**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ**

**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ЕНЕРГЕТИЦІ**

КУРСОВА РОБОТА

**з дисципліни:** ««Основи Веб-програмування»

**на тему:** «Веб-додаток для хостингу зображень»

**Керівник**: **Виконав**:

Гагарін. О.О. Коваленко Данило Олексійович

студент 2 курсу

«Допущено до захисту» групи ТВ-31

(особистий підпис керівника)

« » 20 р.

Захищено з оцінкою (особистий підпис виконавця)

«25» травня 2025р.

(оцінка)

Посилання на **GitHub** репозиторій https://github.com/psy-radiation/WEBkursova/tree/main

**Київ - 2025**

# АНОТАЦІЯ

**Тема курсової роботи:** "Веб-додаток для хостингу зображень".

Мета роботи полягає у створенні простого та зручного веб-додатку, який дозволяє користувачам завантажувати, переглядати та зберігати зображення. Проєкт також включає функції реєстрації, авторизації, ведення профілю користувача, можливість завантажувати аватар та фон, а також перегляд усіх своїх зображень.

У вступній частині описується актуальність подібних веб-додатків, а також пояснюється, чому саме цей функціонал є корисним для користувачів, які хочуть зберігати або ділитися зображеннями.

Основна частина роботи містить опис технологій, які були використані під час розробки. Зокрема, описано використання FastAPI для серверної частини, PostgreSQL для бази даних та HTML, CSS і JavaScript для фронтенду. Описано, як реалізовано навігацію, завантаження файлів, зберігання користувацьких даних, а також взаємодію між клієнтом і сервером.

Також у роботі пояснюється, як була організована структура проєкту, як працює система автентифікації та як користувач може бачити власні зображення або переходити на сторінки інших користувачів.

У висновках підсумовано виконану роботу, визначено переваги створеного додатку, а також можливі напрямки для його покращення в майбутньому.

# ABSTRACT

**Course project topic:** "Image Hosting Web Application".

The goal of the project is to create a simple and user-friendly web application that allows users to upload, view, and store images. The project also includes features such as user registration, login, profile management, the ability to upload an avatar and background, and viewing all images uploaded by the user.

The introduction describes the relevance of such web applications and explains why this functionality is useful for users who want to store or share images.

The main part of the work contains a description of the technologies used during development. In particular, it describes the use of FastAPI for the backend, PostgreSQL for the database, and HTML, CSS, and JavaScript for the frontend. It explains how navigation, file uploads, user data storage, and interaction between the client and server are implemented.

The paper also explains how the project structure is organized, how the authentication system works, and how users can view their own images or visit other users’ pages.

The conclusion summarizes the completed work, outlines the advantages of the created application, and describes possible directions for further improvement.

# ЗМІСТ

[АНОТАЦІЯ 1](#_bookmark0)

[ABSTRACT 2](#_bookmark1)

[ЗМІСТ 3](#_bookmark2)

[ВСТУП 4](#_bookmark3)

1. [ПРО ПРОЄКТ 5](#_bookmark4)
2. [ІНСТРУМЕНТИ РОЗРОБКИ 7](#_bookmark5)
   1. [Фреймворк 7](#_bookmark6)
   2. [Базa даних 8](#_bookmark7)
   3. [Архітектурний шаблон 9](#_bookmark8)
   4. Інструменти розгортання та запуску…………………………11
3. [РЕАЛІЗОВАНИЙ ФУНКЦІОНАЛ СИСТЕМИ 1](#_bookmark9)2
   1. [Авторизація та аутентифікація 1](#_bookmark10)2
   2. [Перегляд деталей зображення 1](#_bookmark11)3
   3. [Завантаження зображень 1](#_bookmark12)3
4. [ВИГЛЯД СТОРІНОК WEB-САЙТУ 1](#_bookmark14)5

[ВИСНОВКИ 2](#_bookmark15)2

[ВИКОРИСТАННІ ДЖЕРЕЛА 24](#_bookmark16)

[ДОДАТКИ 2](#_bookmark17)5

[Додаток А. Фрагмент коду для роботи головної сторінки 2](#_bookmark18)5

# ВСТУП

У сучасному світі актуальність теми проявляється у зростаючій популярності веб-платформ для обміну зображеннями, особливо серед користувачів, які прагнуть ділитися власними роботами або зберігати зображення в зручному та доступному форматі. Додаток VibeJPG надає можливість завантаження та перегляду зображеннями, а також створення особистого профілю з аватаром та фоновим оформленням.

Під час роботи над завданням було проведено детальний аналіз вимог до додатку, визначено основні функціональні можливості, такі як реєстрація користувачів, авторизація, зберігання зображень, а також реалізація особистого профілю та перегляд зображень та профілів інших користувачів.

Створено структуру бази даних, яка відповідає потребам додатку, забезпечено безпеку завдяки механізмам аутентифікації та перевірки доступу до ресурсів. Для бекенду використано фреймворк FastAPI, а для фронтенду — HTML, CSS і JavaScript.

Реалізовано відображення зображень користувачів на головній сторінці та сторінках профілю, а також створено зручний інтерфейс для завантаження нових зображень, аватарів та фонів, а також для більш детального перегляду чи зберігання зображень окремо.

Проведено тестування основної функціональності та усунення виявлених помилок, що забезпечило стабільну роботу додатку та зручність для користувача.

# ПРО ПРОЄКТ

Проєкт націлений на створення інтернет-платформи для користувачів, які зацікавлені у зберіганні, перегляді та обміні зображеннями. Платформа забезпечує користувачам простий і зручний інтерфейс для завантаження своїх зображень, перегляду робіт інших, а також взаємодії з контентом через лайки, коментарі та перегляди.

Користувачі можуть створювати власні облікові записи, заповнюючи особисту інформацію. Після авторизації їм надається можливість керувати своїм профілем, завантажувати зображення, редагувати аватар та фон профілю, переглядати власну галерею та видаляти непотрібні файли.

На головній сторінці сайту відображаються останні додані зображення з можливістю перейти до профілю автора або переглянути повну версію.

Платформа побудована з урахуванням безпеки — особиста інформація зберігається у захищеному вигляді, а всі дії проходять перевірку доступу. Проєкт є гнучким до розширення і в подальшому може бути доповнений новими функціями, такими як категорії, підписки на авторів, персональні повідомлення та інше.

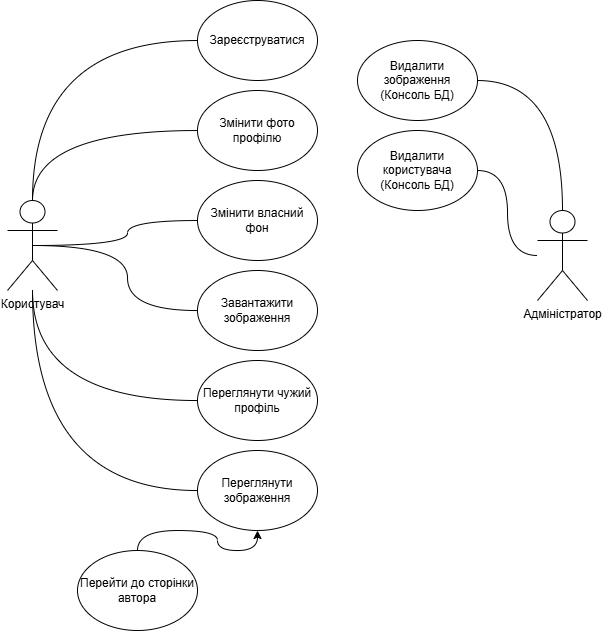


Рис. 1.1 - Діаграма прецедентів

# ІНСТРУМЕНТИ РОЗРОБКИ

## Фреймворк

Для реалізації проєкту було обрано сучасний фреймворк під назвою **FastAPI**.

