Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Дробот Анни Ярославівни*
(ПІБ)

академічної групи *121-18-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка веб-додатку крос-платформеної системи пошуку зображень з використанням технології ASP.NET.*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-18-2* *Дробот Анни Ярославівни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатку крос-платформеної системи пошуку зображень з використанням технології ASP.NET.*

затверджена наказом ректора НТУ «ДП» від №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***Наі основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Дробот А. Я.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 92 с., 35 рис., 3 дод., 20 джерел.

Об'єкт розробки: веб-додаток крос-платформеної системи пошуку зображень з використанням технології ASP.NET.

Мета кваліфікаційної роботи: Розробка веб-додатку крос-платформеної системи пошуку зображень з використанням технології ASP.NET.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці інформаційної системи, що дозволяє користувачам знаходити необхідний товар в одному місці з багатьох магазинів, порівнювати його характеристики відносно інших товарів, відслідковувати динаміку зміни ціни на протязі певного періоду часу, та обирати найвигіднішу пропозицію, яку пропонують інтернет магазини.

Актуальність програмного продукту визначається великим ростом популярності електронної комерції, а особливо сфери продаж, та тим, що сьогодні багато людей намагаються заощаджити час та кошти і данна система допомагатиме користувачам в цьому.

Список ключових слів: ВЕБ, ВЕБ-ДОДАТОК, ПРОГРАМА, ПОШУК ЗОБРАЖЕНЬ, КРОС-ПЛАТФОРМЕНІСТЬ, БРАУЗЕР, КЛІЄНТ, ІНФОРМАЦІЙНА СИСТЕМА.

ABSTRACT

Explanatory note: 92 p., 35 figs., 3 appx., 20 sources.

Object of development: web application for a cross-platform image search system using ASP.NET technology.

Purpose of the qualification work: Development of a web application for a cross-platform image search system using ASP.NET technology.

In the introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiated the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development is carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work of the program is described.

In the economic section it is determined the complexity of the developed software product, calculated the cost of work to create an application and calculated the time to create it.

Of practical importance is the development of an information system that allows users to find the desired product in one place from many stores, compare its characteristics with other products, track the dynamics of price changes over time, and choose the best offer offered, which is offered by online stores.

The relevance of the software product is determined by the growing popularity of e-commerce, and especially sales, and the fact that many people today are trying to save time and money and this system will help users in this.

Keywords: WEB, WEB-APPLICATION, PROGRAM, IMAGE SEARCH, CROSS PLATFORM, BROWSER, CLIENT, INFORMATION SYSTEM.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ЕК – електронна комерція;

БД – база даних;

JSON – JavaScript Object Notation, текстовий формат обміну данним

ІС – інформаційна система;

CSS – Cascading Style Sheets ;

ОС – операційна система;

ПК – персональний комп’ютер;

ПЗ – програмне забезпечення;

ІТ – інформаційні технології.

ВСТУП

Тематика даної кваліфікаційної роботи присвячена проектуванню та розробці веб-додатку крос-платформенної системи пошуку зображень з використанням технології ASP.NET.

У теперішній час великою популярністю користуються веб-сервіси та соціальні мережі, що зосереджені навколо медіа-контенту, зокрема зображень.

Важливою спільною рисою цих сервісів є легкість знаходження контенту, що з найбільшою ймовірністю буде відповідати інтересам користувача. Користувачам подобається бачити нові зображення, що відповідають їх естетичним смакам, без необхідності вручну робити складний пошук по базі даних, а просто “скролячи” веб-додаток. Іншою важливою стороною функціоналу є можливість переглядати історію власних взаємодій з дописами.

Багато типових аспектів соціальних мереж, які не зосереджені саме на медіа-контенті, є зазвичай другорядними у подібних сервісах. Зазвичай може бути відсутня або досить обмежена будь-яка взаємодія між окремими користувачами. Користувачі виступають в ролі глядачів, пасивних споживачів контенту.

Результатом виконання даної роботи є інформаційна система, що дозволяє користувачу знаходити та зберігати у колекцію медіа-контент, який йому сподобався просто з естетичної точки зору або містив якусь корисну інформацію.

Завдання даного дипломного проекту та об'єкт його діяльності безпосередньо пов'язані з напрямом підготовки «Інженерія програмного забезпечення» та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної області

В даний час існує багато різних веб-сервісів, орієнтованих на пасивний пошук користувачем різного медіа-контенту.

Ці сервіси можуть бути імплементовані на найрізноманітніших стеках технологій і типах архітектур. Серед деяких типових рішень можна назвати наступні:

- розділений фронт-енд і бек-енд

Фронт-енд — це частина веб-додатку, яка імплементує графічний інтерфейс користувача.

Бек-енд — це частина веб-додатку, яка здійснює операції з даними і обробляє запити.

Зазвичай це розділення існує на дуже масштабних ентерпрайз-проектах з великою командою розробників, серед яких є ті, що спеціалізуються відповідно на фронт-енді та бек-енді.

Фронт-енд в таких рішеннях найчастіше імплементований з використанням таких популярних Javascript-фреймфорків, як React.js, Vue.js або Angular.

Бек-енд може бути імплементований із застосуванням будь-яких мов програмування, за умови узгодженості формату запитів з фронт-ендом, але зазвичай стандартом формату запитів є REST. Популярними мовами імплементації бек-енду є C#, Java, PHP, C++, Ruby, Python.

Бек-енд зазвичай використовує реляційні бази даних. Серед найуживаніших технологій є MS SQL Server, MySql, Oracle Database, PostgreSQL.

- рішення, побудоване на основі фронт-енду

Такий тип рішень більше підходить для невеликих команд розробників і проектів, які не вимагають складної моделі даних. Додатковою задачею фронт-енду є також деплоймент серверного додатку, який буде обробляти запити і звертатися до бази даних.

Одним з найширше використовуваних є сервер на основі Node.js, який також надає можливість взаємодії з базою даних, зазвичай документо-орієнтованими.

- рішення, побудоване на основі бек-енду

Цей тип рішень також використовується серед невеликих команд розробників, але більше підходить для проектів, що вимагають складної моделі реляційної бази даних.

Популярними архітектурними паттернами є MVC, MVVM, MVP та ін. Ці архітектурні паттерни описують структуру з трьох основних компонентів:

- View – інтерфейс користувача, який наповнюється отриманими даними з бази. Відповідає front-end частині додатку.
- Model – об’єкт з даними, які наповнюють View та які можуть бути отримані в результаті відправлення форми користувачем.
- Controller – рівень сервісу, який обробляє вхідні дані та генерує дані у відповідь. Відповідає за рівень бізнес-логіки та взаємодії з базою даних.

Серед технологій, що можуть використовуватися у даних рішеннях, є платформа ASP.NET, Java, PHP тощо.

У даній кваліфікаційній роботі розглянуто рішення з використанням технології ASP.NET з Razor Pages.

Технологія .NET була вперше представлена компанією Microsoft у жовтні 1999 року і з того часу пройшла великий шлях розвитку, ставши спочатку основним середовищем виконання будь-яких процесів операційної системи Windows, а зрештою — крос-плаформенною технологією, що може виконуватися також і у операційних системах Linux і macOS. .NET не є віртуальною машиною, подібною до віртуальної машини Java, натомість ця

технологія має більшу інтеграцію з системним API цільової операційної системи, що значно покращує перформанс і надає можливість дебаггінгу.

ASP.NET є частиною технології .NET, яка дозволяє створювати веб-додатки з різною архітектурою і цільовим призначенням. Сайти, створені за допомогою технології ASP.NET є динамічними і вихідний код їх фронт-енд частини генерується засобами платформи, після наповнення певної веб-сторінки контентом.

Razor Pages – це гібридна мова синтаксису, що поєднує стандартний HTML-код і C#-код. Назва вихідного формату текстового файлу з кодом (.cshtml) демонструє це поєднання. Razor Pages дозволяє створювати шаблони веб-сторінок, які будуть динамічно наповнюватись способом, описаним логікою наявного C#-коду.

Іншою особливістю є можливість використання Tag Helpers для розширення стандартного функціоналу HTML-тегів. Зокрема, Anchor Tag Helper, який є атрибутом HTML-тегу з назвою, що починається на “asp-”, дозволяє створити логіку роутингу і передачі даних у додатку.

Передача даних у View, який представлений Razor-сторінкою, відбувається за допомогою моделі окремого Data Transfer Object. У межах однієї веб-сторінки можна використовувати декілька інтерфейсів DTO, за умови, що переданий екземпляр об’єкту буде реалізовувати їх всі.

Razor Pages також дозволяють використовувати Partial Views, які самі не є самостійною завершеною веб-сторінкою, а є лише частинами інших веб-сторінок. Partial Views мають свої власні моделі DTO, які формуються і наповнюються у основному View, з якого викликається асинхронний метод рендеру цього Partial View.

Ендпоїнти у технології ASP.NET розглядаються як окремі методи, що належать до певного контролеру. Формат посилання на окремий ендпоїнт зазвичай наступний — назва контролеру / назва методу. Параметри передаються після знаку “?” і розділяються знаком “&”. У одному контролері не можуть бути два метода-ендпоїнта з однаковою назвою, навіть якщо вони

отримують різні DTO у параметрах. Аналогічно, контролери також мають мати унікальні назви.

Запит, що приходить з веб-клієнта на ендпоїнт, та відповідь, що формується сервером, автоматично, можливостями платформи ASP.NET, набувають необхідного формату, описаного стандартом REST.

REST, Representational State Transfer – прийнятий архітектурний паттерн взаємодії клієнта і веб-сервера, який зокрема описує допустимі формати запитів, кешування, протоколів HTTP і очікуваної поведінки серверу.

Збереження медіа-контенту можливе локальне, з використанням емулятору хмарного сховища Azurite, який має абсолютно ідентичний консольний інтерфейс, як і хмарне сховище Azure.

1.2. Призначення розробки та область застосування

Як об’єкт розробки інформаційної системи пошуку зображень розглядається веб-додаток, в якому користувач може знаходити нові зображення, опціонально реагувати на них позитивно чи негативно, переглядати і редагувати свої реакції. Сервіс містить велику базу даних об’єктів-зображень, які описуються деякими об’єктами-тегами за допомогою зв’язку багато-до-багатьох, дані про користувача, які поєднані за допомогою окремого об’єкту — реакції — з об’єктами зображень.

Функціонал, доступний суперадміну:

- Створення нових користувачів, які можуть бути звичайними юзерами, або едіторами — користувачами, які відповідають за додавання і модерацію контенту.
- Пошук користувачів у системі, зміна їх інформації, заборона або дозвіл доступу до сервісу.
- Редагування власного акаунту.

Також суперадміну доступний функціонал користувача-едітора:

- Створення нових тегів з унікальними назвами.
- Пошук тегів, їх редагування, активація або деактивація.
- Додання нових зображень.
- Пошук зображень, редагування їх інформації, додання або видалення пов’язаних тегів, активація для доступу на сервісі та приховування їх.
- Перегляд зображень, починаючи з останніх, що мають певний тег.
- Пошук зображень, назва яких або один з тегів містить певний текст.
- Редагування власного акаунту.

Функціонал, доступний звичайному користувачу:

- Перегляд нових зображень, які сервіс знайшов на основі даних про попередні взаємодії користувача з зображеннями, або випадковим чином, якщо таких взаємодій ще не було або їх було недостатньо.

- Реакція на зображення — лайк або дизлайк — або відсутність реакції і завантаження наступного зображення.
- Перегляд власних лайків та дизлайків з можливістю відмінити або змінити реакцію.
- Перегляд лайкнутих зображень, відсортованих по тегам.
- Перегляд зображень, починаючи з останніх, що мають певний тег.
- Пошук зображень, назва яких або один з тегів містить певний текст.

Структура даного веб-сервісу складається з наступних веб-сторінок:

- Сторінка логіну.
- Сторінка реєстрації.
- Меню суперадміна.
- Меню управління едіторами, що містить можливість додання нового едітора, зміни інформації про наявних едіторів і їх пошуку.
- Меню управління юзерами, що містить можливість додання нового юзера, зміни інформації про наявних юзерів і їх пошуку.
- Меню едітора.
- Меню управління тегами, що містить можливість додання нового тегу, зміни інформації про наявні теги і їх пошук.
- Список зображень, що містять певний тег, з можливістю пошуку текстового паттерну, що міститься у назві зображення або у будь-якому його тегу, та можливістю прибрати тег з певного зображення.
- Вікно перегляду зображення, що також містить його назву і список тегів, без можливості реакції.
- Меню управління зображеннями, що містить можливість пошуку існуючих зображень по текстовому паттерну, який міститься у їхній назві або будь-якому з їхніх тегів, а також можливість зміни назви зображення, активації чи деактивації і редагування тегів, що пов’язані з певним зображенням.
- Сторінка додання нового зображення, з можливістю також вказати певну назву і обрати відповідні теги.

- Вікно перегляду зображення, що містить його назву і список тегів, з можливістю реакції або опції завантаження наступного зображення.
- Вікно пошуку зображень по текстовому паттерну, що міститься у їх назві або у будь-якому з тегів цього зображення.
- Сторінка з списком пролайканих зображень, з можливістю зміни реакції.
- Сторінка з списком продизлайканих зображень, з можливістю зміни реакції.
- Сторінка перегляду колекцій-тегів з пролайканими зображеннями.
- Сторінка перегляду пролайканих зображень, що відносяться до певного тегу, з можливістю зміни реакції.
- Сторінка редагування власної інформації.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу. Тема роботи узгоджується з керівником дипломного проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки «виконання кваліфікаційної роботи» є:

- освітня програма 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка»;
- завдання на кваліфікаційну роботу на тему «Проектування та розробка веб-додатку кросс-платформенної системи пошуку зображень з використанням технології ASP.NET».

1.4. Постановка завдання

Завданням цього дипломного проекту є проектування та розробка веб-додатку кросс-платформенної системи пошуку зображень з використанням технології ASP.NET. Програмне забезпечення призначене для надання універсального інструменту для відображення контенту бази зображень та описових тегів.

Програма повинна реалізувати наступні функції:

- формування web-сторінок з використанням контенту з серверної частини застосунку;
- зберігання і опрацьовування реакцій користувачів.
- можливість редагування інформації у систему.

Для досягнення поставленої мети необхідно:

- вивчити предметну область розв'язуваної задачі;
- створити алгоритм для реалізації поставленого завдання;
- створити базу даних і клієнтську та серверну програми, що працюють з нею.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставлених цілей програмне забезпечення, що розробляється, повинно підтримувати виконання наступних дій:

- надання віддаленого доступу до застосунку через веб-браузер на комп'ютері або мобільному пристрої користувача;
- зчитування вхідних даних з пристроїв, хмарних сервісів, за адресними посиланнями;
- зберігання даних системи в реляційній базі даних.
- для виконання перерахованих вище функцій у застосунку повинні бути реалізовані:
- можливість отримати доступ до програми через веб-браузер;
- наявність типової конфігурації, що забезпечує можливість швидкого введення застосунку в експлуатацію;
- програмно-апаратна переносимість.

1.5.2 Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програми необхідно реалізувати:

- семантичний та синтаксичний контроль вхідних даних;
- обробку виняткових ситуацій;
- виведення повідомлень про помилки;
- можливість повторного введення даних;
- можливість безперервної роботи протягом не менше 24 годин (1 доба);
- платформну незалежність.

1.5.3 Вимоги до складу та параметрів технічних засобів

Для нормального функціонування даного застосунку необхідно, щоб серверний комп’ютер, на якому, буде функціонувати інформаційної система, відповідала наступним вимогам:

- процесор класу Intel i7 2 покоління з тактовою частотою не менш 3.6 ГГц та 2 ядрами;
- доступ до мережі Internet;
- не менше 16 GB оперативної пам'яті;
- 1TB вільного місця на жорсткому диску;
- клавіатура;
- маніпулятор "миша".

