тестування, порядок етапів, як і види тестів, відсоток ручних та автоматичних тестів може коригуватися в залежності від особливостей застосунку, вимог проєкту та обраної методології розробки.

### 5.2 Висновки за розділом

В цьому розділі було розглянуто теоретичну частину процесу тестування з урахуванням особливостей програмного забезпечення, що надійшло до команди

QA спеціалістів для оцінки якості та пошуку помилок чи невідповідностей технічному завданню застосунку.

## **6 КЕРІВНИЦТВО КОРИСТУВАЧА**

### **6.1 Призначення вебзастосунку**

Розроблено вебзастосунок задовольняє потреби користувачів у мережевій соціалізації за допомогою візуального контенту, такого як фотографії, графічні зображення та їх текстовий супровід. Серед можливостей користувача слід виділити наступні: реєстрація, авторизація, перегляд та оцінка контенту інших користувачів, створення власного графічного контенту із текстовим супроводом, отримання фідбеку для власного контенту у вигляді кількісної оцінки аудиторії застосунку (лайків), можливості перегляду власного контенту та контенту, що сподобався раніше. Такий набір функцій дозволяє задовольнити потреби користувачів у споживанні не перевантаженого візуального контенту з максимально низьким рівнем токсичності взаємодії користувачів.

### **6.2 Умови використання вебзастосунку**

Мінімальні технічні вимоги до серверу:

- жорсткий диск з вільним простором – 8 Гб;
- оперативна пам'ять – 4 Гб;
- процесор: 4 ядра, 3,5 ГГц частота;
- доступ до мережі інтернет.

Взаємодія між користувачем та застосунком відбувається через браузер, що потребує наявності мобільного пристрою або персонального комп'ютера із встановленим сучасним браузером, типу Microsoft Edge, Mozilla Firefox, Google Chrome, тощо. Версія браузера має відповідати актуальній.

### 6.3 Опис процесу використання застосунку

Для використання розробленого застосунку користувачеві необхідно обрати веб-браузер та ввести в адресну строку «localhost:4200», оскільки наразі сайт розташовано на локальному сервері а не на хостингу.

Взаємодія користувача з застосунком відбувається за допомогою інтерфейсу користувача. На початку взаємодії користувач може бачити сторінку з елементами керування для задоволення потреб неавторизованого користувача рис. 6.1, а саме, у хедері він може побачити кнопки, що забезпечують можливість реєстрації та авторизації, форма для пошуку, а у тілі сайту – пости інших користувачів без можливості оцінки.

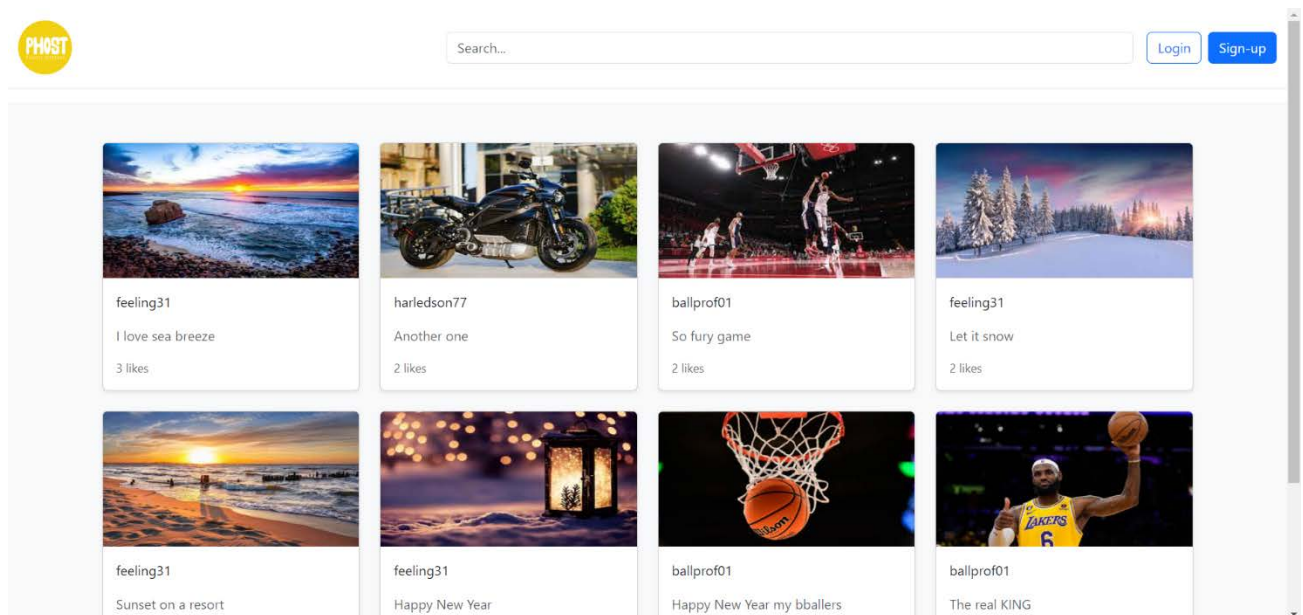


Рисунок 6.1 – Початкова сторінка для неавторизованого користувача

Неавторизований користувач може скористатися пошуком користувацьких постів за ключовими словами або автором шляхом взаємодії з пошуковим рядком сайту. Ця дія змінить контент, що відображується на сторінці на відповідний пошуковому запиту користувача рис. 6.2.

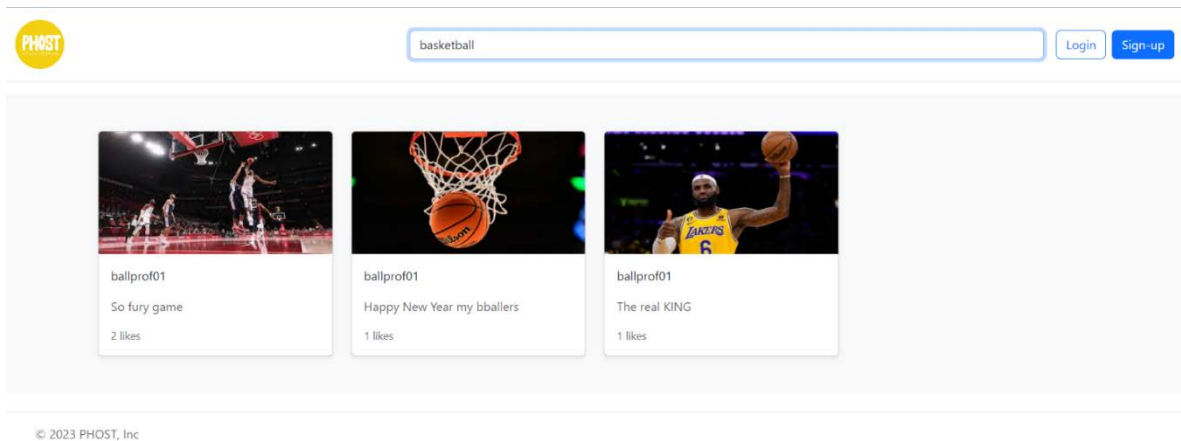


Рисунок 6.2 – Реакція вебзастосунку на пошуковий запит користувача

Користувач може повернутися на головну сторінку зі всіма постами, якщо зітре пошуковий запит або натисне на логотип застосунку в лівому верхньому куті. Натиснувши на ім'я автора, що відображається на кожному окремому пості, користувач зможе побачити картку особистого профілю автора та створені саме ним пости у тілі сайту рис. 6.3.

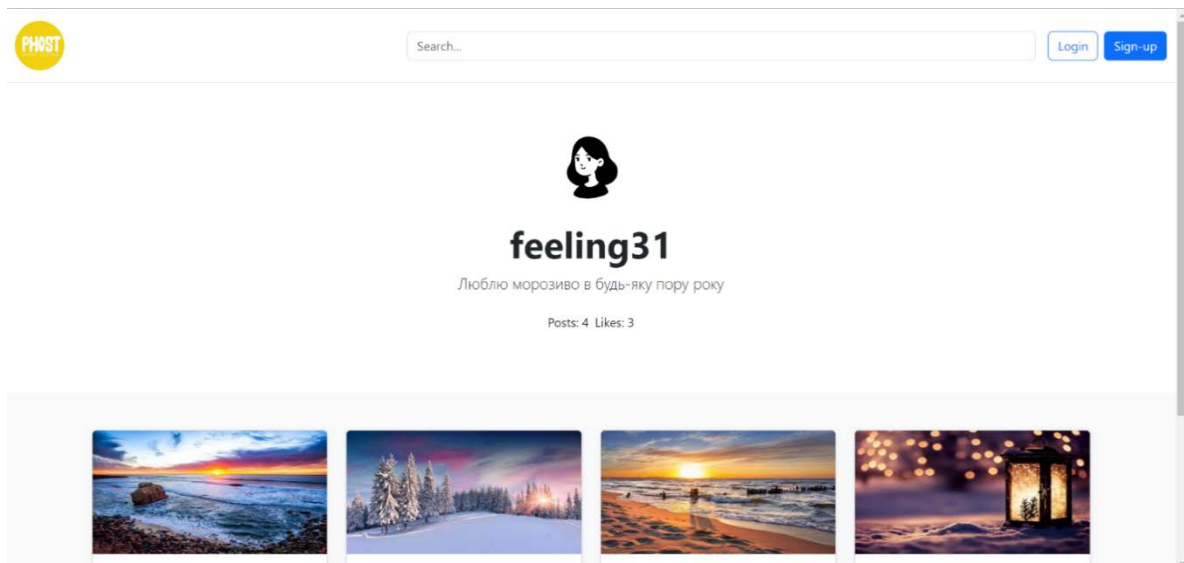
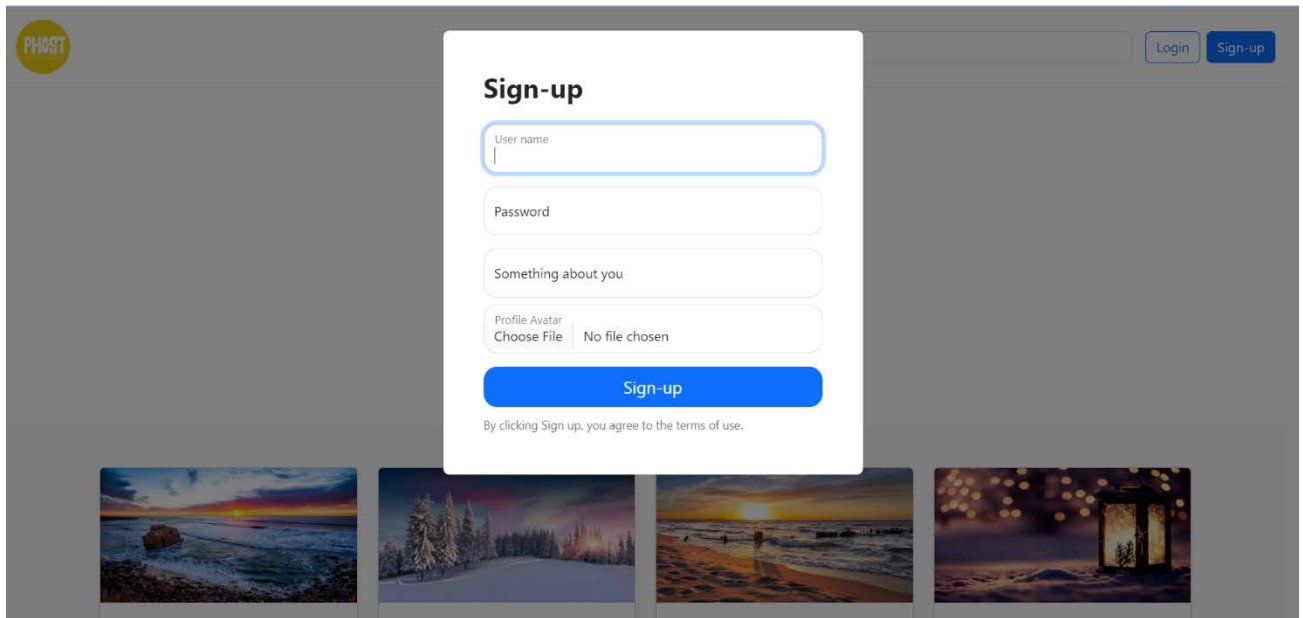