**Фреймворк** (framework) — це набір інструментів, бібліотек та правил, який допомагає розробникам створювати програмні додатки. Він задає структуру проєкту, дозволяє швидко реалізувати типові задачі, полегшує обробку запитів, роботу з базами даних та побудову API.

**FastAPI** — це високопродуктивний вебфреймворк для мови програмування **Python**, створений Себастьяном Раміресом у 2018 році. Він побудований на основі стандарту OpenAPI та бібліотеки Starlette, що забезпечує підтримку асинхронної обробки запитів.

FastAPI відомий своєю швидкодією, простотою у використанні та високою продуктивністю. Він дозволяє створювати API-сервери, які обробляють тисячі запитів на секунду, що робить його ідеальним вибором для вебплатформ, таких як хостинги зображень, де користувачі часто надсилають запити на завантаження, перегляд і взаємодію з контентом.

Ще однією перевагою FastAPI є його тісна інтеграція з автоматичною генерацією документації, що суттєво спрощує розробку та тестування серверної частини. Це дозволяє швидко перевіряти, як працюють ендпоінти, без потреби у зовнішніх інструментах.

Завдяки простому синтаксису, підтримці типізації, високій швидкості роботи та зручності розгортання, FastAPI чудово підходить для створення масштабованих і зручних у підтримці сучасних вебдодатків.

## Базa даних

Одним із ключових елементів будь-якого вебзастосунку є надійне, структуроване та безпечне збереження даних. Це можуть бути облікові дані користувачів, інформація про зображення, лайки, коментарі, історія взаємодій тощо. Після аналізу доступних систем керування базами даних, для реалізації проєкту було обрано **PostgreSQL**.

**PostgreSQL** — це потужна об’єктно-реляційна система керування базами даних з відкритим вихідним кодом. Вона підтримує складні запити, транзакції, збережені процедури, роботу з JSON, розширення геоданих (PostGIS), а також гнучку систему прав доступу, що робить її ідеальним вибором для масштабованих вебплатформ.

У нашому проєкті PostgreSQL використовується для зберігання:

* інформації про користувачів (логін, електронна пошта, аватар тощо);
* завантажених зображень (назва, шлях до файлу, автор, дата публікації тощо);
* взаємодії користувачів (лайки, дизлайки, коментарі);
* технічних метаданих і логів.

PostgreSQL чудово інтегрується з **FastAPI**, завдяки чому можна легко налаштувати підключення, керувати міграціями, створювати запити до бази, а також використовувати ORM-інструменти, такі як **SQLAlchemy** або **Tortoise ORM**, для забезпечення високого рівня абстракції та зручності в роботі з таблицями бази даних.

Завдяки своїй стабільності, масштабованості та широким можливостям, PostgreSQL став надійною основою для серверної частини нашої системи, що забезпечує ефективне та безпечне зберігання всіх необхідних даних.

## Архітектурний шаблон

Архітектурні шаблони відіграють важливу роль у побудові структурованих, підтримуваних та масштабованих вебзастосунків. Вони допомагають чітко розділити логіку застосунку на окремі компоненти, кожен з яких має свою відповідальність. Це значно спрощує процес розробки, тестування, а також полегшує впровадження нових функцій у майбутньому.

Для реалізації нашого проєкту хостингу зображень було обрано **архітектурний шаблон "розділення на шари" (Layered Architecture)**, який чітко структурує застосунок на логічні частини: **API-рівень, рівень сервісів, рівень доступу до даних та рівень моделей**.

**1. API-рівень (Controllers або Routers):**  
Цей шар відповідає за обробку HTTP-запитів користувачів. Він приймає вхідні дані, викликає відповідні сервіси та повертає відповіді у форматі JSON. У FastAPI він реалізується через маршрутизатори (Routers).

**2. Рівень сервісів (Business Logic):**  
Тут розміщується бізнес-логіка застосунку: перевірки, взаємодія з моделями, обробка даних тощо. Це дозволяє тримати API-рівень "тонким" і відповідальним лише за маршрутизацію.

**3. Рівень доступу до даних (Repositories):**  
Цей шар відповідає за взаємодію з базою даних через ORM (у нашому випадку — SQLAlchemy). Він містить запити до бази, такі як створення, оновлення, видалення чи отримання записів.

**4. Рівень моделей (Schemas/Models):**  
Схеми Pydantic використовуються для валідації та серіалізації даних, а моделі SQLAlchemy — для відображення структури бази даних.

Такий підхід забезпечує **чисту архітектуру**, що легко тестується, масштабовується та дозволяє різним розробникам паралельно працювати над окремими модулями без конфліктів. Шаблон "розділення на шари" ідеально підходить для сучасних REST API-застосунків, таких як наш сайт-хостинг зображень, побудований на FastAPI.

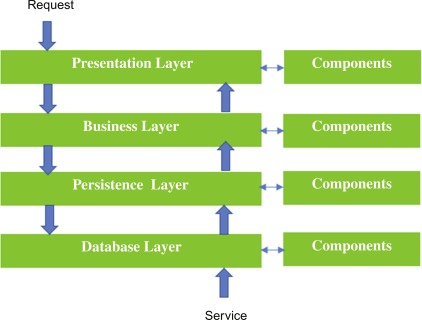


Рис. 2.1 – Архітектурний патерн LA

### **2.4 Інструменти розгортання та запуску**

Для зручного запуску та перенесення проєкту на будь-який сервер була використана технологія **Docker**. Docker дозволяє створювати ізольовані контейнери з усіма залежностями, необхідними для роботи застосунку. Завдяки цьому запуск бекенду (FastAPI), бази даних (PostgreSQL) та інших компонентів здійснюється швидко, стабільно і незалежно від середовища, в якому виконується система.

У межах проєкту створено Dockerfile для бекенду, а також конфігураційний файл docker-compose.yml, що забезпечує одночасний запуск усіх необхідних сервісів. Це суттєво спрощує як локальну розробку, так і розгортання на продакшн-сервері.

Для **перманентного зберігання даних** бази даних та завантажених користувачами зображень були змонтовані **volume-теки** на хост-машині. Це забезпечує збереження всієї важливої інформації навіть після перезапуску або повного ребілду контейнерів. Зокрема:

* база даних PostgreSQL використовує зовнішній том для збереження таблиць та іншої службової інформації;
* каталог для завантажених зображень також винесений у спільну теку між хостом і контейнером, що дозволяє зберігати фото незалежно від життєвого циклу контейнера.

Для хостингу **фронтенду** було використано просте рішення на основі **Python HTTP-сервера**. За допомогою стандартного модуля http.server або пакета uvicorn, користувачам надається доступ до HTML/CSS/JS-файлів без потреби в додатковому веб-сервері. Такий підхід є легким у реалізації та ефективним для невеликих або середніх проєктів, дозволяючи швидко розгортати інтерфейс при чому моделюючи, що бекенд працює на іншому сервері, в нашому випадку просто іншому порті localhost.

# РЕАЛІЗОВАНИЙ ФУНКЦІОНАЛ СИСТЕМИ

## Авторизація та автентифікація

Користувач має можливість зареєструватися на сайті для отримання повного функціоналу. Перегляд загальнодоступних зображень, сторінок інших користувачів або головної стрічки можливий без облікового запису, однак для завантаження зображень необхідна авторизація.