Вимоги до клієнтського персонального комп’ютеру:

- процесор класу Intel Pentium з тактовою частотою не менш 2.4 ГГц та двума ядрами;
- доступ до мережі Internet;
- не менше 8 GB оперативної пам'яті;
- 3 GB вільного місця на жорсткому диску;
- клавіатура;
- маніпулятор "миша".

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний застосунок буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутих замовником.

1.5.4 Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно, щоб програмне забезпечення персонального комп’ютера або мобільного пристрою, на якому буде функціонувати веб-орієнтована система, відповідало наступним вимогам:

- операційна система Windows (7+) / Linux / MacOs;
- веб-браузер Google Chrome / Opera / Safari/ Firefox / Microsoft Edge.

Застосунок має бути реалізовано на мові програмування C# з використанням технологій ASP.NET і Razor Pages, локального сховища Azurite і бази даних MS SQL.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом виконання даної кваліфікаційної роботи є веб-додаток крос-платформеної системи пошуку зображень. Крос-платформенність проявляється як зі сторони серверного середовища (можливість виконання середовища .NET на операційних системах Microsoft Windows, macOS, Linux), так і зі сторони клієнту, оскільки клієнтський інтерфейс є веб-інтерфейсом, що доступний для відображення з будь-якого сучасного браузеру.

Головна задача даного додатку — це видача користувачу зображень за допомогою сформованих рекомендацій на основі історії попередніх взаємодій користувача з іншими зображеннями.

У додатку представлені наступні ролі:

- рівень доступу «Суперадмін» («SuperAdmin»);
- рівень доступу «Едітор» («Editor»);
- рівень доступу «Юзер» («User»).

Користувач з рівнем доступу «Суперадмін» («SuperAdmin») має доступ до наступного функціоналу:

- перегляд наявних у системі едіторів;
- зміна даних наявних у системі едіторів;
- пошук наявних у системі едіторів;
- додавання у систему нових едіторів;
- перегляд наявних у системі юзерів;
- зміна даних наявних у системі юзерів;
- пошук наявних у системі юзерів;
- додавання у систему нових юзерів.

Доступ до наступного функціоналу мають користувачі з рівнем доступу «Суперадмін» («SuperAdmin») та «Едітор» («Editor»):

- перегляд наявних у системі тегів зображень;
- зміна даних наявних у системі тегів зображень;
- пошук наявних у систему тегів зображень;
- додавання у систему нових тегів зображень;
- перегляд зображень, які пов’язані зв’язком “багато-до-багатьох” з певним тегом зображень;
- зміна статусу зв’язка “багато-до-багатьох” між зображенням і його тегом;
- пошук зображень, які пов’язані зв’язком “багато-до-багатьох” з певним тегом зображень;
- перегляд наявних у системі зображень;
- зміна даних наявних у системі зображень;
- пошук наявних у систему зображень;
- додавання у систему нових зображень;
- перегляд тегів, які пов’язані зв’язком “багато-до-багатьох” з певним зображенням;
- пошук тегів, які пов’язані зв’язком “багато-до-багатьох” з певним зображенням.

Користувач з рівнем доступу «Юзер» («User») має доступ до наступного функціоналу:

- реєстрація у системі;
- перегляд нового зображення і його додаткових даних, що було знайдене системою;
- пошук зображень, чия назва містить певний текст, та зображень, назва одного з чиїх тегів містить певний текст;
- пошук зображень з певним тегом;
- перегляд зображень з власною позитивною або негативною реакцією;
- зміна реакції на протилежну або скасування реакції;
- перегляд зображень з позитивною реакцією, відсортованих по тегам;
- можливість пошуку зображень з комбінованими умовами.

Доступ до наступного функціоналу мають користувачі з усіма рівнями доступу:

- вхід до власного акаунту;
- перегляд та зміна власної інформації;
- вихід з власного акаунту.

2.2. Опис застосованих математичних методів

Під час проектування та розробки даної інформаційної системи використовувалися лише прості арифметичні дії. Математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проектування

Інформаційна система реалізована за допомогою паттерну MVC.

Паттерн MVC, або Model-View-Controller, описує архітектуру веб-додатку з трьома найголовнішими компонентами:

- Model або модель – об’єкт, який може передаватися як від контролеру у представлення, так і у зворотному напрямку. У першому випадку такий об’єкт містить дані, які мають бути відображені на веб-сторінці засобами HTML, CSS та JavaScript. У другому випадку містять дані запиту, які мають бути опрацьовані контролером. Модель є фактично посередником між контролером, який здійснює опрацювання даних з використанням бази даних та опціональних інших сервісів, та представленням, яке є графічним інтерфейсом користувача;
- View або представлення — веб-сторінка, з якою взаємодії кінцевий клієнт, та яка наповнюється даними, отриманими з моделі. Представлення відрізняється від класичної HTML-сторінки тим, що воно будується на сервері по частинах і клієнт отримує вже готовий результат, який не є статичним файлом на сервері. Відправлення даних з представлення можливе за допомогою HTML-форм або за допомогою посилань з необхідними додатковими параметрами.
- Controller або контролер — найголовніший елемент паттерну MVC, який здійснює опрацювання отриманих даних та генерацію нових даних у відповідь. Контролер отримує дані як параметри методу певного ендпоїнта, після чого відбувається опрацювання даних. Контролер є своєрідним шлюзом до глибокої інфраструктури бекенду, яка має свою власну архітектуру. Взаємодія з іншими сервісами та базою даних відбувається засобами, що вже специфічні для вищезгаданої інфраструктури, і не має суттєвий вплив на сам контролер, оскільки це різні елементи системи.

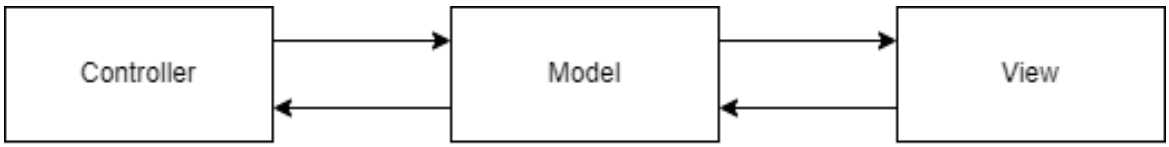


Рис. 1. – Графічне представлення шаблону MVC

Серед переваг паттерну MVC можна зазначити наступні:

- прозорість структури класу моделі у представленні;
- динамічна генерація контенту представлення;
- швидкість опрацювання даних контролером.

Серед недоліків найголовнішими є наступні:

- довга генерація контенту представлення;
- підвищене навантаження на сервер.

У даній інформаційній системі контролер взаємодіє з глибинною інфраструктурою за допомогою механізму ін’єкції залежностей.

В рамках середовища ASP.NET контролери розглядаються як спеціальний тим сервісів. Поведінка, що спільна для сервісів усіх типів — це можливість отримати посилання на інший сервіс у момент виклику конструктору класу при надходженні нового запиту. Таким чином, контролер отримує посилання на необхідні сервіси, методи яких викликаються при обробці запитів.

Сервіси, що виконують обробку запитів, аналогічним образом отримують посилання на репозиторії, які також є сервісами. Репозиторії виконують всю роботу, пов’язану з базою даних або файловим сховищем системи. Репозиторії отримують посилання на сервіс контексту бази даних та інші додаткові сервіси — у даній інформаційній системі таким прикладом є сервіс доступу до хмарного файлового сховища Azure (або емулятор Azurite, який має абсолютно ідентичний консольний інтерфейс).

2.4. Опис використаних технологій та мов програмування

При розробці даної інформаційної системи були використані наступні технології:

- C#;
- .NET;
- ASP.NET;
- ASP.NET MVC;
- Razor Pages;
- HTML;
- CSS;
- JavaScript;
- jQuery;
- Azure CLI;
- Azurite;
- MS SQL.

C# є однієї з найпопулярніших на даних момент мов програмування, яка активно розвивається і підтримується компанією Microsoft. Для розробки програмних застосунків за допомогою цієї мови існує велика кількість різних фреймворків та платформ.

C# є мультипарадигменною мовою програмування, хоча основними парадигмами, з яких почалася історія розвитку даної мови, є об'єктно-орієнтоване програмування та сильна типізація даних. З мов, які безпосередньо вплинули на створення C# можна назвати C++ і Java. Окремі парадигми, які не є домінуючими у цій мові, такі, як функціональне програмування та окремі випадки слабої або динамічної типізації, з'явилися під впливом мов Haskell та інших.

.NET є на сьогоднішній момент основним виконавчим середовищем операційної системи Microsoft Windows та окрім цього має реалізації на інших операційних системах, зокрема Linux та macOS. Важливою перевагою

платформи .NET є спільне середовище виконання програм написаних на мовах, сумісних з Common Language Runtime. Це дозволяє використовувати одночасно модулі і компоненти, написані різними мовами, що дозволяє у обирати технології таким чином, щоб прискорювати виконання програм, підвищувати швидкість розробки та покращувати аспекти кібербезпеки програмних застосунків.

ASP.NET – це крос-платформений веб-фреймворк, у основі якого знаходиться технологія .NET, як свідчить його назва. Цей фреймворк може використовуватись у розробці API та бекенду будь-яких веб-застосунків. Широке використання цей фреймворк набув завдяки високій швидкості обробки запитів, високого ступеню захисту даних та інтеграції з багатьма сервісами Microsoft та хмарними сервісами інших провайдерів, таких як Amazon.

ASP.NET MVC – це технологія, реалізована за допомогою ASP.NET, яка імплементує базову архітектуру паттерну MVC з нуля. Саме ASP.NET MVC додає власну реалізацію функціональних можливостей взаємодії між трьома основними компонентами цього паттерну. Найпростіша програма, майже пустий шаблон, що реалізує таку технологію, дуже схожа на таку саму шаблонну пусту програму, написану тільки з використанням ASP.NET. Можна сказати, що ASP.NET MVC є надмножиною над чистим ASP.NET, оскільки не вимагає позбавлення або модифікації якихось компонентів своєї основної технології.

Razor Pages є однією з найголовніших частин технології ASP.NET MVC. Саме Razor Pages дозволяють створювати динамічно згенеровані веб-сторінки. У цієї технології є окремий синтаксис, який поєднує код мовами C# та код стандартних HTML-сторінок, які також можуть містити CSS або JavaScript код.

HTML – це мова розмітки усіх веб-сторінок, хоча у наш час ця мова набула використання і за межами веб-додатків. Усі елементи веб-сторінки є тегами, які поєднані у спільну структуру — DOM-дерево.

Мова стилів CSS дозволяє модифікувати зовнішній вигляд HTML сторінки і навіть додати нову логіку відображення елементів та взаємодії з ними.

JavaScript є мовою програмування, що широко використовується у веб-інтерфейсах і дозволяє додавати більш складну логіку взаємодії з користувачем, а також робити самостійні запити, відправляти та отримувати дані. JavaScript є інтерпретованою мовою програмування, що має динамічну типізацію. У контексті ASP.NET MVC додатків JavaScript використовується головним чином для візуальних ефектів та інших не критично важливих функціональностей фронтенду.

jQuery є популярним та широковживаним JavaScript фреймворком, який за допомогою свого спрощеного синтаксису полегшує доступ до певних елементів DOM-дерева.

Azure CLI (Command-Line Interface) — це мова запитів до хмарного сервісу Azure. Оскільки це мова, що лише описує запити у вигляді команд, файли коду можуть бути будь-якими скриптами, що здатні виконуватись на певній операційній системі: .bat, .cmd, .sh тощо.

Azurite є емулятором хмарного середовища Azure, що має абсолютно ідентичний консольний інтерфейс та логіку зберігання файлів у контейнерах. Цей емулятор доцільно використовувати під час розробки додатків, що мають використовувати хмарне сховище Azure. З точки зору коду самого додатку не існує різниці між емулятором на справжнім сховищем.

MS SQL – це імплементація мови запитів до бази даних SQL для серверу бази даних, розробленого компанією Microsoft. Дозволяє оперувати додавати, модифікувати та видаляти бази даних, їхні таблиці, дані у тих таблицях та інші об’єкти бази даних, такі як: процедури, функції, представлення, користувачі, рівні доступу та ін. Разом з популярною технологією Entity Framework є одним з найбільш уживаних варіантів реалізації зберігання не-файлових даних у інформаційних системах різних типів. Дуже швидкий пошук і доступ до необхідних даних імплементований за допомогою індексів записів у таблицях, що зберігаються у окремій кластерній структурі даних.

2.5. Опис структури програми та алгоритмів її функціонування

Сутності, наявні у системі, представлені на наступній схемі бази даних (Рис. 2):

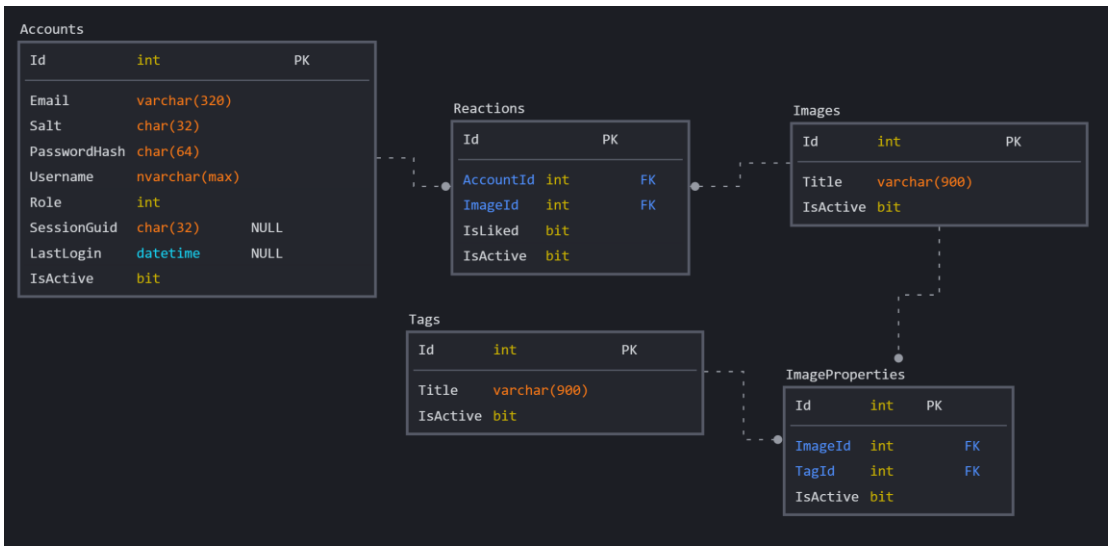


Рис. 2. – Схема бази даних

У базі наявні сутності 5 типів:

- Акаунти;
- Теги;
- Зображення;
- Реакції;
- Властивості зображень.

Запис про кожного зареєстрованого користувача системи міститься в таблиці Акаунтів.

Первинний ключ акаунта — його унікальний ідентифікатор. Перше значення такого ідентифікатора у кожній таблиці дорівнює 1 і збільшується з шагом 1.

Кожний акаунт обов’язково має містити електронну адресу, що є унікальною.

Дані про пароль акаунта не зберігаються у відкритому вигляді, бо це несе у собі критичні кібербезпекові ризики. Натомість зберігаються два значення, які отримані при шифруванні отриманого від користувача пароля при його реєстрації або при його ручному створенні, алгоритмом шифрування SHA256

(його реалізація HMACSHA256 була використана у даному випадку) — сіль та хеш паролю. Сіль — це випадкова величина розміром у 16 байтів, яка є унікальною для кожного зашифрованого значення. Сіль для зручності зберігається у базі як HEX-строка довжиною 32 байти. Хеш паролю — це результат роботи функції хешування, аргументами якої являються вхідне значення паролю та сіль. Цей хеш має довжину 256 бітів (що відображено у назві алгоритму шифрування), або 32 байти. Для зручності хеш також зберігається як HEX-строка довжиною 64 байти. Функція хешування незворотня, тобто знаючи тільки сіль і хеш неможливо відтворити оригінальний пароль. Тому при кожній спробі залогуватися в систему додаток обчислює хеш отриманого від користувача значення, використовуючи значення солі для акаунту зі значенням електронної пошти рівному тому, яке прийшло у запиту. Успішна авторизація користувача наступає за обов'язкової умови збігу цих двох хешів.

