МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ
ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ УПРАВЛІННЯ, ПСИХОЛОГІЇ ТА БЕЗПЕКИ

Кафедра інформаційних технологій

МОДЕЛІ ТА ЗАСОБИ ЗБЕРІГАННЯ ЗОБРАЖЕНЬ НА ПЛАТФОРМІ ХМАРНИХ ОБЧИСЛЕНЬ

Кваліфікаційна робота
здобувача вищої освіти
4 курсу заочної форми навчання
Вікторії-Мішель АНЦУПОВОЇ

Науковий керівник:
доцент, кандидат технічних наук
Андрій ГОЛОВАТИЙ

Рецензент:

_____
вчене звання, науковий ступінь

_____
(Ім'я ПРІЗВИЩЕ рецензента)

Кваліфікаційна робота допущена до захисту
«___» _____ 2026 р., протокол № ____

Завідувач кафедри інформаційних технологій
_____ Олег ЗАЧЕК
(підпис)

Львів
2026

АНОТАЦІЯ

АНЦУПОВА В.-М. Моделі та засоби зберігання зображень на платформі хмарних обчислень. – Рукопис.

Дослідження на здобуття освітнього ступеня «бакалавр» за спеціальністю 126 «Інформаційні системи та технології». – Львівський державний університет внутрішніх справ, МВС України, Львів, 2026.

У роботі досліджено та розроблено сучасний веб-застосунок для зберігання, обробки та поширення медіа-даних (фотографій та відео) у хмарному середовищі з використанням прогресивних технологій ASP.NET Core та Angular. Актуальність теми зумовлена стрімким зростанням обсягів медіа-контенту, необхідністю забезпечення надійного, безпечного та кросплатформенного доступу до даних незалежно від пристрою користувача, а також обмеженнями традиційних локальних систем зберігання

Ключові слова: хмарне сховище, веб-сервіс, архітектура, бази даних.

ABSTRACT

OLENYUK F.-A. Name Models and means of storing images on the cloud computing platform. – Manuscript.

Research for obtaining a bachelor’s degree in specialty 126 «Information systems and technologies». – Lviv State University of Internal Affairs, MIA of Ukraine, Lviv, 2026.

The paper researches and develops a modern web application for storing, processing and distributing media data (photos and videos) in a cloud environment using progressive technologies ASP.NET Core and Angular. The relevance of the topic is due to the rapid growth of media content, the need to ensure reliable, secure and cross-platform access to data regardless of the user's device, as well as the limitations of traditional local storage systems.

Keywords: cloud storage, web service, architecture, databases.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API (API)	Application Programming Interface (Інтерфейс програмування застосунків)
ASP.NET	Active Server Pages .NET
ASP.NET Core	Active Server Pages .NET Core
Web API	Web Application Programming Interface (Веб-інтерфейс програмування застосунків)
MVC	Model-View-Controller (Модель-Представлення-Контролер)
SPA	Single Page Application (Односторінковий застосунок)
REST	Representational State Transfer (Передача репрезентативного стану)
HTTP	HyperText Transfer Protocol (Протокол передачі гіпертексту)
JSON	JavaScript Object Notation (Нотація об’єктів JavaScript)
CRUD	Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення)
ORM	Object-Relational Mapping (Об’єктно-реляційне відображення)
SQL	Structured Query Language (Структурована мова запитів)
UI	User Interface (Користувацький інтерфейс)
HTML	HyperText Markup Language (Мова розмітки гіпертексту)
URL	Uniform Resource Locator (Уніфікований локатор ресурсу)
.NET	.NET Platform
IDE	Integrated Development Environment (Інтегроване середовище розробки)
XML	eXtensible Markup Language (Розширювана мова розмітки)
UML	Unified Modeling Language (Уніфікована мова моделювання)

ВСТУП

В умовах швидкого розвитку інформаційних технологій та зростання обсягів медіа-контенту важливим завданням є забезпечення зручного, надійного та безпечного зберігання та доступу до фотографій і відео з будь-якого пристрою. Традиційні локальні накопичувачі часто стають причиною втрати даних через зношення, обмежений об’єм або відсутність синхронізації між пристроями. У зв’язку з цим актуальним є створення сучасних веб-застосунків, що використовують хмарні технології зберігання.

У роботі програмних систем, пов’язаних із медіа-даними, існує комплекс взаємопов’язаних компонентів: серверна частина, клієнтський інтерфейс, база даних, система автентифікації та хмарне сховище. Відсутність чіткої архітектури, застарілі технології або неправильна інтеграція цих компонентів можуть призвести до низької продуктивності, проблем з безпекою та погіршення користувацького досвіду.

Мета роботи полягає в дослідженні та розробці веб-застосунку для збереження та доступу до медіа-даних у хмарному сховищі з використанням сучасних технологій ASP.NET Core та Angular.

Для досягнення поставленої мети необхідно виконати такі **завдання:**

- Провести аналіз сучасних хмарних технологій та існуючих рішень;
- Виконати системний аналіз предметної області та сформувати вимоги до системи;
- Розробити концептуальну модель бази даних та архітектуру програмного рішення;
- Реалізувати серверну частину на базі ASP.NET Core Web API;
- Розробити клієнтську частину на платформі Angular з інтуїтивним інтерфейсом;
- Провести тестування та розгортання системи.

Об’єктом дослідження є процеси збереження, обробки та доступу до медіа-даних у хмарному середовищі.

Предметом дослідження виступають методи, алгоритми та програмні засоби розробки веб-застосунків із використанням хмарних технологій.

У роботі використано комплекс методів, що поєднує теоретичні та практичні підходи. Теоретичну основу склали методи системного аналізу, структурно-функціонального моделювання та огляд науково-технічної літератури. Практична частина базується на об’єктно-орієнтованому дизайні (SOLID), багатошаровій архітектурі (N-Tier), а також методах тестування та розгортання в хмарному середовищі.

Практична значущість роботи полягає в створенні прототипу веб-сервісу, який дозволяє користувачам зручно зберігати, ділитися та отримувати доступ до своїх медіа-даних з будь-якого пристрою. Розроблене рішення може бути використане як приватними користувачами, так і в навчальних закладах для поширення навчальних матеріалів.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг основного тексту роботи становить 45 сторінок, містить 9 рисунків, 1 таблиця та 20 бібліографічних джерел.

РОЗДІЛ 1
АНАЛІЗ ОБ'ЄКТУ ДОСЛІДЖЕННЯ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

Хмарне сховище (англ. Cloud) це модель сховища даних, де дані зберігаються в логічних пулах, а фізичне зберігання може охоплювати навіть декілька серверів, що у свою чергу називається масштабуванням (англ. Scalability). Фізичне середовище розташування застосунку належить хостинговим компаніям, які у свою чергу керують ним. Хостинги хмарних систем відповідають за збереження наявної інформації та надають доступ до неї.

Користувачі купують у постачальників послуг хмарного сховища змогу зберігати там дані. Ця модель забезпечує зручний доступ на вимогу через мережу до спільного пулу обчислювальних ресурсів, що підлягають налаштуванню, і які можуть бути оперативно надані та звільнені з мінімальними управлінськими затратами та зверненнями до провайдера.

При використанні хмарних обчислень програмне забезпечення надається користувачеві як Інтернет-сервіс. Користувач має доступ до власних даних, але не може керувати і не повинен піклуватися про інфраструктуру, операційну систему і програмне забезпечення, з яким він працює. «Хмарою» метафорично називають інтернет, який приховує всі технічні деталі. Згідно з документом IEEE, опублікованим у 2008 році, «Хмарні обчислення – це парадигма, в рамках якої інформація постійно зберігається на серверах у мережі інтернет і тимчасово кешується на клієнтській стороні, наприклад на персональних комп'ютерах, ігрових приставках, ноутбуках, смартфонах тощо».

На сьогоднішній день сфера інформаційних технологій розвивається дедалі більше. Постійно появляються нові системи розробки програмного забезпечення. За останні роки особливої популярності набули веб-

орієнтовані рішення, які дають змогу перенести усі дані з комп'ютера у «хмару» та відділити самим роботу з даними від конкретного пристрою.

Настільні застосунки дедалі більше втрачають свою актуальність через те, що вони, як правило, прив'язані до тої чи іншої платформи та версії операційної системи чи допоміжного програмного забезпечення для їх встановлення та запуску. Багато таких рішень переходить у хмарні сховища. Наприклад, продукт від Microsoft OneNote, колись мав властивість бути лише настільним додатком для серії Microsoft Windows, а віднедавна являє собою суміш настільного застосунку та хмарного додатку, які синхронізуються між собою, та надає можливість отримувати доступ до своїх нотаток з будь-якого пристрою незалежно від статусу встановленого додаткового програмного забезпечення.

Разом із іншими веб-застосунками та веб-ресурсами появляється дедалі більше хмарних сервісів, де користувач має можливість відвантажувати свої медіа дані та ділитися ними з друзями, рідними та знайомими. Існує багато різних рішень як і вузького, так і більш широкого призначення. Так наприклад, Instagram є найбільшим у світі лідером серед обміну фотографій та відео між користувачами. Він також користується не аби якою популярністю серед українських користувачів. Існує необхідність розробки програмного забезпечення, яке б було актуальним у нашій країні та вивело українську індустрію інформаційних технологій на новий рівень.

Даний застосунок також можна адаптувати для навчальних закладів, для створення презентацій з красивим графічним інтерфейсом, поширення інформації викладачів для усіх студентів тощо.

На сьогоднішній день є багато технологій, які дозволяють це все реалізувати. Однією з найпопулярніших технологією створення веб-сервісів є ASP.NET, що є складовою частиною платформи Microsoft .NET, що грунтується на CRL (Common Language Runtime), та надає можливість розробляти високопродуктивні веб-рішення.

1.1. Аналіз існуючих хмарних сховищ

Існує багато різних хмарних сервісів для збереження зображень. Усі системи мають як і переваги, так і недоліки. Сервіси відомих світових брендів, таких як Microsoft, Apple, Google, не виключення. Наприклад, розмір безкоштовного сховища, що надає своїм користувачам Apple, становить лише 5 гігабайт та синхронізація відбувається для усіх файлів системи, а не для обраних, так як хоче цього користувач. У OneDrive (у минулому SkyDrive), також пам'ять з недавніх пір була обмежена 5 гігабайтами дискового простору. На перший погляд уся справа у обсязі пам'яті, що надає постачальник. Іншим недоліком та найбільших недоліком цих систем є те, що вони не достатньо зрозумілі усім користувачам пристроїв. Інтерфейс для таких застосунків повинен бути максимально простим та зрозумілим.

Ринок хмарних сховищ завжди відкритий для свіжих ідей. Кожен розробник має можливість принести щось нове та унікальне або й розширити функціональність існуючих. На сьогодні існує чимало різних технологій та мов програмування, за допомогою яких цього можна досягнути.

1.2. Технологія ASP.NET Core

ASP.NET Core – це сучасна, відкрита та кросплатформенна технологія для розробки веб-застосунків і веб-сервісів, створена компанією Microsoft. ASP.NET Core є самостійним фреймворком і становить ключову частину єдиної платформи .NET (версії 5.0 і вище).

Розробка ASP.NET Core була розпочата Microsoft у 2015 – 2016 роках як повна перебудова попередніх веб-технологій. Метою створення нової платформи було подолання фундаментальних обмежень класичного ASP.NET (.NET Framework), таких як прив’язаність до операційної системи Windows, монолітність, висока ресурсозатратність та складність розгортання в хмарному середовищі. Нова платформа була розроблена з нуля з урахуванням сучасних вимог до веб-розробки, хмарних обчислень та мікросервісної архітектури.

ASP.NET Core базується на .NET Runtime (раніше відомому як .NET Core) та використовує модульну архітектуру. ASP.NET Core радикально відрізняється як за внутрішньою будовою, так і за принципами роботи. Розробники можуть створювати застосунки, використовуючи переважно мову **C#**, а також F# (VB.NET підтримується, але використовується значно рідше).

Основні переваги ASP.NET Core:

- Кросплатформенність: працює на Windows, Linux та macOS;
- Висока продуктивність і низьке споживання ресурсів завдяки легкому HTTP-серверу Kestrel;
- Модульна архітектура: розробник підключає лише необхідні компоненти (middleware), що зменшує розмір застосунку та підвищує безпеку;
- Вбудована підтримка Dependency Injection на рівні фреймворку;
- Сучасні підходи до розробки: Minimal APIs, Razor Pages, MVC, Blazor, gRPC;
- Покращена підтримка контейнеризації (Docker) та оркестрації (Kubernetes);
- Вбудовані інструменти для моніторингу, логування, аутентифікації, OpenAPI (Swagger) та захисту від поширених веб-атак;
- Значно швидша компіляція та виконання завдяки сучасному runtime;
- Активна розробка та довгострокова підтримка (LTS-версії).

Актуальною є версія ASP.NET Core 10.0, яка входить до складу .NET 10.

ASP.NET Core дозволяє ефективно реалізовувати як традиційні веб-додатки, так і сучасні API-сервіси для роботи з різноманітними клієнтами (веб, мобільні застосунки, IoT тощо). Завдяки своїй гнучкості, продуктивності та кросплатформенності дана технологія є одним з найбільш поширених і перспективних рішень для розробки веб-систем у сучасній індустрії.

1.3. ASP.NET Core Web API

ASP.NET Web API – це технологія, розроблена компанією Microsoft для створення HTTP-сервісів (веб-API), які можуть обслуговувати широкий спектр клієнтів, включаючи веб-браузери, мобільні застосунки та сторонні сервіси. У сучасній версії ASP.NET Core Web API є одним з основних інструментів для побудови RESTful-сервісів і є невід'ємною частиною платформи.

У ASP.NET Core технологія була повністю переосмислена та інтегрована в єдину модульну архітектуру. Вона дозволяє розробникам створювати легкі, високопродуктивні та кросплатформенні веб-сервіси, які можуть працювати як самостійно, так і в складі більших веб-застосунків.

Сучасна реалізація Web API в ASP.NET Core підтримує два основних підходи до створення ендпоінтів:

- Minimal APIs – спрощений, високопродуктивний стиль створення HTTP-ендпоінтів без використання контролерів (рекомендується для простих сервісів).
- Controller-based Web API – класичний підхід на основі контролерів, зручний для складних бізнес-логік, автоматичної валідації та генерації документації.

Архітектура ASP.NET Core Web API базується на потужній системі middleware та endpoint routing, яка забезпечує гнучке керування маршрутами запитів. У ASP.NET Core використовується сучасна система Endpoint Routing, яка є більш продуктивною та гнучкою.

Добре спроєктоване Web API повинно відповідати таким ключовим принципам:

- Незалежність від платформи клієнта – API має бути доступним для будь-яких клієнтів (веб, iOS, Android, десктоп тощо) через стандартні HTTP-протоколи.