Реєстрація починається з переходу до відповідного розділу або натискання на кнопку. Користувачу пропонується вказати **електронну пошту** та **пароль**.  
З метою безпеки реалізовано такі перевірки:

* електронна пошта має бути унікальною та відповідати правильному формату (перевірка через регулярні вирази),
* пароль також має бути унікальним.

У разі помилки введення даних користувачу виводяться зрозумілі повідомлення про некоректність або повторність введеної інформації.

Після успішної реєстрації користувач автоматично потрапляє назад на головну.

Також передбачена сторінка входу для вже зареєстрованих користувачів. Після авторизації завантаження зображень стають доступними.

Для реалізації механізмів реєстрації та авторизації у проєкті використовувалася стандартна інтеграція з JWT (JSON Web Tokens), яка дозволяє надійно зберігати сесію користувача після входу. Усі дії, що потребують авторизації, захищені перевіркою токена доступу.

## Перегляд деталей зображення

Основна мета сайту – дати людям можливість ділитися зображеннями, продивлятися зображення інших та дізнаватися хто їх автор.

Користувачі бачать на сайті галерею зображень, та можуть натиснути на будь-яку щоб дізнатися про автора, побачити її в повному розмірі, чи скачати через контекстне меню. Дані про те чи інше зображення зберігаються в базі даних і можуть бути змінені у СУБД.

## Завантаження зображень

Користувач має змогу завантажувати зображення на сайт після авторизації. Ця функціональність є однією з ключових у межах платформи, адже вона дозволяє користувачам ділитися власними роботами або цікавими зображеннями з іншими.

На головній сторінці, або у своєму профілі, користувач натискає на кнопку **«Завантажити зображення»**, після чого відкривається форма, де можна обрати файл із локального пристрою. Користувач також має можливість додати короткий опис до зображення перед його публікацією.  
Після надсилання форма перевіряється:  
– зображення повинно бути у форматі .jpg, .jpeg, або .png;  
– розмір файлу не повинен перевищувати задане обмеження (наприклад, 10 МБ).

Після успішного завантаження зображення стає доступним на головній сторінці сайту, а також зберігається у розділі **"Мої зображення"** в особистому кабінеті користувача. Звідти його можна видалити або редагувати опис.

Окрім публікацій, користувач може **оновлювати свій аватар та фонове зображення профілю**. Це також виконується через особистий кабінет. При зміні аватара або фону користувач завантажує відповідне зображення, після чого воно автоматично оновлюється на сторінці профілю та відображається на всіх публікаціях, які він створив.

Для реалізації цієї функціональності використовується асинхронна взаємодія між клієнтом та сервером за допомогою HTTP-запитів FastAPI. Збереження сесій користувача, перевірка авторизації перед завантаженням, а також керування файлами реалізовані на основі JWT (JSON Web Tokens) та контролю прав доступу на серверному боці.