При успішній авторизації або успішному створенні нового юзера генерується унікальний ідентифікатор його сесії, який зберігається у cookie-файлах клієнта протягом доби. Цей ідентифікатор також зберігається у базі і регулярно оновлюється при кожному візиті користувача. При надсиланні будь-якого запиту на сервер відбувається перевірка наявності і валідності cookie-файлу з даним ідентифікатором. При неуспішній спробі перевірки даний користувач не може використовувати функціонал даного веб-додатку і відбувається редірект на сторінку входу в систему.

Окремо зберігаються дані про час останнього логіну у систему користувача. Ці дані слугують для додаткової перевірки віку cookie-файлу з унікальним ідентифікатором сесії.

Роль користувача також зберігається у базі. Роль може бути лише одного з 3 типів — Суперадмін, Едітор або Юзер.

Також у кожної сутності у системі є поле бітове IsActive, яке слугує маркером того, чи дозволено використовувати певний запис у будь-якому запиті, крім окремих випадків, коли у відповіді мають бути абсолютно усі

записи. За замовчуванням це поле має значення true. У контексті використання для сутності Акаунту це поле має означати, чи дозволено певному користувачу отримувати доступ до системи. Для зображення це значення означає, чи буде показуватись це зображення юзерам та чи можна буде його знайти під час пошуку. Для тегу це значення означає, чи буде цей тег видний у списку тегів для зображення і чи можна буде знаходити зображення, що мають зв’язок з цим тегом, за умови, що назва цього тегу містить текст пошуку. Для властивості зображення це значення означає, чи активний зв’язок певного тегу з певним зображенням. Для реакції це значення означає, чи вважається зображення таким, що користувач його вже бачив і прореагував на нього.

Центральною сутністю даної системи є сутність зображення. Крім спільних для всіх типів сутностей полів унікального ідентифікатора і маркеру активного статусу, зображення має єдине поле — його назва. У контейнері хмарного сховища зображення також зберігаються за своїм унікальним ідентифікатором у базі даних.

Другою найголовнішою сутністю є сутність тега. Тег має абсолютно ідентичну структуру до структури сутності зображення.

Зображення і теги пов’язані між собою зв’язком “багато-до-багатьох” за допомогою лінковочної таблиці, яка створює додаткову сутність під назвою Властивість зображення. Крім пари вторинних ключів (унікальний ідентифікатор зображення і унікальний ідентифікатор тегу) яка є додатковим унікальним ключем до первинного ключа сутності, ця сутність також має маркер активного статусу.

Аналогічним образом побудована сутність Реакція — унікальна пара вторинних ключів у даному разі виступає парою унікального ідентифікатора користувача і унікального ідентифікатору зображення. Маркер активного статусу відображає, чи бачив і реагував цей користувач на певне зображення.

Далі наведені діаграми класів, які описують частини інформаційної системи.

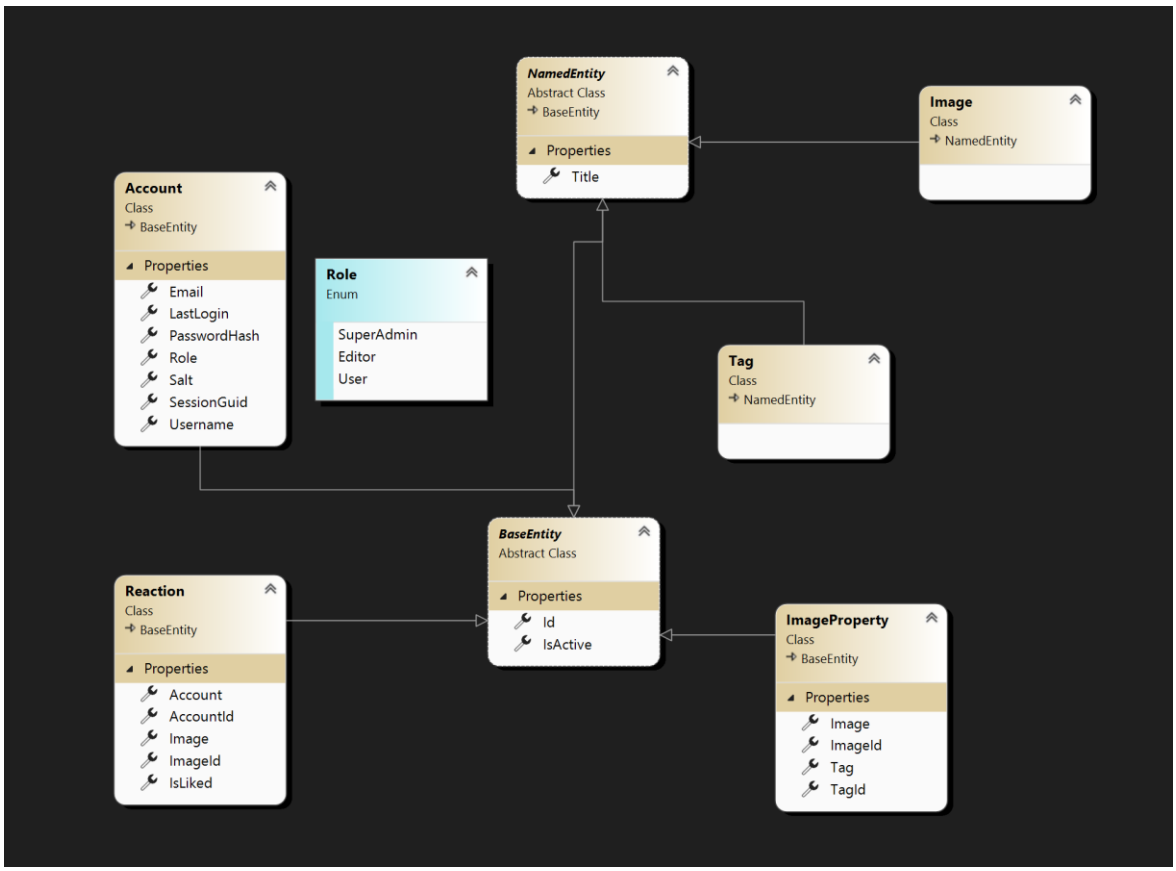


Рис. 3. – Діаграма класів сутностей.

Діаграма класів на Рис. 3 відображає зв’язки між класами, що описують наявні сутності у базі даних. Додатково були додані два абстрактних класи – BaseEntity і NamedEntity, які описують спільні поля для наявних сутностей.

Серед класів і інтерфейсів, які не є специфічними саме для цієї області даного додатку, є абстрактний клас FailableRequest, який дозволяє додати повідомлення про помилку у разі, якщо запит не був успішно виконаний, інтерфейс ISearchable, який дозволяє надати функціонал пошуку, і інтерфейс IManageEntitiesModel, який є моделлю у частковому відображенні, де надається інформація про загальну кількість певних стуностей у базі та кількість сутностей з активним статусом.

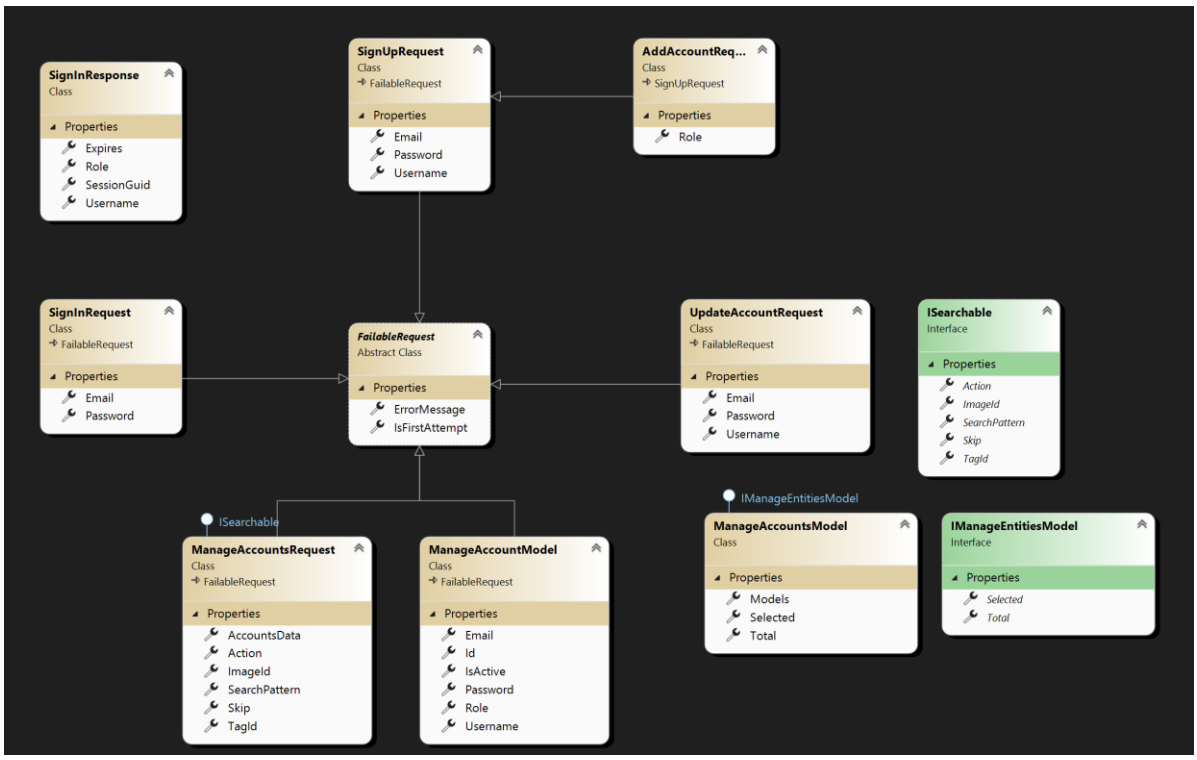
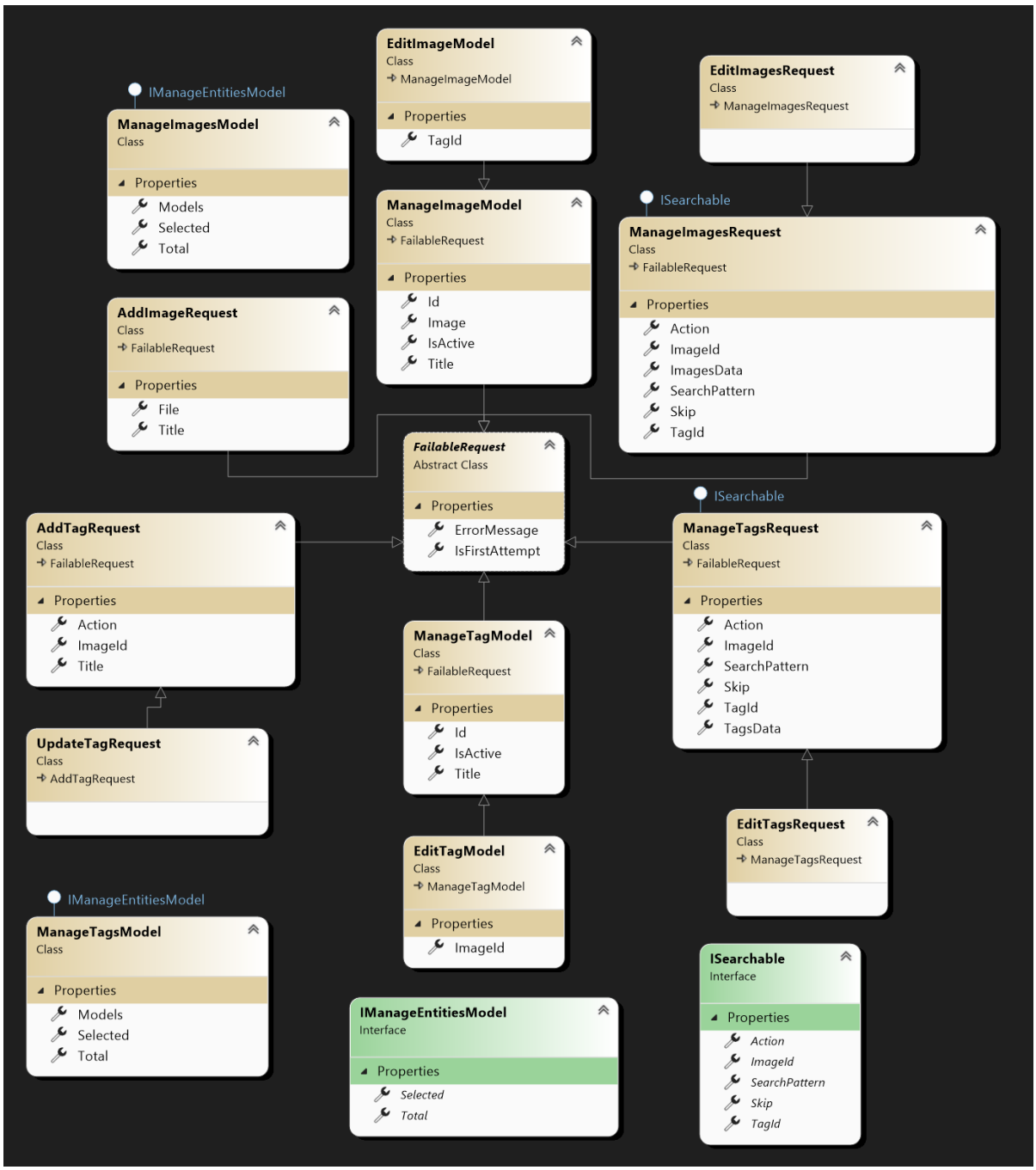


Рис. 4. – Діаграма класів моделей, пов’язаних з контролером акаунтів.

Діаграма класів на Рис. 4 описує усі моделі, які використовуються у ендпоїнтах контролеру, пов’язаного з опрацюванням даних акаунтів. Абстрактний клас FailableRequest, а також інтерфейси ISearchable та IManageEntitiesModel також присутні на діаграмі.

Діаграма класів на Рис. 5 описує моделі, які використовуються у ендпоїнтах контролеру, пов’язаного з опрацюванням даних медіа, що призначені для управління даними користувачами з правами Суперадміну або Едітора. На цій діаграмі також присутні абстрактний клас FailableRequest, інтерфейси ISearchable і IManageEntitiesModel.

Діаграма класів на Рис. 6 описує моделі, які використовуються у ендпоїтах контролеру, пов’язаного з опрацюванням даних медіа, що призначені для відображення даних зображення для юзеру.