- Еволюційна гнучкість – можливість змінювати та розширювати функціональність сервісу без порушення роботи існуючих клієнтів (backward compatibility).
- Чисті та зрозумілі URL (RESTful routing) – використання зрозумілих, ресурс-орієнтованих адрес, що покращує читабельність, SEO та зручність використання API.
- Документованість – автоматична генерація документації за допомогою OpenAPI/Swagger.

Сучасні URL-адреси в стилі REST є зрозумілими та користувацько-орієнтованими порівняно. Вони не розкривають внутрішню структуру застосунку, що підвищує безпеку та дозволяє змінювати реалізацію без впливу на зовнішні посилання.

Приклад сучасного підходу до маршрутизації в ASP.NET Core:

/api/posts – отримання списку публікацій (GET)

/api/posts/{id} – робота з конкретною публікацією

/api/users/{username}/followers – отримання підписників користувача

ASP.NET Core Web API тісно інтегрується з такими технологіями, як Entity Framework Core, ASP.NET Core Identity, JWT Bearer Authentication, а також підтримує автоматичну валідацію моделей, версіонування API, rate limiting та кешування відповідей.

Станом на 2026 рік ASP.NET Core Web API залишається одним з найбільш популярних і ефективних рішень для створення масштабованих веб-сервісів у .NET-екосистемі.

1.4. Концепції та принципи об'єктно-орієнтованого дизайну

SOLID – це акронім, утворений з перших літер п’яти фундаментальних принципів об’єктно-орієнтованого дизайну, запропонованих відомим американським інженером програмного забезпечення Робертом Сесілом Мартіном (відомим також як «Uncle Bob»).

Принципи SOLID є одним із наріжних каменів сучасного об’єктно-орієнтованого програмування. Їх дотримання дозволяє створювати програмні системи, які є зрозумілими, гнучкими, легко розширюваними та стійкими до змін вимог протягом тривалого часу. Застосування цих принципів суттєво знижує технічний борг і спрощує підтримку та розвиток програмного забезпечення.

Розшифрування акроніму SOLID наведено в таблиці 1.1.

Таблиця 1.1

Принципи об’єктно-орієнтованого дизайну SOLID

Принцип	Повна назва	Опис
S	Single Responsibility Principle (Принцип єдиної відповідальності)	Кожен клас повинен мати лише одну причину для зміни.
O	Open/Closed Principle (Принцип відкритості/закритості)	Програмні сутності повинні бути відкритими для розширення, але закритими для модифікації.
L	Liskov Substitution Principle (Принцип підстановки Лісков)	Об’єкти похідних класів повинні бути повністю взаємозамінними з об’єктами базового класу.
I	Interface Segregation Principle (Принцип розділення інтерфейсів)	Клієнти не повинні залежати від інтерфейсів, які вони не використовують.
D	Dependency Inversion Principle (Принцип інверсії залежностей)	Модулі високого рівня не повинні залежати від модулів низького рівня. Обидва повинні залежати від абстракцій.

Коротка характеристика принципів SOLID:

1. Принцип єдиної відповідальності (Single Responsibility Principle) Клас повинен виконувати лише одну обов’язкову функцію. Якщо клас виконує декілька завдань (наприклад, одночасно працює з базою даних, відправляє email і генерує звіти), його важко підтримувати та тестувати. Приклад: Замість одного великого класу UserService, що відповідає за реєстрацію, автентифікацію та відправку сповіщень, краще створити окремі класи: UserRegistrationService, AuthService та NotificationService.

2. Принцип відкритості/закритості (Open/Closed Principle) Класи повинні бути спроектовані таким чином, щоб їх поведінку можна було

розширювати без зміни існуючого коду (наприклад, через успадкування або використання стратегій). Приклад: Замість додавання умов if для кожного нового типу файлу в методі завантаження, краще використовувати стратегію (IFileUploader) або плагіну архітектуру.

3. Принцип підстановки Лісков (Liskov Substitution Principle) Об'єкти дочірнього класу повинні бути здатні повністю замінити об'єкти батьківського класу без порушення коректності програми. Приклад: Якщо клас Rectangle має методи SetWidth() та SetHeight(), то похідний клас Square не повинен порушувати поведінку базового класу, інакше це призведе до помилок у коді, який очікує поведінки прямокутника.

4. Принцип розділення інтерфейсів (Interface Segregation Principle) Краще мати багато спеціалізованих інтерфейсів, ніж один великий універсальний. Приклад: Замість інтерфейсу IUser з методами Save(), SendEmail(), GenerateReport(), краще створити IUserRepository, INotificationService та IReportGenerator.

5. Принцип інверсії залежностей (Dependency Inversion Principle) Високорівневі модулі не повинні залежати від низькорівневих. Обидва рівні повинні залежати від абстракцій (інтерфейсів). Приклад: Клас PostService не повинен безпосередньо створювати екземпляр SqlPostRepository. Замість цього він повинен отримувати IPostRepository через конструктор (Dependency Injection).

Принципи SOLID тісно взаємопов'язані з базовими концепціями об'єктно-орієнтованого програмування. Зокрема, принцип підстановки Лісков є формальним уточненням механізму наслідування, принцип інверсії залежностей розширює ідею поліморфізму, а принцип єдиної відповідальності тісно пов'язаний з інкапсуляцією та приховуванням інформації.

Дотримання принципів SOLID сприяє створенню слабкозв'язаних компонентів, що особливо важливо при розробці складних систем, таких як веб-застосунки та хмарні Сервіси.

1.5. Мережеві протоколи

REST (Representational State Transfer – «передача репрезентативного стану») – це архітектурний стиль (підхід) проектування мережевих протоколів і розподілених систем, який забезпечує ефективний доступ до інформаційних ресурсів. Концепцію REST було запропоновано Роєм Філдінгом у його дисертації 2000 року. Цей стиль базується на наборі принципів, які роблять систему масштабованною, надійною та зручною для еволюції.

Відповідно до REST, дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Мережевий протокол (зокрема HTTP) повинен підтримувати механізми кешування, не залежати від конкретного мережевого прошарку, а також не зберігати інформацію про стан між парами «запит–відповідь» (stateless). Такий підхід, на думку автора, забезпечує високий рівень масштабування системи та дозволяє їй ефективно розвиватися відповідно до нових вимог і умов експлуатації.

Важливо зазначити, що REST фактично не залежить від будь-якого конкретного базового протоколу і не обов’язково пов’язаний з HTTP. Однак найбільш поширені реалізації систем, побудованих на принципах REST, використовують саме протокол HTTP як основний транспортний протокол для відправки та отримання запитів. Цей підрозділ зосереджується на відображенні принципів REST саме для систем, що працюють з використанням HTTP.

HTTP (HyperText Transfer Protocol – протокол передачі гіпертексту) – це один з основних протоколів передачі даних, який широко використовується в комп’ютерних мережах. Назва протоколу походить від англійських слів HyperText Transfer Protocol. HTTP призначений передусім для передачі веб-сторінок – текстових файлів з розміткою HTML, хоча за його допомогою успішно передаються й інші типи файлів, пов’язані з веб-

сторінками (зображення, відео, скрипти, стилі), а також файли, не пов’язані безпосередньо з веб (у цьому аспекті HTTP успішно конкурує зі складнішим протоколом FTP).

HTTP належить до протоколів 7-го (прикладного) рівня моделі OSI. Основним призначенням протоколу є забезпечення зручної взаємодії між клієнтською програмою (найчастіше веб-браузером) та сервером. Протокол дозволяє клієнтській програмі відображати гіпертекстові веб-сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення контенту HTTP надає можливість клієнту дізнаватися мову та кодування веб-сторінки, а також запитувати потрібну версію документа з використанням позначень стандарту MIME (Multipurpose Internet Mail Extensions).

HTTP є протоколом прикладного рівня. Серед подібних до нього протоколів можна назвати FTP (File Transfer Protocol) та SMTP (Simple Mail Transfer Protocol). Обмін повідомленнями в HTTP відбувається за класичною схемою «запит–відповідь». Для ідентифікації ресурсів протокол використовує глобальні уніфіковані ідентифікатори ресурсів (URI).

На відміну від багатьох інших протоколів, HTTP не зберігає свого стану (stateless). Це означає, що протокол не зберігає проміжного стану між окремими парами «запит–відповідь». Кожен запит обробляється сервером незалежно від попередніх. Компоненти системи, що використовують HTTP, можуть самостійно зберігати інформацію про стан, пов’язаний з попередніми запитами та відповідями (наприклад, за допомогою cookies, JWT-токенів або сесій на сервері). Браузер, який надсилає запити, може відстежувати затримки відповідей, а сервер – зберігати IP-адреси та заголовки запитів останніх клієнтів. Однак, згідно з протоколом, ні клієнт, ні сервер не зобов’язані зберігати інформацію про попередні взаємодії. У самому протоколі не передбачено внутрішньої підтримки стану, і він не висуває таких вимог до учасників обміну.

Кожен HTTP-запит або відповідь складається з трьох основних частин:

- стартовий рядок (request line для запиту або status line для відповіді);
- заголовки (headers);
- тіло повідомлення (message body), яке може містити дані запиту, запитаний ресурс або опис проблеми у випадку помилки.

Обов’язковим мінімальним елементом запиту є стартовий рядок. Починаючи з версії HTTP/1.1, обов’язковим став заголовок Host, який дозволяє розрізняти кілька доменів, що мають одну й ту саму IP-адресу.

Під час розробки RESTful-сервісів найпопулярнішими є такі стандартні методи виконання HTTP-запитів:

GET – запитує вміст вказаного ресурсу;

POST – передає дані, призначені для користувача (наприклад, з HTML-форм), заданому ресурсу для створення нового ресурсу або виконання певної дії;

PUT – завантажує вказаний ресурс на сервер або повністю оновлює існуючий ресурс;

DELETE – видаляє вказаний ресурс;

PATCH – виконує часткове оновлення ресурсу.

Ці методи є ключовими для реалізації CRUD-операцій (Create, Read, Update, Delete) у сучасних веб-сервісах.

JSON (JavaScript Object Notation – нотація об’єктів JavaScript) – це текстовий, легкий для читання людиною формат обміну даними між комп’ютерами. JSON базується на текстовому представленні та може бути легко прочитаний і згенерований як людиною, так і машиною. Формат дозволяє описувати об’єкти та інші структури даних. Головним чином JSON використовується для передачі структурованої інформації через мережу завдяки процесу, який називається серіалізацією.

JSON будується на двох базових структурах даних:

- Набір пар «ім’я/значення». У різних мовах програмування це реалізовано як об’єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативний масив.

- Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність.

Ці дві структури є універсальними і підтримуються практично всіма сучасними мовами програмування. Оскільки JSON використовується для обміну даними між різними мовами програмування та платформами, його доцільно будувати саме на цих базових структурах.

У JSON використовуються такі основні форми представлення даних:

Об'єкт – це невпорядкована множина пар «ім'я: значення». Об'єкт починається символом { і закінчується символом }. Кожне значення слідує після двокрапки:, а пари «ім'я/значення» відділяються комами.

Масив – це впорядкована множина значень. Масив починається символом [і закінчується символом]. Значення в масиві відділяються комами. Значення може бути рядком у подвійних лапках, числом, логічними значеннями true або false, значенням null, об'єктом або масивом. Ці структури можуть бути вкладені одна в одну на будь-яку глибину.

Рядок – це впорядкована множина з нуля або більше символів Unicode, обмежена подвійними лапками. Для представлення спеціальних символів використовуються escape-послідовності, що починаються зі зворотної косої риски (\).

Завдяки своїй простоті, компактності та широкій підтримці JSON став одним з основних форматів даних у сучасних RESTful веб-сервісах, повністю витіснивши в багатьох випадках старий формат XML.

1.6. Angular

Angular – це сучасний відкритий фронтенд-фреймворк, написаний на мові TypeScript, який розробляється під керівництвом команди Angular Team компанії Google за активної участі спільноти приватних розробників і провідних корпорацій. Angular є потужним інструментом для створення складних клієнтських веб-застосунків.

Сучасний Angular не є простим продовженням попередньої версії. Він представляє собою повну переробку та еволюцію бібліотеки **Angular** (версія 1.x), яка була повністю переписана командою розробників Google. Якщо Angular був побудований на JavaScript і використовував підхід MVC, то Angular (починаючи з версії 2.0, випущеної у 2016 році) був розроблений з нуля з урахуванням усіх накопичених недоліків попередника та сучасних вимог до веб-розробки.

Angular призначений насамперед для розробки Single Page Applications (SPA) – односторінкових веб-застосунків, які складаються з однієї HTML-сторінки, що динамічно оновлюється за допомогою CSS і JavaScript (TypeScript) без необхідності повного перезавантаження сторінки при переході між розділами. Фреймворк розширює можливості традиційних веб-застосунків, побудованих на шаблоні Model-View-Controller (MVC), а також значно спрощує процес тестування та розробки складних інтерфейсів.

Платформа працює безпосередньо зі сторінкою HTML, збагачуючи її додатковими атрибутами та директивами. Angular встановлює двосторонній зв’язок між областями введення даних (input) та областями виведення (output) на сторінці та моделлю даних, яка представлена звичайними змінними у TypeScript. Значення цих змінних можуть задаватися як вручну розробником, так і динамічно отримуватися зі статичних джерел або через асинхронні запити до сервера у форматі JSON.

Angular спроектований з урахуванням філософії, згідно з якою декларативне програмування найкраще підходить для побудови інтерфейсів користувача та опису компонентів, тоді як імперативне програмування більш ефективне для реалізації бізнес-логіки. Платформа суттєво адаптує та розширює стандартний HTML, вводячи власні директиви, компоненти та синтаксис шаблонів. Це дозволяє забезпечити двосторонню прив’язку даних (Two-Way Data Binding) для динамічного контенту, завдяки чому модель і представлення (view) автоматично синхронізуються між собою. У результаті Angular значно зменшує необхідність у ручних маніпуляціях з DOM

(Document Object Model), що призводить до підвищення продуктивності застосунку та суттєвого полегшення процесу тестування.

HTML як мова розмітки відмінно підходить для опису статичних документів, проте виникають значні труднощі при спробі описати динамічні об’єкти та складну взаємодію у сучасних веб-додатках. Angular розширює синтаксис HTML, дозволяючи розробникам створювати більш виразний, читабельний і легко підтримуваний код. Завдяки цьому підходу значно спрощується робота з динамічними інтерфейсами.

Основні цілі розробки Angular включають:

Відділення маніпуляцій з DOM від бізнес-логіки застосунку, що суттєво покращує тестування коду та його читабельність;

Підвищення уваги до тестування як до невід’ємної частини процесу розробки. Складність тестування безпосередньо залежить від рівня структурованості та модульності коду;

Чітке розділення клієнтської та серверної частин застосунку, що дозволяє вести розробку цих частин паралельно та незалежно одна від одної;

Проведення розробника через весь цикл створення програми – від проектування користувацького інтерфейсу, через реалізацію бізнес-логіки, до написання тестів та розгортання.