# ВИГЛЯД СТОРІНОК WEB-САЙТУ

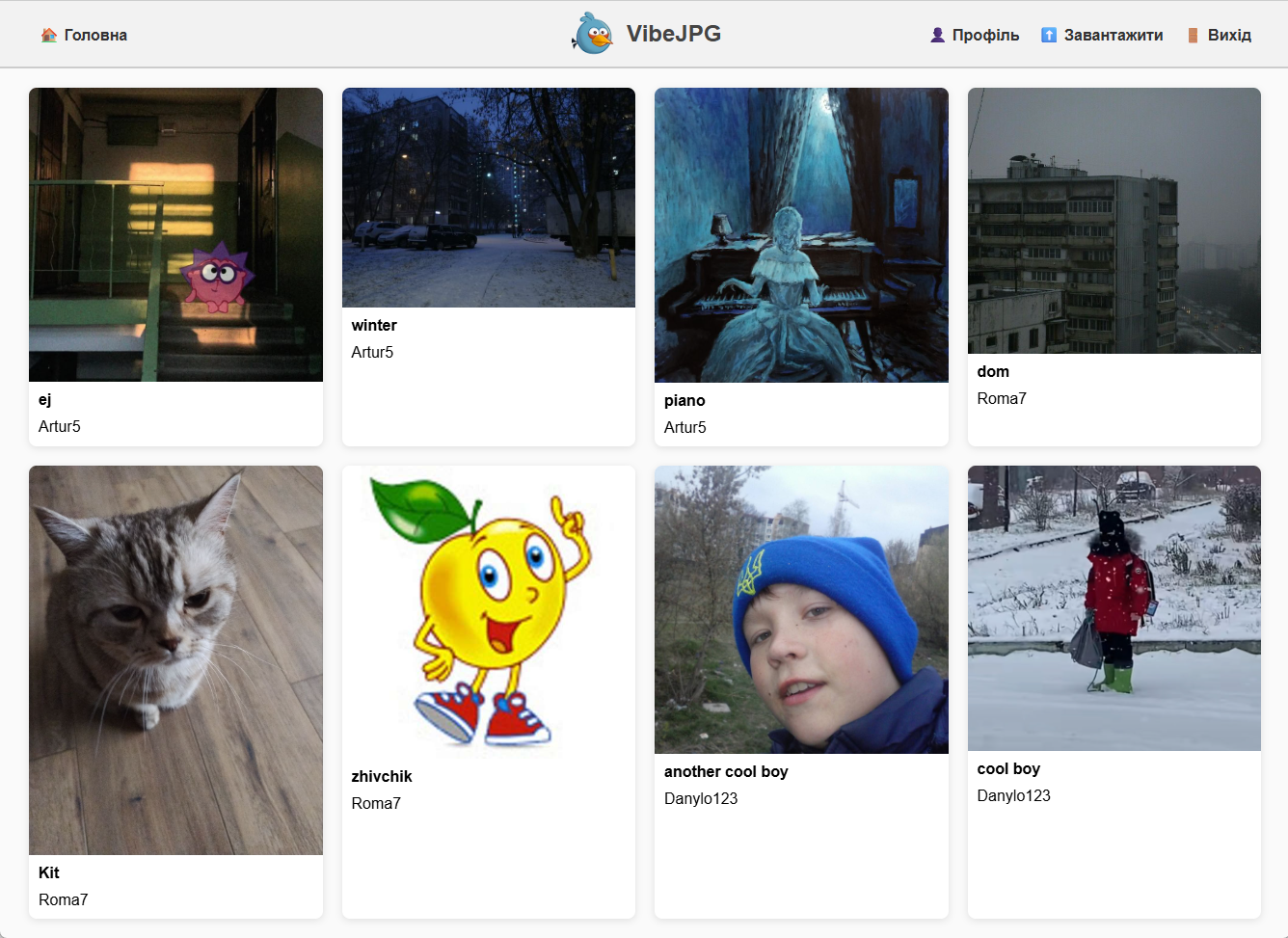
****

Рис 4.1 – Галерея на головній сторінці сайту

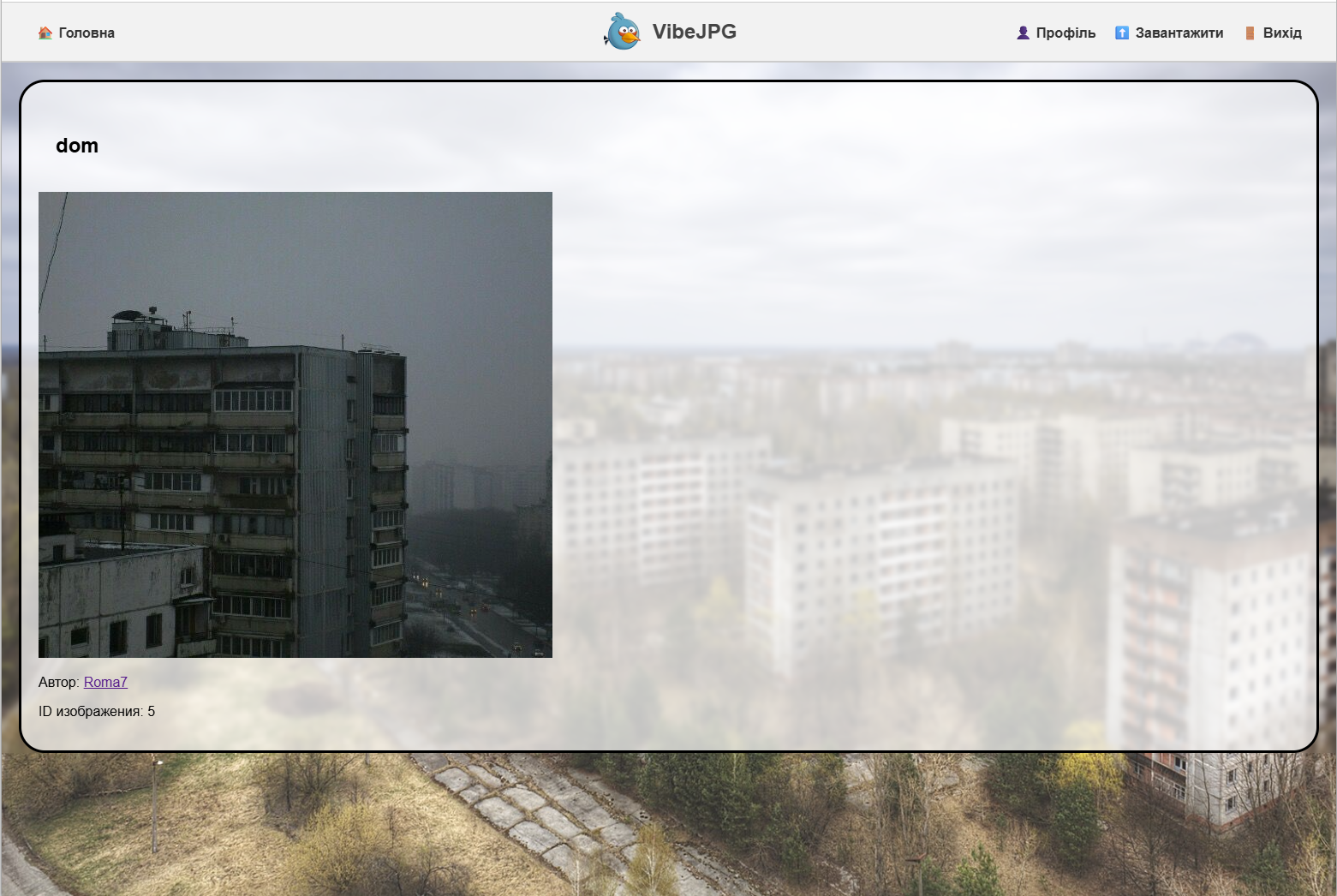


Рис 4.2 – Типовий вигляд окремої сторінки для картинки