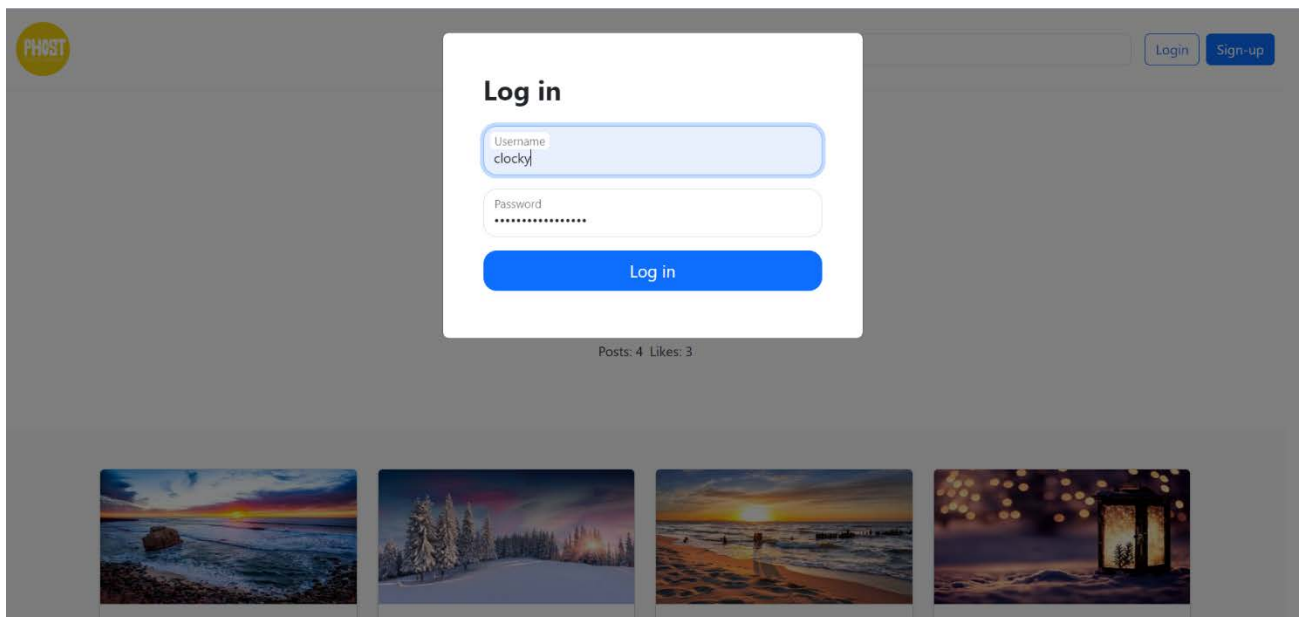
Рисунок 6.3 – Профіль автора зі створеними їм постами

Неавторизований користувач має змогу реєстрації у формі реєстрації рис. 6.4, та авторизації у формі авторизації рис. 6.5.



The screenshot shows a web browser window with the PHOST logo in the top left corner. In the top right corner, there are two buttons: "Login" and "Sign-up". The "Sign-up" button is highlighted. A white modal form titled "Sign-up" is centered on the screen. The form contains the following fields: "User name" (with a cursor), "Password", "Something about you", and "Profile Avatar" (with a "Choose File" button and "No file chosen" text). Below these fields is a blue "Sign-up" button. At the bottom of the form, there is a small text line: "By clicking Sign up, you agree to the terms of use." The background of the page is a dark gray with a grid of four landscape images at the bottom.

Рисунок 6.4 – Форма реєстрації нового користувача



The screenshot shows the same web browser window as Figure 6.4. The "Login" button in the top right corner is now highlighted. A white modal form titled "Log in" is centered on the screen. The form contains two fields: "Username" (with the text "clocky") and "Password" (with masked characters "\*\*\*\*\*"). Below these fields is a blue "Log in" button. Below the form, the text "Posts: 4 Likes: 3" is visible. The background of the page is a dark gray with a grid of four landscape images at the bottom.

Рисунок 6.5 – Форма авторизації існуючого користувача

Після авторизації, зі зміною статусу користувача, він отримає додаткові можливості. По-перше, користувач може побачити зміни у картках постів, де з'явилася можливість відмітити пости лайком рис. 6.6. Додатковий функціонал авторизованого користувача можна побачити у хедері, в випадаючому списку біля логотипу сайту рис. 6.6. При натисканні на пункт «Main» в цьому випадаючому меню, ми повернемо сторінку на початковий стан, натискання на пункт «Create new post» відкриє форму створення користувацького посту рис 6.7, в якому користувач обирає зображення, пише опис до посту, записує ключові слова для пошуку. В цій формі користувач може побачити обране ним зображення, а натиснувши кнопку «Upload» створено і відображено для всіх користувачів. В цьому ж списку, користувач може натиснути кнопку «Liked posts». Ця дія оновлює відображення контенту авторів, залишаючи на сторінці тільки пости, що були відмічені лайком у авторизованого користувача рис 6.8. Натиснувши на пункт «Log out», користувач змінює свій статус на неавторизованого, зі всіма наслідками.

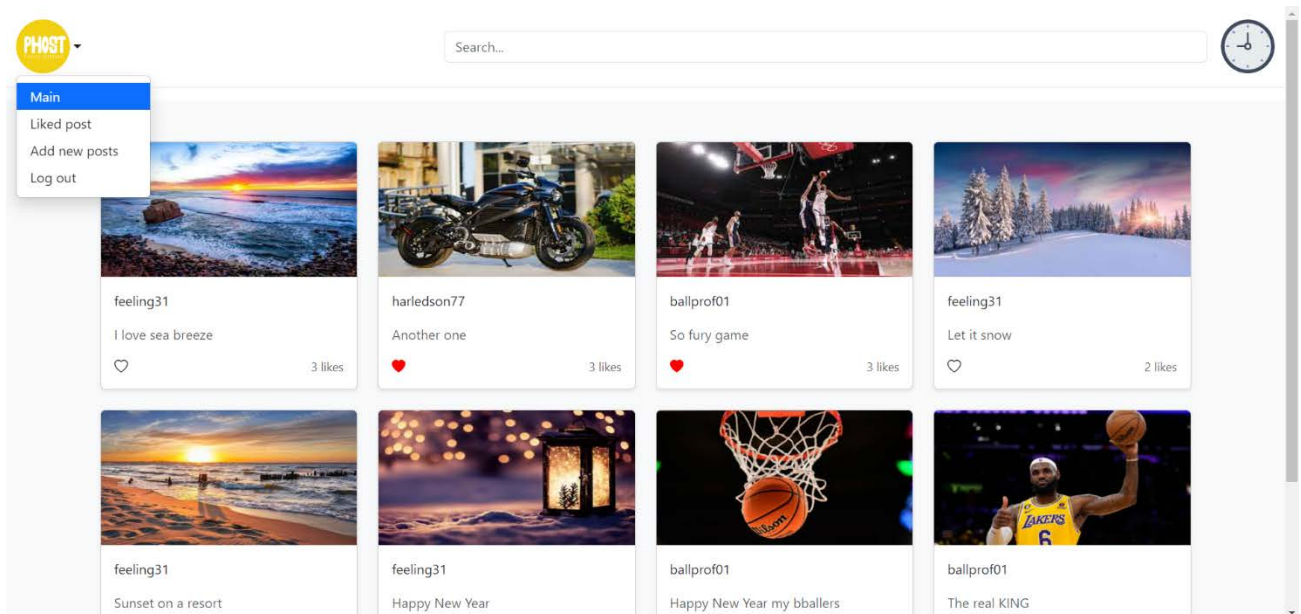


Рисунок 6.6 - Стартовий екран авторизованого користувача з доступним функціоналом