Angular дотримується архітектурного шаблону MVVM (Model-View-ViewModel), що сприяє створенню слабкого зв’язку між графічним представленням (View), даними (Model) та бізнес-логікою (ViewModel). Використовуючи потужний механізм впровадження залежностей (Dependency Injection), фреймворк дозволяє передавати на клієнтську сторону сервіси, подібні до класичних серверних служб, такі як взаємопов’язані контролери та репозиторії. Завдяки цьому значно зменшується навантаження на сервер, підвищується швидкодія клієнтської частини та спрощується масштабованість усього застосунку.

1.7. Microsoft Azure як хмарна платформа

Microsoft Azure – це комплексна хмарна платформа та інфраструктура, розроблена компанією Microsoft для створення, розгортання та управління сучасними хмарними застосунками. Основною метою платформи є значне спрощення процесу створення, тестування та експлуатації онлайн-програмного забезпечення, а також забезпечення високого рівня масштабованості, надійності та безпеки. Azure надає розробникам широкий спектр сервісів, які дозволяють будувати як прості веб-додатки, так і складні розподілені системи, що працюють у глобальному масштабі.

Історично платформа спочатку називалася Windows Azure і була орієнтована переважно на технології Microsoft. Однак з часом Azure перетворився на універсальну хмарну екосистему, яка підтримує широкий спектр технологій. Для створення додатків на платформі Azure можна використовувати практично всі сучасні мови програмування та фреймворки. Крім технологій Microsoft .NET, розробники можуть застосовувати Java, PHP, Node.js, Python, Ruby, Go, C/C++ та багато інших. Така поліглотність робить Azure привабливим для різноманітних команд розробників і підприємств.

Важливою складовою частиною Azure є інструменти для локальної розробки та налагодження. Раніше для цього використовувався SDK-емулятор хмари (Compute Emulator), який дозволяв розробникам імітувати роботу хмарного середовища на локальному комп’ютері. Емулятор надавав можливість підключатися до локального управління, виконувати налагодження в режимі реального часу, реєструвати діагностичну інформацію та використовувати сервіси, подібні до тих, що доступні в реальній хмарі. Однак емулятор мав певні обмеження: він не повністю відтворював поведінку реальної платформи, зокрема механізми балансування навантаження, масштабування та деякі аспекти безпеки. Крім того, локальні екземпляри працювали з підвищеними привілеями (адміністратор), тоді як у реальній хмарі застосовувався принцип найменших привілеїв. Сучасні

інструменти Azure (Visual Studio, Azure CLI, Azure Developer CLI) значно розширили можливості локальної розробки та інтеграції з хмарою.

Архітектура Microsoft Azure побудована таким чином, щоб забезпечувати гнучкість і масштабованість. Платформу можна використовувати як для розробки та запуску додатків безпосередньо в хмарі, так і для гібридних сценаріїв, коли частина застосунку працює локально на комп'ютерах користувачів або в приватній інфраструктурі, а інша частина – у хмарному середовищі.

Microsoft Azure складається з великої кількості взаємопов'язаних сервісів. Основними компонентами платформи є:

Compute (Обчислення) – набір сервісів для виконання додатків, включаючи віртуальні машини (Azure Virtual Machines), контейнери (Azure Kubernetes Service, Azure Container Instances), безсерверні обчислення (Azure Functions) та спеціалізовані середовища для веб-додатків (Azure App Service).

Storage (Зберігання) – потужна система зберігання даних у хмарі, яка включає Blob Storage для неструктурованих даних (зображення, відео, файли), Table Storage, Queue Storage, а також File Storage.

Azure SQL Database (раніше SQL Azure) – повністю керована реляційна база даних як сервіс (Database as a Service), яка забезпечує високу доступність, автоматичне масштабування та резервне копіювання.

Networking та інші інфраструктурні сервіси – забезпечують безпечне з'єднання, балансування навантаження, доставку контенту та захист від DDoS-атак.

Application Insights – це розширювана служба моніторингу продуктивності додатків (Application Performance Management – APM), призначена для веб-розробників і підтримує багато платформ. Служба дозволяє здійснювати глибокий моніторинг працюючих веб-додатків, автоматично виявляти аномалії продуктивності, збирати телеметрію, аналізувати поведінку користувачів, діагностувати проблеми та постійно покращувати якість і зручність використання застосунку. Application Insights

працює з різноманітними технологіями, включаючи .NET, Node.js, Java, Python та інші, як у хмарному, так і в локальному середовищі. Служба тісно інтегрується з процесами DevOps і має зручні точки підключення до багатьох інструментів розробки.

Для використання Application Insights розробник встановлює невеликий пакет інструментування (SDK) у свій додаток і налаштовує відповідний ресурс на порталі Microsoft Azure. Після цього пакет відстежує роботу застосунку і відправляє дані телеметрії на портал Azure. Важливою перевагою є те, що Application Insights може працювати з будь-яким додатком незалежно від місця його розміщення. Крім веб-сервісів, інструментування можна застосовувати до фонових компонентів, черг, баз даних, а також JavaScript-коду на клієнтських веб-сторінках.

Microsoft Azure постійно розвивається та пропонує все нові сервіси, такі як штучний інтелект (Azure AI), машинне навчання, Internet of Things, серверless-архітектура та гібридні рішення. Завдяки високому рівню надійності, глобальній присутності (десятки регіонів по всьому світу), моделі оплати за фактичне використання ресурсів та потужній екосистемі інструментів, Azure є однією з провідних хмарних платформ у світі та ідеально підходить для реалізації сучасних веб-сервісів, у тому числі хмарних сховищ медіа-даних.

1.8. ADO.NET Entity Framework

ADO.NET Entity Framework (EF) є об’єктно-орієнтованою технологією доступу до даних, яка представляє собою рішення типу object-relational mapping (ORM) від компанії Microsoft. Вона забезпечує можливість взаємодії з даними у вигляді об’єктів, підтримуючи два основні механізми виконання запитів: LINQ to Entities та Entity SQL. Для побудови сучасних веб-рішень Entity Framework інтегрується з різними технологіями Microsoft, зокрема з ASP.NET, що дозволяє реалізовувати багаторівневі архітектури на основі шаблонів проектування MVC, MVP чи MVVM.

Entity SQL є спеціалізованою мовою запитів, синтаксично подібною до Transact-SQL, яка призначена для роботи з концептуальними моделями в Entity Framework. Вона дає змогу виконувати запити безпосередньо до абстракції даних, незалежно від конкретної реалізації сховища.

Історично перші версії Entity Framework підтримували три основні підходи до створення моделі даних. Підхід Database First дозволяв на основі вже існуючої бази даних автоматично генерувати концептуальну модель у форматі EDMX-файлу, яка потім використовувалася для підключення та роботи з даними. Пізніше був доданий підхід Model First, що передбачав ручне створення моделі за допомогою візуального дизайнера в середовищі Visual Studio з подальшим генеруванням бази даних на її основі.

Починаючи з версії 4.1 (і особливо з 5.0) найбільш сучасним і рекомендованим став підхід Code First [5]. Його принцип полягає в тому, що розробник спочатку описує доменну модель у вигляді звичайних C#-класів (POCO), після чого Entity Framework на основі цих класів, конвенцій та конфігурації (за допомогою Fluent API або Data Annotations) автоматично створює або оновлює схему бази даних. При цьому використання EDMX-файлів стає непотрібним, що значно спрощує процес розробки та роботи в команді.

У контексті сучасної розробки на платформі ASP.NET Core класичний Entity Framework 6 поступово витісняється Entity Framework Core – повноцінним переосмисленням технології, яке є кросплатформенним, більш продуктивним, модульним та оптимізованим для хмарних середовищ. EF Core зберігає основні принципи роботи з LINQ, міграціями та change tracking, але пропонує значно кращу продуктивність, підтримку асинхронних операцій за замовчуванням, shadow properties, global query filters, value converters та інші сучасні можливості. Він є стандартним вибором для нових проектів на .NET 8 та .NET 10.

1.9. Автомасштабування

Автомасштабування (autoscaling) – це процес динамічного розподілу обчислювальних ресурсів відповідно до поточних вимог до продуктивності системи. У міру зростання обсягу робіт (збільшення кількості користувачів, підвищення навантаження на сервіс) додаток може потребувати додаткових ресурсів для підтримання бажаного рівня швидкодії та дотримання угод про рівень обслуговування (Service Level Agreement – SLA). Навпаки, при зниженні попиту надлишкові ресурси автоматично звільняються, що дозволяє мінімізувати експлуатаційні витрати.

Автомасштабування ефективно використовує ключову перевагу хмарних середовищ – еластичність. Воно суттєво зменшує необхідність постійного ручного моніторингу продуктивності системи оператором та прийняття рішень щодо додавання або вилучення ресурсів [2]. Завдяки цьому досягається оптимальне співвідношення між продуктивністю, доступністю та вартістю володіння системою.

Існує два фундаментальні способи масштабування додатків:

Вертикальне масштабування (scale up / scale down) передбачає зміну потужності окремого ресурсу – наприклад, збільшення обсягу оперативної пам’яті, кількості ядер процесора або типу віртуальної машини. Такий підхід часто вимагає тимчасового призупинення роботи системи під час перерозподілу ресурсів, що ускладнює його автоматизацію та робить менш придатним для високонавантажених сервісів з високими вимогами до доступності.

Горизонтальне масштабування (scale out / scale in) полягає в динамічному додаванні або видаленні екземплярів (інстансів) ресурсу – наприклад, додаткових віртуальних машин, контейнерів або інстансів App Service. Додаток продовжує працювати безперервно, оскільки нові ресурси поступово вводяться в роботу. Після завершення ініціалізації нові екземпляри починають обробляти запити. При зниженні навантаження надлишкові інстанси коректно завершують роботу та звільняються. Цей тип

масштабування є найбільш поширеним у хмарних середовищах завдяки високій відмовостійкості та можливості повної автоматизації.

Більшість сучасних хмарних платформ, зокрема Microsoft Azure, надають потужні вбудовані інструменти для автоматичного горизонтального масштабування. У Azure це реалізується через Azure Autoscale, Virtual Machine Scale Sets, Azure Kubernetes Service (AKS) з Horizontal Pod Autoscaler, Azure App Service та інші сервіси. Автомасштабування може здійснюватися на основі метрик (CPU, пам’ять, кількість запитів, черги), розкладу або комбінації обох підходів, що забезпечує високу адаптивність системи до реальних умов експлуатації.

1.10. Постанова задачі

Завданням є розробити веб-сервіс за допомогою платформи ASP.NET та розмістити його на Azure. Веб-сервіс повинен надавати API для програм клієнтів: javascript, ios, android тощо. Також розробити веб-клієнт з інтуїтивним графічним інтерфейсом за допомогою платформи Angular. Клієнтський застосунок повинен реалізувати усі API методи для взаємодії з веб-сервісом.

РОЗДІЛ 2
СИСТЕМНИЙ АНАЛІЗ

Системний – аналіз це наукова методологія, об'єктом аналізу якої є проблема, незалежно від сфери діяльності, де вона виникла, а метою системного аналізу є проект вирішення проблеми. Системний аналіз є напрямом, в якому поєднано методологію і досягнення математичних і прикладних наук. Системний аналіз у технічній галузі орієнтований на вирішення складних проблем аналізу та створення комп'ютерних, комунікаційних, інформаційних та інших технічних систем, і ґрунтується на принципах інженерних наук, імітаційному та інформаційному моделюванні об'єктів і процесів та націлений на застосування в конкретних проєктах, розробленнях, прикладних дослідженнях і дослідницько-конструкторських роботах.

3.1. Опис предметної області та перелік вимог до системи

Веб-застусунок призначений для широкої аудиторії користувачів, котрі хочуть поділитися цікавими моментами знятими на об'єктив камери. Доступ передбачається для усіх користувачів, котрі користуються останніми версіями інтернет переглядачів.

Веб-сервіс дає змогу реєструватися і логінитися під своїм іменем, підписуватися на оновлення інших користувачів, переглядати записи інших користувачів, ставити відмітки, ділитися ними з друзями та знайомими, залишати та редагувати коментарі, відвантажувати та редагувати свої фотографії.

Користувачеві надається можливість налаштувати свій обліковий запис, встановити зображення свого профілю, залишити деталі про себе та свої захоплення. Користувач також має можливість зробити свій профіль прихованим. Для того, щоб інші користувачі мали можливість переглядати

такий профіль, їм необхідно відправити запит, щоб власник приватної сторінки надав доступ.

2.1. Особливості програмного продукту

Зареєстрований у системі користувач має можливість налаштовувати стрічку оновлень, переглядати останні оновлення друзів, ділитися зображеннями. Не зареєстрований у системі користувач не має доступу до стрічки оновлень користувачів та не має можливості відвантажувати свої зображення, залишати коментарі під світлинами інших користувачів та ставити відмітки. У такого типу користувачів є лише доступ за посиланням до сторінок користувачів, які не мають включеної функції приватності у налаштуваннях свого профілю.

У системі передбачена система заявок. Завдяки цьому, користувач має можливість підписатися на приватну сторінку іншого користувача після того як останній підтвердить запит першого. Користувач із приватною сторінкою має можливість відхилити або заблокувати запит користувача. Якщо користувач заблокував доступ до своєї сторінки іншому користувачеві, то заблокований користувач не матиме можливості відправляти повторні запити на надання доступу до стрічки оновлень користувача.

У налаштуваннях профілю є доступна опція деактивація аккаунту. Після деактивації аккаунту усі зображення зберігаються. Такий аккаунт перестає бути видимим для інших активних користувачів.

Обліковий запис користувача можна також зробити приватним. Такий обліковий запис буде лише доступний для підтверджених у запиті користувачів. Якщо інші користувачі підписалися на оновлення аккаунту до того як він став приватним, то їм не буде необхідності відправляти повторний запит. Вхідний запит також можна відхилити, а надоїдливого користувача перемістити у чорний список. Такий користувач більше не зможе відправляти повторні запити, до моменту видалення його із списку.

Перелік учасників системи. Система передбачає три типи користувачів: адміністратор, система, користувач. Окремо можна також виділити незареєстрованих користувачів – гостей.

Роль системи полягає у виконанні запланованих операцій: очистка системи від непотрібних зображень, які з'являються при наступному сценарії: користувач відвантажив зображення та не закріпив його за записом впродовж п'яти хвилин. Роль системи використовує системна задача, яка автоматично за вказаним інтервалом у налаштуваннях Microsoft Azure запускається та виконує вказані операції.

Адміністратор – це користувач з додатковими можливостями, а саме: блокування інших користувачів із невідповідним контентом, опрацьовувати скарги інших користувачів тощо.