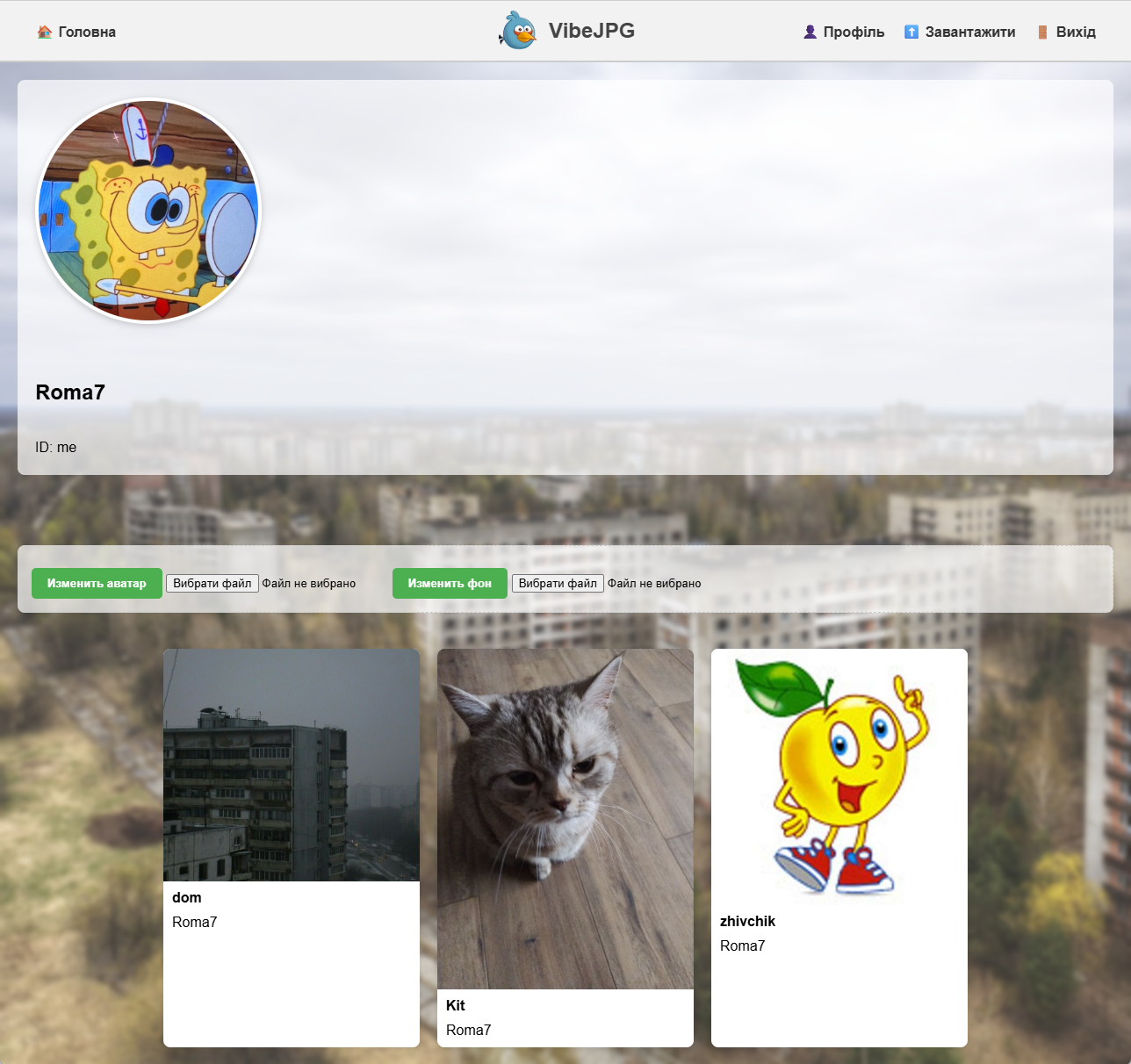


Рис 4.3 – Перегляд власної сторінки

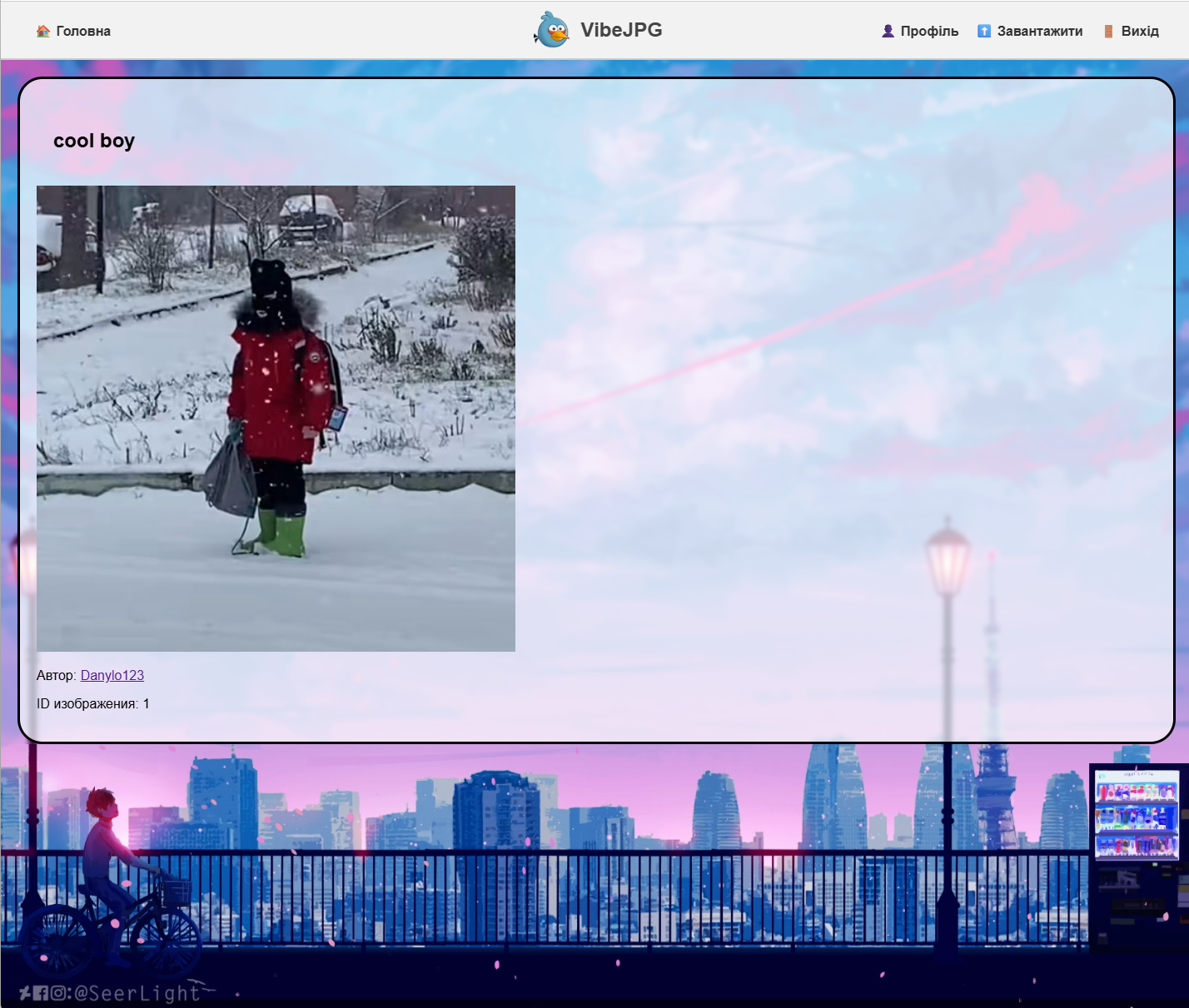


Рис 4.4 – Перегляд чужої картинки, вже з іншим фоном.