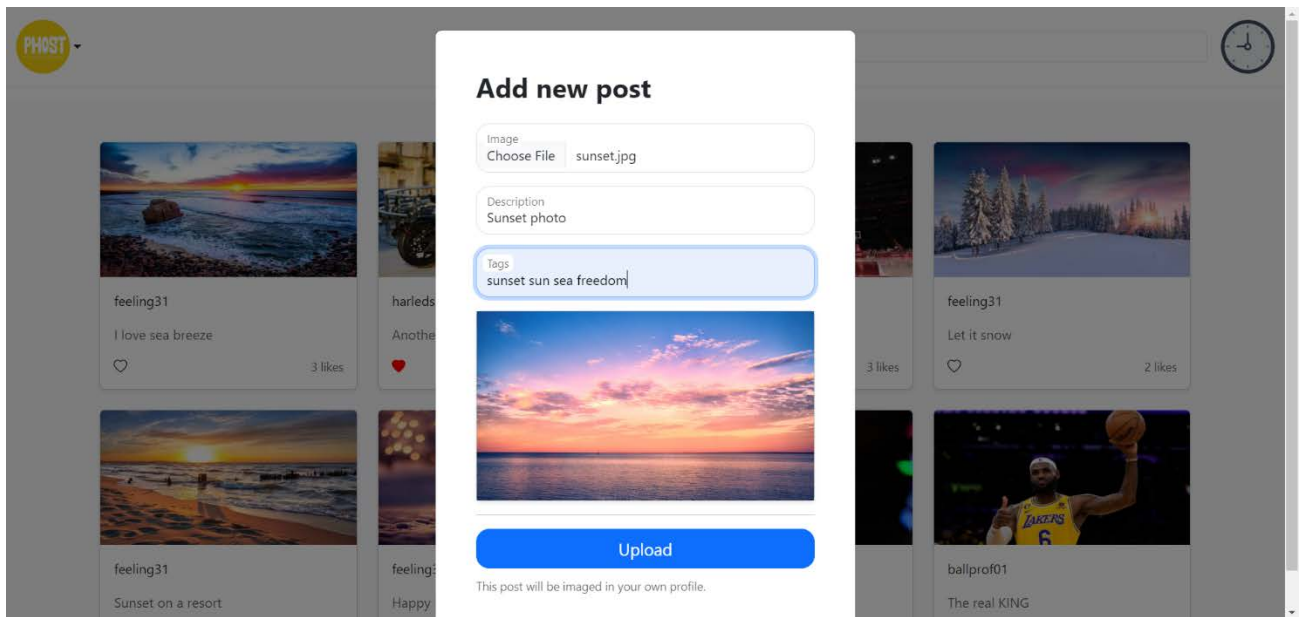


Рисунок 6.7 - Форма створення власного посту для авторизованого користувача

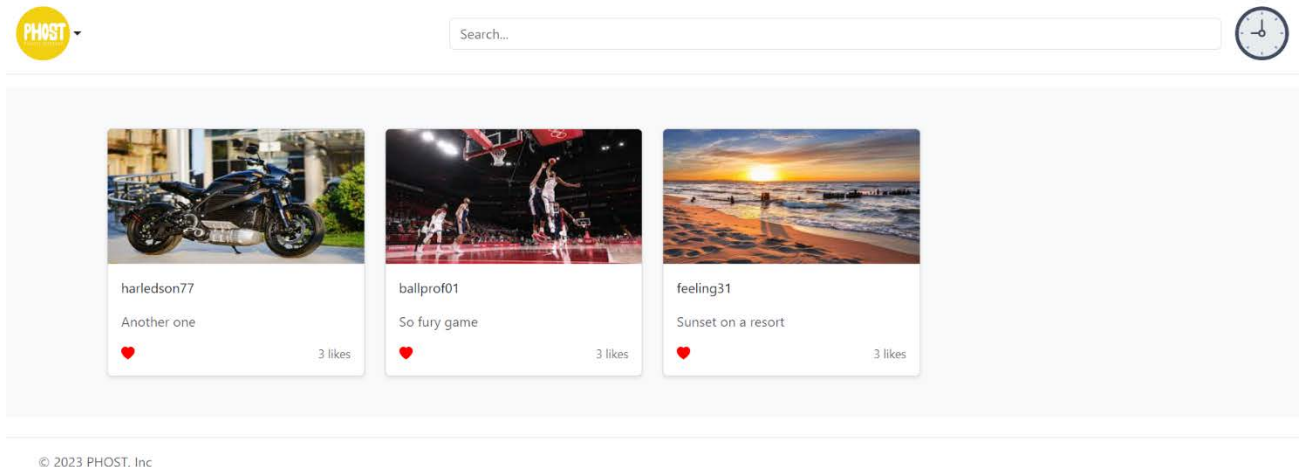


Рисунок 6.8 – Відображення постів, що були відмічені лайком авторизованого користувача

Авторизований користувач може переглядати сторінку з карткою свого профілю та тільки власними постами, автором яких він є рис. 6.9, після натискання на власний аватар, що розташовано в хедері праворуч.

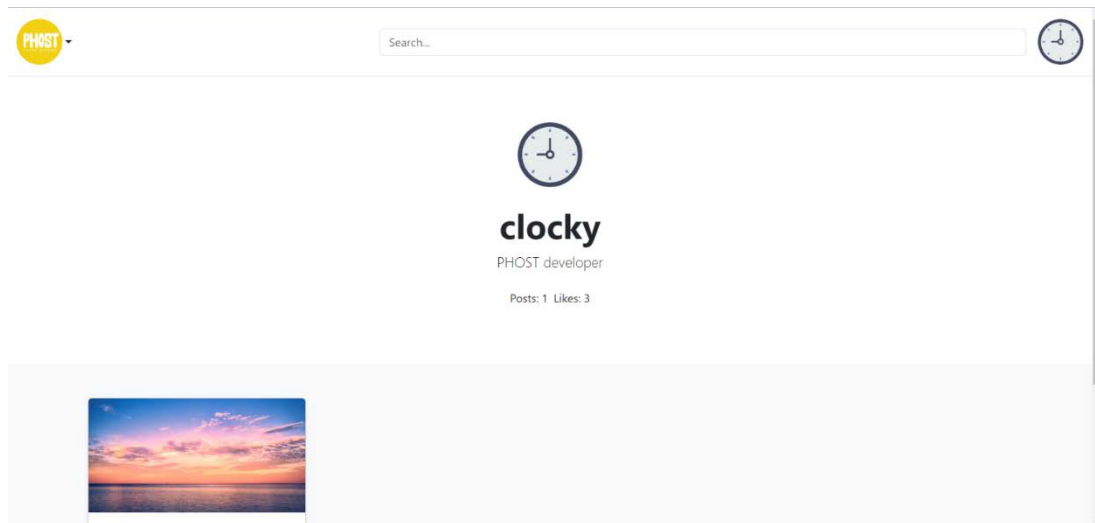


Рисунок 6.9 – Картка профілю авторизованого користувача зі створеними ним постами

При взаємодії з будь-яким елементом керування на сторінці, який пов'язано з відображенням контенту, картка профілю зникає, а список постів оновлюється.

## 6.4 Висновки за розділом

У цьому розділі було описано загальне призначення вебзастосунку, що розробляється. Також, визначено умови використання.

Вебзастосунок є односторінковим, при взаємодії з інтерфейсом оновлює список контенту, що відображається на цій сторінці за допомогою http запитів. Застосунок має просту логіку використання, поділяючи користувачів за статусом на авторизованих та не авторизованих. У розділі детально описано й процес використання сайту, до кожного прецеденту наведено приклад роботи власне розробленого вебзастосунку.

## ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи було розв'язано наступні задачі:

- дослідження життєвого циклу вебзастосунку та участь представників експертних ролей у розробці;

- аналіз предметної області та постановка завдання, шляхом дослідження специфіки роботи вебзастосунків соціальних мереж та хостингів візуального контенту, таких як зображення та відео, при цьому були розглянуті найпопулярніші застосунки, що в різних пропорціях поєднують риси притаманні соціальним мережам та хостингам;

- за результатами аналізу створено мінімальний перелік функціональних вимог, що забезпечать життєздатність проєкту, моделі роботи програми у вигляді діаграм прицедентів та концепти користувацького інтерфейсу;

- розробка інтерфейсу користувача, під час якої було описано основні вимоги до навантаженості інтерфейсу, способів взаємодії користувача з інтерфейсом та перелік рекомендацій щодо дизайну.

- проектування розробленого вебзастосунку, шляхом визначення архітектури, обґрунтуванням засобів розробки, таких як мова програмування, фреймворки, система керування БД, середовища розробки, з подальшим визначенням структури проєкту, описом програмних файлів застосунку та обґрунтуванням взаємодії клієнтської та серверної частини вебдодатку;

- загальний опис процесу тестування застосунку та його підтримки;

- створення керівництва користувача з описом призначення застосунку, мінімальних технічних вимог до сервера, детальним описом та прикладами роботи розробленого сайту.

Розроблений вебзастосунок фотохостингу надає користувачеві можливість ділитися власним контентом у вигляді зображень чи фото з

текстовим супроводом, передивлятися контент інших авторів та інформацію про них, задовольнити соціальні потреби в можливості залишати або переглядати кількісну оцінку кожного окремого посту у вигляді лайків. Також, користувач може змінювати вектор своєї уваги на контент користуючись пошуком за автором або ключовими словами, переглядаючи виключно свій контент, чи контент окремих авторів, маючи змогу передивлятися в окремому списку вподобані раніше та відмічені лайком пости.

Загалом, в цій роботі, на прикладі розробленого застосунку фотохостингу, досліджено комплексні підходи та методи реалізації вебдодатку впродовж всього циклу розробки. Послідовно розглянуті етапи розробки, та підходи до реалізації застосунку з точки зору різних експертних ролей.

Практична значущість роботи полягає у тому, що був розроблений вебзастосунок фотохостингу з елементами соціальної мережі з описом повного процесу розробки починаючи з бізнес аналітики і закінчуючи тестуванням робочої версії застосунку.

Наукова новизна дослідження полягає у впровадженні реорганізації процесу розробки для досягнення цілей розробки оптимальним шляхом.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Pinterest [Electronic resource]. – Access mode : <https://www.pinterest.com/>.
2. Instagram [Electronic resource]. – Access mode : <https://www.instagram.com/>.
3. YouTube [Електронний ресурс]. – Режим доступу : <https://www.youtube.com/>.
4. HTML [Electronic resource]. – Access mode : <https://html.spec.whatwg.org/multipage/>.
5. CSS [Електронний ресурс] – Режим доступу : <https://css.in.ua/css/properties>.
6. JS Фреймворки [Електронний ресурс]. – Режим доступу : [https://uk.wikipedia.org/wiki/Порівняння\\_фреймворків\\_JavaScript](https://uk.wikipedia.org/wiki/Порівняння_фреймворків_JavaScript).
7. User Interfaces – Information, People, and Technology [Electronic resource]. – Access mode : <https://unizin.org>.
8. Довідник по HTML тегам [Електронний ресурс]. – Режим доступу : <https://css.in.ua/html/tags>.
9. JavaScript [Electronic resource]. – Access mode : [https://developer.mozilla.org/uk/docs/Learn/Getting\\_started\\_with\\_the\\_web/JavaScript\\_basics](https://developer.mozilla.org/uk/docs/Learn/Getting_started_with_the_web/JavaScript_basics).
10. Веб приложение. Разница между сайтом и веб-приложением, SPA И PWA [Електронний ресурс]. – Режим доступу : <https://webcase.com.ua/blog/cho-takoe-web-prilozhenie-vse-vidy/>.
11. Microsoft Visual Studio Code [Electronic resource]. – Access mode : <https://code.visualstudio.com/>.
12. Vue.js [Електронний ресурс]. – Режим доступу : <http://xn--80adh0aefm3i.xn--j1amh/Vue.js>.