Користувач – це стандартна для усіх зареєстрованих користувачів роль, яка надає доступ до створення постів, перегляду стрічок оновлень, пошук користувачів та записів за назвою тега, ставити відмітки «мені подобається» та видаляти їх, коментувати записи та видаляти свої коментарі і записи, редагувати свій профіль. Незареєстровані учасники системи матимуть доступ лише до стрічки новин неприватних облікових записів та постів цих записів.

Для доступу до системи необхідний пристрій з доступом до інтернету та останньою версією веб-переглядачів: Google Chrome, Safari, Opera, , Microsoft Edge, Mozilla Firefox.

До компонентів системи відносяться:

- графічний інтерфейс клієнтської програми;
- Сервер автентифікації;
- API-сервер;
- Azure Web Jobs;
- Azure Blob Storage;
- Azure SQL Database.

2.2. Функціональні вимоги

Для початку роботи в системі користувач повинен створити обліковий запис через форму реєстрації або авторизуватися за допомогою зовнішніх сервісів (Google, Facebook, Apple тощо). Після реєстрації користувач перенаправляється на сторінку авторизації.

До функціональних вимог системи належать:

- Автономна реєстрація користувача або авторизація через зовнішні сервіси;
- Автентифікація та авторизація на основі протоколу OAuth 2.0 / OpenID Connect;
- Налаштування профілю (фотографія, особиста інформація, інтереси);
- Створення публікацій із можливістю додавання кількох медіафайлів;
- Додавання текстового опису до публікації;
- Редагування власних публікацій;
- Автоматичне генерування хештегів на основі опису;
- Підписка на оновлення інших користувачів;
- Редагування профілю та налаштувань доступу;
- Налаштування рівнів приватності та деактивація облікового запису;
- Перегляд стрічки оновлень в антихронологічному порядку;
- Видалення власних публікацій;
- Додавання та видалення коментарів;
- Постановка та зняття вподобань («лайків»);
- Копіювання посилання на публікацію;
- Пошук користувачів та публікацій за хештегами;
- Управління запитами на підписку, блокування та розблокування користувачів.

2.3. Нефункціональні вимоги

Вимоги до продукту:

- Доступ до веб-застосунку з будь-якого сучасного інтернет-браузера;
- Оптимізація роботи з базою даних з використанням асинхронних запитів для забезпечення високої продуктивності при значному одночасному навантаженні;
- Захист від шахрайства, зокрема блокування вразливостей на рівні браузера;
- Надійна автентифікація та авторизація з використанням протоколів OAuth 2.x / OpenID Connect;
- Захист від Cross-Site Request Forgery (CSRF), а також інших поширених веб-атак;
- Підтримка багатомовного інтерфейсу;
- Підтримка зміни теми оформлення (світла/темна).

2.4. Побудова дерева цілей

Дерево цілей – це графічне зображення взаємозв'язку і підпорядкованості цілей, що відображає розподіл місії і мети на цілі, під цілі, завдання та окремі дії. Дерево цілей можна визначити, як цільовий каркас організації, явища чи діяльності. Дерево цілей зображено у додатку 1.

Для створення порталу визначено наступні цілі:

1. Розробка UI частини:

- Підключення сучасної версії платформи Angular;
- Розробка компонентів та сервісів для взаємодії з серверною частиною;
- Створення HTML-шаблонів з використанням компонентної архітектури Angular;
- Розробка сервісів для HTTP-запитів до сервера;
- Реалізація модулів модальних вікон та сповіщень;
- Розробка інтерактивних елементів інтерфейсу.

2. Розробка серверної частини (WEB API):

- Налаштування модулів безпеки та автентифікації;
- Реалізація авторизації та автентифікації за допомогою протоколу OAuth 2.0;
- Створення системи ролей та збереження контексту користувача в базі даних;
- Оптимізація серверної частини для забезпечення високої швидкодії;
- Розробка моделей даних для API;
- Реалізація мапінгу між API-моделями та доменними моделями;
- Створення контролерів для обробки бізнес-логіки;
- Захист системи від автоматизованих атак та зловмисників.

3. Розробка рівня доступу до бази даних:

- Проектування доменної моделі;
- Створення контексту бази даних;
- Реалізація репозиторіїв для кожної сутності;
- Впровадження шаблону Unit of Work;
- Підключення Entity Framework Core;

Налаштування механізму автоматичного створення та оновлення бази даних.

2.5. Побудова дерева проблем

Для виконання поставленого завдання необхідно врахувати усі проблеми та проаналізувати їх. Дерево проблем зображено у додатку 2.

Перша гілка дерева містить такі складові:

1.1. Головним призначенням є збереження медіа-даних у хмарі.

1.2. Умови керування даною системою.

Доступ до програми можна отримати із будь-якого сучасного пристрою із доступом до мережі інтернет.

Друга гілка дерева містить такі складові.

2.1. Функція системи.

Основною функцією базових компонент є можливість зберігати зображення у хмарному сховищі.

2.2. Структура частини забезпечення системи.

Серверна частина забезпечує постійний доступ до веб-ресурсу.

2.3. Механізм функціонування системи.

Дані базові компоненти можуть функціонувати режимі реального часу.

Третя гілка дерева містить такі складові:

3.1. Характеристика способу організації розробки системи.

3.2. Автомасштабування системи.

Розміщення системи на декількох серверах, щоб забезпечити продуктивність при максимальній навантаженні.

2.6. Аналіз та вибір архітектури проекту

Ефективність функціонування системи та зручність її подальшої підтримки значною мірою залежать від обраної архітектури. Важливим принципом є чітке інкапсулювання компонентів та розділення відповідальності між шарами системи. Серверна та клієнтська частини розробляються як окремі рішення (Рис. 2.1).

Проект реалізовано з використанням багатошарової (N-Tier) архітектури, яка включає такі основні проекти:

- Infrastructure: PhotoCloud.Infrastructure, PhotoCloud.Infrastructure.Data, PhotoCloud.Infrastructure.EF, PhotoCloud.Infrastructure.WebApiClient;
- Application: PhotoCloud.Services, PhotoCloud.DataContracts, PhotoCloud.Managers, PhotoCloud.Mapping, PhotoCloud.Proxies, PhotoCloud.Resources, PhotoCloud.Security;
- Azure: PhotoCloud.Azure.Jobs, PhotoCloud.Azure.Jobs.Core;
- Domain: PhotoCloud.Domain, PhotoCloud.Domain.Entities;
- Repositories: PhotoCloud.Repositories, PhotoCloud.Repositories.EF;
- Storage: PhotoCloud.CloudStorage, PhotoCloud.Database;
- Presentation: PhotoCloud.Web.

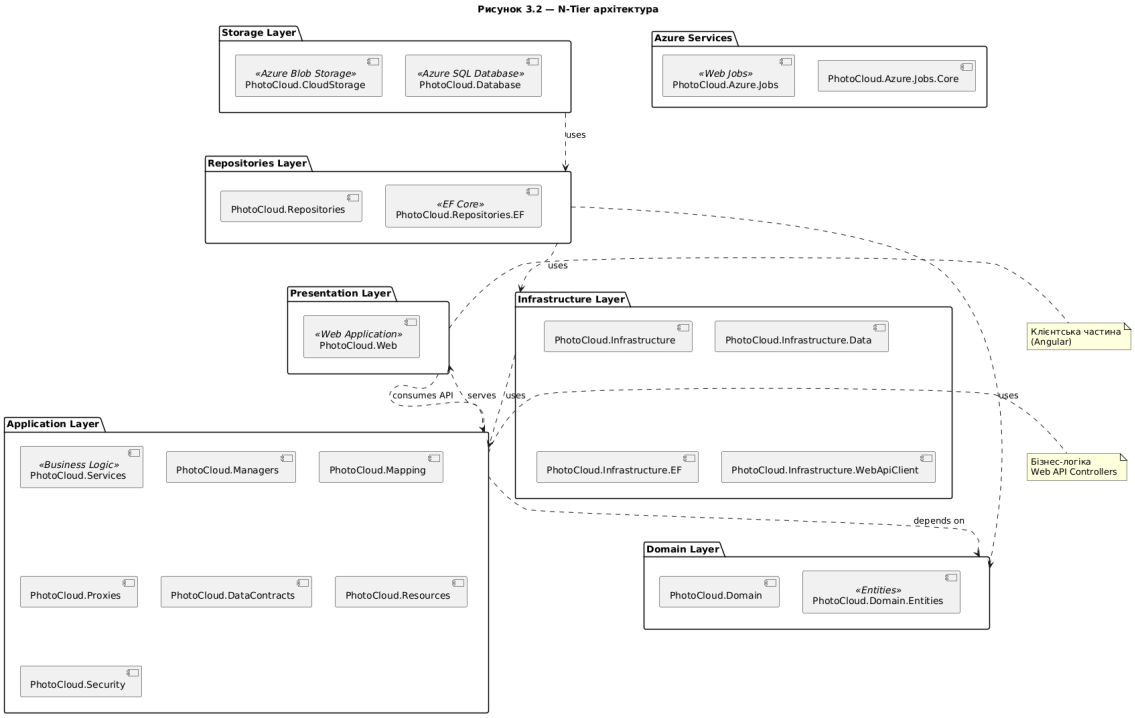


Рис. 2.1. N-Tier архітектура

N-Tier архітектура забезпечує високу підтримуваність, масштабованість та зручність внесення змін у систему.

2.7. Аналіз засобів та інструментів для розробки та проєктування системи

Вибір сучасних і ефективних засобів розробки та проєктування є одним із ключових факторів, що визначають успішність реалізації програмного проєкту, його якість, продуктивність розробки та зручність подальшої підтримки. Правильно підібраний інструментарій дозволяє суттєво скоротити час розробки, підвищити надійність коду, забезпечити ефективну командну взаємодію та спростити процес тестування й розгортання системи.

Для реалізації серверної частини веб-застосунку було обрано Microsoft Visual Studio як основне інтегроване середовище розробки. На момент розробки використовувалася версія Visual Studio 2022 Enterprise, яка є одним із найбільш потужних і функціональних інструментів для створення корпоративних застосунків на платформі .NET. Visual Studio надає широкі

можливості для розробки, налагодження, рефакторингу коду, автоматичної генерації коду, інтеграції з системами контролю версій, а також вбудовану підтримку Entity Framework Core, ASP.NET Core Web API, Dependency Injection та багатьох інших сучасних технологій. Серед ключових переваг даного середовища слід виділити потужний IntelliSense, вбудований профайлер продуктивності, інструменти для статичного аналізу коду (Code Analysis), а також можливість розгортання безпосередньо в середовище Microsoft Azure.

У якості основної системи керування базами даних обрано Microsoft SQL Server 2022 у поєднанні з Azure SQL Database. Microsoft SQL Server є надійною, високопродуктивною і масштабованою реляційною системою керування базами даних (РСУБД), яка повністю відповідає вимогам проєкту щодо зберігання даних користувачів, медіа-файлів, взаємозв’язків та налаштувань приватності. Для роботи з базою даних використовується мова Transact-SQL – розширена реалізація стандарту ANSI/ISO SQL, що включає потужні засоби процедурного програмування, тригери, збережені процедури та вбудовані функції. Використання Azure SQL Database забезпечує автоматичне резервне копіювання, високу доступність, геореплікацію та автоматичне масштабування ресурсів, що є критично важливим для хмарного веб-застосунку.

Для організації контролю версій коду, управління завданнями та налаштування процесів безперервної інтеграції та розгортання (CI/CD) було прийнято рішення використовувати Azure DevOps Services (раніше відома як Team Foundation Server). Дана платформа є сучасним еволюційним розвитком TFS і пропонує повний набір інструментів DevOps: Git-репозиторії, дошки завдань (Boards), пайплайни CI/CD, тестові плани, артефакти та детальне управління релізами. Перевагами Azure DevOps є глибока інтеграція з Visual Studio, Microsoft Azure, а також можливість організації як приватних, так і відкритих проєктів. Використання Git замість

централізованої системи TFVC дозволило забезпечити гнучкість і зручність паралельної розробки.

Для фронтенд-розробки застосовувався Visual Studio Code з розширеннями для Angular (Angular Language Service, ESLint, Prettier), що забезпечує швидке редагування, автодоповнення та статичний аналіз TypeScript-коду. Збірка фронтенд-частини здійснювалася за допомогою Angular CLI та npm, що є сучасним стандартом для проєктів на базі Angular.

- Додаткові інструменти, які використовувалися в процесі розробки:
- Postman та Swagger UI – для тестування та документування Web API;
- Azure Storage Explorer – для роботи з Azure Blob Storage;
- GitHub Desktop та Git – як допоміжні інструменти контролю версій;
- SQL Server Management Studio – для адміністрування бази даних;

Application Insights – для моніторингу продуктивності та помилок у реальному часі після розгортання в Azure.

Вибір саме цих засобів обумовлений кількома факторами: по-перше, їх повною сумісністю з обраним технологічним стеком (.NET та Azure); по-друге, широкою поширеністю в індустрії, що полегшує подальшу підтримку системи; по-третє, можливістю безкоштовного використання студентських та освітніх ліцензій; і, нарешті, високою ефективністю при реалізації хмарних веб-застосунків.

Таким чином, комплексне використання сучасного інструментарію дозволило забезпечити високу якість реалізації проєкту, дотримання кращих практик програмної інженерії та створення масштабованого, надійного та зручного в підтримці програмного рішення.

РОЗДІЛ 3
РОЗРОБКА АЛГОРИТМІВ ТА ПРОГРАМНОГО РІШЕННЯ ВЕБ-СЕРВІСУ

3.1. Концептуальна модель бази даних

Концептуальна модель бази даних є ключовим елементом проектування інформаційної системи, оскільки визначає структуру зберігання даних, їхні взаємозв’язки та забезпечує цілісність інформації. У розробленій системі дані зберігаються в реляційній базі даних Microsoft SQL Server. Основними таблицями є:

Users – містить основну інформацію про користувачів системи (ідентифікатор, ім’я користувача, повне ім’я, електронна пошта, дата реєстрації, налаштування профілю, параметри приватності, статус активності облікового запису тощо);

UserRoles – таблиця зв’язку «багато-до-багатьох» між користувачами та ролями;

Roles – довідкова таблиця ролей системи (користувач, адміністратор, системна роль);

Posts – таблиця записів (публікацій) користувачів, яка включає опис, дату створення, ідентифікатор автора, кількість вподобань, коментарів тощо;

Attachments – таблиця вкладень (зображення, відео), що містить посилання на файл у хмарному сховищі, дату завантаження та зв’язок із записом;

Comments – таблиця коментарів до записів;

UserLikes – таблиця вподобань («подобається») користувачів до записів;

UserRelationships – проміжна таблиця для моделювання підписки/дружби між користувачами;

BlockedUsers – таблиця заблокованих користувачів;

RelationshipRequests – таблиця запитів на підписку (дружбу).