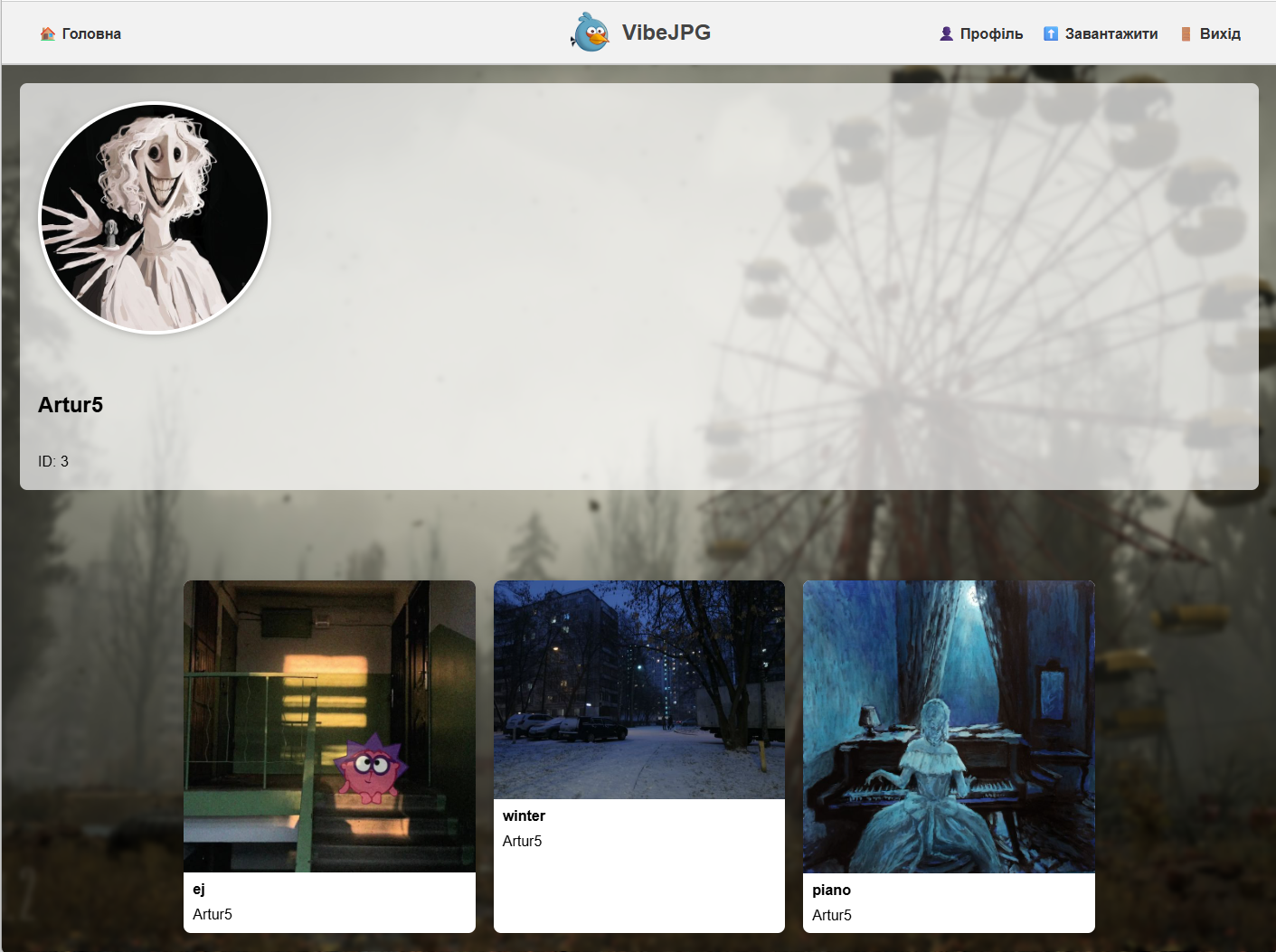


Рис 4.5 – Перегляд профіля іншого користувача



Рис 4.6 – Панель навігації, коли не виконано авторизацію

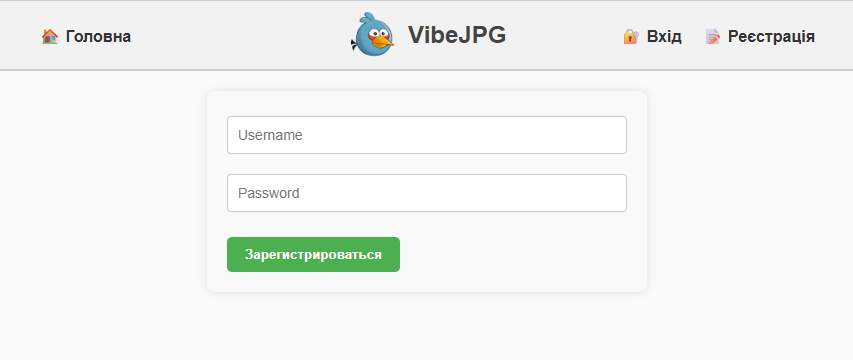


Рис 4.7 - Сторінка реєстрації

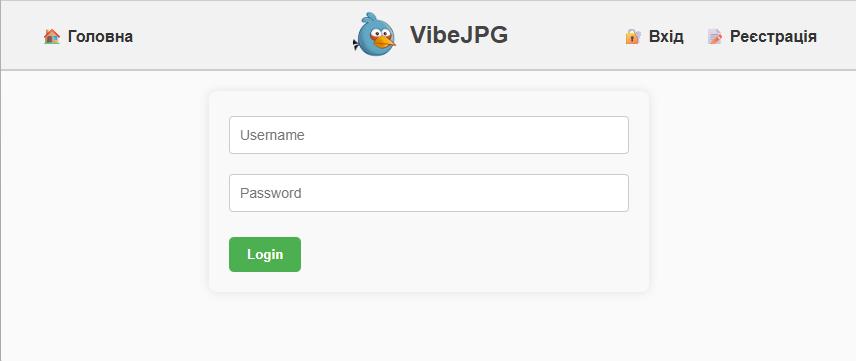


Рис 4.8 – Сторінка входу

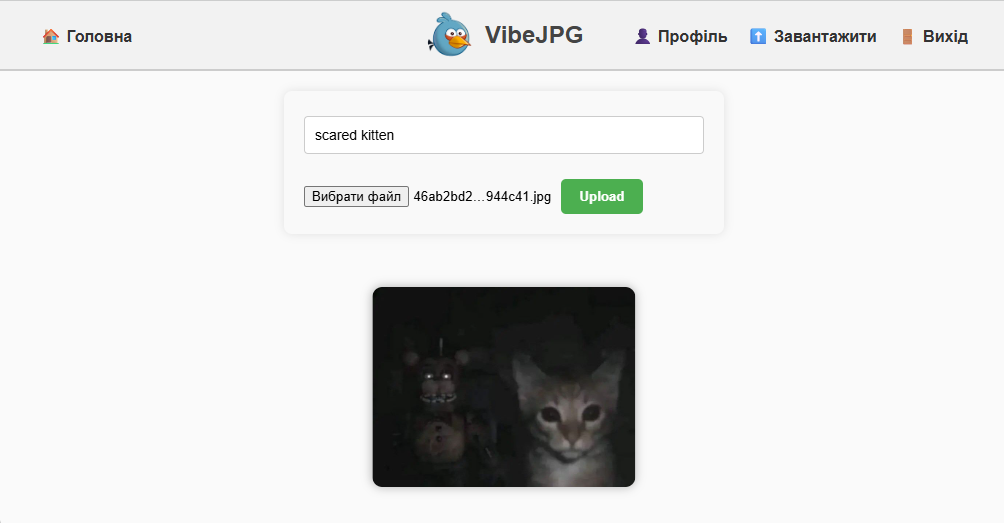


Рис 4.9 – Сторінка завантаження фото

# ВИСНОВКИ

У ході курсової роботи було реалізовано вебплатформу для творчих людей, яка дозволяє користувачам створювати профілі, завантажувати свої зображення, змінювати аватар та фон профілю, а також взаємодіяти з контентом інших користувачів. Основна увага під час розробки приділялась простоті у використанні, безпеці та масштабованості застосунку.

Для реалізації серверної частини було обрано **FastAPI** — сучасний асинхронний фреймворк на Python, що забезпечує високу продуктивність та зручну структуру REST API. Його використання дозволило ефективно реалізувати обробку запитів, авторизацію користувачів за допомогою JWT-токенів, а також обробку завантаження зображень та їх зберігання.

У якості системи керування базами даних було використано **PostgreSQL** — потужну, надійну реляційну СКБД з підтримкою транзакцій, індексів, розширеної типізації та високої продуктивності. Вона дозволила гнучко організувати зберігання інформації про користувачів, зображення та інші пов’язані дані.

Застосунок було контейнеризовано з використанням **Docker**, що забезпечило простоту розгортання, розмежування середовищ, ізоляцію залежностей та стабільну роботу незалежно від платформи. Для збереження зображень користувачів та даних бази після перезапуску або ребілду контейнерів були змонтовані відповідні зовнішні томи, що гарантує збереження важливої інформації.

Фронтенд частина розміщується окремо та може бути запущена навіть за допомогою простого Python-сервера (python3 -m http.server), що дозволяє швидко перевіряти та розгортати клієнтську частину під час розробки або на етапі тестування.

### **Основні досягнення проєкту:**

* **Безпека:** Реалізована авторизація з використанням JWT, захист маршрутів, валідація даних при реєстрації, а також перевірка доступу до користувацьких дій.
* **Функціональність:** Можливість створення облікового запису, зміни аватара та фону профілю, завантаження зображень із заголовком, перегляд профілів інших користувачів.
* **Зручність у використанні:** Мінімалістичний інтерфейс користувача, простота у навігації та інтуїтивно зрозумілий процес взаємодії з сайтом.
* **Масштабованість:** Завдяки використанню FastAPI, PostgreSQL та Docker, застосунок легко масштабувати, підтримувати та переносити між середовищами.

У результаті розробки було створено надійну основу для подальшого розвитку функціоналу платформи, включно з можливістю впровадження коментарів, лайків, повідомлень або системи рейтингів. Проєкт є прикладом сучасного повноцінного вебзастосунку з чітким розділенням серверної та клієнтської частин, побудованого з урахуванням вимог до продуктивності, безпеки та зручності підтримки.