IManageEntitiesModel
ManageImagesModel
Class
Properties
Models
Selected
Total
EditImageModel
Class
ManageImageModel
Properties
TagId
EditImagesRequest
Class
ManageImagesRequest
ISearchable
ManageImageModel
Class
FailableRequest
Properties
Id
Image
IsActive
Title
ManageImagesRequest
Class
FailableRequest
Properties
Action
ImageId
ImagesData
SearchPattern
Skip
TagId
AddImageRequest
Class
FailableRequest
Properties
File
Title
FailableRequest
Abstract Class
Properties
ErrorMessage
IsFirstAttempt
ISearchable
AddTagRequest
Class
FailableRequest
Properties
Action
ImageId
Title
ManageTagsRequest
Class
FailableRequest
Properties
Action
ImageId
SearchPattern
Skip
TagId
TagsData
ManageTagModel
Class
FailableRequest
Properties
Id
IsActive
Title
UpdateTagRequest
Class
AddTagRequest
EditTagsRequest
Class
ManageTagsRequest
EditTagModel
Class
ManageTagModel
Properties
ImageId
IManageEntitiesModel
ManageTagsModel
Class
Properties
Models
Selected
Total
IManageEntitiesModel
Interface
Properties
Selected
Total
ISearchable
Interface
Properties
Action
ImageId
SearchPattern
Skip
TagId

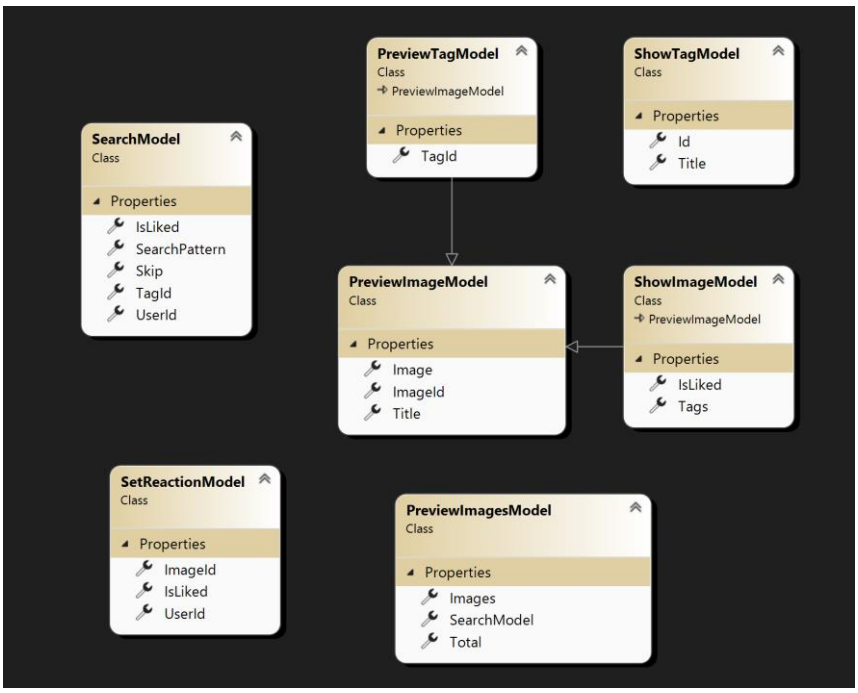


Рис. 5. – Діаграма класів моделей, пов’язаних з контролером медіа, що використовуються для управління даними.

Рис. 6. – Діаграма класів моделей, пов’язаних з контролером медіа, що використовуються відображення даних для юзера.

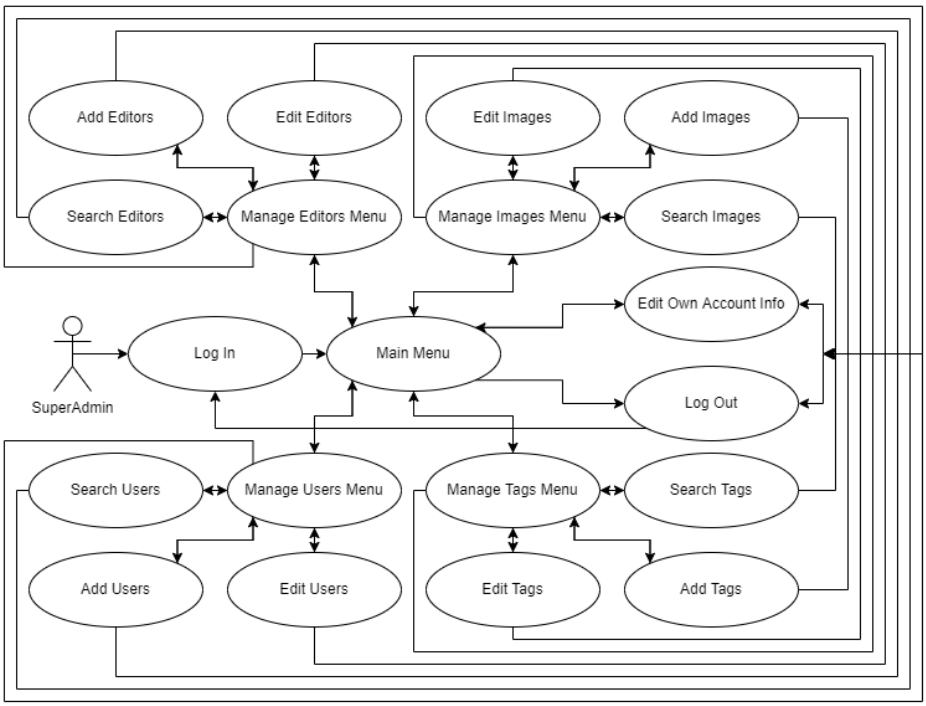


Рис. 7. – Діаграма варіантів використання для Суперадміну.

Діаграма варіантів використання на Рис. 7 зображує можливі сценарії використання додатку для Суперадміну.

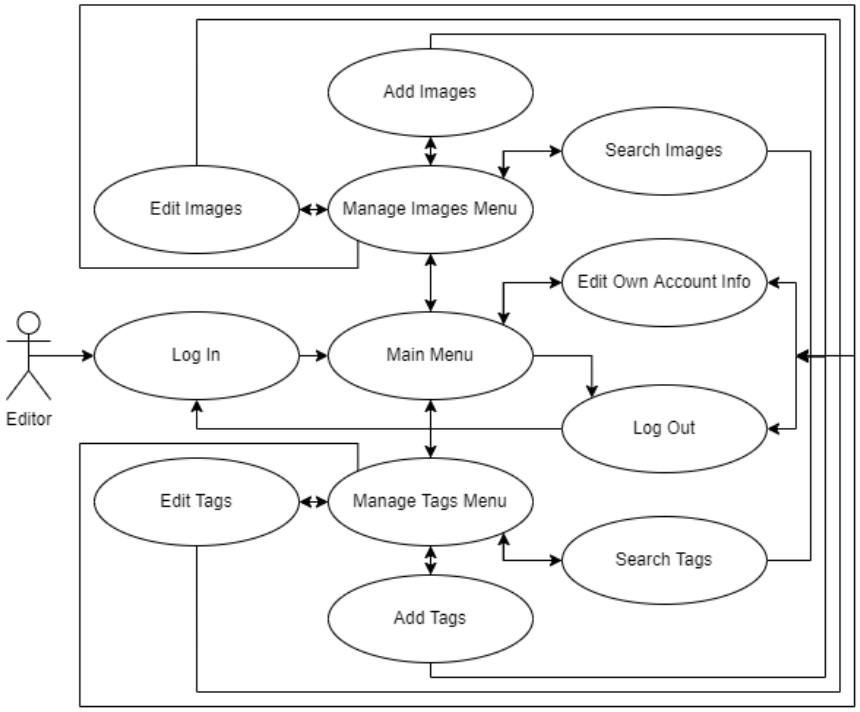


Рис. 8. – Діаграма варіантів використання для Едітору.

Діаграма варіантів використання на Рис. 8 зображує можливі сценарії використання додатку для Едітору.

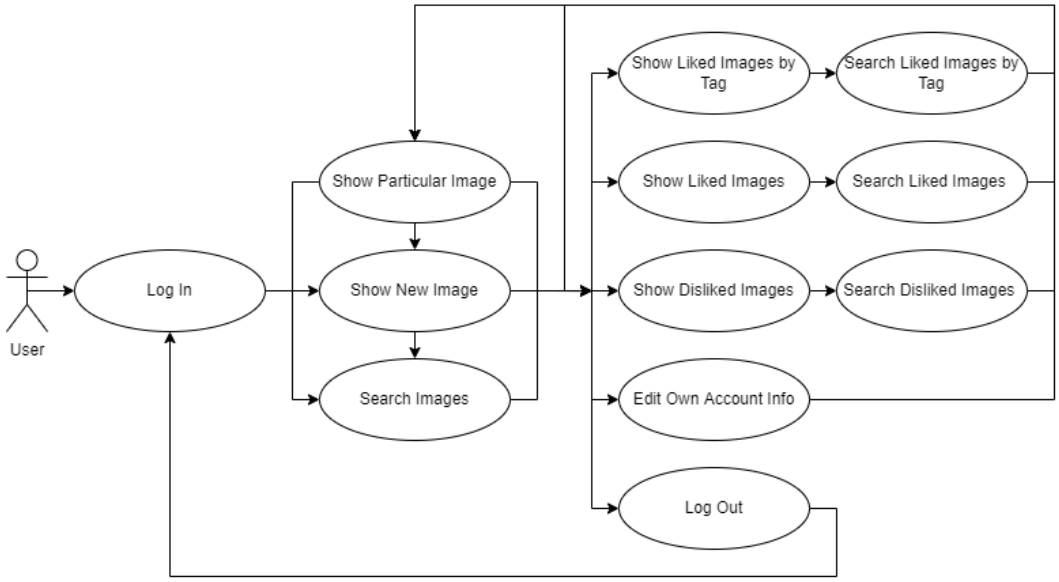


Рис. 9. – Діаграма варіантів використання для Юзеру.

Діаграма варіантів використання на Рис. 9 зображує можливі сценарії використання додатку для Юзеру.

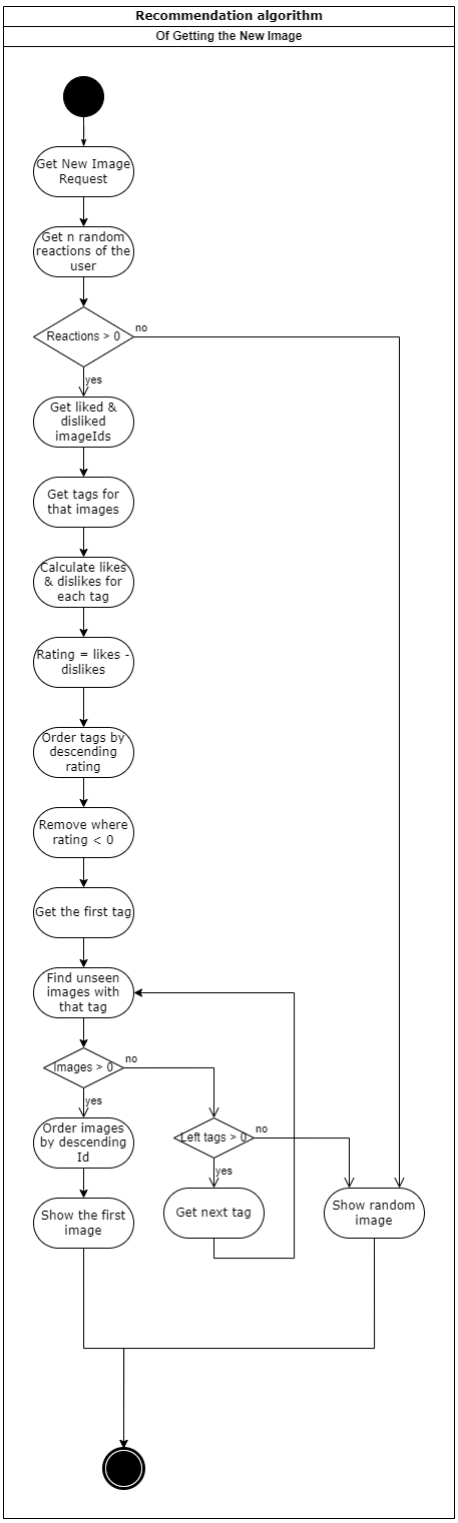


Рис. 10. – Діаграма активності пошуку нового зображення.

Діаграма активності на Рис. 10 зображує алгоритм пошуку нового зображення при отриманні відповідного запиту (цей запит відбувається при логіні юзера або при натисканні кнопки генерації нового зображення).

Спочатку знаходиться n (у реалізації даної системи n = 100) довільних реакцій даного користувача на зображення. Якщо даних про реакції у базі даних немає для цього користувача, то виводиться довільне зображення.

При наявності даних про реакції формується два списки унікальних ідентифікаторів зображень — перший список з ідентифікаторами зображень з позитивною реакцію, інший список — з негативною.

Наступним кроком формуються два списки з унікальними ідентифікаторами тегів, які мають зв’язок з зображеннями з двох списків. Таким чином, формуються список тегів, які асоціюються з позивною реакцію, і список тегів, що асоціюються з негативною реакцію. Ідентифікатор одного і того самого тега може повторюватись кілька разів всередині одного списку, а також бути одночасно присутнім у обох списках. Потім відбувається підраховування кількість появ кожного тегу в кожному з двох списків. Кількість появ тегу у списку, що асоціюється з позитивною реакцією, визначає кількість лайків тегу. Аналогічним образом визначається кількість дизлайків тегу.

Рейтинг тегу визначається різницею між кількістю лайків і дизлайків. Теги поєднуються у спільний список з рейтингом, звідки виключаються всі теги з від’ємним рейтингом. Теги сортуються за рейтингом від більшого до меншого. Теги перебираються у циклі, для кожного шукаються зображення, які користувач ще не бачив (тобто не має активної реакції з ними). Якщо для певного тегу таких зображень немає, то черга переходить до наступного тегу у циклі. Як тільки знаходиться зображення, яке користувач ще не бачив, воно відправляється у відповідь. Якщо жодного зображення не було знайдено, відправляється у відповідь довільне зображення, включно з тими, які користувач вже між бачити.

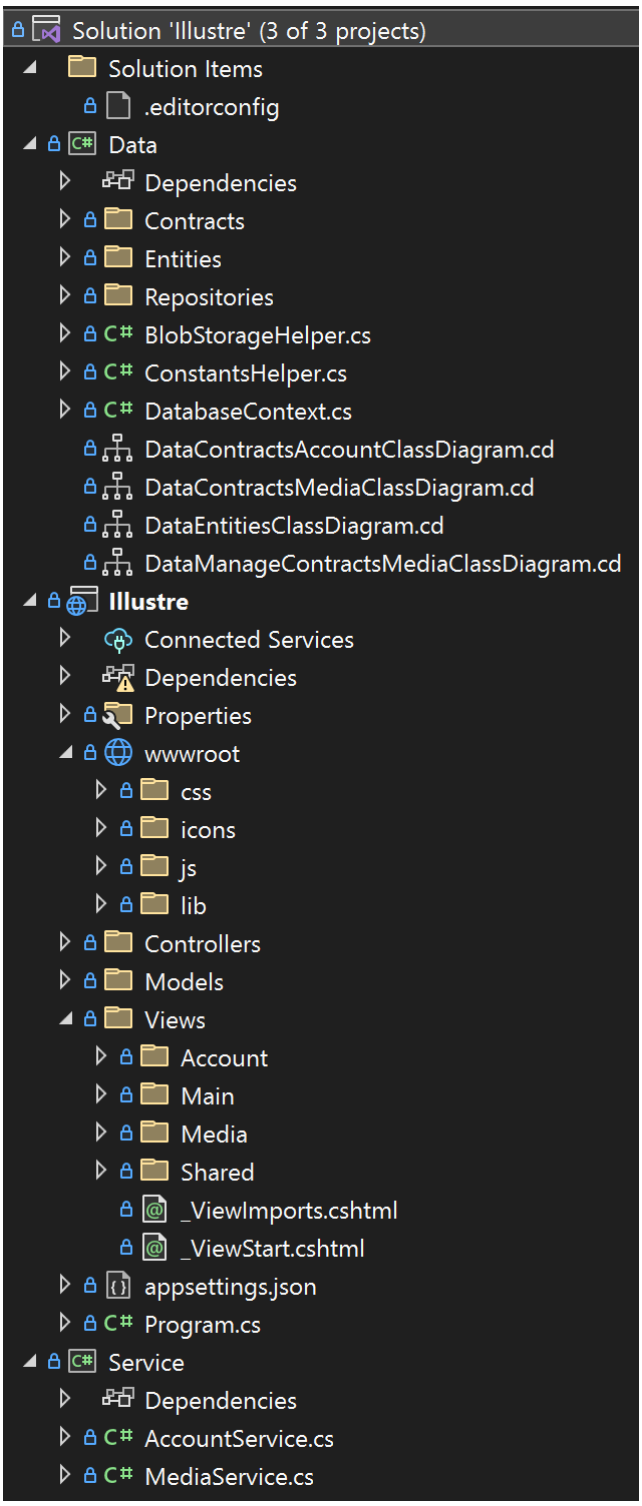


Рис. 11. – Загальна структура проекту.