13. Angular [Электронный ресурс]. – Режим доступа : <https://angular.io/>.
14. Bootstrap [Электронный ресурс]. – Режим доступа : <https://getbootstrap.com/docs/5.0/getting-started/introduction/>.
15. Express.js [Электронный ресурс]. – Режим доступа : <https://expressjs.com/>.
16. MongoDB [Электронный ресурс]. – Режим доступа : [https://www.mongodb.com/cloud/atlas/lp/try4?utm\\_source=google&utm\\_campaign=search\\_gs\\_pl\\_evergreen\\_atlas\\_general\\_retarget-brand\\_gic-null\\_emea-all\\_ps-all\\_desktop\\_eng\\_lead&utm\\_term=mongodb&utm\\_medium=cpc\\_paid\\_search&utm\\_ad=p&utm\\_ad\\_campaign\\_id=14412646458&adgroup=151115418215&cq\\_cmp=14412646458&gad\\_source=1&gclid=CjwKCAiAkp6tBhB5EiwANTCx1EnFzutAg77SMHrtIVWb7DEMJVhBi0HliK4EzrMLJZRK9vSlv33jZxoCh5cQAvD\\_BwE](https://www.mongodb.com/cloud/atlas/lp/try4?utm_source=google&utm_campaign=search_gs_pl_evergreen_atlas_general_retarget-brand_gic-null_emea-all_ps-all_desktop_eng_lead&utm_term=mongodb&utm_medium=cpc_paid_search&utm_ad=p&utm_ad_campaign_id=14412646458&adgroup=151115418215&cq_cmp=14412646458&gad_source=1&gclid=CjwKCAiAkp6tBhB5EiwANTCx1EnFzutAg77SMHrtIVWb7DEMJVhBi0HliK4EzrMLJZRK9vSlv33jZxoCh5cQAvD_BwE).
17. Git [Электронный ресурс]. – Режим доступа : <https://git-scm.com/>.
18. GitHub [Электронный ресурс]. – Режим доступа : <https://github.com/>.
19. NgRx/Store [Электронный ресурс]. – Режим доступа : <https://ngrx.io/guide/store>.
20. What is NoSQL [Электронный ресурс]. – Режим доступа : <https://aws.amazon.com/nosql/>.
21. JSON Web Token [Электронный ресурс]. – Режим доступа : <https://www.npmjs.com/package/jsonwebtoken>.

**ДОДАТОК А**  
**Текст програми**

### A.1 Код файла index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>PhostFront</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

### A.2 Код файла app.component.html

```
<app-header></app-header>
<router-outlet></router-outlet>
<app-post-list></app-post-list>
<app-footer></app-footer>
```

### A.3 Код файла app.config.ts

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
```

```
import { RouterModule, RouterOutlet, provideRouter } from
'@angular/router';

import { FooterComponent } from './components/footer/footer.component';
import { HeaderComponent } from './components/header/header.component';
import { PostListComponent } from './components/post-list/post-
list.component';

import { UserProfileComponent } from './components/user-profile/user-
profile.component';

import { HttpClientModule } from '@angular/common/http';
import { PostService } from './services/post.service';
import { UserService } from './services/user.service';
import { FormsModule } from '@angular/forms';
import { AuthService } from './services/auth.service';
import { NgbActiveModal } from '@ng-bootstrap/ng-bootstrap';
import { bootstrapApplication } from '@angular/platform-browser';
import { provideStore } from '@ngrx/store';
import { ModalReducer } from './store/modal.reducer';
import { AuthReducer } from './store/auth.reducer';
import { HomeComponent } from './components/home/home.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule,
            RouterOutlet,
            FooterComponent,
            HeaderComponent,
            PostListComponent,
```

```

        UserProfileComponent,
        HttpClientModule,
        FormsModule,
        RouterModule
    ],
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css'],
    providers: [UserService, PostService, AuthService, NgbActiveModal],
  })
export class AppComponent {
  title = 'phost_front';
}

```

#### A.4 Код файла header.component.html

```

<header class="py-3 mb-3 border-bottom">
  <div class="container-fluid d-grid gap-3 align-items-center" style="grid-
template-columns: 1fr 2fr;">
    
    <div ngbDropdown class="dropdown" *ngIf="isLoggedIn">
      <a ngbDropdownToggle class="d-flex align-items-center col-lg-4 mb-2
mb-lg-0 link-dark text-decoration-none dropdown-toggle" href="#"
id="dropdownNavLink" data-bs-toggle="dropdown" aria-expanded="true">
        
      </a>

```

```

<ul ngbDropdownMenu class="dropdown-menu text-small shadow" aria-
labelledby="dropdownNavLink">
    <li><a ngbDropdownItem class="dropdown-item active" href="#" aria-
current="page">Main</a></li>
    <li><a ngbDropdownItem class="dropdown-item" href="#"
(click)="getUserLikes()">Liked post</a></li>
    <li><a ngbDropdownItem class="dropdown-item" href="#"
(click)="open(newContent)">Add new posts</a></li>
    <li><a ngbDropdownItem class="dropdown-item" href="#"
(click)="logout()">Log out</a></li>
</ul>
</div>

<div class="d-flex align-items-center">
    <form class="w-100 me-3">
        <input type="text" (input)="onSearch($event)" class="form-control"
placeholder="Search..." aria-label="Search">
    </form>

    <div class="flex-shrink-0 dropdown">
        <a *ngIf="isLoggedIn" class="d-block link-dark text-decoration-none"
id="User2">
            <img src={{this.userAvatar}} alt="mdo" width="64"
height="64" class="rounded-circle" (click)="getUserProfile()">
        </a>
        <a *ngIf="!isLoggedIn" class="d-block link-dark text-decoration-none"
id="User2">

```

```

        <button type="button" class="btn btn-outline-primary me-2"
(click)="open(logContent)">Login</button>
        <button type="button" class="btn btn-primary"
(click)="open(signContent)">Sign-up</button>
    </a>
</div>
</div>
</div>
</div>

<!--Modals-->
<ng-template #logContent let-modal><app-log-form></app-log-form></ng-
template>
    <ng-template #signContent let-modal><app-sign-form></app-sign-
form></ng-template>
    <ng-template #newContent let-modal><app-create-form></app-create-
form></ng-template>
<!------>

</header>

```

### A.5 Код файла **header.component.ts**

```

import { Component, OnDestroy, OnInit, TemplateRef } from
'@angular/core';
import { CommonModule } from '@angular/common';
import {
    NgbDropdownModule,
    ModalDismissReasons,

```

```

    NgbDatepickerModule,
    NgbModal,
  } from '@ng-bootstrap/ng-bootstrap';
import { SignFormComponent } from '../sign-form/sign-form.component';
import { LogFormComponent } from '../log-form/log-form.component';
import { CreateFormComponent } from '../create-form/create-
form.component';

import { AuthService } from '../../services/auth.service';
import { Subject, debounceTime, distinctUntilChanged } from 'rxjs';
import { PostService } from '../../services/post.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-header',
  standalone: true,
  imports: [
    CommonModule,
    NgbDropdownModule,
    NgbDatepickerModule,
    SignFormComponent,
    LogFormComponent,
    CreateFormComponent,
  ],
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css'],
})
export class HeaderComponent implements OnInit {
  isLoggedIn = false;

```

```

postArr: any;
private inputChangedSubject = new Subject<string>();
userName: string = "";
userAvatar: string = "";
constructor(
    private modalService: NgbModal,
    private authService: AuthService,
    private postService: PostService,
    private router: Router
) {
    this.authService.status$.subscribe((status) => {
        this.isLoggedIn = status;
    });
}
ngOnInit() {
    this.inputChangedSubject
        .pipe(
            debounceTime(300),
            distinctUntilChanged()
        )
        .subscribe((value) => {
            this.postService
                .findAll(value)
                .subscribe((res) => this.postService.updateArray(res));
        });
    this.postService.postArr$.subscribe(postArr => this.postArr = postArr);
    this.authService.getUserProfile();
    this.authService.myUserProfile$.subscribe((userProfile) => {

```

```

    if (userProfile) {
        this.userName = userProfile.userName;
        this.userAvatar = userProfile.userAvatar;
    }
});
}
closeResult = "";
onSearch(event: any) {
    let search = event.target.value;
    this.inputChangedSubject.next(search);
}
open(content: TemplateRef<any>) {
    const modalRef = this.modalService.open(content, {
        ariaLabelledBy: 'modal-basic-title',
    });
    modalRef.result.then(
        (result) => {
            this.closeResult = `Closed with: ${result}`;
        },
        (reason) => {
            this.closeResult = `Dismissed ${this.getDismissReason(reason)}`;
        }
    );
}
getUserLikes(){

}
getUserProfile() {

```

```

    this.router.navigate(['/', this.userName]);
    this.postService.updateArray(this.postArr);
  }
  logout() {
    localStorage.removeItem('accessToken');
    this.authService.updateStatus(false);
  }
  private getDismissReason(reason: any): string {
    switch (reason) {
      case ModalDismissReasons.ESC:
        return 'by pressing ESC';
      case ModalDismissReasons.BACKDROP_CLICK:
        return 'by clicking on a backdrop';
      default:
        return `with: ${reason}`;
    }
  }
}

```

## A.6 Код файла **home.component.ts**

```

import { Component, OnInit } from '@angular/core';
import { UserProfileComponent } from '../user-profile/user-
profile.component';
import { AuthService } from '../services/auth.service';
import { ActivatedRoute } from '@angular/router';
import { CommonModule } from '@angular/common';

```

```

@Component({
  selector: 'app-home',
  standalone: true,
  imports: [UserProfileComponent, CommonModule],
  template: '<app-user-profile *ngIf="showProfileSection"></app-user-
profile>',
  styleUrls: ['./home.component.css'],
})
export class HomeComponent implements OnInit {
  public showProfileSection = false;
  constructor(private route: ActivatedRoute) {}
  ngOnInit(): void {
    const currentRoute = this.route.snapshot;
    const paramValue = currentRoute.params['id'];
    if (paramValue) {
      this.showProfileSection = true;
    }
  }
}

```

### A.7 Код файла user-profile.component.html

```

<div class="px-4 py-3 my-5 text-center">
  <img class="d-block mx-auto mb-4" src={{userProfile.userAvatar}} alt=""
width="90" height="90">
  <h1 class="display-5 fw-bold">{{userProfile.userName}}</h1>
  <div class="col-lg-6 mx-auto">
    <p class="lead mb-4">{{userProfile.userDescription}}</p>

```

```

<div class="d-grid gap-2 d-sm-flex justify-content-sm-center">
  <p>Posts: {{userProfile.posts}}</p>
  <p>Likes: {{userProfile.likes}}</p>
</div>
</div>
</div>

```

### A.8 Код файла `user-profile.component.ts`

```

import { Component, Input, OnInit } from '@angular/core';
import { UserService } from '../services/user.service';
import { CommonModule } from '@angular/common';
import { User } from '../interfaces/user';
import { AuthService } from '../services/auth.service';
import {
  ActivatedRoute,
  NavigationStart,
  Route,
  Router,
} from '@angular/router';
import { PostService } from '../services/post.service';
import { filter } from 'rxjs';

@Component({
  selector: 'app-user-profile',
  standalone: true,
  imports: [],
  templateUrl: './user-profile.component.html',

```