# ВИКОРИСТАНІ ДЖЕРЕЛА

1. **Що таке FastAPI? Офіційна документація.**  
   FastAPI – сучасний вебфреймворк для створення API на Python, відомий своєю швидкодією та простотою.  
   URL: https://fastapi.tiangolo.com/uk/ (дата звернення: 25.05.2024)
2. **Документація PostgreSQL (укр. / англ.)**  
   PostgreSQL – потужна об’єктно-реляційна система керування базами даних з відкритим кодом.  
   URL: <https://www.postgresql.org/docs/> (дата звернення: 25.05.2024)
3. **Docker: Офіційний гід для початківців.**  
   Docker дозволяє створювати, запускати та керувати контейнерами для програм і сервісів.  
   URL: https://docs.docker.com/get-started/ (дата звернення: 25.05.2024)
4. **Що таке REST API? Вступ до REST-архітектури – MDN Web Docs.**  
   Базові принципи REST, які використовуються для взаємодії між фронтендом і бекендом.  
   URL: <https://developer.mozilla.org/en-US/docs/Glossary/REST> (дата звернення: 25.05.2024)
5. **JWT.io – Що таке JSON Web Token (JWT)?**  
   JWT використовується для безпечної передачі інформації між клієнтом і сервером.  
   URL: https://jwt.io/introduction/ (дата звернення: 25.05.2024)
6. **Python HTTP Server – Швидкий запуск локального сервера для фронтенду.**  
   Опис вбудованого HTTP-сервера Python, який можна використовувати для простого хостингу.  
   URL: <https://docs.python.org/3/library/http.server.html> (дата звернення: 25.05.2024)
7. **Підключення та використання PostgreSQL з FastAPI – TUTORIAL (DEV.to)**  
   Практичний посібник зі з'єднання FastAPI і PostgreSQL.  
   URL: https://dev.to/mbuthi/devops-with-fast-api-postgresql-how-to-containerize-fast-api-application-with-docker-1jdb (дата звернення: 25.05.2024)

# ДОДАТКИ

## Додаток А. Фрагменти коду для головної сторінки

**index.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<link rel="stylesheet" href="css/style.css">

<title>Галерея</title>

<link rel="icon" href="/static/logo48.png" type="image/png" />

</head>

<body>

<div id="navbar-container"></div>

<script src="/js/navbar.js" defer></script>

<div id="gallery"></div>

<script src="js/api.js"></script>

<script>

async function showImages() {

const images = await fetchImages();

const gallery = document.getElementById("gallery");

gallery.innerHTML = ""; // Очищаем при каждом вызове

images.forEach(img => {

const div = document.createElement("div");

div.className = "image-card";

div.innerHTML = `

<a href="/image.html?id=${img.id}" style="text-decoration: none; color: inherit;">

<img src="http://localhost:8000/uploads/${img.filename}" alt="${img.title}" />

<h4>${img.title}</h4>

<p>${img.owner?.username || 'Неизвестно'}</p>

`;

gallery.appendChild(div);

});

}

showImages();

</script>

</body>

</html>

**main.py**

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from fastapi.staticfiles import StaticFiles

import os

from app.routes import users, images

UPLOAD\_DIR = "uploads"

os.makedirs(UPLOAD\_DIR, exist\_ok=True)

app = FastAPI()

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

app.include\_router(users.router)

app.include\_router(images.router)

app.mount("/uploads", StaticFiles(directory="uploads"), name="uploads")

from app.database import Base, engine

from app.models import User

# Создаём таблицы, если их нет

Base.metadata.create\_all(bind=engine)

@app.get("/")

def read\_root():

return {"message": "FastAPI работает!"}

**models.py**

from sqlalchemy import Column, Integer, String, ForeignKey, DateTime

from sqlalchemy.orm import relationship

from datetime import datetime

from .database import Base

class User(Base):

\_\_tablename\_\_ = "users"

id = Column(Integer, primary\_key=True, index=True)

username = Column(String, unique=True, index=True)

password = Column(String)

avatar = Column(String, nullable=True)

background = Column(String, nullable=True)

images = relationship("Image", back\_populates="owner")

class Image(Base):

\_\_tablename\_\_ = "images"

id = Column(Integer, primary\_key=True, index=True)

title = Column(String)

filename = Column(String)

upload\_time = Column(DateTime, default=datetime.utcnow)

owner\_id = Column(Integer, ForeignKey("users.id"))

owner = relationship("User", back\_populates="images")

**users.py**

from fastapi import APIRouter, Depends, HTTPException, File, UploadFile

from sqlalchemy.orm import Session

import shutil, os

from app import schemas, models, database, auth

from datetime import datetime

router = APIRouter(prefix="/users", tags=["Users"])

def get\_db():

db = database.SessionLocal()

try:

yield db

finally:

db.close()

UPLOAD\_DIR = "uploads"

os.makedirs(UPLOAD\_DIR, exist\_ok=True)

ALLOWED\_IMAGE\_TYPES = ["image/jpeg", "image/png", "image/gif"]

@router.get("/me", response\_model=schemas.UserOut)

async def get\_current\_user(

current\_user: models.User = Depends(auth.get\_current\_user),

db: Session = Depends(get\_db)

):

return current\_user

# Получение данных любого пользователя

@router.get("/{user\_id}", response\_model=schemas.UserOut)

async def get\_user(

user\_id: int,

db: Session = Depends(get\_db)

):

user = db.query(models.User).filter(models.User.id == user\_id).first()

if not user:

raise HTTPException(status\_code=404, detail="User not found")

return user

@router.post("/register")

def register(user: schemas.UserCreate, db: Session = Depends(get\_db)):

db\_user = db.query(models.User).filter\_by(username=user.username).first()

if db\_user:

raise HTTPException(status\_code=400, detail="User already exists")

new\_user = models.User(

username=user.username,

password=auth.hash\_password(user.password),

background="defaultback",

avatar="defaultav.png"

)

db.add(new\_user)

db.commit()

db.refresh(new\_user)

return {"message": "Registered"}

@router.post("/login", response\_model=schemas.Token)

def login(user: schemas.UserLogin, db: Session = Depends(get\_db)):

db\_user = db.query(models.User).filter\_by(username=user.username).first()

if not db\_user or not auth.verify\_password(user.password, db\_user.password):

raise HTTPException(status\_code=401, detail="Invalid credentials")

token = auth.create\_token({"sub": db\_user.username})

return {"access\_token": token}

@router.post("/me/avatar")

async def upload\_avatar(

file: UploadFile = File(...),

current\_user: models.User = Depends(get\_current\_user),

db: Session = Depends(get\_db)

):

try:

# 1. Проверка типа файла

if file.content\_type not in ALLOWED\_IMAGE\_TYPES:

raise HTTPException(

status\_code=400,

detail=f"Allowed types: {', '.join(ALLOWED\_IMAGE\_TYPES)}"

)

# 2. Подготовка директории

os.makedirs(UPLOAD\_DIR, exist\_ok=True)

# 3. Генерация имени файла

ext = file.filename.split('.')[-1].lower()

filename = f"avatar\_{current\_user.id}\_{int(datetime.utcnow().timestamp())}.{ext}"

filepath = os.path.abspath(os.path.join(UPLOAD\_DIR, filename))

# 4. Проверка безопасности пути

if not filepath.startswith(os.path.abspath(UPLOAD\_DIR)):

raise HTTPException(

status\_code=400,

detail="Invalid file path"

)

# 5. Сохранение файла

with open(filepath, "wb") as buffer:

shutil.copyfileobj(file.file, buffer)

# 6. Удаление старого аватара

old\_avatar = None

if current\_user.avatar:

old\_avatar = os.path.join(UPLOAD\_DIR, current\_user.avatar)

if os.path.exists(old\_avatar):

try:

os.remove(old\_avatar)

except Exception as e:

print(f"Warning: Could not delete old avatar: {str(e)}")

try:

user = db.merge(current\_user)

user.avatar = filename

db.commit()

db.refresh(user)

except Exception as e:

# Откатываем изменения в БД если ошибка

db.rollback()