Така структура моделі бази даних є оптимальною, оскільки вона дозволяє гнучко розширювати функціональність системи без суттєвої перебудови схеми. Центральною сутністю системи виступає користувач, навколо якого організовано всі основні бізнес-процеси. Концептуальну модель бази даних зображено на рис. 3.1.

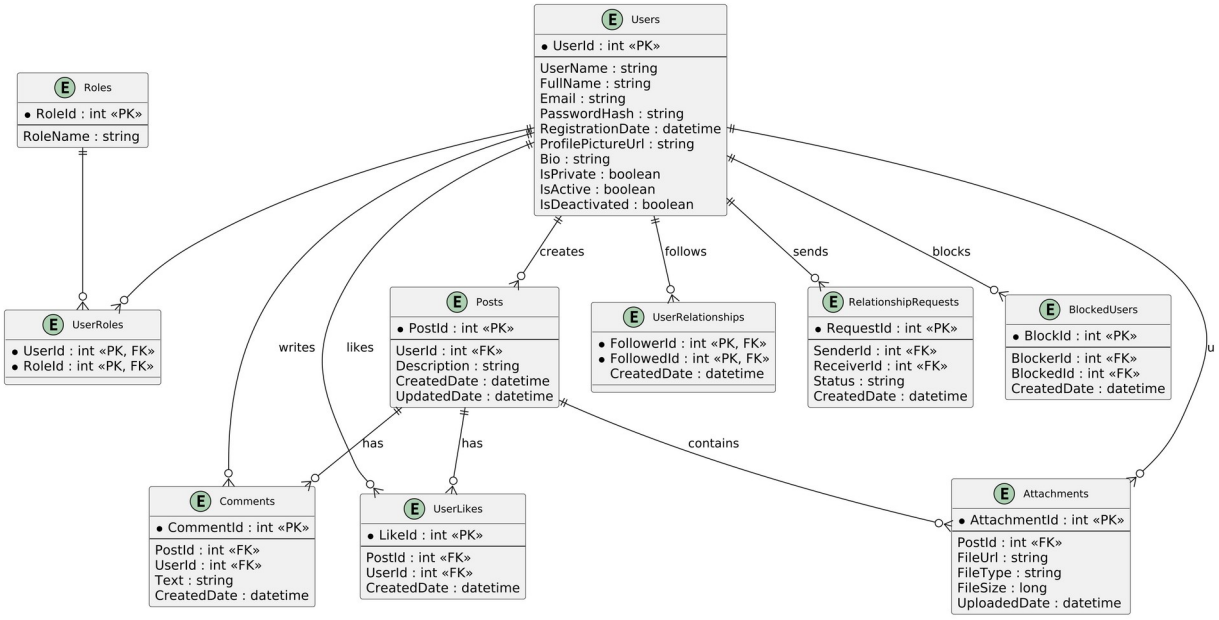


Рис. 3.1. Концептуальна модель бази даних

У процесі проектування було враховано принципи мікросервісної архітектури. Хоча основна реалізація виконана у вигляді монолітного застосунку з чітким розподілом на шари, архітектурні рішення (окремі проекти для доменної моделі, репозиторіїв, сервісів, API) полегшують потенційний перехід до мікросервісів у майбутньому. Комунікація між компонентами здійснюється через чітко визначені інтерфейси, що відповідає принципам слабкого зв’язування.

Діаграма прецедентів (Use Case) - це діаграма для відображення відношень між користувачами (акторами) та прецедентами у системі, що розробляється.

Діаграму прецедентів можна вважати графом, який складається з акторів, прецедентів. Граф обмежується границею, яка називається границя системи. Між акторами та прецедентами проводяться лінії асоціацій. Між прецедентами проводяться лінії відношень. Загалом, діаграма прецедентів відображає варіанти використання системи користувачем, тобто клієнтом.

На рисунку 3.2 зображено діаграму прецедентів системи, що розробляється.

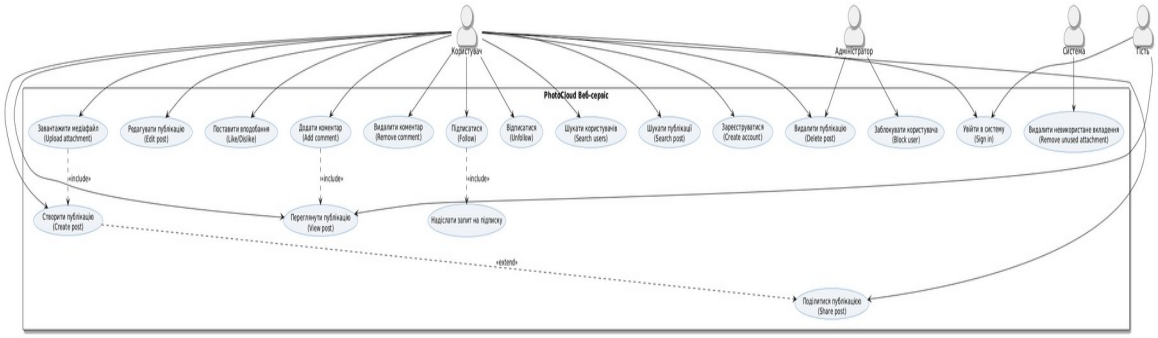


Рис. 3.2 Діаграма прецедентів

Діаграма класів – це графічне представлення структури моделі. На діаграмі класів відображаються класи та інтерфейси, їх вміст, тобто поля та методи, відносини між класами. На такій діаграмі також можуть позначатися поведінка елементів. Після написання програмного коду, Visual Studio дозволяє згенерувати діаграму класів, що набагато полегшує роботу та створення документації розробнику, оскільки не має необхідності малювати вручну усі таблиці та заповнювати їх внутрішніми елементами (методами, полями, властивостями, подіями тощо). Нижче на рис. 3.3 зображено діаграму класів WEB API сервера, який відповідає за звернення до бази даних та повернення даних на користувацький інтерфейс.

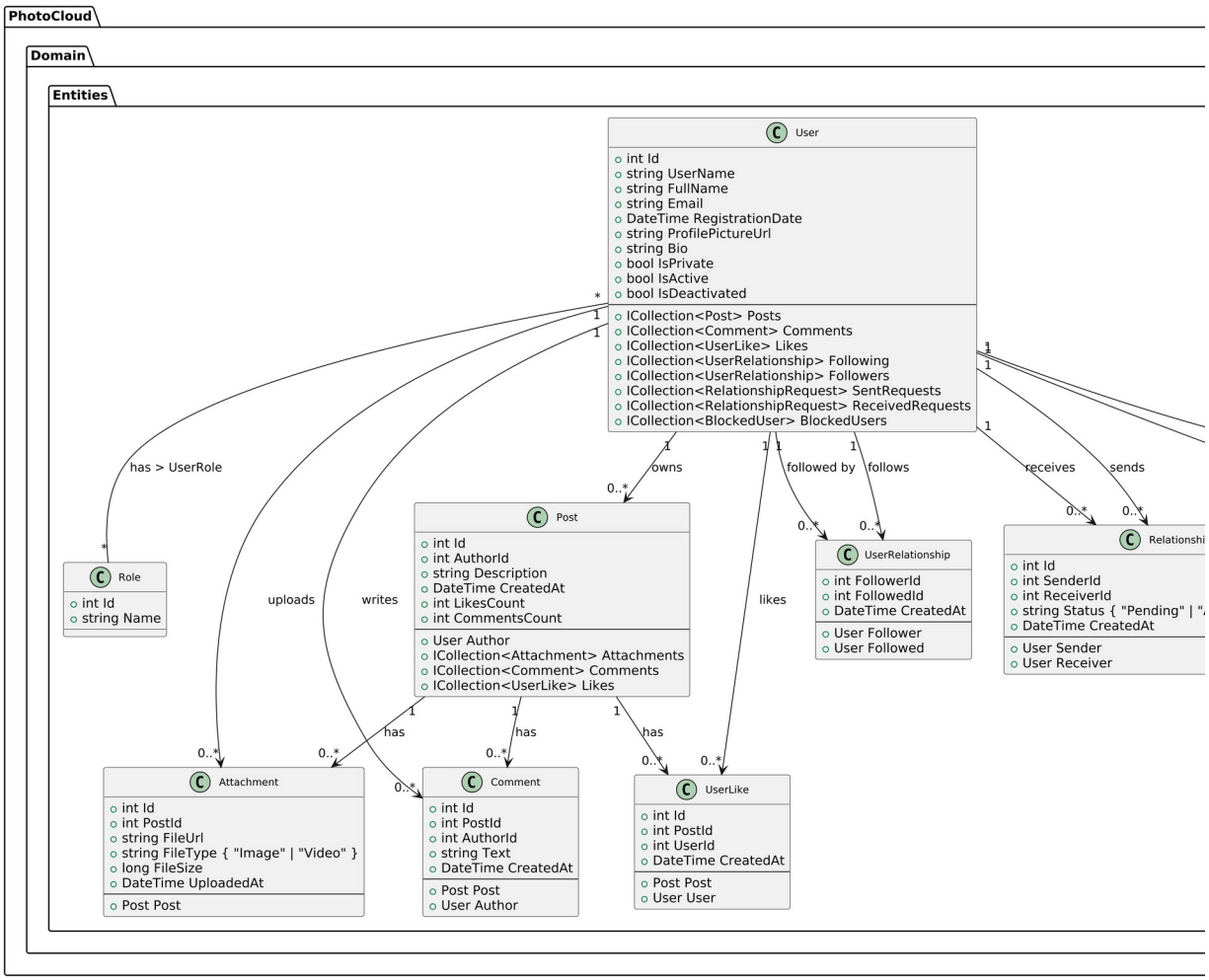


Рис. 3.3. Діаграма класів доменних моделей

Доменні моделі – це відображення сутностей, що зберігаються та читаються з бази даних. Властивості цих моделей повністю відповідають полям та їхнім типам у базі даних.

Репозиторій є посередником між рівнями області визначення та відображення даних. Він містить класи та методи для роботи із основними сутностями системи та їх взаємодією із базою даних. Його можна назвати рівнем доступу до бази даних.

3.2. Схема бази даних

Схема бази даних це структура системи баз даних описана формальною мовою, яка підтримується системою управління баз даних і відноситься до організації даних для створення плану побудови бази даних з розподілом на

таблиці. Формально схема баз даних являє собою набір формул, які називаються обмеженнями цілісності. Обмеження цілісності забезпечують сумісність між всіма частинами схеми. Всі обмеження виражаються однією мовою. На рис. 3.4 зображено діаграму бази даних з усіма таблицями та їхніми полями, а також зв'язками між ними.

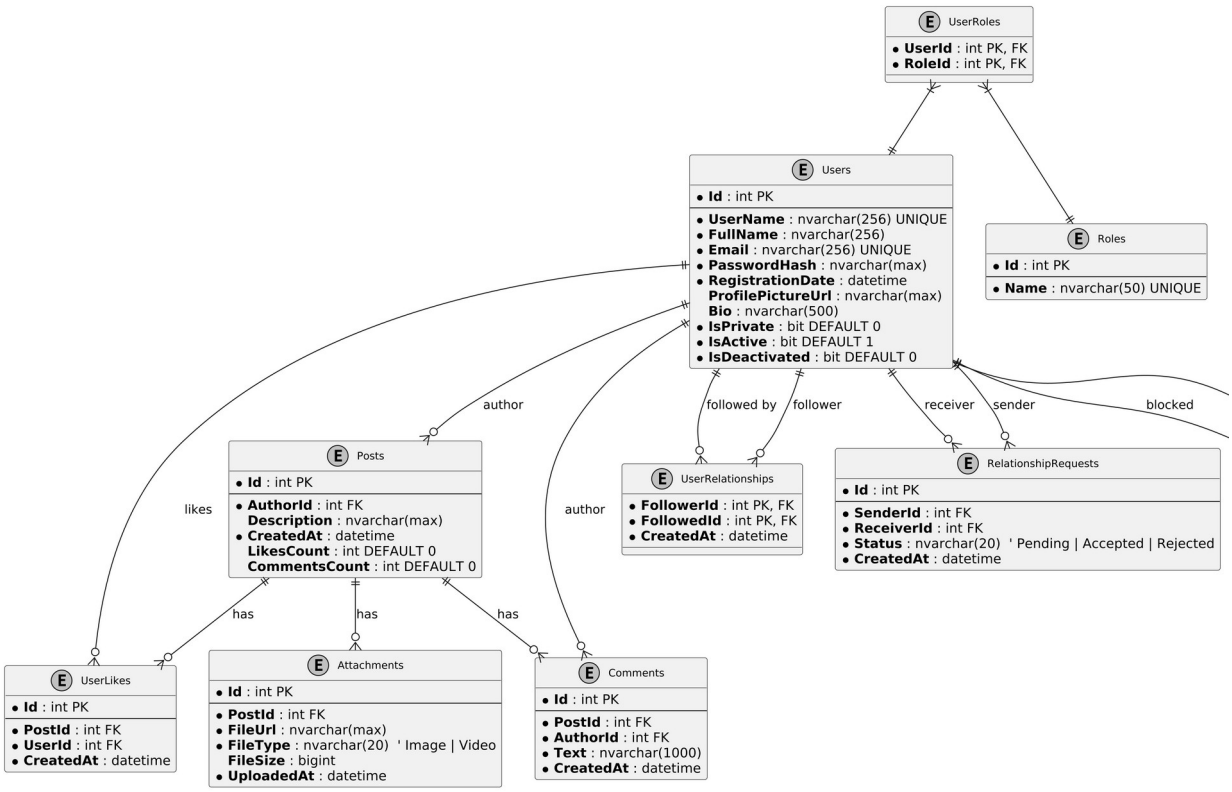


Рис. 3.4. Діаграма бази даних

Основні таблиці та їх призначення: Users – основна таблиця користувачів. Містить інформацію про профіль, налаштування приватності та статус аккаунту. Roles / UserRoles – реалізація ролей (User, Administrator, System) через зв’язок багато-до-багатьох. Posts – публікації користувачів (записи зі зображеннями/відео). Attachments – медіафайли (зображення та відео), прив’язані до публікацій. Посилання зберігаються в Azure Blob Storage. Comments – коментарі до публікацій. UserLikes – таблиця вподобань («лайків»). UserRelationships – підписки/дружба між користувачами. RelationshipRequests – запити на підписку (для приватних профілів). BlockedUsers – блокування користувачів.