```

    styleUrls: ['./user-profile.component.css'],
  })
export class UserProfileComponent implements OnInit {
  public userProfile: any;
  constructor(
    private authService: AuthService,
    private route: ActivatedRoute,
    private postService: PostService
  ) {}
  ngOnInit(): void {
    this.route.url.subscribe(() => {
      const paramValue = this.route.snapshot.params['id'];
      this.authService.getUserProfile(paramValue);
    });
    this.authService.userProfile$.subscribe((profile) => {
      this.userProfile = { ...profile };
      this.postService.postArr$.subscribe((array) => {
        if (array) {
          this.userProfile.posts = 0;
          this.userProfile.likes = 0;
          array.map((el: any) => {
            if (el.postCreator === this.userProfile.userName) {
              this.userProfile.posts++;
            }
            if (el.postLikes.includes(this.userProfile.userName)) {
              this.userProfile.likes++;
            }
          });
        }
      });
    });
  }
}

```

```

    }
  });
});
}
}

```

### A.9 Код файла `post-list.component.html`

```

<div class="album py-5 bg-light">
  <div class="container">
    <div class="row row-cols-1 row-cols-sm-2 row-cols-md-4 g-4">

      <app-post-card *ngFor="let post of posts" [post]="post"></app-post-
card>

    </div>
  </div>
</div>

```

### A.10 Код файла `post-list.component.ts`

```

import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { PostCardComponent } from './post-card/post-card.component';
import { PostService } from '../services/post.service';
import { Post } from '../interfaces/post';
import { AuthService } from '../services/auth.service';
import { ActivatedRoute } from '@angular/router';

```

```

@Component({
  selector: 'app-post-list',
  standalone: true,
  imports: [CommonModule, PostCardComponent],
  templateUrl: './post-list.component.html',
  styleUrls: ['./post-list.component.css'],
})
export class PostListComponent implements OnInit {
  posts: Post[] = [];
  userName: string = "";
  myUserName: string = "";
  isLoggedIn = false;
  constructor(
    private postService: PostService,
    private authService: AuthService,
    private route: ActivatedRoute
  ) {}
  ngOnInit(): void {
    this.authService.userProfile$.subscribe((user) => {
      if (user) this.userName = user.userName;
      if (this.userName && this.posts) {
        this.posts.filter((post) => post.postCreator == this.userName);
      }
    });
    this.authService.myUserProfile$.subscribe((user) => {
      if (user) this.myUserName = user.userName;
    });
    this.authService.userProfile$.subscribe((user) => {

```

```

if (user) this.userName = user.userName;
this.postService.postArr$.subscribe((postArr) => {
  if (postArr) {
    this.posts = postArr;
    if (this.myUserName && this.isLoggedIn) {
      this.posts.map((post) => {
        post.isLiked = post.postLikes.includes(this.myUserName);
        post.isOwn = post.postCreator == this.myUserName;
        return post;
      });
    }
    if (this.userName && this.posts) {
      this.posts = this.posts.filter((post) => {
        return post.postCreator == this.userName;
      });
    }
  });
});
this.authService.status$.subscribe((status) => (this.isLoggedIn = status));

this.postService.findAll().subscribe((res) => {
  this.postService.updateArray(res);
});
}
}

```

### A.11 Код файла `post-card.component.html`

```

<div class="col">
  <div class="card shadow-sm">
    <img class="bd-placeholder-img card-img-top" src={{post.postImage}}
width="100%" height="160">

    <div class="card-body">
      <p class="card-text creator"
(click)="getUserProfile()">{{post.postCreator}}</p>
      <p class="text-muted">{{post.postDescription}}</p>
      <div class="d-flex justify-content-between align-items-center">
        <div *ngIf="isLoggedIn" class="btn-group">
          <svg *ngIf="post.isLiked" xmlns="http://www.w3.org/2000/svg"
width="16" height="16" fill="red" class="bi bi-heart-fill" viewBox="0 0 16 16"
(click)="likeUpdate()">
            <path fill-rule="evenodd" d="M8 1.314C12.438-3.248 23.534 4.735
8 15-7.534 4.736 3.562-3.248 8 1.314"/>
          </svg>
          <svg *ngIf="!post.isLiked" xmlns="http://www.w3.org/2000/svg"
width="16" height="16" fill="currentColor" class="bi bi-heart" viewBox="0 0 16 16"
(click)="likeUpdate()">
            <path d="m8 2.748-.717-.737C5.628 2.514 8.78 1.4 3.053c-.523
1.023-.641 2.531 4.385.92 1.815 2.834 3.989 6.286 6.357 3.452-2.368 5.365-4.542
6.286-6.357.955-1.886.838-3.362.314-4.385C13.486 8.78 10.428 8.717 2.01z"
M8 15C-7.333 4.868 3.279-3.04 7.824 1.143c.06.055.119.112.176.171a3.12 3.12 0 0 1
.176-.17C12.72-3.042 23.333 4.867 8 15"/>
          </svg>

```

```

    </div>
    <small class="text-muted">{{post.postLikes.length}} likes</small>
  </div>
</div>
</div>
</div>

```

### A.12 Код файла `post-card.component.ts`

```

import { Component, Input } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AuthService } from '../services/auth.service';
import { Post } from '../interfaces/post';
import { PostService } from '../services/post.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-post-card',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './post-card.component.html',
  styleUrls: ['./post-card.component.css'],
})
export class PostCardComponent {
  @Input() post: Post | any = null;
  isLoggedIn = false;
  constructor(
    private authService: AuthService,

```

```

    private postService: PostService,
    private router: Router
  ) {
    this.authService.status$.subscribe((status) => {
      this.isLoggedIn = status;
    });
  }
  getUserProfile() {
    this.router.navigate(['/'], this.post.postCreator);
  }
  likeUpdate(): void {
    this.postService.updateLikes(this.post.postId);
  }
}

```

### A.13 Код файла `sign-form.component.html`

```

<div class="modal-header p-5 pb-4 border-bottom-0">
  <h2 class="fw-bold mb-0">Sign-up</h2>
</div>

<div class="modal-body p-5 pt-0">
  <form class="" (submit)="userCreate($event)">
    <div class="form-floating mb-3">
      <input type="text" class="form-control rounded-4" id="floatingName"
placeholder="floatingName" [(ngModel)]="userName"
[ngModelOptions]="{standalone: true}">
      <label for="floatingInput">User name</label>
    </div>
  </form>
</div>

```

```

</div>
<div class="form-floating mb-3">
  <input type="password" class="form-control rounded-4"
id="floatingPassword" placeholder="floatingPassword" [(ngModel)]="userPassword"
[ngModelOptions]="{standalone: true}">
  <label for="floatingPassword">Password</label>
</div>
<div class="form-floating mb-2">
  <input type="description" class="form-control rounded-4"
id="floatingDescr" placeholder="floatingDescr" [(ngModel)]="userDescription"
[ngModelOptions]="{standalone: true}">
  <label for="floatingDescr">Something about you</label>
</div>
<div class="form-floating mb-3">
  <input type="file" class="form-control rounded-4" id="floatingAvatar"
placeholder="floatingAvatar" (change)="onFileChange($event)">
  <label for="floatingAvatar">Profile Avatar</label>
</div>
<button class="w-100 mb-2 btn btn-lg rounded-4 btn-primary"
type="submit">Sign-up</button>
  <small class="text-muted">By clicking Sign up, you agree to the terms of
use.</small>
</form>
</div>

```

#### **A.14 Код файла sign-form.component.ts**

```
import { Component } from '@angular/core';
```

```
import { UserService } from '../services/user.service';
import { AuthService } from '../services/auth.service';
import { User } from '../interfaces/user';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
```

```
@Component({
  selector: 'app-sign-form',
  standalone: true,
  imports: [ CommonModule, FormsModule ],
  templateUrl: './sign-form.component.html',
  styleUrls: ['./sign-form.component.css']
})
export class SignFormComponent {

  constructor(private userService: UserService){}

  userName: string = "";
  userPassword: string = "";
  userDescription: string = "";
  userAvatar: string = 'assets/images/user_default.png';

  onFileChange(event: any): void {
    let image = event.target.files[0];
    if(image) {
      const reader = new FileReader();
      reader.onload = (e) => {
        this.userAvatar = reader.result as string;
      }
    }
  }
}
```

```

    }
    reader.readAsDataURL(image);
  }
}

```

```

userCreate(event: Event): void{
  event.preventDefault();
  this.userService.createUser({
    userName: this.userName,
    userPass: this.userPassword,
    userDescription: this.userDescription,
    userAvatar: this.userAvatar
  });
}
}

```

### A.15 Код файлы `log-form.component.html`

```

<div class="modal-header p-5 pb-4 border-bottom-0">
  <h2 class="fw-bold mb-0">Log in</h2>
</div>

```

```

<div class="modal-body p-5 pt-0">
  <form class="" (submit)="userCreate($event)">
    <div class="form-floating mb-3">
      <input class="form-control rounded-4" id="floatingInput"
placeholder="floatingInput" [(ngModel)]="userName"
[ngModelOptions]="{standalone: true}">

```

```

    <label for="floatingInput">Username</label>
  </div>
  <div class="form-floating mb-3">
    <input type="password" class="form-control rounded-4"
id="floatingPassword" placeholder="floatingPassword" [(ngModel)]="userPass"
[ngModelOptions]="{standalone: true}">
    <label for="floatingPassword">Password</label>
  </div>
  <button class="w-100 mb-2 btn btn-lg rounded-4 btn-primary"
type="submit">Log in</button>
</form>
</div>

```

#### **A.16 Код файла `log-form.component.ts`**

```

import { Component } from '@angular/core';
import { NgbActiveModal, NgbDatepickerModule } from '@ng-bootstrap/ng-
bootstrap';
import { FormsModule } from '@angular/forms';
import { AuthService } from '../services/auth.service';

@Component({
  selector: 'app-log-form',
  standalone: true,
  imports: [NgbDatepickerModule, FormsModule],
  templateUrl: './log-form.component.html',
  styleUrls: ['./log-form.component.css']
})

```

```

export class LogFormComponent {

    constructor(private authService: AuthService, private modalRef:
NgbActiveModal ) { }

    userPass: string = "";
    userName: string = "";

    userCreate(event: Event): void {
        event.preventDefault();
        this.authService.login(this.userName, this.userPass).then(() => {
            this.modalRef.dismiss();
        });
    }
}

```

#### **A.17 Код файла create-form.component.html**

```

<div class="modal-header p-5 pb-4 border-bottom-0">
    <h2 class="fw-bold mb-0">Add new post</h2>
</div>

<div class="modal-body p-5 pt-0">
    <form class="" (submit)="postCreate($event)">
        <div class="form-floating mb-3">
            <input type="file" class="form-control rounded-4" id="floatingUrl"
placeholder="floatingUrl" (change)="onFileChange($event)">
            <label for="floatingUrl">Image</label>
        </div>

```