# Удаляем только что сохранённый файл

if os.path.exists(filepath):

os.remove(filepath)

raise HTTPException(

status\_code=500,

detail=f"Database error: {str(e)}"

)

return {

"message": "Avatar uploaded successfully",

"filename": filename,

"avatar\_url": f"/{UPLOAD\_DIR}/{filename}"

}

except HTTPException:

raise # Пробрасываем уже обработанные ошибки

except Exception as e:

raise HTTPException(

status\_code=500,

detail=f"Internal server error: {str(e)}"

)

@router.post("/me/background")

async def upload\_background(

file: UploadFile = File(...),

current\_user: models.User = Depends(get\_current\_user),

db: Session = Depends(get\_db)

):

# Проверка типа файла

if file.content\_type not in ALLOWED\_IMAGE\_TYPES:

raise HTTPException(status\_code=400, detail="Only JPEG, PNG and GIF images are allowed")

# Генерируем уникальное имя файла

ext = file.filename.split('.')[-1]

filename = f"bg\_{current\_user.id}\_{int(datetime.utcnow().timestamp())}.{ext}"

filepath = os.path.join(UPLOAD\_DIR, filename)

try:

# Сохраняем файл

with open(filepath, "wb") as buffer:

shutil.copyfileobj(file.file, buffer)

# Удаляем старый фон, если он существует

if current\_user.background:

old\_filepath = os.path.join(UPLOAD\_DIR, current\_user.background)

if os.path.exists(old\_filepath):

os.remove(old\_filepath)

# Обновляем пользователя в БД

user = db.merge(current\_user)

user.background = filename

db.commit()

db.refresh(user)

return {

"message": "Background uploaded successfully",

"filename": filename,

"background\_url": f"/{UPLOAD\_DIR}/{filename}" # URL для доступа к фону

}

except Exception as e:

raise HTTPException(status\_code=500, detail=f"Could not upload background: {str(e)}")

**images.py**

from fastapi import APIRouter, UploadFile, File, Depends, Form

from sqlalchemy.orm import Session

from .. import models, database, schemas, auth

import shutil, os

from datetime import datetime

router = APIRouter(prefix="/images", tags=["Images"])

UPLOAD\_DIR = "uploads"

os.makedirs(UPLOAD\_DIR, exist\_ok=True)

def get\_db():

db = database.SessionLocal()

try:

yield db

finally:

db.close()

@router.post("/upload", response\_model=schemas.ImageOut)

async def upload\_image(title: str = Form(...), file: UploadFile = File(...), db: Session = Depends(get\_db), current\_user: models.User = Depends(auth.get\_current\_user)):

filename = f"{datetime.utcnow().timestamp()}\_{file.filename}"

filepath = os.path.join(UPLOAD\_DIR, filename) # Убедитесь, что UPLOAD\_DIR здесь тот же самый

print(f"Attempting to save file to: {filepath}") # <--- ДОБАВЬТЕ ДЛЯ ДЕБАГА

try:

with open(filepath, "wb") as buffer:

shutil.copyfileobj(file.file, buffer)

print(f"File {filename} saved successfully to {filepath}") # <--- ДОБАВЬТЕ ДЛЯ ДЕБАГА

except Exception as e:

print(f"Error saving file: {e}") # <--- ДОБАВЬТЕ ДЛЯ ДЕБАГА

# Можно возбудить HTTPException, если сохранение не удалось

raise HTTPException(status\_code=500, detail=f"Could not save file: {e}")

image = models.Image(title=title, filename=filename, owner\_id=current\_user.id)

db.add(image)

db.commit()

db.refresh(image)

return image

@router.get("/latest", response\_model=list[schemas.ImageOut])

def get\_images(db: Session = Depends(get\_db)):

return db.query(models.Image).order\_by(models.Image.upload\_time.desc()).limit(10).all()

@router.get("/latestfr/{user\_id}", response\_model=list[schemas.ImageOut])

def get\_images(user\_id: int, db: Session = Depends(get\_db)):

return db.query(models.Image)\

.filter(models.Image.owner\_id == user\_id)\

.order\_by(models.Image.upload\_time.desc())\

.limit(10)\

.all()

@router.get("/{image\_id}", response\_model=schemas.ImageOut)

def get\_image(image\_id: int, db: Session = Depends(get\_db)):

image = db.query(models.Image).filter(models.Image.id == image\_id).first()

if not image:

raise HTTPException(status\_code=404, detail="Image not found")

return image

**style.css**

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

background-color: #fafafa;

background-size: cover;

background-attachment: fixed;

background-repeat: no-repeat;

background-position: center center;

backdrop-filter: blur(3px);

}

nav {

padding: 10px;

background-color: #f2f2f2;

}

nav a {

margin: 0 10px;

text-decoration: none;

color: #333;

}

h2 {

padding: 20px;

}

#gallery {

display: flex;

flex-wrap: wrap;

justify-content: center;

gap: 20px;

padding: 20px;

}

.image-card {

width: calc(25% - 20px);

min-width: 250px;

box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);

border-radius: 8px;

overflow: hidden;

background-color: #fff;

transition: transform 0.2s;

}

.image-card:hover {

transform: scale(1.02);

}

.image-card img {

width: 100%;

height: auto;

display: block;

}

.image-card h4, .image-card p {

margin: 10px;

}

button,

input[type="submit"] {

background-color: #4CAF50;

color: white;

padding: 10px 18px;

margin-top: 10px;

border: none;

border-radius: 5px;

cursor: pointer;

font-weight: bold;

transition: background-color 0.3s ease;

}

button:hover,

input[type="submit"]:hover {

background-color: #45a049;

}

/\* Текстовые поля \*/

input[type="text"],

input[type="password"],

input[type="email"],

textarea {

width: 100%;

padding: 10px;

margin-top: 5px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

font-size: 14px;

}

/\* Общий стиль форм \*/

form {

max-width: 400px;

margin: 20px auto;

padding: 20px;

background-color: #f9f9f9;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

form h2 {

text-align: center;

}

#image-container {

margin: 20px;

padding: 20px;

border: 3px solid black;

border-radius: 30px;

background-color: rgb(255 255 255 / 70%);;

}

#idtext{

margin-left: 20px;

}

.profile-header {

position: relative;

margin:20px;

height: 450px;

margin-bottom: 80px;

border-radius: 8px;

overflow: hidden;

background-color: rgb(255 255 255 / 70%);;

}

.profile-background {

width: 100%;

height: 100%;

object-fit: cover;

}

.profile-avatar {

margin: 20px;

width: 250px;

height: 250px;

border-radius: 50%;

border: 4px solid white;

object-fit: cover;

background-color: #f0f0f0;

box-shadow: 0 2px 8px rgba(0, 0, 0, 0.2);

}

.profile-info {

margin-top: 60px;

padding: 20px;

}

.upload-section {

margin: 20px;

padding: 15px;

border: 1px dashed #ccc;

border-radius: 8px;

background-color: rgb(255 255 255 / 70%);;

}

.upload-section h3 {

margin-top: 0;

}

@media (max-width: 768px) {

.profile-avatar {

width: 100px;

height: 100px;

bottom: -30px;

}

.profile-header {

height: 200px;

margin-bottom: 50px;

}

}

#preview-container {

position: fixed;

bottom: 20px;

left: 50%;

transform: translateX(-50%);

z-index: 1000;

text-align: center;

}

#preview-container img {

max-width: 300px;

max-height: 200px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.3);

}