3.3. Загальна структура програмного продукту

Для того, щоб програмний продукт міг в подальшому розвиватися необхідно розробити відповідну архітектуру програмного продукту та використати відповідні технології, щоб цього досягнути та надати можливість іншим розробникам використовувати можливості доступу до ресурсів сервера. Тому було прийнято рішення використати ASP.NET Core Web API, а для більшої оптимальності програмне рішення було розділено на підпроекти та окремі компоненти:

- API Client – клієнти, що можуть звертатися до сервера через API; клієнтами є програмні рішення з можливістю виконання HTTP запитів до ресурсу;
- OAuth 2.0 Authentication Server – окремий сервер автентифікації та авторизації.
- ASP.NET Core Web API – основний сервер, що надає RESTful API через контролери.
- Application services – служить проміжною ланкою між рівнем доступу до бази даних та рівнем презентації. У цьому рівні міститься основна бізнес логіка системи;
- Azure Web Job – заплановані операції підтримки системи;
- Repositories – описує колекції даних та реалізовує методи інтерфейсів для роботи з цими даними;
- Domain – шар доменних моделей та бізнес-правил.
- Infrastructure – інфраструктурні компоненти (робота з Azure Blob Storage, Entity Framework Core тощо).
- Presentation (Web Client) – клієнтська частина на базі Angular.

Завдяки такій архітектурі і реалізації програма може в подальшому розвиватися і бути орієнтована не лише на веб, але і настільних, мобільних додатках, оскільки робота з даними відбувається завдяки відповідному API.

3.4. Розробка та опис програмних модулів

Для розробки програмного рішення було використано Microsoft Visual Studio. Для написання коду модулів сервера, сервісів та взаємодії з базою даних було використано мову C#. Для Front-end частити було використано JavaScript.

Під час розробки було використано такі технології, бібліотеки та інструменти:

- Entity Framework Core – сучасний ORM для роботи з базою даних.
- ASP.NET Core Identity – система управління ідентифікацією та авторизацією.
- AutoMapper – бібліотека для мапінгу між доменними та транспортними моделями.
- FluentValidation – валідація вхідних даних.
- Ninject / Microsoft.Extensions.DependencyInjection – контейнери впровадження залежностей.
- Angular (з Angular Material) – для створення сучасного Single Page Application.
- Azure SDK for .NET – для роботи з Azure Blob Storage та Azure Web Jobs.

Модуль автентифікаціх та авторизації

Автентифікація дозволяє однозначно ідентифікувати користувача. Наприклад, користувач може увійти у систему за допомогою логіна та пароля, а система використовує ці дані для автентифікації користувача. **Авторизація** визначає чи може користувач виконувати ті чи інші дії. Наприклад, чи може користувач переглядати той чи інший ресурс.

Автентифікація забезпечує однозначну ідентифікацію користувача в системі, тоді як авторизація визначає дозволені йому дії. У розробленому рішенні реалізовано гібридний підхід до автентифікації:

- Локальна реєстрація та авторизація (логін/пароль).
- Авторизація через зовнішні провайдери (Google, Facebook тощо) за допомогою протоколу OAuth 2.0 / OpenID Connect.

Для управління доступом використовуються політики авторизації ASP.NET Core та рольова модель. Контекст безпеки користувача (ClaimsPrincipal) формується на основі JWT-токенів, що забезпечує stateless-авторизацію та високу масштабованість.

Користувацький модуль авторизації. Для гнучкого керування доступом було розроблено власні атрибути авторизації, що успадковуються від AuthorizationFilterAttribute та реалізують інтерфейс IAuthorizationFilter. Це дозволило реалізувати складну бізнес-логіку перевірки прав доступу (перевірка приватності профілю, статусу блокування тощо).

Авторизація всередині методу контролера. У деяких випадках можна дозволити виконання запиту, але контролювати поведінку на основі контексту безпеки. Наприклад, результат запиту залежить від ролі користувача. Усередині методу контролера можна отримати поточний контекст безпеки звернувшись до властивості APIController.User.

У випадку створення проекту за допомогою середовища Visual Studio, програмісту надається три параметри для автентифікації:

Індивідуальні акаунти використовують серверну базу даних, тобто ту що створена на хостингу;

Організаційні акаунти користувачі мають змогу увійти за допомогою Azure, Office 365 тощо;

Автентифікація за допомогою Windows у випадку, коли користувач використовує у глобальній мережі IIS як модуль автентифікації.

Індивідуальні акаунти надають два шляхи для користувача, щоб увійти:

- Локальний вхід. Користувач реєструється на сайті, ввівши ім'я користувача та пароль. Додаток зберігає хеш пароля в базі даних. Коли користувач входить в систему, система ідентифікації ASP.NET перевіряє пароль.
- Вхід через соціальні мережі та зовнішні служби. Користувач входить в систему за допомогою зовнішньої служби, як Facebook, Microsoft. або Google тощо. Додаток створює запис для користувача в базі членства, але не

зберігати будь-яких повноважень. Користувач входить у систему шляхом надання даних зовнішнього сервісу, який він використовує.

Одним із найпопулярніших засобів проведення автентифікації та авторизації є протокол OAuth 2.0. Цей протокол також використовують такі відомі соціальні мережі як Facebook, Twitter, Google тощо.

Для того, щоб виконувати запити необхідно отримати token, який є обов'язковим параметром виконання запиту. Token отримується після авторизації в мережі та повертається у адресному рядку браузера.

Для того, щоб можна було здійснювати get/post запити до платформи необхідно використовувати обов'язковий параметр token, який отримується після авторизації. Авторизація неофіційного додатку відбувається за протоко лом OAuth2.0.

OAuth *(англ. Oppen Authorization)* це відкритий стандарт для проведення авторизації, що надає можливість користувачам відкривати доступ до даних, що зберігаються на одному сервері іншому застосунку, без необхідності вводу імені користувача та паролю, тобто автентифікація відбувається на стороні сервера до якого користувач пробує під'єднатися.

OAuth дозволяє користувачам роздавати сайтам маркери доступу, до даних що розміщуються на сайтах-сервісах. Кожен маркер доступу надає доступ конкретному сайту (наприклад, сайту редагування відео) до конкретних ресурсів (наприклад, тільки відео від конкретного альбому) та на визначений термін (наприклад, на наступні 2 години). Це дозволяє користувачам надавати до-ступ третім сайтам до їх інформації, що зберігається на інших сайтах постачальниках послуг, не передаючи повною мірою самих даних та без застосування імені/паролю.

Під час роботи з даним протоколом необхідно виділити наступні терміни:

- Ресурс - захищені дані, тобто дані для доступу до яких необхідно пройти автентифікацію;
- Сервер ресурсу сервер - що містить ресурс;

- Власник ресурсу - особа, що має доступ до ресурсу;
- Клієнт - додаток, який взаємодіє з ресурсом (найчастіше клієнтом виступає веб-переглядач);
- Ключ доступу (access token) ключ, що надає праву переглядати ресурс;
- Bearer token - тип ключа, що надає декільком користувачам використовувати один і той же ж ключ.
- Сервер авторизації - сервер, що генерує унікальний ключ для кожного клієнта.
- Під час проходження автентифікації за допомогою протоколу OAuth виконуються наступні дії:
- Користувач вводить логін та пароль у клієнті;
- Клієнт надсилає ці дані до сервера;
- Сервер автентифікує користувача та повертає ключ доступу (token).
- Для доступу до ресурсів у HTTP запиті використовується token.

При виборі індивідуальних акаунтів в шаблоні проекту Web API, проект включає в себе сервер авторизації, який перевіряє облікові дані і видає ключі.

Базовий потік отримання ключа додатком:

3.5. Розробка та опис інтерфейсу користувача

Для front-end частини було використано HTML, CSS, Javascript ra архітектуру Single Page Application. Було також використано таблицю стилів Material, що знаходиться у відкритому доступі для розробників. Дизайн Material було розроблено компанією Google. Основним рушієм роботи з графічним інтерфейсом у програмному рішенні була платформа Angular.

Структура модулів графічного інтерфейсу користувача:

```
src/
├── app/
|   ├── core/
|   ├── shared/
|   ├── features/
```

```
|   |   ├── home/
|   |   |   ├── components/
|   |   |   ├── services/
|   |   |   ├── home.component.ts
|   |   |   ├── home.component.html
|   |   |   └── home.module.ts
|   |   ├── posts/
|   |   └── profile/
|   ├── models/
|   ├── services/
|   ├── guards/
|   ├── interceptors/
|   ├── app.component.ts
|   ├── app.config.ts
|   └── app.routes.ts
├── assets/
|   ├── img/
|   └── icons/
├── environments/
└── index.html
```

Для збірки і компіляції веб-клієнта було використано gulp.

Після входу на веб-портал, неавторизованому користувачу буде завантажено сторінку опису проекту. Для того, щоб увійти в систему потрібно натиснути кнопку Sign In, а для того, щоб зареєструватися Create Account додаток 3.

3.5. Опис альтернативних підходів, що розглядалися під час розробки

На етапі проектування для розробки системи також розглядалися й інші технології. Серед них можна виділити ASP.NET MVC з Razor Pages або Razor Views. Але було прийнто рішення, що використовуючи Razor могли б виникнути проблеми у роботі front-end, оскільки необхідно було б використовувати C# код всередині HTML, що принесло би труднощі з розширенням HTML клієнтських елементів управління, тестування та розділення обов’язків між фронтендом і бекендом.

Також аналізувався фреймворк Bootstrap як альтернатива Angular Material. Проте перевага була віддана Angular Material через кращу інтеграцію з Angular, сучасні компоненти та відповідність принципам Material Design від Google.

У результаті було обрано сучасний стек ASP.NET Core Web API + Angular, який на момент розробки та станом на 2026 рік залишається одним із найбільш ефективних і перспективних для створення масштабованих веб-додатків.

3.6. Інструкції щодо встановлення системи на IIS

Для розгортання веб-застосунку на локальному або віддаленому сервері під управлінням операційної системи Windows Server необхідно увімкнути Internet Information Services (IIS) – потужний веб-сервер Microsoft, що забезпечує високу продуктивність, надійність та гнучке керування.

Процес встановлення починається з активації компонента IIS. Для цього необхідно перейти у «Панель керування» обрати «Програми» обрати «Увімкнення або вимкнення компонентів Windows». У вікні, що відкриється, слід відмітити пункт «Internet Information Services» та всі необхідні підкомпоненти (зокрема, ASP.NET Core Hosting Bundle, Application Development Features тощо). Після внесення змін потрібно перезавантажити сервер. Основні параметри, які необхідно активувати, представлено на рис. 3.5.

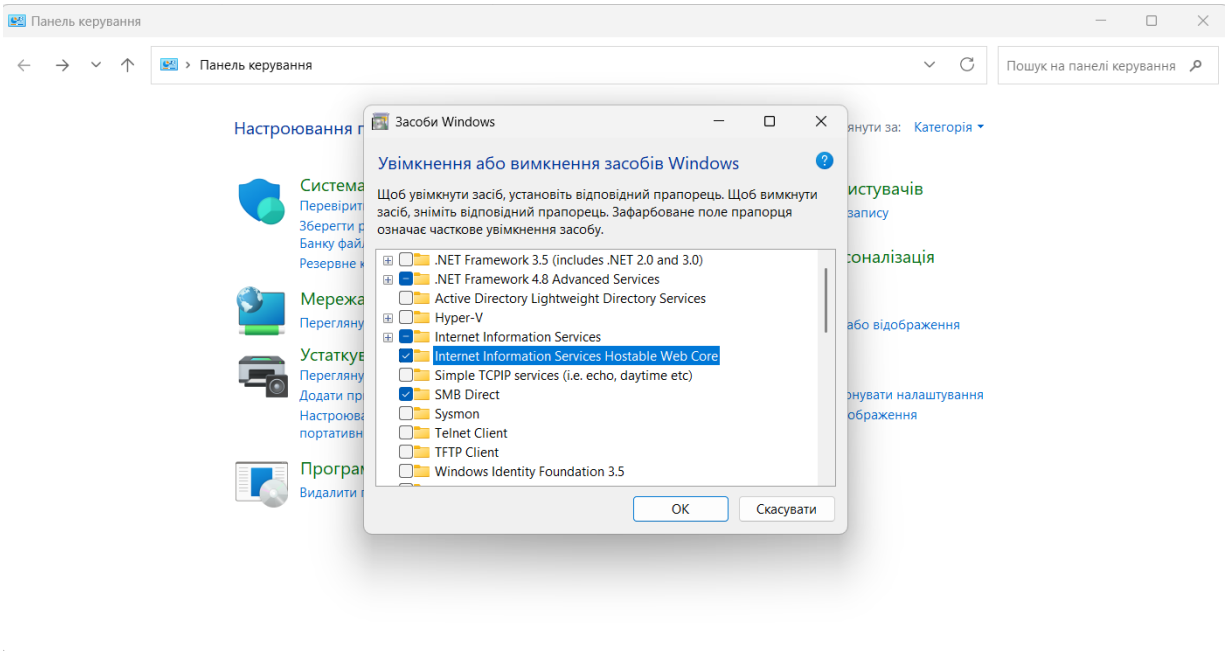


Рис. 3.5 Увімкнення IIS

Після перезавантаження слід запустити IIS Manager. У лівій панелі дерева об’єктів потрібно розгорнути вузол «Sites», клацнути правою кнопкою миші та обрати «Add Website». У діалоговому вікні вказуються: назва сайту, фізичний шлях до опублікованих файлів застосунку, порт (за замовчуванням 80 або 443 для HTTPS), а також пул застосунків (Application Pool) з налаштуваннями для ASP.NET Core.

3.7. Інструкції щодо розгортання системи на Azure

Для того, щоб розгорнути систему на Microsoft Azure необхідно перейти за посиланням https://portal.azure.com/ та створити веб-застосунок (рис. 3.6).

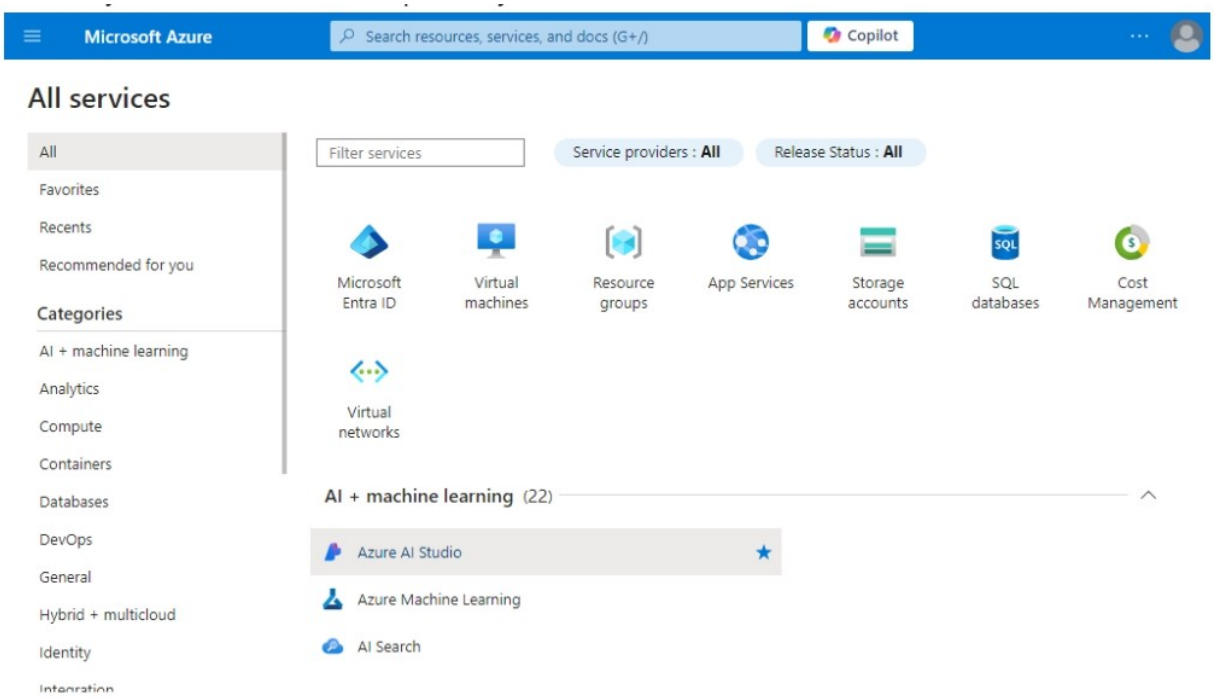


Рис. 3.6. Портал Azure

Після цього потрібно налаштувати Continuous Integration (CI) через Build Definition на порталі Azure DevOps. Це зроблено для того, щоб після кожного відвантаження коду на систему керування версіями відбувся деплой.