```

<div class="form-floating mb-3">
  <input type="text" class="form-control rounded-4" id="floatingInput"
placeholder="floatingInput" [(ngModel)]="postDescription"
[ngModelOptions]="{standalone: true}">
    <label for="floatingInput">Description</label>
</div>
<div class="form-floating mb-3">
  <input type="text" class="form-control rounded-4" id="floatingString"
placeholder="floatingString" [(ngModel)]="postTags"
[ngModelOptions]="{standalone: true}">
    <label for="floatingString">Tags</label>
</div>
<div class="card shadow-sm">
  <img class="bd-placeholder-img" src={{postImage}} width="100%"
height="225">
</div>
<hr>
<button class="w-100 mb-2 btn btn-lg rounded-4 btn-primary"
type="submit">Upload</button>
  <small class="text-muted">This post will be imaged in your own
profile.</small>
</form>
</div>

```

#### **A.18 Код файла create-form.component.ts**

```

import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

```

```
import { FormsModule } from '@angular/forms';
import { PostService } from '../services/post.service';
```

```
@Component({
  selector: 'app-create-form',
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: './create-form.component.html',
  styleUrls: ['./create-form.component.css'],
})
export class CreateFormComponent {
  constructor(private postService: PostService) {}

  postImage: string = "";
  postDescription: string = "";
  postTags: string = ""
  onFileChange(event: any): void {
    let image = event.target.files[0];
    if(image) {
      const reader = new FileReader();
      reader.onload = (e) => {
        this.postImage = reader.result as string;
      }
      reader.readAsDataURL(image);
    }
  }
  postCreate(event: Event): void {
    event.preventDefault();
```

```

let postTags = this.postTags.split(' ');

this.postService.createPost({
  postDescription: this.postDescription,
  postImage: this.postImage,
  postTags,
});
}
}

```

### **A.19 Код файла post.ts**

```

export interface Post {
  postDate: string,
  postId: string,
  postDescription: string,
  postImage: string,
  postCreator: string,
  postTags: string[],
  postLikes: string[],
  isLiked?: boolean,
  isOwn? : boolean
}

export type CreatePostDTO = Omit<Post, "postId" | "postDate" | "postLikes" |
"postCreator">;

export type PostSubject = Post[] | null;

```

### A.20 Код файла **user.ts**

```
export interface User {  
    userName: string,  
    userDescription: string,  
    userAvatar: string,  
    userPass: string,  
}
```

### A.21 Код файла **auth.service.ts**

```
import { HttpClient } from '@angular/common/http';  
import { Injectable } from '@angular/core';  
import { BehaviorSubject } from 'rxjs';  
import { User } from '../interfaces/user';  
import { initializeHeaders } from '../helpers/utills';  
  
@Injectable({  
    providedIn: 'root',  
})  
export class AuthService {  
    constructor(private http: HttpClient) {}  
    private statusSubject = new BehaviorSubject<boolean>(  
        !!localStorage.getItem('accessToken')  
    );  
    private userSubject = new BehaviorSubject<any>(null);  
    private myUserSubject = new BehaviorSubject<any>(null);  
    myUserProfile$ = this.myUserSubject.asObservable();
```

```

userProfile$ = this.userSubject.asObservable();
status$ = this.statusSubject.asObservable();
updateMyUserProfile(profile: User): void {
    this.myUserSubject.next(profile);
}
updateUserProfile(profile: User): void {
    this.userSubject.next(profile);
}
updateStatus(status: boolean): void {
    this.statusSubject.next(status);
}

login(userName: string, userPass: string) {
    return this.http
        .post<{ accessToken: string }>(`http://localhost:3000/user/auth/`, {
            userName,
            userPass,
        })
        .toPromise()
        .then(
            (res: any) => {
                localStorage.setItem('accessToken', res);
                this.updateStatus(true);
                return Promise.resolve();
            },
            (error) => {
                return Promise.reject();
            }
        )
}

```

```

    );
  }

  getUserProfile(paramValue?: string) {
    const urlParams = paramValue ? paramValue : "";
    const headers = paramValue ? {} : initializeHeaders();
    return this.http
      .get<User>(`http://localhost:3000/user/getProfileInfo/` + urlParams, {
        headers,
      })
      .subscribe((res) => {
        paramValue
          ? this.updateUserProfile(res)
          : this.updateMyUserProfile(res);
      });
  }
}

```

## A.22 Код файла **post.service.ts**

```

import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { CreatePostDTO, Post, PostSubject } from '../interfaces/post';
import { BehaviorSubject } from 'rxjs';
import { AuthService } from './auth.service';
import { initializeHeaders } from './helpers/utills';

@Injectable({

```

```

    providedIn: 'root',
  })
export class PostService {
  protected LoggedIn = false;
  constructor(private http: HttpClient, private authService: AuthService) {
    this.authService.status$.subscribe((status) => (this.LoggedIn = status));
  }

  private postArrSubject = new BehaviorSubject<any>(null);
  private postUserArrSubject = new BehaviorSubject<any>(null);
  // private postUserLikesArrSubject = new BehaviorSubject<any>(null);
  // postUserLikesArr$ = this.postUserLikesArrSubject.asObservable();
  postUserArr$ = this.postUserArrSubject.asObservable();
  postArr$ = this.postArrSubject.asObservable();
  updateArray(postArr: Post[]): void {
    this.postArrSubject.next(postArr);
  }

  updateUserArray(postArr: Post[]): void {
    this.postUserArrSubject.next(postArr);
  }

  updateLocalLikes(postId: string, userName: string): void {
    let localValue: any;
    this.postArr$.subscribe((value) => (localValue = value));
    localValue.map((postEl: Post) => {
      if (postId == postEl.postId) {
        postEl.isLiked = !postEl.isLiked;
      }
    });
  }
}

```

```

        if (postEl.isLiked) {
            postEl.postLikes = [userName, ...postEl.postLikes];
        } else {
            postEl.postLikes.splice(postEl.postLikes.indexOf(userName), 1);
        }
    }
    return postEl;
});
this.updateArray(localValue);
}

findByLikes(userName: string) {
    return this.http.get<Post>('http://localhost:3000/post/byLikes/');
}

findAll(search?: string, userPage?: boolean) {
    let config = {};
    if (this.LoggedIn) {
        let headers = initializeHeaders();
        config = { headers };
    }
    const urlParams = search ? `?search=${search}` : "";
    return this.http.get<Post[]>(
        'http://localhost:3000/post/' + urlParams,
        config
    );
}

createPost(post: CreatePostDTO) {
    const headers = initializeHeaders();

```

```

return this.http
  .post<Post>('http://localhost:3000/post/', post, {
    headers: headers,
  })
  .subscribe((el: Post) => {
    let postArr = [];
    this.postArr$.subscribe((value) => (postArr = value));
    postArr.push(el);
    this.updateArray(postArr);
  });
}

updateLikes(postId: string) {
  const headers = initializeHeaders();
  return this.http
    .get<any>(`http://localhost:3000/post/updateLikes/${postId}/`, {
      headers,
    })
    .subscribe((res) => {
      this.updateLocalLikes(postId, res);
    });
}
}

```

### A.23 Код файла `user.service.ts`

```

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

```

```

import { User } from '../interfaces/user';

@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor(private http: HttpClient) { }

  createUser(user: User) {
    return this.http.post<{user: User}>('http://localhost:3000/user/',
user).subscribe((user)=>{console.log(user)});
  }
}

```

#### **A.24 Код файла auth.reducer.ts**

```

import { createReducer, createSelector, on } from '@ngrx/store';

export interface AuthState {
  loggedIn: Boolean;
}

export const initialState: AuthState = {
  loggedIn: !!localStorage.getItem('accessToken'),
};

```

```
export const loggedInSelector = createSelector((state: AuthState) => state,
(auth) => auth)
```

```
export const AuthReducer = createReducer(
  initialState
);
```

#### **A.25 Код файла modal.action.ts**

```
import { createAction, props } from '@ngrx/store';

export const setCurrentModal = createAction('[Modal] Call to Close', props<{
modal: any }>());
```

#### **A.26 Код файла modal.reducer.ts**

```
import { createReducer, on } from '@ngrx/store';
import * as modalActions from './modal.actions';
```

```
export interface ModalState {
  modal: any;
}
```

```
export const initialState: ModalState = {
  modal: null,
};
```

```
export const ModalReducer = createReducer(
  initialState,
```

```

    on(modalActions.setCurrentModal, (state, { modal }) => ({ ...state, modal
  })),
);

```

### A.27 Код файла index.js

```

const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const webtoken = require('jsonwebtoken');
const app = express();

mongoose.connect(
  'mongodb+srv://clockytema:OqF8I9HUtY6irQ9D@cluster0.pwso5wu.mong
odb.net/?retryWrites=true&w=majority',
  {useNewUrlParser: true,
   useUnifiedTopology: true,
  });

app.use(cors());
app.use(express.json());
app.use(express.urlencoded({extended: true}));
app.use((req, res, next) => {

  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With,
Content-Type, Accept');

```

```

    res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE,
OPTIONS');
    next()

  });

  require('./src/modules/post/post.router')(app);
  require('./src/modules/user/user.router')(app);

  app.use(function (err, req, res, next) {
    let errorContent;
    try {
      errorContent = JSON.parse(err.message);
    } catch (e) {
      const isJoiError = !!err?.error?.details;
      const joiMessage = isJoiError ?
        err.error.details.map(detail => detail.message,) : null;

      errorContent = { message: joiMessage || err.message };
    }
    res.status(500).send({ status: 'Internal Server Error', ...errorContent });

    next();
  });

  app.listen(process.env.PORT || 3000, () => console.log(`Server started at port
${process.env.PORT || 3000}`));

```

**A.28 Код файла auth.middleware.js**

```
const webtoken = require('jsonwebtoken')

module.exports = (req, res, next) => {
  if (req.method === 'OPTIONS') {
    return next()
  }

  try {

    const token = req.headers.authorization

    if (!token) {
      return res.status(401).json({ message: 'Unauthorized' })
    }

    const decoded = webtoken.verify(token, 'enigmamachine')
    req.user = decoded
    next()

  } catch (e) {
    res.status(401).json({ message: 'Unauthorized' })
  }
}
```

**A.29 Код файла non-auth.middleware.js**

```
const webtoken = require("jsonwebtoken");
```

```

module.exports = (req, res, next) => {
  if (req.method === "OPTIONS") {
    return next();
  }

  try {
    if (req.headers.authorization) {
      const token = req.headers.authorization;
      const decoded =webtoken.verify(token, "enigmamachine");
      req.user = decoded;
    }
    next();
  } catch (e) {
    next();
  }
};

```

### **A.30 Код файла `post.model.js`**

```

const {Schema, model} = require('mongoose');

const schema = new Schema({
  postDate: {
    type: Date,
    default: new Date(Date.now()),
  },
  postId: {

```

```
    type: String,
    require: true,
  },
  postDescription: {
    type: String,
  },
  postImage: {
    type: String,
    require: true,
  },
  postCreator: {
    type: String,
    require: true,
  },
  postTags: {
    type:[String],
  },
  postLikes: {
    type:[String],
    default: [],
    require: false
  }
});
```