Проект має наступну структуру:

- .editorconfig – файл налаштування текстового редактору IDE;
- 3 проекти: Data (сутності, контракти, репозиторії, контекст бази даних, сервіс для роботи з хмарним сховищем), Services (сервіси, проміжний шар між

контролерами та рівнем доступу до даних) і Illustre – проект з назвою веб-застосунку, у якому містяться усі компоненти фреймворку MVC.

2.7. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними є дані, які передаються у моделях та дані cookie-файлів. Дані з моделей необхідні для коректної роботи ендпоїнтів, а дані з cookie-файлів необхідні для можливості ідентифікації користувача та надання або обмеження прав доступу до веб додатку.

Серед вхідних даних, що надходять у моделях можуть бути:

- унікальні ідентифікатори сутностей;
- текстові дані;
- електронна адреса;
- пароль;
- тощо.

Вихідними даними є представлення, які бачить користувач у вікні свого браузеру. Користувач не завжди має змогу побачити чисті дані моделі, які приходять з контролеру у представлення. У той же час саме представлення теж має свою логіку побудови і тому його структура також є результатом певних обчислень.

Представлення може надсилати дані у контролер за допомогою HTML-форм або за допомогою параметрів запитів. Контролер надсилає дані у моделі, яка є основою побудови представлення.

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Для користувача є важливим мати систему, яка може запускатися та працювати з сучасними браузерами, тож необхідно мати такі мінімальні параметри ЕОМ які рекомендують розробники більшості браузерів:

- ЦП: Pentium 4.
- Відеоадаптер: 3D адаптер nVidia, Intel, AMD/ATI.
- Відеопам’ять: 128 МБ.
- Накопичувач: 150 Гб.
- Оперативна пам’ять: 2 ГБ.

2.7.2. Використані програмні засоби

Під час розробки даного застосунку були використані такі програмні засоби:

- Microsoft Visual Studio 2022;
- Microsoft SQL Server Management Studio 18;
- Git Bash;
- GitHub;
- Node.js;
- Azure CLI;
- Azurite.

Microsoft Visual Studio 2022 – це найновіша версія популярної IDE для багатьох мов програмування від компанії Microsoft. Перевагою цього середовища розробки є інтеграція всіх необхідних для розробки компонентів і можливість додати велику кількість плагінів для роботи. У цього додатку дуже зручний інтерфейс і він має усе необхідне для розробки в одному місці.

Microsoft SQL Server Management Studio 18 – це клієнт для роботи з сервером бази даних MS SQL, який також був розроблений компанією Microsoft. Цей продукт є дуже потужним і водночас зручним інструментом при розробці баз даних і створенні складних запитів до них.

Git Bash – це консольний інтерфейс програми Git, яка була розроблена відомим програмістом Лінусом Торвальдсом, автором ядра операційної системи GNU Linux. Вперше вийшла у реліз у 2005 року, і з того часу стала фактично стандартом будь-яких систем контролю версій. Це є найзручніший інструмент для віддаленого зберігання файлів проекту при розробці як у командах, так і у проектах з одним програмістом. Консольний інтерфейс має велику перевагу над будь-яким графічним інтерфейсом у тому, що жодний графічний інтерфейс не імплементує повністю весь функціонал програми Git. Окремою перевагою консольного інтерфейсу є прозорість команд і можливість завжди дізнатись, що саме відбувається.

GitHub – найбільший сервіс зберігання репозиторіїв будь-яких проектів. GitHub надає багато можливостей для командної роботи з спільної верифікації якості коду.

Node.js — потужна платформа, яка має можливість слугувати самостійною серверною оболонкою. У контексті цього проекту використовувалась для роботи з Azurite.

Azure CLI – мова запитів у вигляді команд до сервісів Azure.

Azurite – емулятор хмарного сховища Azure, який має абсолютно ідентичний консольний інтерфейс та структуру зберігання файлів у контейнерах.

2.7.3. Виклик та завантаження програми

Для запуску серверної частини додатку необхідно:

− інсталювати останню версію Node.js;

− інсталювати останню версію Azure CLI;

− інсталювати емулятор Azurite за допомогою команди у терміналі “npm install -g azurite”;

− створити папку для зберігання файлів контейнеру, відкрити її у терміналі і запустити Azurite за допомогою команди “azurite”;

− інсталювати останню версію Microsoft SQL Server та останню версію Microsoft SQL Server Management Studio;

− підключитися до локального серверу бази даних у програмі Microsoft SQL Server Management Studio;

− запустити скрипт Setup.sql, який знаходиться у каталозі Illustre/Database.

На даному етапі завершено базове налаштування серверу і серверним додатком можна користуватись. Електронна адреса і пароль Суперадміна знаходяться у файлі Setup.sql. З міркувань безпеки ці дані можна замінити у скрипті власноруч, перш ніж створювати базу даних. Для цього можна використати додану консольну утиліту PasswordGenerator, яка генерує значення пароля, солі і хешу.

Для того, щоб заповнити додаток тестовими даними зображень і тегів потрібно:

− запустити скрипт Import.cmd, який знаходиться у каталозі Demo;

− запустити скрипт Import.sql з того же каталогу.

З точки зору користувача, єдине, що необхідно зробити — відкрити веб-додаток за посиланням. Цей додаток працює у всіх сучасних браузерах для будь яких клієнтів — персональних комп’ютерів, мобільних пристроїв, планшетів тощо.

2.7.4. Опис інтерфейсу користувача

При першому запуску веб-додатку користувач бачить вікно як на Рис. 12. Оскільки користувач не має cookie-файлу, який зберігає факт його згоди на використання таких файлів, то він бачить відповідне попередження. Після цього cookie-файл вже збережено. Оскільки інший важливий cookie-файл, з інформацією про ідентифікатор сесії, також відсутній, користувачу автоматично пропонується або увійти в свій акаунт або зареєструватися як юзер.

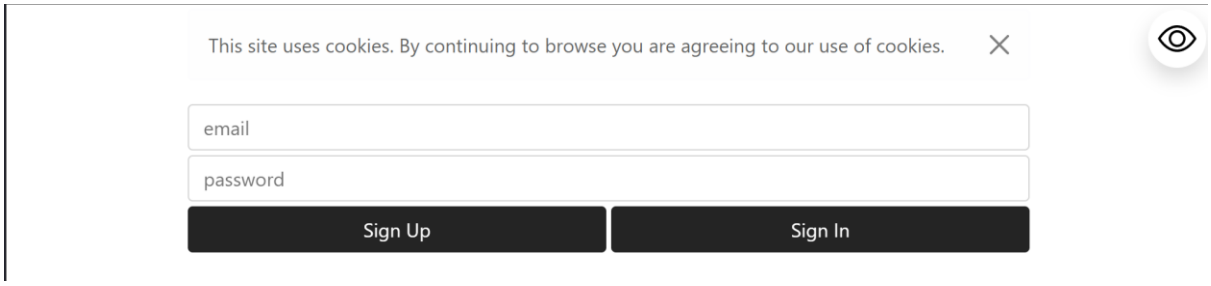


Рис. 12. – Сторінка авторизації нового користувача.

Рис. 13. – Пуста бокова панель.

Також у додатку використовується бокова панель, яка є поки що пустою, оскільки користувач не є зареєстрованим.

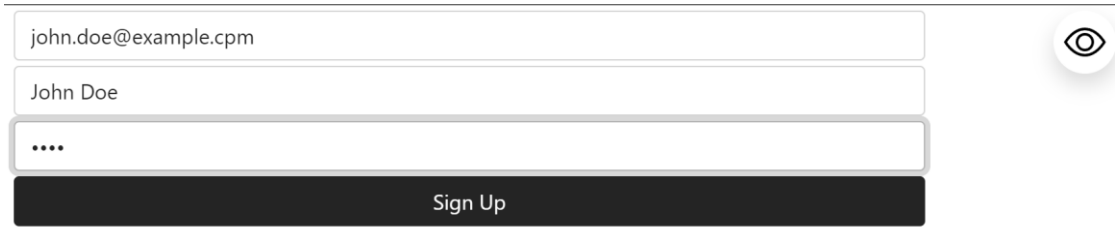


Рис. 14. – Форма реєстрації.

Після успішної реєстрації бачимо таке веб-сторінку з таким зображенням:

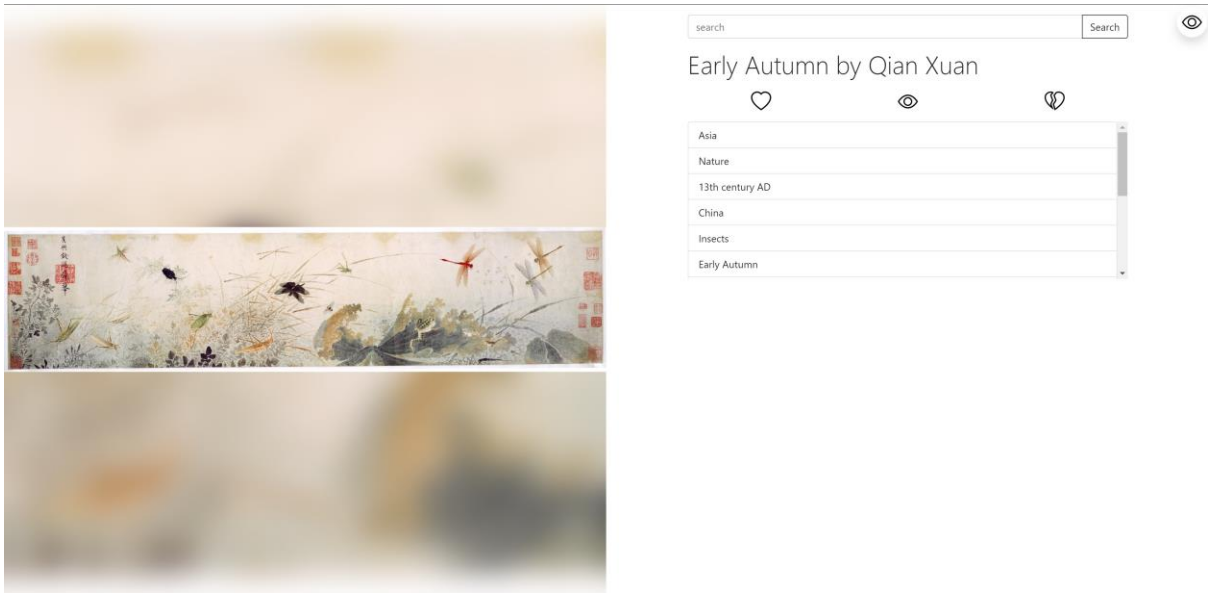


Рис. 15. – Головна сторінка користувача.

Для пристроїв з вертикальним дисплеєм це буде мати наступний вигляд:

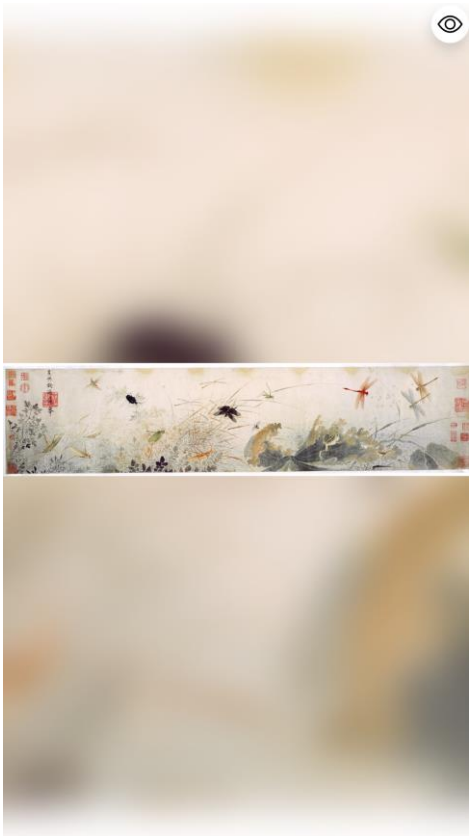


Рис. 16. – Верхня частина головної сторінки користувача.

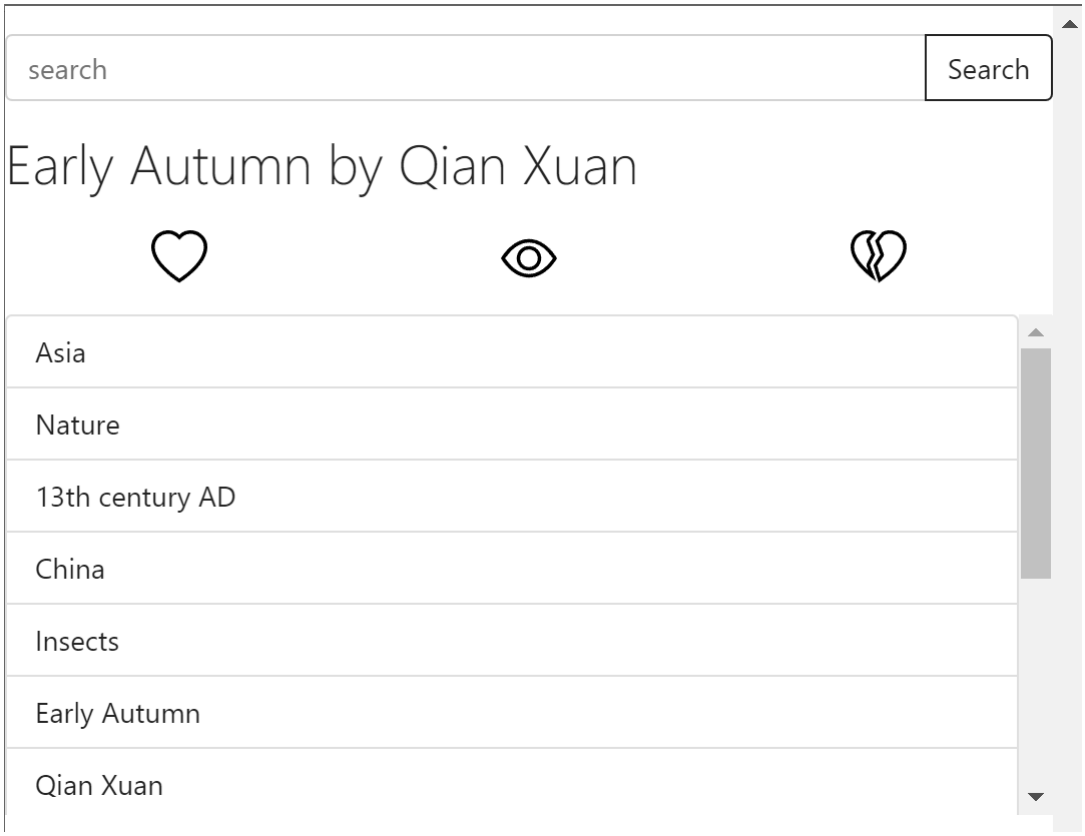


Рис. 17. – Нижня частина головної сторінки користувача.

Після додавання декількох лайків можемо побачити їх як на Рис. 18.

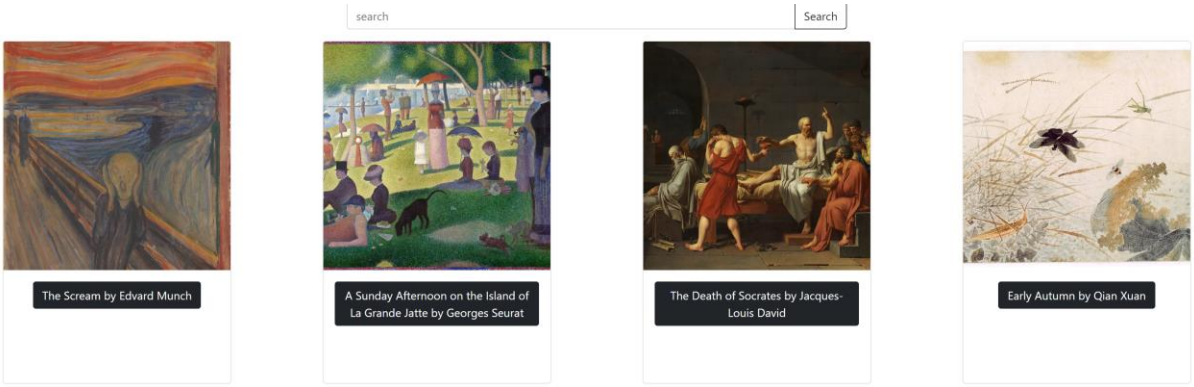


Рис. 18. – Лайки користувача.