Для забезпечення безперервної інтеграції та розгортання (CI/CD) було налаштовано Azure DevOps Services. Створено Build Definition та Release Pipeline, які автоматично запускаються при кожному push-коді у репозиторій. Налаштування Continuous Integration. Такий підхід дозволяє автоматизувати процес тестування, збірки та розгортання, суттєво зменшуючи ймовірність помилок та прискорюючи вихід нових версій.

3.8. Тестування системи

Система була розроблена з використанням методології Test Driven Development, що передбачало написання тестів перед реалізацією функціональності.

Це дозволило забезпечити високу якість коду та мінімізувати кількість дефектів на етапі інтеграції.

Окрім модульних тестів (NUnit), було виконано комплексне автоматизоване тестування за допомогою інструментів Selenium WebDriver та Coded UI Tests. Основні цілі тестування включали:

- Виявлення функціональних помилок у ключових модулях системи;
- Перевірку стабільності роботи при різних рівнях мережевого навантаження та повільному з’єднанні;
- Тестування коректності автентифікації, авторизації та управління доступом;
- Перевірку роботи з медіафайлами (завантаження, зберігання, відображення);

Тестування інтерфейсу користувача на відповідність вимогам до зручності використання.

- Були протестовані такі основні компоненти:
- Модуль автентифікації та авторизації;
- Модуль створення, редагування та видалення публікацій;
- Модуль коментарів та вподобань;
- Система підписки та управління приватністю;
- Графічний інтерфейс користувача (responsive behavior, навігація, модальні вікна).

3.9. Робота з системою

Після розгортання застосунку користувач потрапляє на головну сторінку, де пропонується авторизуватися (рис 3.7). Підтримуються як локальна авторизація, так і вхід через зовнішні провайдери (Facebook, Google тощо).

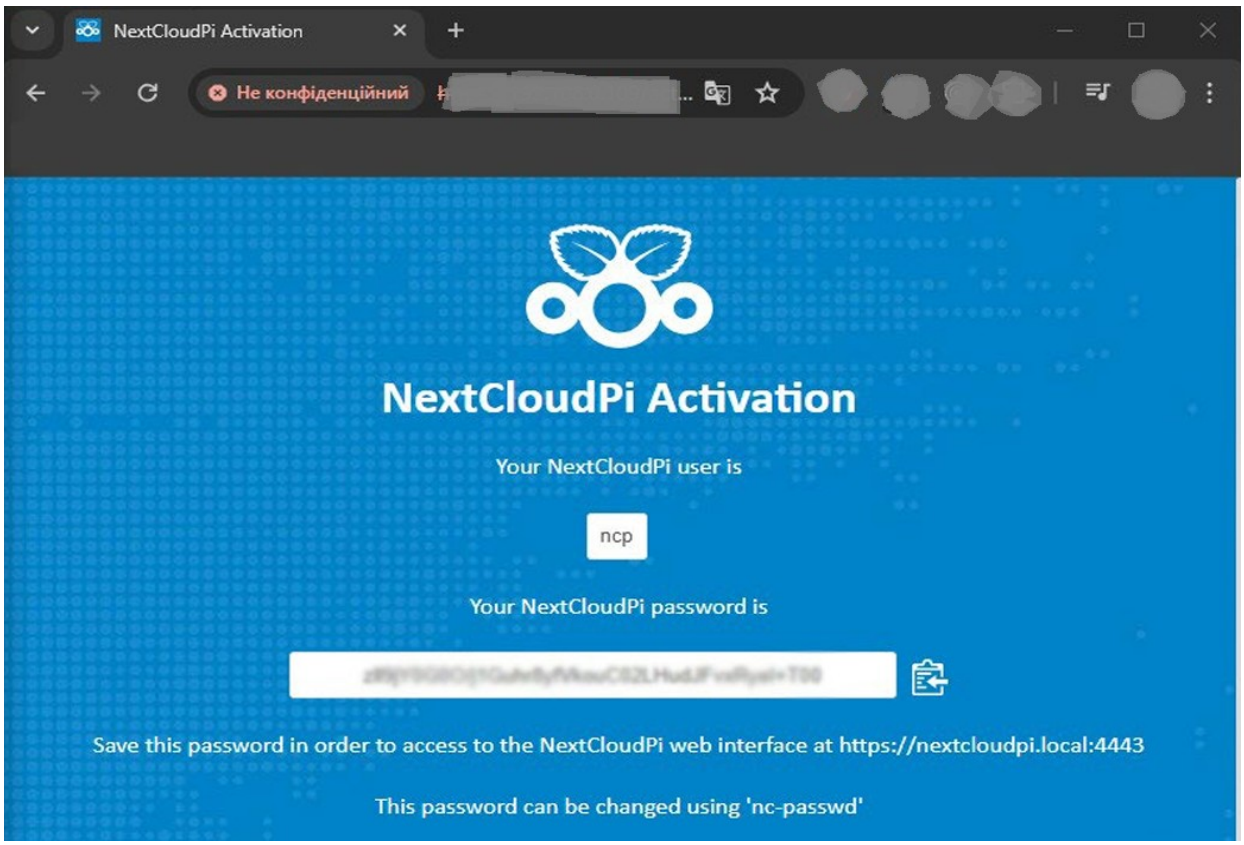


Рис. 3.7 Сторінка авторизації у системі

У випадку, якщо у користувача немає облікового запису, тоді його необхідно створити заповнивши усі необхідні поля на сторінці реєстрації

Після входження у систему користувачу буде відображено вітальну сторінку, яку зображено на рис. (рис 3.8).

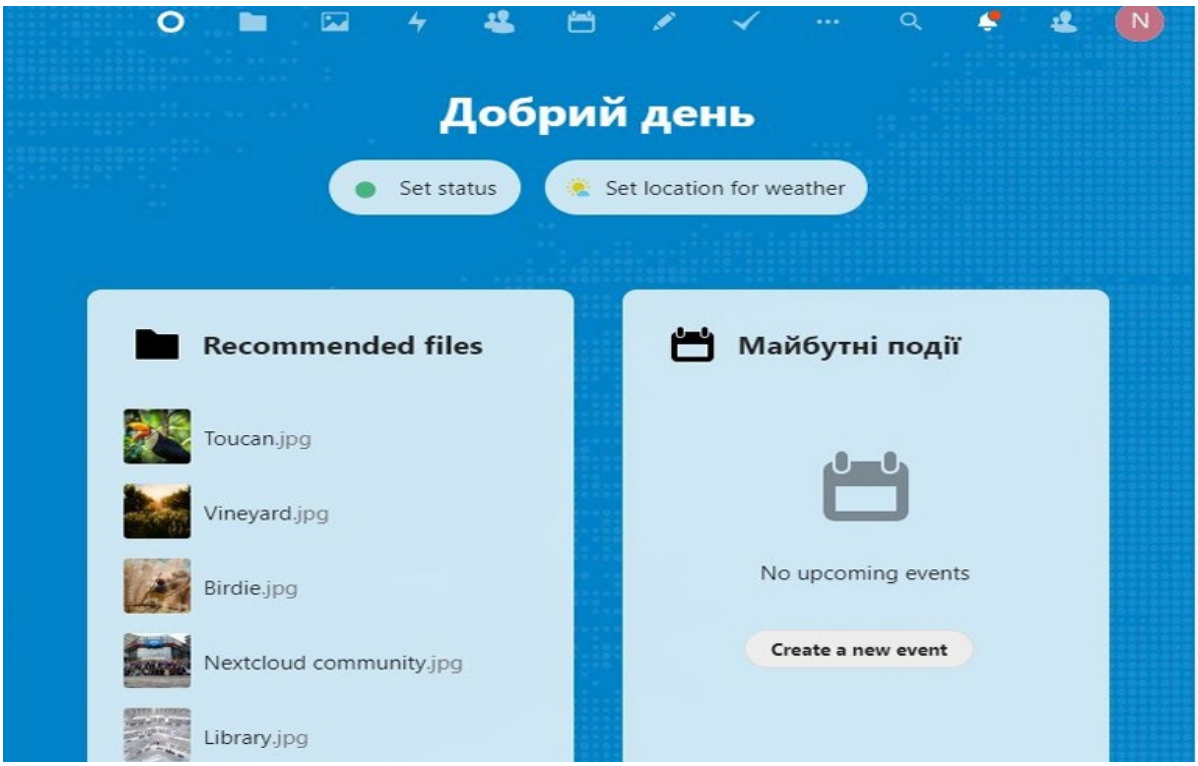


Рис. 3.8. Привітальна сторінка

Спробуємо з мобільного пристрою зайти на сторінку, та переглянути фото. Відкривши браузер, в полі пошуку, вводимо IP-адресу персонального хмарного сховища. Після цього авторизовуємось за допомогою логіну та паролю. На рис. 3.9. зображено вміст цієї сторінки.

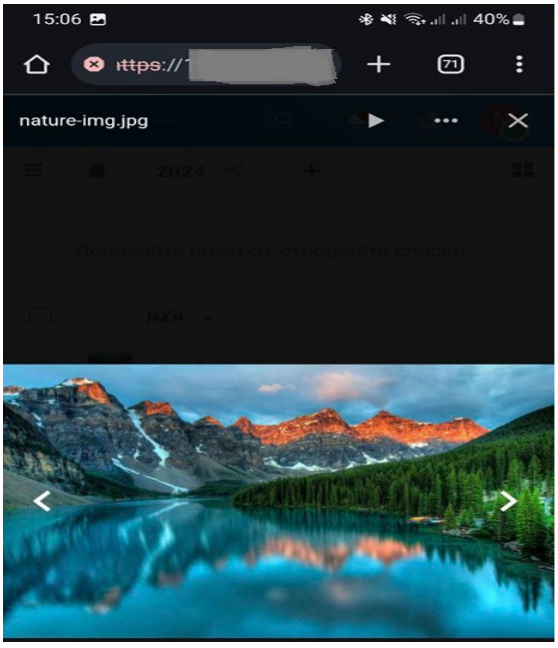


Рис. 3.9 Перегляд фотографії

3.10. Перспективи розвитку системи

Розроблена система побудована з дотриманням принципів об’єктно-орієнтованого дизайну (SOLID), багатошарової архітектури та сучасних практик програмної інженерії. Це забезпечує високу гнучкість та можливість подальшого розширення функціональності без суттєвої переробки існуючого коду.

Серед перспектив розвитку можна виділити:

- Розробку мобільних застосунків (iOS та Android) з використанням створеного Web API;
- Впровадження системи рекомендацій публікацій на основі машинного навчання;
- Додавання функцій реального часу (чати, живі трансляції) за допомогою SignalR;
- Розширення можливостей модерації та аналізу контенту;
- Інтеграцію з додатковими хмарними сервісами Azure (Cognitive Services, CDN тощо);
- Оптимізацію продуктивності та впровадження більш досконалих механізмів кешування.

Завдяки використанню хма рної платформи Microsoft Azure система готова до масштабування без втрати продуктивності.

ВИСНОВОКИ

У результаті виконання бакалаврської кваліфікаційної роботи проведено комплексне дослідження сучасних технологій розробки хмарних веб-застосунків, здійснено системний аналіз предметної області та успішно реалізовано веб-сервіс для зберігання та поширення зображень.

У першому розділі роботи виконано детальний огляд сучасних хмарних технологій, проаналізовано популярні сервіси зберігання зображень, досліджено ключові можливості та переваги платформи ASP.NET Core, технології ASP.NET Core Web API, принципів об’єктно-орієнтованого дизайну SOLID, архітектурного стилю REST, протоколу HTTP, формату JSON, фронтенд-фреймворку Angular, хмарної платформи Microsoft Azure, ORM-технології Entity Framework Core, механізмів автомасштабування та інших складових сучасної веб-розробки. Показано, що обраний технологічний стек є одним із найбільш перспективних для створення масштабованих, кросплатформенних і високопродуктивних веб-рішень.

У другому розділі здійснено системний аналіз предметної області, сформовано повний перелік функціональних та нефункціональних вимог до системи, визначено типи користувачів та їхні ролі, побудовано дерево цілей і дерево проблем, проаналізовано та обґрунтовано вибір багатошарової (N-Tier) архітектури проєкту. Детально описано засоби та інструменти розробки, що забезпечують високу якість коду, ефективну командну взаємодію та подальшу підтримку системи.

У третьому розділі представлено результати безпосередньої реалізації програмного рішення: розроблено концептуальну модель бази даних та її фізичну реалізацію в Azure SQL Database, спроєктовано та реалізовано основні програмні, створено сучасний клієнтський інтерфейс на платформі Angular з використанням компонентної архітектури та Material Design. Розроблено механізми завантаження, зберігання та управління медіафайлами, систему публікацій, коментарів, вподобань, підписки, управління

приватністю та блокуваннями. Здійснено розгортання системи в середовищі Microsoft Azure з використанням Azure Web Apps, Azure SQL Database, Azure Blob Storage та Azure DevOps для CI/CD.

Під час розробки та тестування виявлено та успішно вирішено низку технічних викликів, пов’язаних з безпекою, продуктивністю, управлінням доступом до приватного контенту. Комплексне тестування підтвердило коректність реалізації основних функціональних можливостей.

Розроблений веб-сервіс забезпечує зручний, інтуїтивний та responsive інтерфейс, надійну автентифікацію, гнучке управління приватністю, високий рівень безпеки та можливість подальшого масштабування. Використання хмарної платформи Microsoft Azure гарантує автоматичне масштабування ресурсів, високу доступність та ефективне управління витратами.