```
module.exports = model('post', schema);
```

**A.31 Код файла user.model.js**

```
const { Schema, model } = require('mongoose');
```

```
const schema = new Schema({  
  userName: {  
    type: String,  
    require: true,  
    unique: true  
  },  
  userDescription: {  
    type: String,  
  },  
  userAvatar: {  
    type: String,  
  },  
  userPass: {  
    type: String,  
    require: true  
  },  
});
```

```
module.exports = model('user', schema);
```

**A.32 Код файла post.controller.js**

```
const post = require("../post.service");  
const uuidv4 = require("uuid").v4;
```

```

const postController = {
  findByLikes: async (req, res, next) => {
    try {
      const likes = req.query.likes;
      res.json(await post.findByLikes(likes));
    } catch (e) {
      next(e);
    }
  },
  findAll: async (req, res, next) => {
    const search = req.query.search;
    try {
      let response = search
        ? await post.findByTagsOrCreator(search)
        : await post.findAll();
      res.json(response);
    } catch (e) {
      next(e);
    }
  },
  create: async (req, res, next) => {
    try {
      const item = req.body;
      item.postCreator = req.user.userName;
      item.postId = uuidv4();
      res.json(await post.create(item));
    } catch (e) {
      next(e);
    }
  }
};

```

```

    }
  },
  delete: async (req, res, next) => {
    try {
      const id = req.params.postId;
      res.json(await post.delete(id));
    } catch (e) {
      next(e);
    }
  },
  updateLikes: async (req, res, next) => {
    try {
      const userName = req.user.userName;
      const id = req.params.postId;
      await post.updateLikes(id, userName);
      res.json(userName);
    } catch (e) {
      next(e);
    }
  },
};
module.exports = postController;

```

### **A.33 Код файла post.router.js**

```

const controller = require('./post.controller');
const express = require('express');
const { createValidator } = require('express-joi-validation');

```

```

const {
  postCreateSchema,
  updateLikesSchema,
  getByLikesSchema,
  postSearchSchema
} = require('./post.validation.scheme');
const auth = require("../middleware/auth.middleware")
const nonReqAuth = require("../middleware/non-auth.middleware")

const router = express.Router();
const validator = createValidator();
module.exports = app => {
  router.get('/byLikes', auth, validator.query(getByLikesSchema, {passError:
true})), controller.findByLikes);
  router.get('/', nonReqAuth, controller.findAll);
  router.post('/', auth, validator.body(postCreateSchema, {passError: true}),
controller.create);
  router.delete('/postId', auth, controller.delete);
  router.get('/updateLikes/:postId', auth, controller.updateLikes);
  app.use('/post', router);
}

```

#### **A.34 Код файла post.service.js**

```

const Post = require("../models/post.model");

const postService = {
  findByTagsOrCreator: async (search) => {

```

```

const items = await Post.find({
  $or: [{ postTags: { $in: search } }, { postCreator: search }],
});
return items;
},
findByLikes: async (postLikes) => {
  const items = await Post.find({ postLikes: { $in: postLikes } });
  return items;
},
findAll: async () => {
  let items = await Post.find();
  return items;
},
create: async (item) => {
  const post = new Post(item);
  const savedPost = await post.save();
  return savedPost;
},
delete: async (postId) => {
  return Post.deleteOne({ postId });
},
updateLikes: async (postId, userName) => {
  const post = await Post.findOne({ postId });
  let postLikes = [];
  if (!post.postLikes.includes(userName)) {
    postLikes = [userName, ...post.postLikes];
  } else {
    postLikes.splice(postLikes.indexOf(userName), 1);
  }
}

```

```

    }
    return Post.updateOne({ postId: post.postId }, { postLikes });
  },
};

```

```
module.exports = postService;
```

### A.35 Код файла `post.validation.scheme.js`

```

const Joi = require('joi');

module.exports = {
  postCreateSchema: Joi.object({
    postDescription: Joi.string().optional(),
    postImage: Joi.string().required(),
    postTags: Joi.array().items(Joi.string()).optional(),
  }).options({ abortEarly: false }),

  updateLikesSchema: Joi.object({
    userName: Joi.string().required(),
  }).options({ abortEarly: false }),

  getByLikesSchema: Joi.object({
    likes: Joi.array().items(Joi.string()).required(),
  }),

  postSearchSchema: Joi.object({
    search: Joi.string().required()
  })
};

```

```

    }),
  };

```

### A.36 Код файла `user.controller.js`

```

const user = require('./user.service');

const userController = {
  findName: async (req, res, next) => {
    try {
      const name = req.params.userName;
      res.json(await user.findName(name));
    } catch (e) { next(e); }
  },
  create: async (req, res, next) => {
    try {
      const unit = req.body;
      res.json(await user.create(unit));
    } catch (e) { next(e); }
  },
  auth: async (req, res, next) => {
    try{
      const {userName, userPass} = req.body;
      const accessToken = await user.auth(userName, userPass);
      res.json(accessToken);
    } catch (e) {
      next(e);
    }
  }
}

```

```

},
getProfileInfo: async(req, res, next) => {
  try {
    const user = req.user;
    res.json(user);
  } catch (e) {next(e); }
}};

```

```
module.exports = userController;
```

### **A.37 Код файла user.router.js**

```

const controller = require('./user.controller');
const express = require('express');
const { createValidator } = require('express-joi-validation');
const {
  userCreateSchema,
  userLoginSchema
} = require('./user.validation.scheme');
const auth = require("../middleware/auth.middleware")

const router = express.Router();
const validator = createValidator();
module.exports = app => {
  router.get('/getProfileInfo/:userName', controller.findName);
  router.get('/getProfileInfo', auth, controller.getProfileInfo);
  router.post('/', validator.body(userCreateSchema, {passError: true}),
controller.create);

```

```

    router.post('/auth', validator.body(userLoginSchema, {passError: true}),
controller.auth);

    app.use('/user', router)});

```

### A.38 Код файла **user.service.js**

```

const User = require("../models/user.model");
constwebtoken = require("jsonwebtoken");

const userService = {
  findName: async (name) => {
    return await User.findOne({ userName: name });
  },
  create: async (unit) => {
    const user = new User(unit);
    const savedUser = await user.save();
    return savedUser;
  },
  auth: async (userName, userPass) => {
    try {
      const user = await User.findOne({
        userName,
        userPass,
      });
      if (user) {
        const accessToken = jwt.sign(JSON.stringify(user),
"enigmamachine");
        return accessToken;

```

```

    }
  } catch (e) {
    console.log(e)
    throw new Error(
      "Помилка авторизації: користувача не знайдено, або введено не
вірний пароль"
    );
  }
},});
module.exports = userService;

```

#### **A.39 Код файлу `user.validation.scheme.js`**

```

const Joi = require('joi');