Спробуємо знайти серед лайків зображення зі словом “Autumn”:

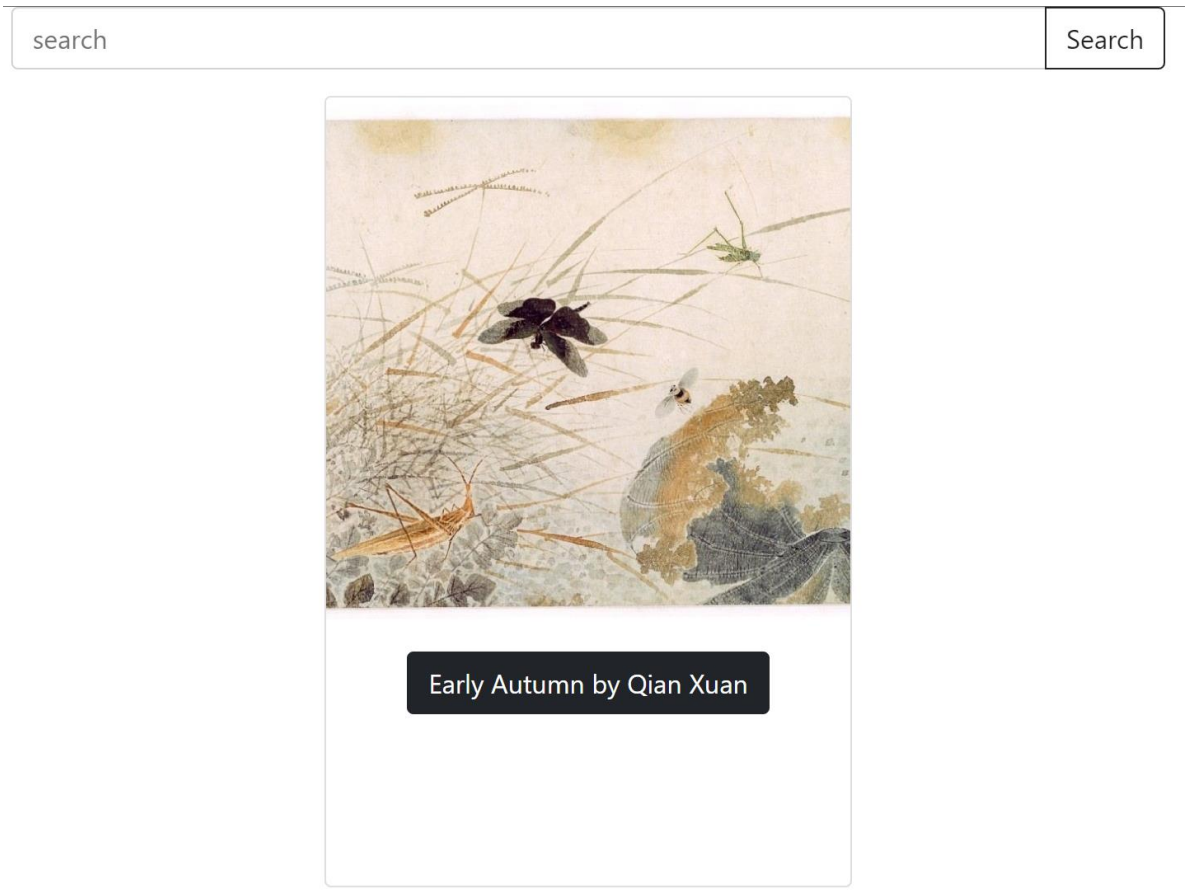


Рис. 19. – Результат пошуку.

Спробуємо відкрити дизлайки:

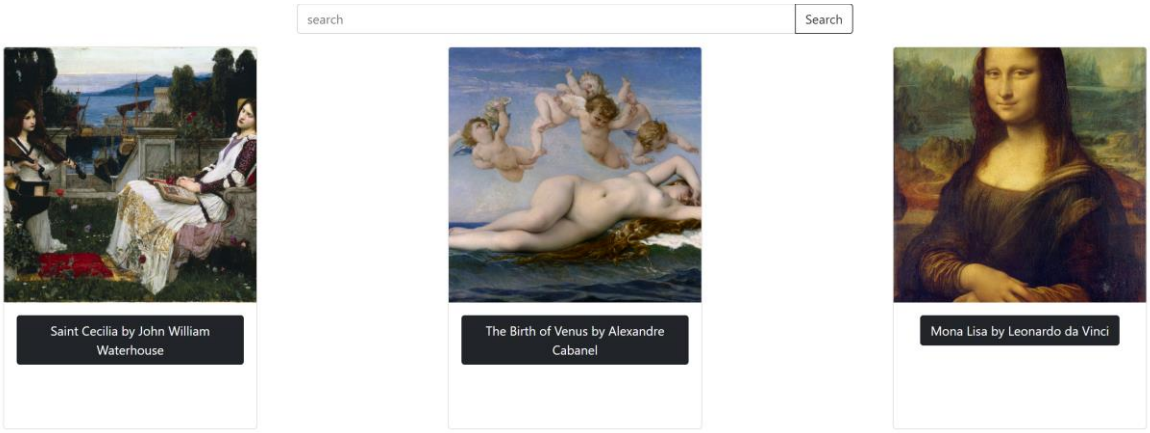


Рис. 20. – Дизлайки користувача.

Спробуємо знайти зображення за текстом “Mona Lisa”:

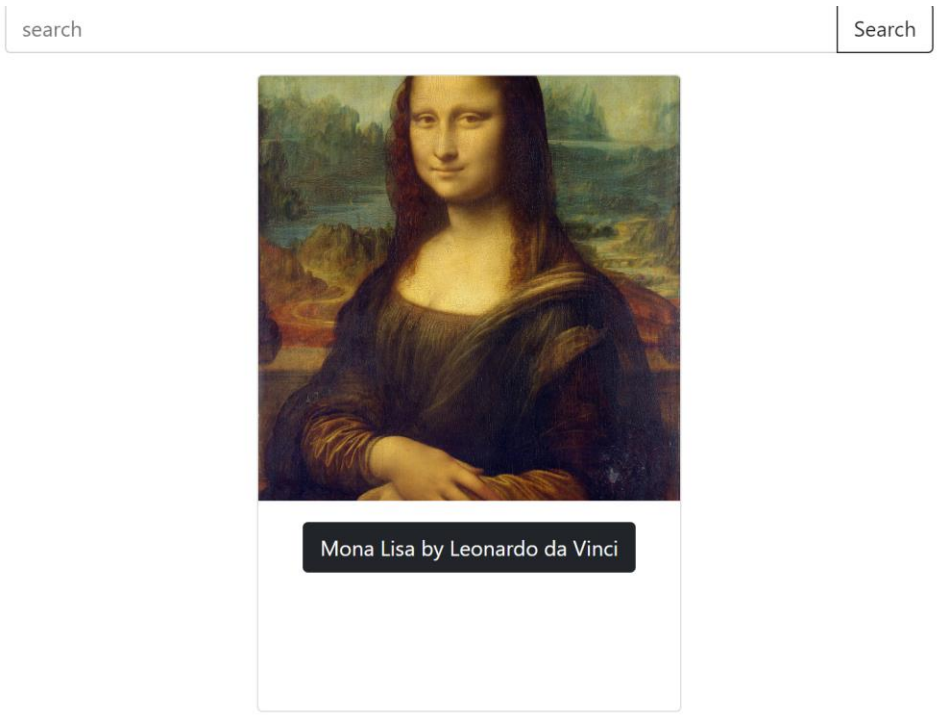


Рис. 21. – Результат пошуку.

Спробуємо відкрити теги користувача. Бачимо, що через велику кількість наявних тегів на сторінці з’являється пагінація.

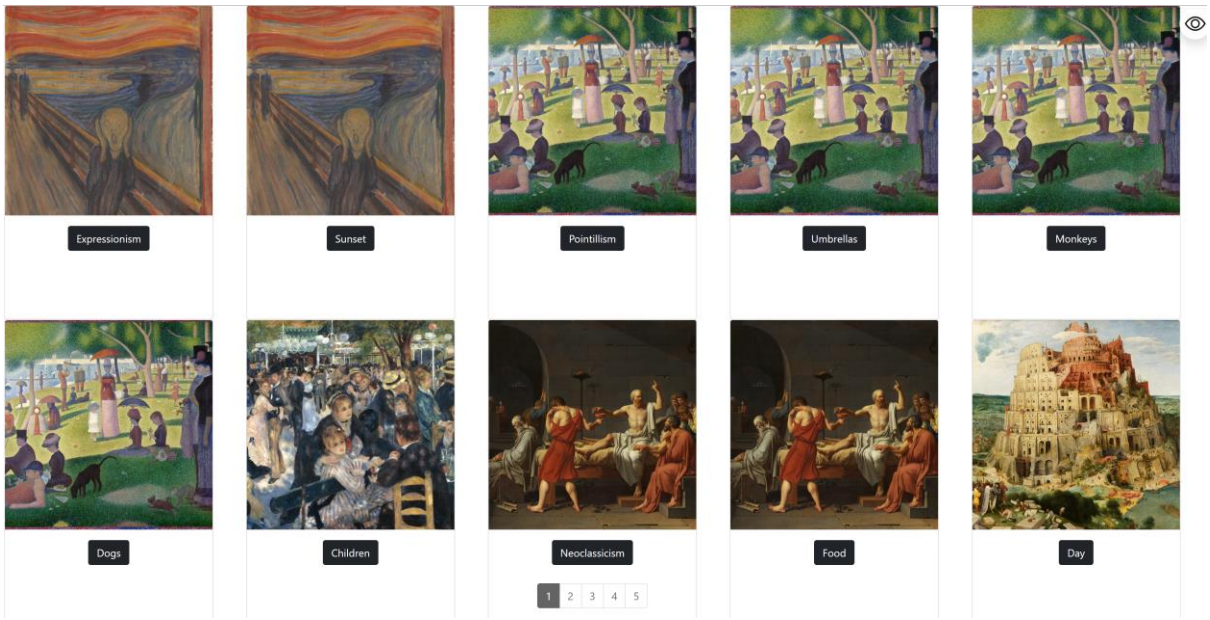


Рис. 22. – Теги користувача.

Відкриваємо будь-який з тегів, наприклад “Food”.

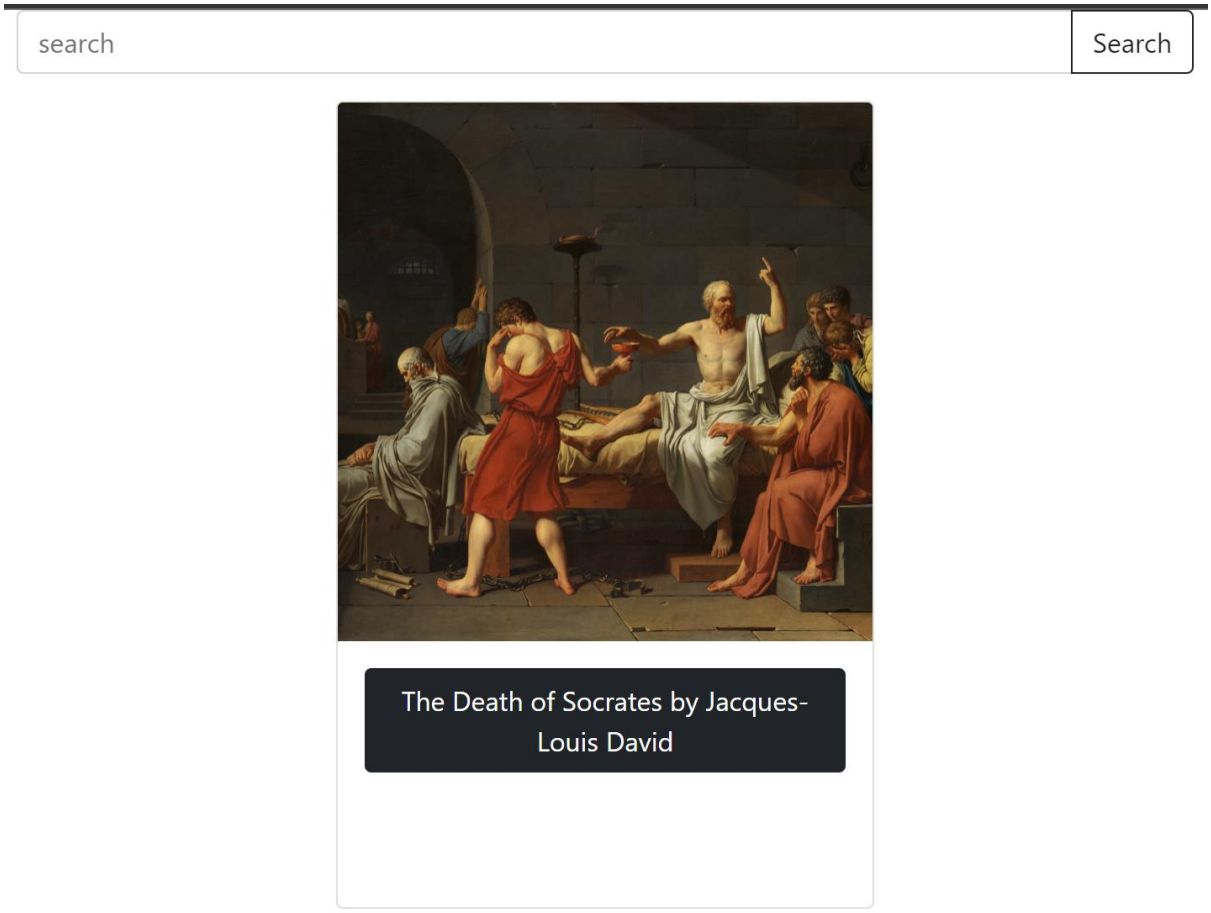


Рис. 23. – Теги користувача “Food”.

Відкриваємо це зображення. Бачимо, що у нього є статус активної реакції.

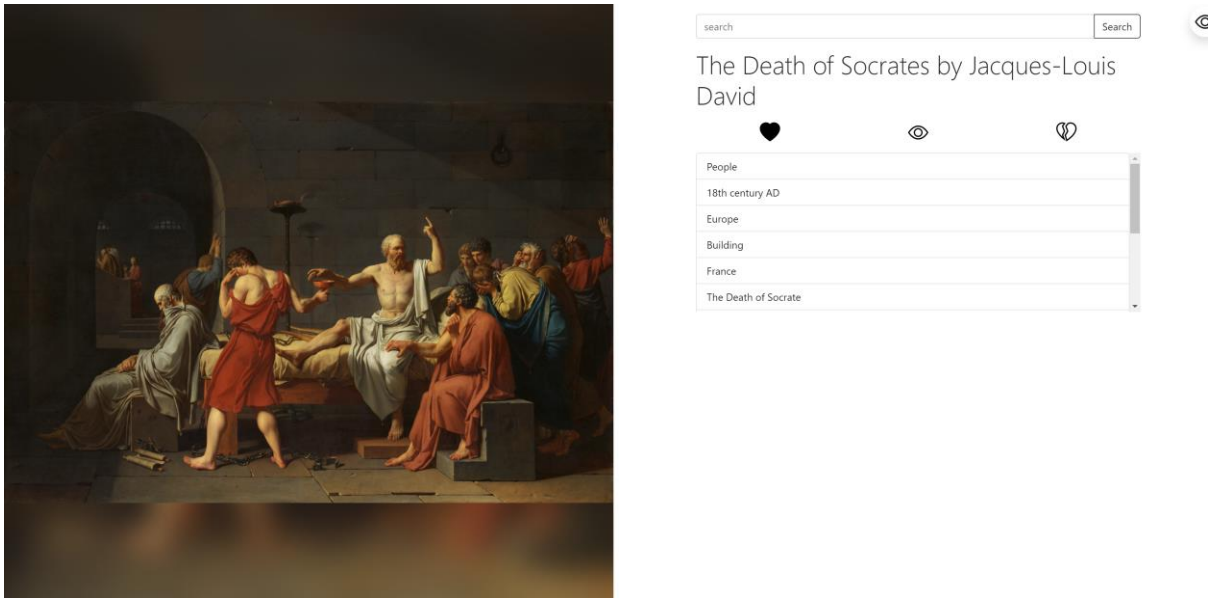


Рис. 24. – Лайкнуте зображення.