Таким чином, у ході виконання кваліфікаційної роботи повністю досягнуто поставленої мети та всіх визначених завдань. Створено конкурентоспроможний прототип сучасного хмарного сервісу, який відповідає актуальним вимогам ринку та може бути використаний як у приватному, так і в корпоративному/освітньому секторі.

Перспективи подальшого розвитку системи включають: розробку мобільних застосунків (iOS та Android), впровадження рекомендаційних систем на базі машинного навчання, інтеграцію сервісів реального часу (SignalR), розширення можливостей модерації контенту за допомогою Azure Cognitive Services, оптимізацію продуктивності та впровадження просунутих механізмів кешування й CDN.

Отримані результати свідчать про високий рівень володіння сучасними технологіями веб-розробки, хмарними рішеннями та принципами програмної інженерії, що підтверджує готовність автора до самостійної професійної діяльності у сфері інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Олексюк В. П., Спірін О. М. Основи хмарних технологій навчальний посібник. Київ: ІЦО НАПН України. 2023. 188 с.

2. Зінченко О. В., Іщеряков С. М., Прокопов С. В. та ін. Хмарні технології: навчальний посібник. URL: https://duikt.edu.ua/uploads/l_2048_32915773.pdf

3. Пасічник В. В., Пасічник О. В., Угрін Д. І. Веб-технології та веб-дизайн: підручник. Львів. Магнолія 2006, 2025. 368 с.

4. Коноваленко І. В. Платформа .NET та мова програмування C# 8.0: навчальний посібник. Тернопіль: ТНТУ. 2020. 322 с.

5. Добришин І. В., Жовтяк І. В. Розробка веб-застосувань ASP.NET MVC на платформі .NET Core. методичний посібник. Київ: Крок. 2019. 37 с.

6. Lock A. ASP.NET Core in Action. 3rd ed. Shelter Island: Manning Publications, 2023. 720 p.

7. Price M. J. C# 13 and .NET 9. Modern Cross-Platform Development Fundamentals. 9th ed. Birmingham. Packt Publishing. 2024. 828 p.

8. Freeman A. Pro Angular: Build Powerful and Dynamic Web Apps. 6th ed. New York: Apress, 2023. 752 p.

9. Andersson J. Learning Microsoft Azure. Cloud Computing and Development Fundamentals. New York: Apress, 2024. 456 p.

10. Lock A., Dykstra T. Entity Framework Core in Action. 2nd ed. Shelter Island: Manning Publications, 2021. 592 p.

11. Erl T., Monroy E. B. Cloud Computing: Concepts, Technology, and Architecture. 2nd ed. Boston: Pearson, 2023. 608 p.

12. Microsoft Docs. ASP.NET Core documentation Microsoft, 2026. URL: https://learn.microsoft.com/en-us/aspnet/core/

13. Microsoft Docs. Entity Framework Core documentation . Microsoft, 2026. URL: https://learn.microsoft.com/en-us/ef/core/

14. Microsoft Docs. Azure Architecture Center Microsoft, 2026. URL: https://learn.microsoft.com/en-us/azure/architecture/

15. Кодекс М. Дж. Реальний світ веб-розробки з .NET 9. Birmingham: Packt Publishing, 2025. 680 с.

16. Хроленко В. М. Хмарні та GRID-технології: підручник. Запоріжжя: ЗНУ, 2025. 100 с.

17. Мухін В. Є. та ін. Хмарні технології: навчальний посібник. Київ: ДУІКТ, 2020. 74 с.

18. Code Maze Team. Ultimate ASP.NET Core Web API – Second Edition. Code Maze, 2024. URL: https://courses.code-maze.com/courses/ultimate-aspnetcore-webapi/

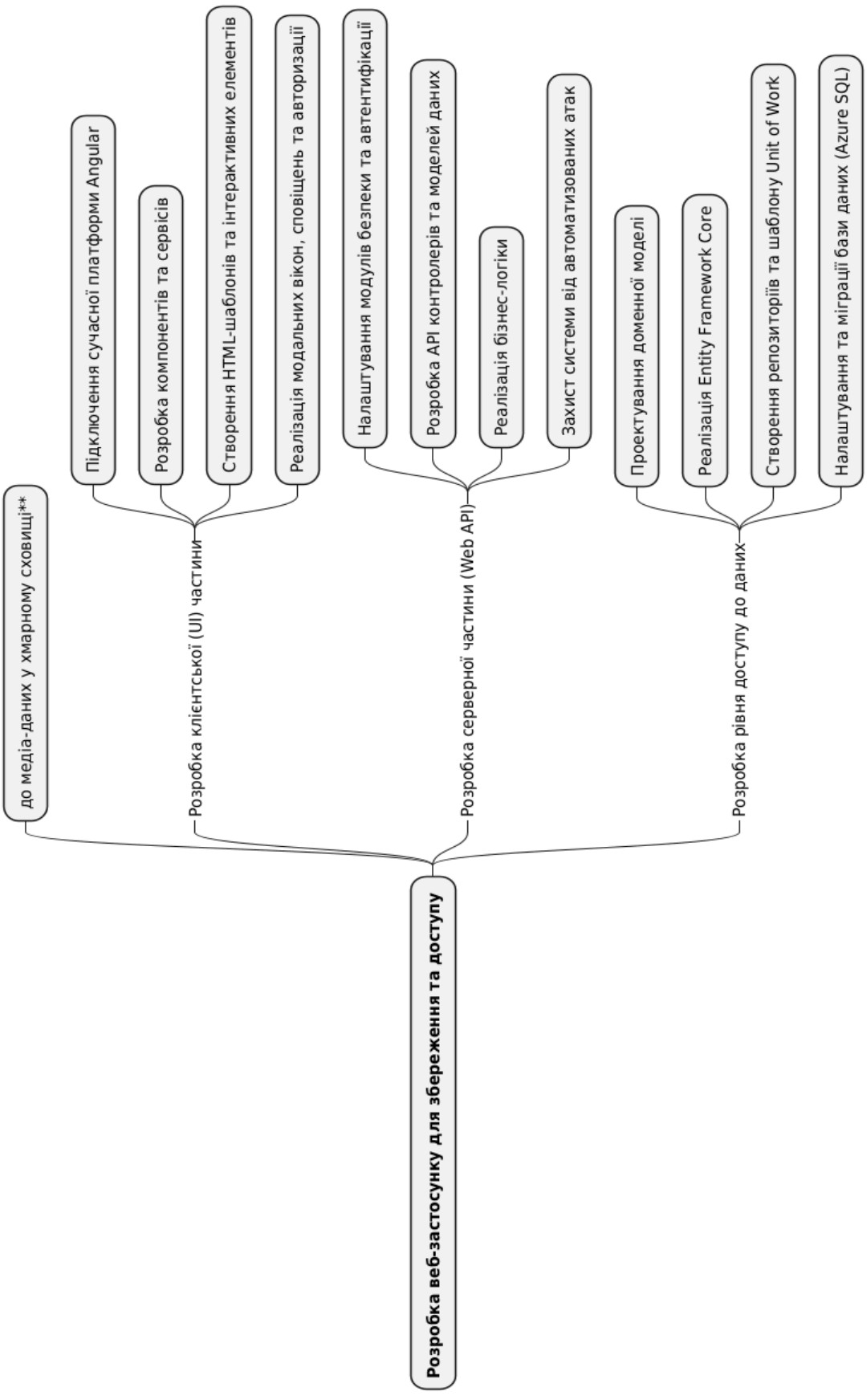
19. Koskela L. Test Driven Development. Manning Publications, 2010. 300 p.

20. Freeman A. Expert ASP.NET Web API 2 for MVC Developers. Apress, 2014. 480 p.

ДОДАТКИ

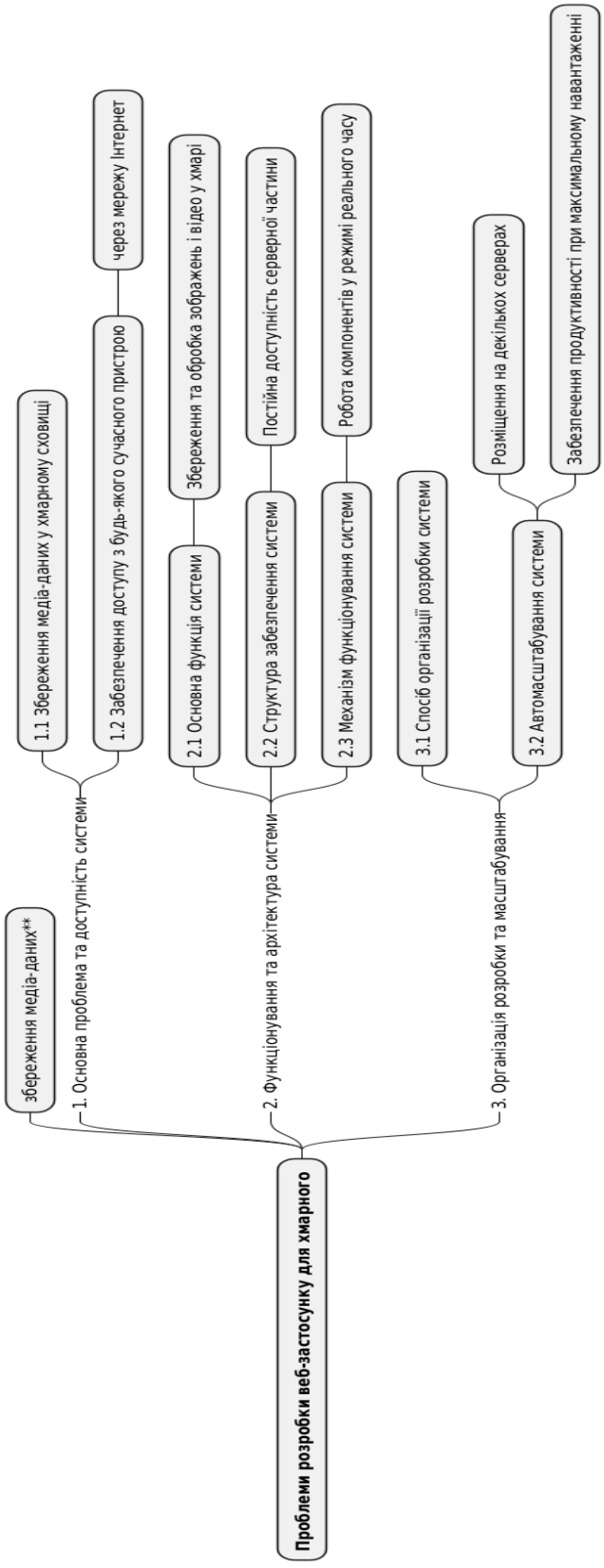
ДОДАТОК 1.

Дерево цілей



ДОДАТОК 2

Структура дерева проблем



ДОДАТОК 3.

Код клієнтського додатка

```
const gulp = require('gulp');
const HubRegistry = require('gulp-hub');
```

```
const browserSync = require('browser-sync').create();
const conf = require('./conf/gulp.conf');

// Load all tasks from gulp_tasks folder
const hub = new HubRegistry([conf.path.tasks('*.js')]);
gulp.registry(hub);

gulp.task('inject', gulp.series(
    gulp.parallel('styles', 'scripts'),
    'inject'
));

gulp.task('build', gulp.series('partials', gulp.parallel('inject', 'other'),
'build'));
gulp.task('serve', gulp.series('inject', 'watch', 'browsersync'));
gulp.task('serve:dist', gulp.series('default', 'browsersync:dist'));
gulp.task('default', gulp.series('clean', 'build'));

// Watch task
function watch(done) {
    gulp.watch([conf.path.src('index.html'), 'bower.json'],
gulp.parallel('inject'));
    gulp.watch(conf.path.src('app/**/*.html'), gulp.series('partials',
reloadBrowserSync));
    gulp.watch(conf.path.src('**/*.css'), gulp.series('styles'));
    gulp.watch(conf.path.src('**/*.js'), gulp.series('inject'));
    done();
}

function reloadBrowserSync(cb) {
    browserSync.reload();
    cb();
}

gulp.task('watch', watch);
node:
```

conf/gulp.conf.js

```
const path = require('path');
const gutil = require('gulp-util');
```

```
exports.ngModule = 'photocloud';

exports.paths = {
    src: 'src',
    dist: 'dist',
    tmp: '.tmp',
    tasks: 'gulp_tasks'
};

exports.path = {};
for (const pathName in exports.paths) {
    if (Object.prototype.hasOwnProperty.call(exports.paths, pathName)) {
        exports.path[pathName] = function () {
            const pathValue = exports.paths[pathName];
            const funcArgs = Array.prototype.slice.call(arguments);
            const joinArgs = [pathValue].concat(funcArgs);
            return path.join.apply(this, joinArgs);
        };
    }
}

exports.htmlmin = {
    ignoreCustomFragments: [/{{.*?}}/]
};

exports.errorHandler = function (title) {
    return function (err) {
        gutil.log(gutil.colors.red('[' + title + ']'), err.toString());
        this.emit('end');
    };
};

exports.wiredep = {
    exclude: [/bootstrap\.js$/],
    directory: 'bower_components'
};
```

gulp_tasks/styles.js

```
const gulp = require('gulp');
const browserSync = require('browser-sync');
const sourcemaps = require('gulp-sourcemaps');
const postcss = require('gulp-postcss');
const autoprefixer = require('autoprefixer');

const conf = require('../conf/gulp.conf');

gulp.task('styles', function styles() {
    return gulp.src(conf.path.src('**/*.css'))
        .pipe(sourcemaps.init())
        .pipe(postcss([autoprefixer()]))
        .on('error', conf.errorHandler('Autoprefixer'))
        .pipe(sourcemaps.write())
        .pipe(gulp.dest(conf.path.tmp()))
        .pipe(browserSync.stream());
});
```

gulp_tasks/scripts.js

```
const gulp = require('gulp');
const eslint = require('gulp-eslint');

const conf = require('../conf/gulp.conf');

gulp.task('scripts', function scripts() {
    return gulp.src(conf.path.src('**/*.js'))
        // .pipe(eslint())
        // .pipe(eslint.format())
        .pipe(gulp.dest(conf.path.tmp()));
});
```

gulp_tasks/partials.js

```
const gulp = require('gulp');
const htmlmin = require('gulp-htmlmin');
const angularTemplatecache = require('gulp-angular-templatecache');

const conf = require('../conf/gulp.conf');

gulp.task('partials', function partials() {
    return gulp.src(conf.path.src('app/**/*.html'))
```

```
        .pipe(htmlmin(conf.htmlmin))
        .pipe(angularTemplatecache('templateCacheHtml.js', {
            module: conf.ngModule,
            root: 'app'
        }))
        .pipe(gulp.dest(conf.path.tmp()));
});
```