module.exports = {
  userCreateSchema: Joi.object({
    userName: Joi.string().required(),
    userPass: Joi.string().required(),
    userDescription: Joi.string().optional(),
    userAvatar: Joi.string().optional(),
  }).options({ abortEarly: false }),
  userLoginSchema: Joi.object({
    userName: Joi.string().required(),
    userPass: Joi.string().required(),
  }).options({ abortEarly: false }),
};

```

**ДОДАТОК Б**  
**Слайди презентації**

Національний університет «Запорізька політехніка»  
Кафедра програмних засобів

Дипломна кваліфікаційна робота магістра

ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ КОМПЛЕКСНИХ СИСТЕМНИХ ПІДХОДІВ ДО РОЗРОБКИ  
ВЕБЗАСТОСУНКІВ НА ПРИКЛАДІ ФОТОХОСТИНГУ

Виконав  
Студент групи КНТ-122м

Артем КЛОКОВ

Керівник  
к.т.н.

Ольга ГЛАДКОВА

Рисунок Б.1 – Слайд №1

**Об'єкт дослідження** – процес розробки вебзастосунку.

**Предмет дослідження** – комплексні системні підходи до розробки вебзастосунку.

**Мета роботи** полягає в дослідженні комплексних системних підходів до розробки вебзастосунків на прикладі фотохостингу, розгляд етапів та ролей у ході розробки.

**Для досягнення мети необхідно розв'язати такі завдання:**

- розгляд послідовності етапів розробки вебзастосунків;
- аналіз експертних ролей під час розробки;
- встановлення функціональних вимог до власного застосунку;
- розробка користувацького інтерфейсу застосунку;
- визначення архітектури та структури застосунку;
- реалізація вебзастосунку фотохостингу;
- розгляд підходів та методів тестування вебзастосунків.

Рисунок Б.2 – Слайд №2

## Життєвий цикл вебзастосунку та задіяні в розробці експертні ролі

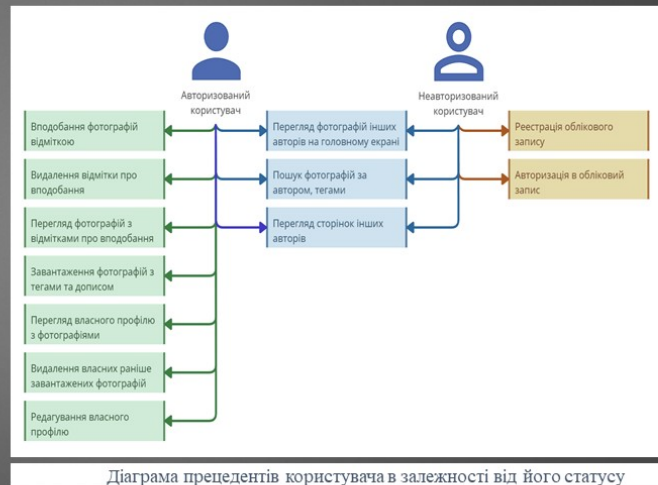
Реєстрація та авторизація користувача;

Створення власних та відображення існуючих постів;

Можливість пошуку постів за тегами або автором;

Перегляд профілей користувачів;

Можливість оцінки та перегляду раніше оцінених користувачем постів.



Діаграма прецедентів користувача в залежності від його статусу

Рисунок Б.3 – Слайд №3

## Методи вибору засобів розробки

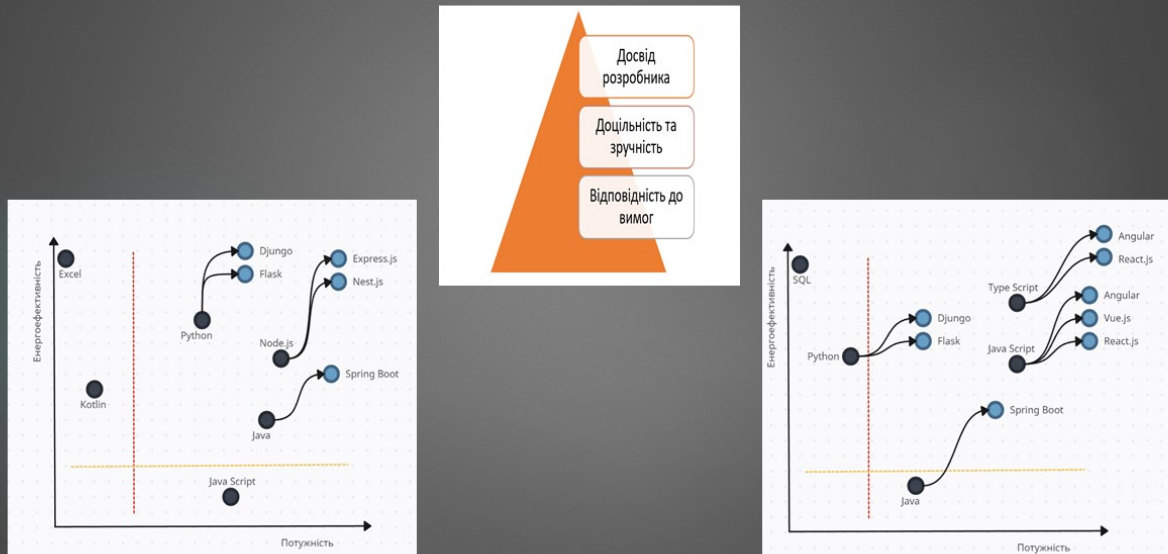


Рисунок Б.4 – Слайд №4

## Обрані засоби розробки

| Мови розмітки та стилів           | HTML та CSS            |
|-----------------------------------|------------------------|
| Мови програмування                | JavaScript, TypeScript |
| Фреймворк для клієнтської частини | Angular, Bootstrap 5   |
| Фреймворк для серверної частини   | Express.js             |
| СКБД                              | MongoDB                |

Рисунок Б.5 – Слайд №5

## Вибір середовища розробки

| Назва                     | Переваги                                                                                                                                                           | Недоліки                                                                                                                                                                                                                          |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Visual Studio Code</b> | Безоплатність використання<br>Підтримка великої кількості розширень<br>Зручність в розгортанні декількох проектів одночасно<br>Широка аудиторія продукту           | Ресурсоемність розширень<br>пригальмує швидкість роботи з проектами<br>Мала кількість вбудованих функцій<br>Залежність багатьох функцій від мережі                                                                                |
| <b>WebStorm</b>           | Досить швидкий<br>Має велику кількість вбудованих функцій у тому числі перевірки коду, що набагато спрощує написання коду<br>Має можливість встановлення розширень | Комерційне програмне забезпечення, потребує оплати для використання<br>Перевантаження вбудованими функціями, які не будуть використані<br>Обмежена підтримка мов програмування<br>Складність використання при відсутності досвіду |

Рисунок Б.6 – Слайд №6

## Архітектура застосунку

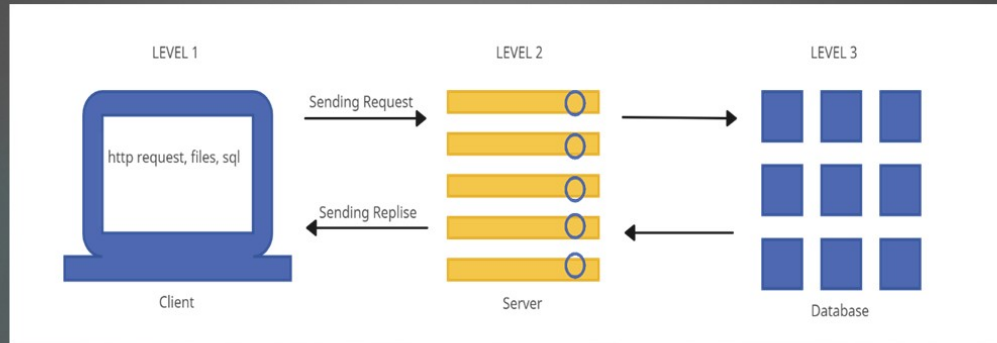
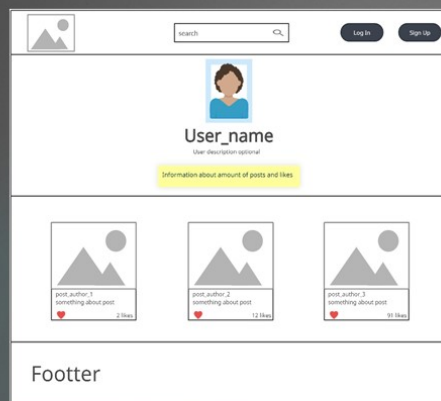


Рисунок Б.7 – Слайд №7

## Прототип інтерфейсу користувача



Прототип інтерфейсу для неавторизованого користувача

Прототип інтерфейсу для авторизованого користувача

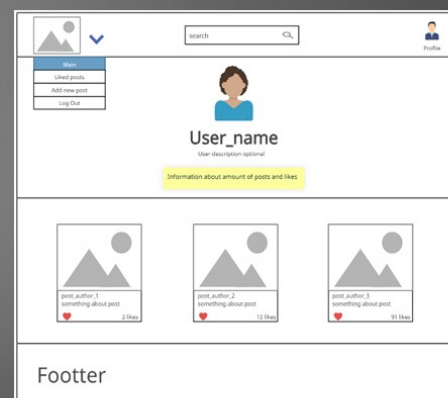


Рисунок Б.8 – Слайд №8

## Структура застосунку

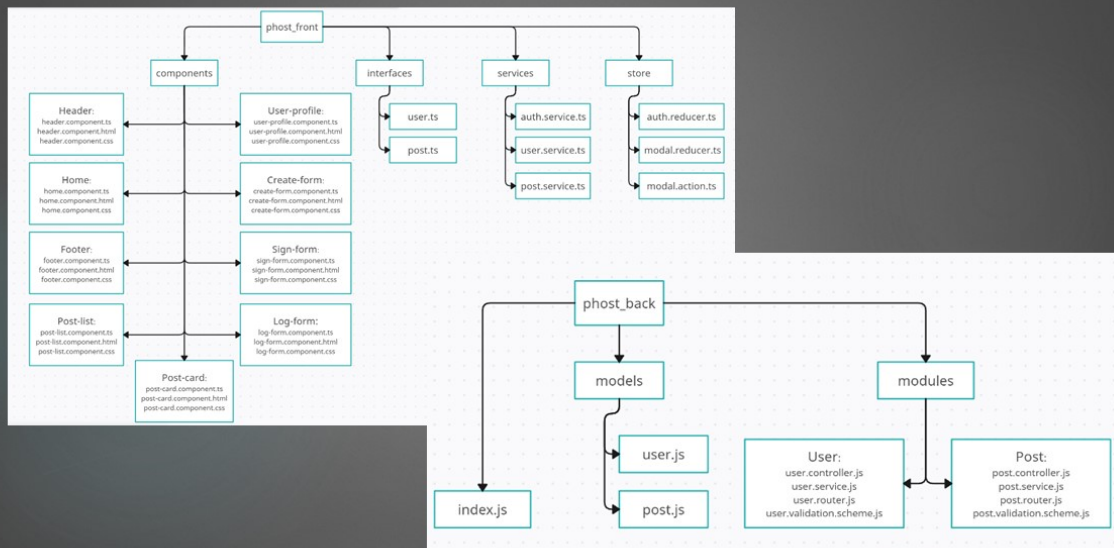


Рисунок Б.9 – Слайд №9

## Результати розробки

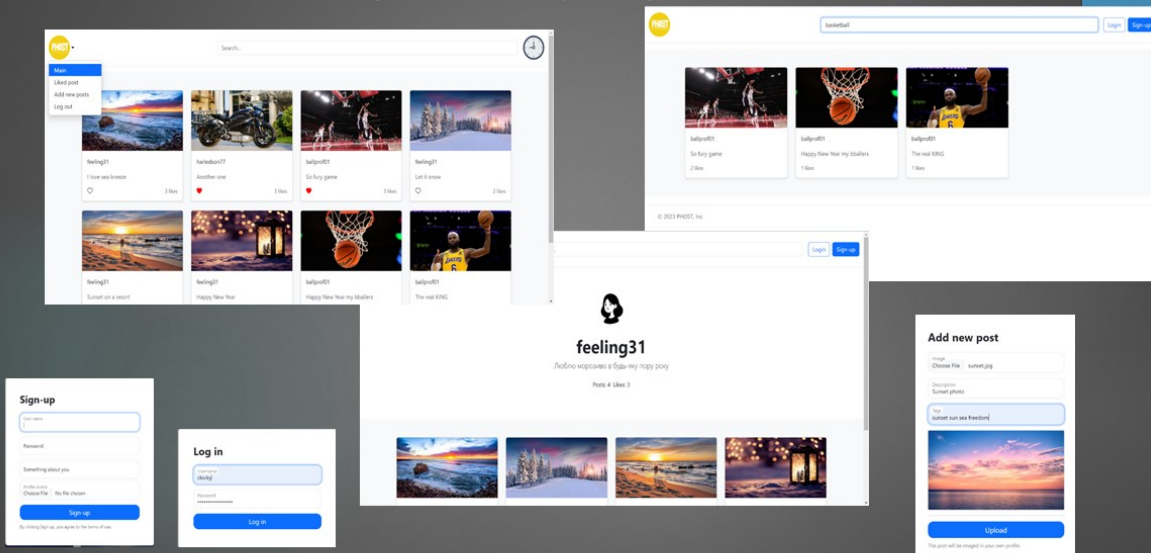


Рисунок Б.10 – Слайд №10

## Висновки

В ході виконання дипломної кваліфікаційної роботи магістра було розв'язано такі задачі:

- розглянуто послідовність етапів розробки вебзастосунків;
- проаналізовано життєвий цикл програмного забезпечення та експертні ролі;
- проаналізовано та обрано інструментарій розробки застосунку;
- розроблено та обґрунтовано концепт інтерфейсу користувача;
- визначено модель архітектури застосунку та його структуру;
- кінцева реалізація програмного забезпечення з описом алгоритмів роботи програми;
- створення керівництва користувача з визначенням призначення застосунку, технічних вимог, описом та прикладами роботи розробленого рішення.

Загалом у цій роботі, на прикладі розробленого застосунку фотохостингу, досліджено комплексні підходи та методи реалізації вебзастосунку впродовж всього циклу розробки. Послідовно розглянуті етапи розробки, та підходи до реалізації застосунку з точки зору різних експертних ролей.

**Практична значущість** роботи полягає у тому, що був розроблений вебзастосунок фотохостингу з елементами соціальної мережі з описом повного процесу розробки починаючи з бізнес аналітики і закінчуючи тестуванням робочої версії застосунку.

**Наукова новизна** дослідження полягає у впровадженні реорганізації процесу розробки для досягнення цілей розробки оптимальним шляхом.

Рисунок Б.11 – Слайд №11