Спробуємо перейти зі сторінки цього зображення то тегу “Food” і бачимо, що у системі є набагато більше зображень з таким тегом.

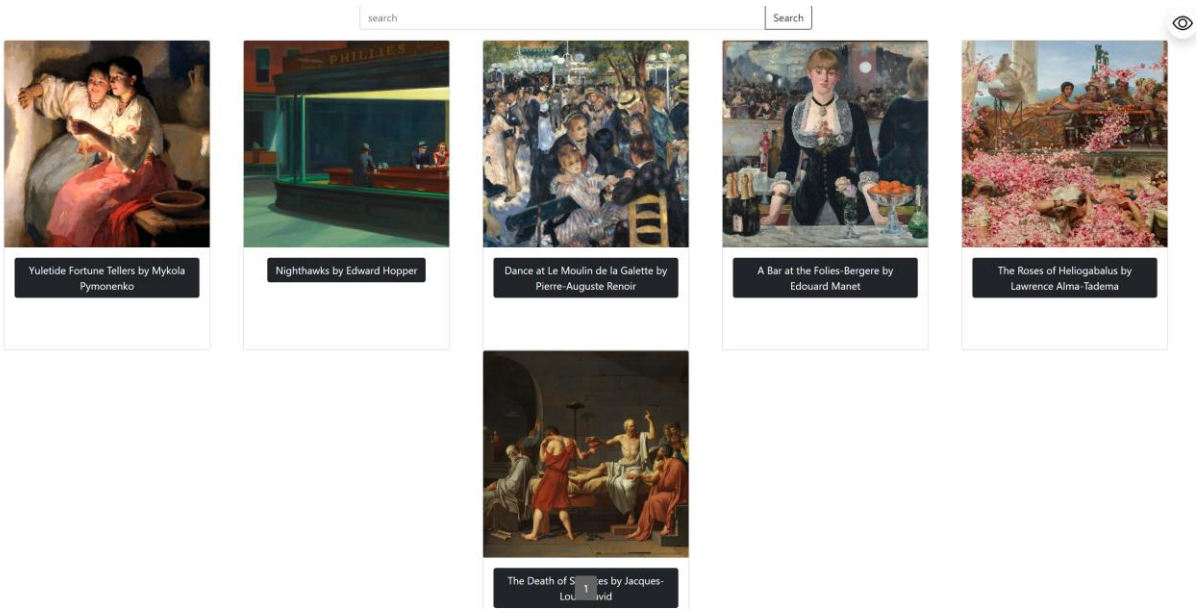


Рис. 25. – Зображення з тегом “Food”.

Повертаємось на попередню сторінку і спробуємо знайти зображення за тегом “Yuletude”.

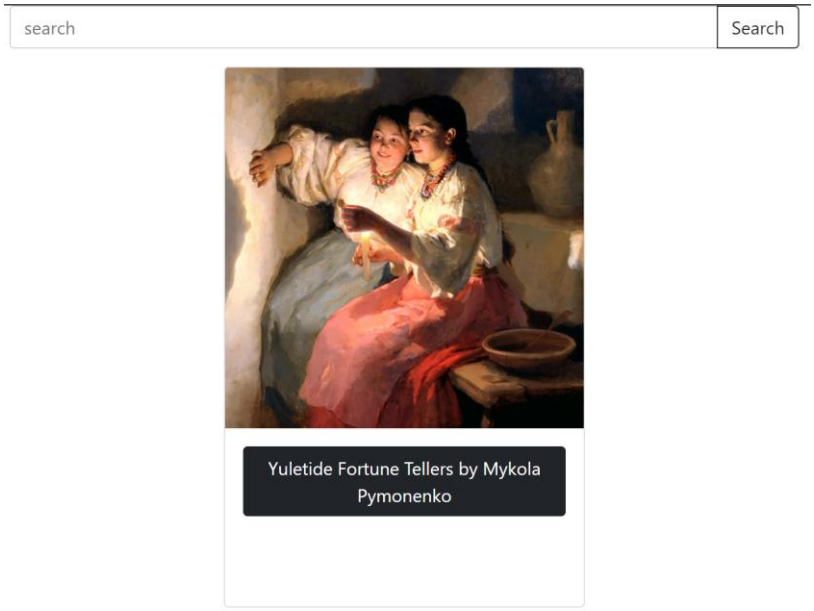


Рис. 26. – Результат пошуку.

Виходимо з системи і заходимо з електронною поштою і паролем суперадміна:

Рис. 27. – Меню СуперАдміну.

Заходимо у меню “Manage Editors” і бачимо, що Едіторів у системі ще немає:

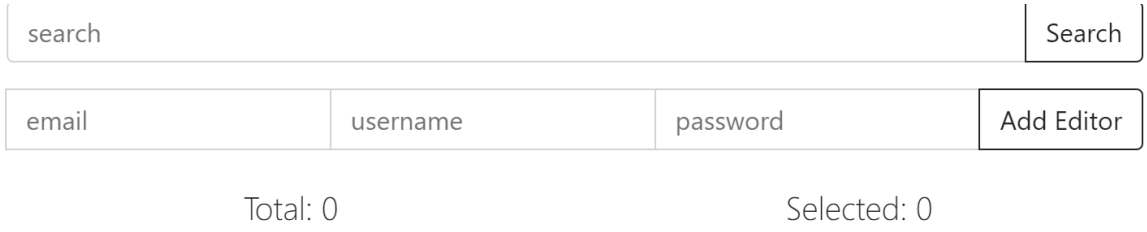


Рис. 28. – Меню “Manage Editors”.

Додаємо одного Едітора:

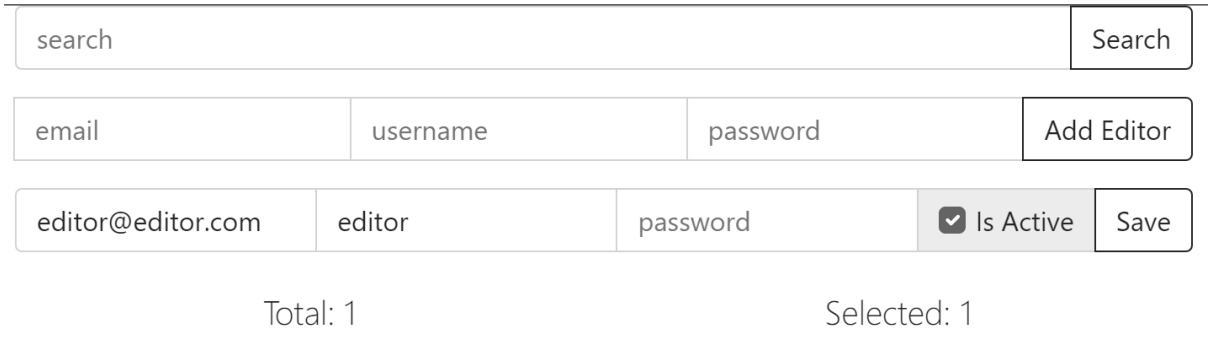


Рис. 29. – Новий Едітор.

Натомість у меню “Manage Users” бачимо, що кілька юзерів вже є:

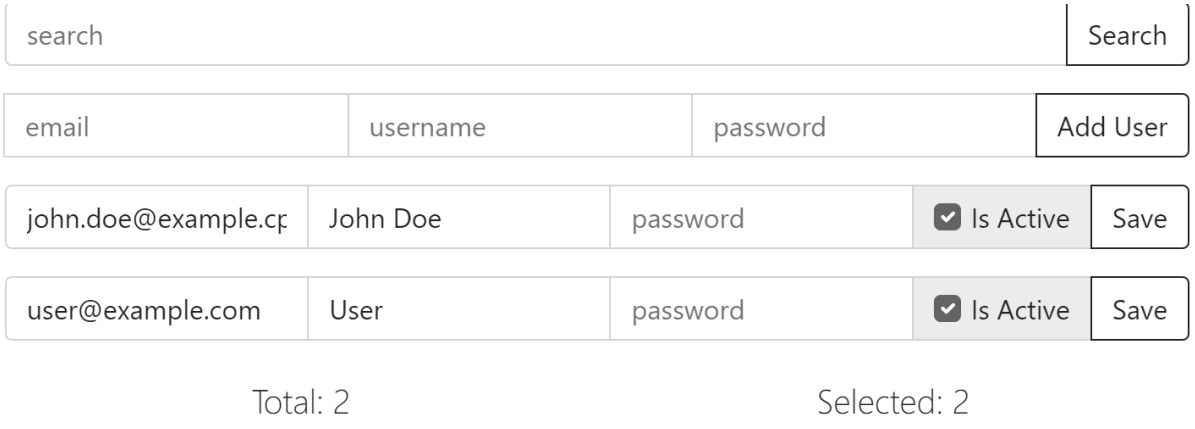


Рис. 30. – Меню “Manage Users”.

Виходимо з системи і логуємось як новостворений Едітор:

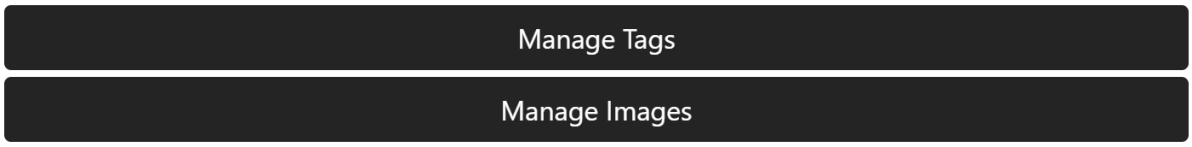


Рис. 31. – Меню Едітора.

Переглядаємо список наявних зображень:

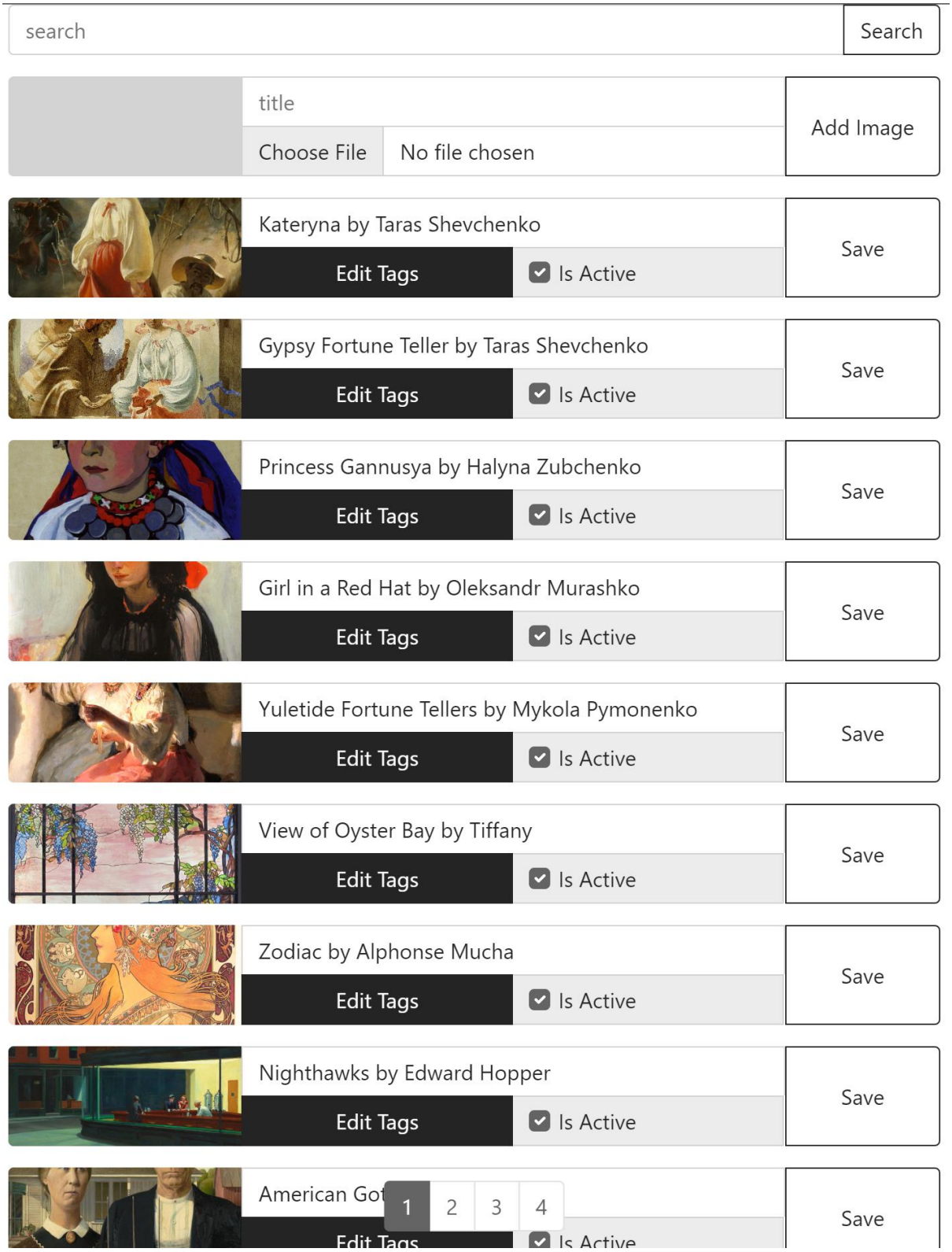


Рис. 32. – Наявні зображення.

Спробуємо натиснути на одне з них:

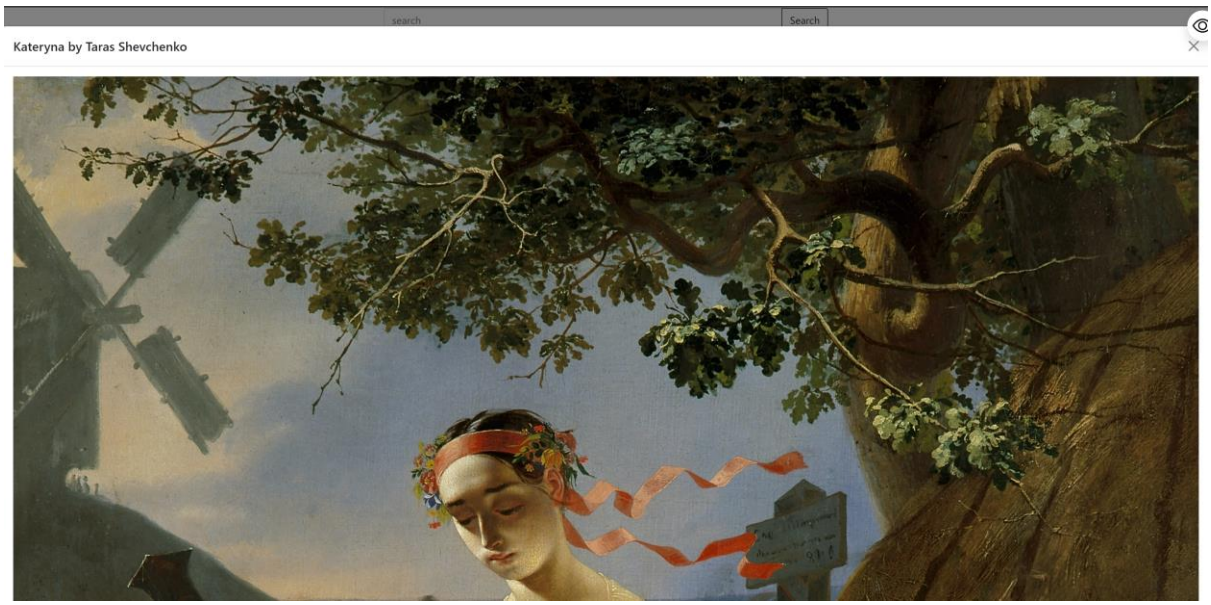


Рис. 33. – Детальний перегляд зображення.

Спробуємо додати нове зображення:

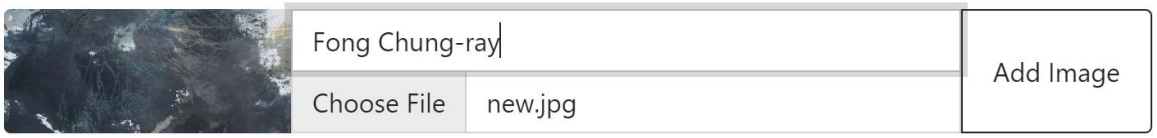


Рис. 34. – Додання зображення.

Переглядаємо теги іншого зображення:

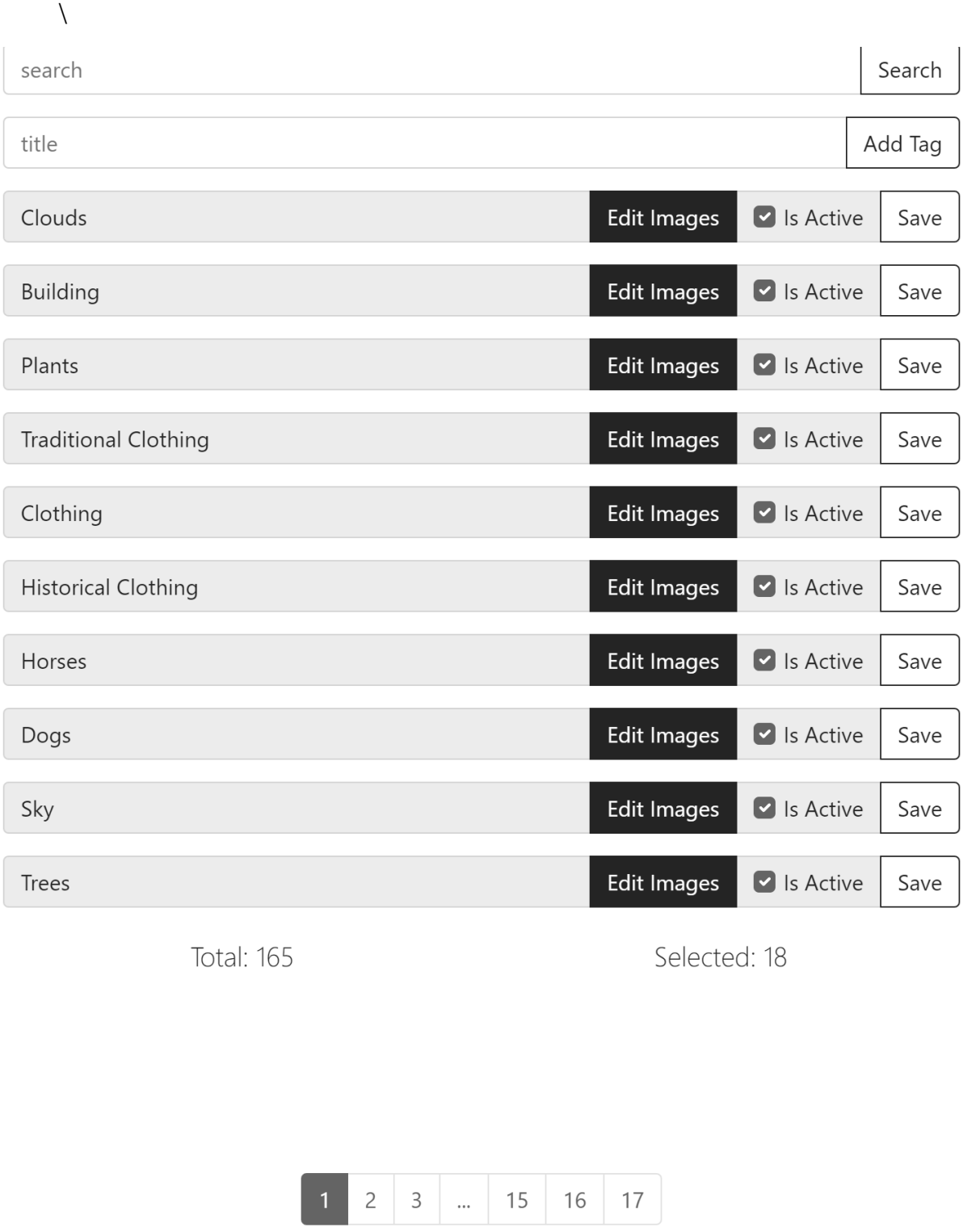


Рис. 35. – Теги зображення.

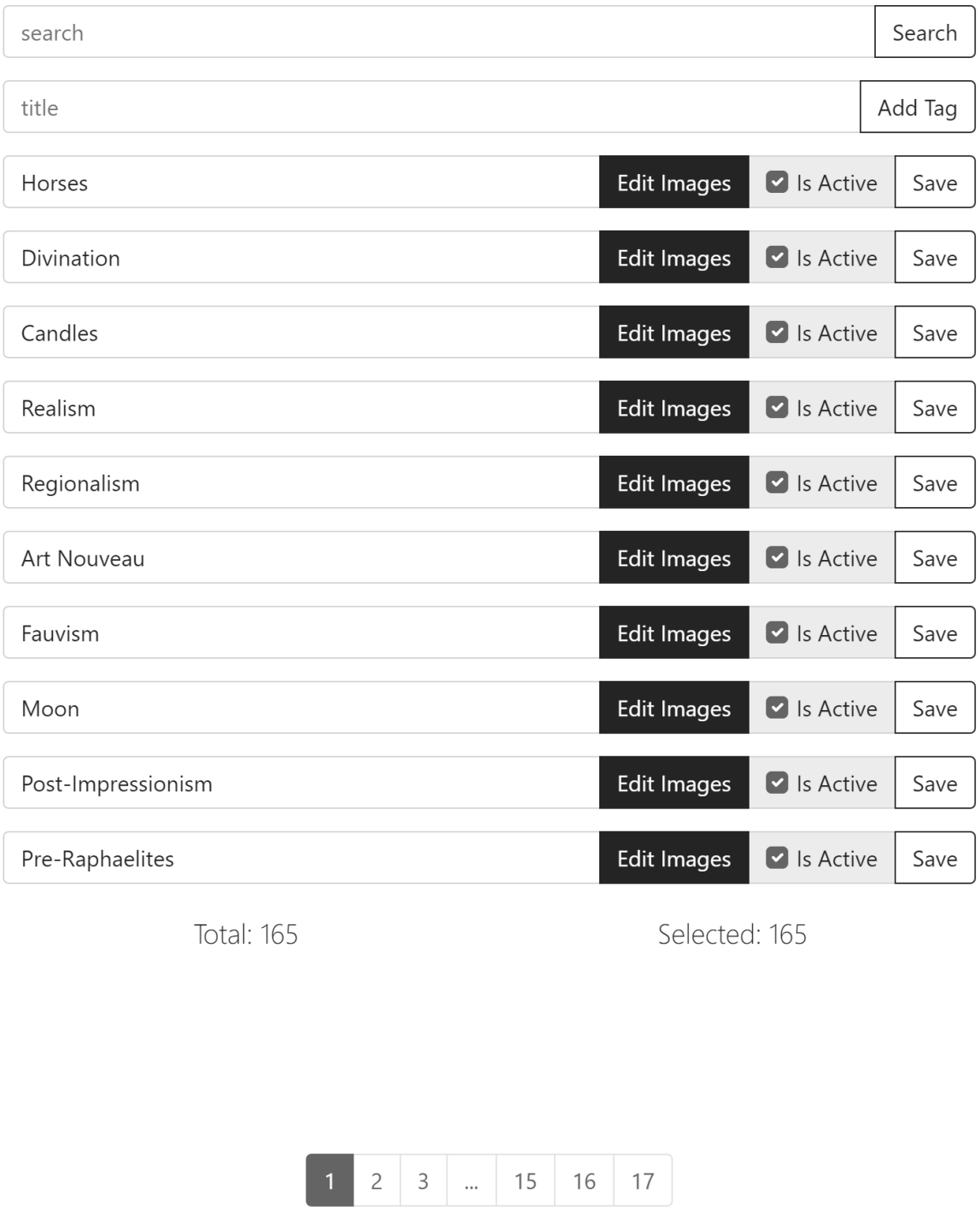


Рис. 35. – Наявні теги.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

- передбачуване число операторів програми – 1512;
- коефіцієнт складності програми – 1,27;
- коефіцієнт корекції програми в ході її розробки – 0,07;
- годинна заробітна плата програміста – 56 грн/год;
- коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,5;
- коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,5;
- вартість машино-години ЕОМ – 10 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \tag{3.2}$$

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1512 * 1,5 * (1 + 0,07) \approx 2426,76$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85) \cdot K}, \tag{3.3}$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$t_u = (2426,76 * 1,5) / (85 * 1,5) \approx 28,55$ людино-годин.

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K}, \tag{3.4}$$

$t_a = 2426,76 / (25 * 1,5) \approx 64,71$ людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot K}, \quad (3.5)$$

$t_n = 2426,76 / (24 * 1,5) \approx 67,41$ людино-годин.

Витрати праці на налагодження програми на ЕОМ:

1. за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5) \cdot K}, \quad (3.6)$$

$t_{отл} = 2426,76 / (5 * 1,5) \approx 323,57$ людино-годин,

2. за умови комплексного налагодження завдання:

$$t^к_{отл} = 1,5 * t_{отл}, \quad (3.7)$$

$t^к_{отл} = 1,5 * 323,57 \approx 485,36$ людино-годин

Витрати праці на підготовку документації:

$$t_д = t_{дp} + t_{до}, \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_д = \frac{Q}{(15...20)\cdot K}, \quad (3.9)$$

$t_д = 2426,76 / (20 * 1,5) \approx 80,89$ людино-годин.

$t_{до}$– трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}, \quad (3.10)$$

$t_{до} = 0{,}75 * 64{,}71 \approx 48{,}53$ людино-годин.

$t_{д} = 64{,}71 + 48{,}53 = 113{,}24$ людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

$t = 50 + 28{,}55 + 64{,}71 + 67{,}41 + 485{,}36 + 113{,}24 = 809{,}27$ людино-годин.

У результаті ми розрахували, що в загальній складності необхідно близько 809,27 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ включають витрати на заробітну плату виконавця програми З/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ} \text{ грн.}, \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР} \text{ грн.}, \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 809,27 * 56 = 45319,12 \text{ грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М} \text{ грн.,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 485,36 * 10 = 4853,6 \text{ грн.}$$

$$К_{ПО} = 45319,12 + 4853,6 = 50172,72 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.,} \qquad (3.14)$$

де B_k- число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$T = 809,27 / (1 * 176) \approx 4,6 \text{ міс.}$$

Висновки. На розробку веб-додатку крос-платформеної системи пошуку зображень піде близько 809,27 людино-години. Тобто, ймовірна очікувана тривалість розробки складатиме 4,6 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 50172,72 грн.

ВИСНОВКИ

Під час виконання практики було поставлено завдання зпроектувати та розробити веб-додаток кросс-платформенної системи пошуку зображень з використанням технології ASP.NET. Ця інформаційна система призначена для надання універсального інструменту для відображення контенту бази зображень та тегів. Практичне призначення даної системи полягає в забезпеченні користувачам легкого пасивного пошуку зображень і можливості реакцій на них. Пошук зображень відбувається за допомогою аналізу минулих взаємодій користувача з даною інформаційною системою. Інші користувачі — такі як суперадмін та едітори — наповнюють веб-додаток контентом та модерують його.

Під час виконання даного проекту були виконані наступні задачі:

− вивчено предметну область розв'язуваної задачі;

− створено алгоритм для реалізації поставленого завдання;

− створено макети інтерфейсу системи, базу даних і клієнтську та серверну програму.

Розроблене програмне забезпечення дозволяє:

− формувати web-сторінки з використанням контенту з серверної частини застосунку;

− зберігати і опрацьовувати дані реакцій користувачів;

− редагувати інформацію у системі.

Програма реалізована на мові програмування C# з використанням технологій ASP.NET і Razor Pages, локального сховища Azurite і бази даних MS SQL.

Під час виконання кваліфікаційної роботи також було визначено трудомісткість розробленого програмного продукту (809,27 людино-годин), проведений підрахунок вартості роботи по створенню програми (50172,72 грн) та розраховано час на його створення (4,6 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Офіційна документація Microsoft Docs: https://docs.microsoft.com/en-us/
- Pro Git 2nd Edition - Scott Chacon, Ben Straub - 2014 - ISBN-13: 978-1484200773 - https://doi.org/10.1007/978-1-4842-0076-6
- Application Techniques for Checking and Transformation of Names - J. Klensin - 2004 - https://www.rfc-editor.org/rfc/rfc3696
- Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec - S. Kelly, S. Frankel - 2007 - https://www.rfc-editor.org/rfc/rfc4868
- Microsoft Password Guidance - Robyn Hicock - 2016 - https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/Microsoft_Password_Guidance-1.pdf
- Bootstrap documentation - https://getbootstrap.com/docs/
- Pro ASP.NET Core 6 - Adam Freeman - 2022 - ISBN: 978-1-4842-7957-1 - https://doi.org/10.1007/978-1-4842-7957-1
- Pro C# 10 with .NET 6 - Andrew Troelsen, Philip Japikse — 2022 - ISBN-13 : 978-1484278680
- C# in Depth - Eric Lippert – 2019 - ISBN 9781617294532
- Code like a Pro in C# - Jort Rodenburg – 2021 - ISBN 9781617298028
- Entity Framework Core in Action - Jon P Smith - ISBN 9781617294563
- CLR via C# (Developer Reference) - Jeffrey Richter - ISBN-10 : 0735667454
- The Unified Modeling Language User Guide - Grady Booch , James Rumbaugh, Ivar Jacobson - 2005 - ISBN-10 : 0321267974
- C# Data Structures and Algorithms: Explore the possibilities of C# for developing a variety of efficient applications - Marcin Jamro – 2018 - ISBN-13: 978-1788833738

- Software Architecture with C# 9 and .NET 5: Architecting software solutions using microservices, DevOps, and design patterns for Azure, 2nd Edition – 2020 - Gabriel Baptista, Francesco Abbruzzese - ISBN-13: 978-1800566040
- T-SQL Fundamentals (Developer Reference) – 2016 - Itzik Ben-Gan - ISBN-13: 978-1509302000
- SQL Antipatterns: Avoiding the Pitfalls of Database Programming (Pragmatic Programmers) – 2017 - Bill Karwin - ISBN-13: 978-1934356555
- Designing Software Architectures. A Practical Approach – 2016 - Humberto Cervantes, Rick Kazman – ISBN978-0134390789
- Professional ASP.NET MVC 5 – 2014 - Jon Galloway, Brad Wilson, K. Scott Allen, David Matson - ISBN-13: 978-1118794753
- ASP.NET Core in Action – 2021 - Andrew Lock - ISBN 9781617298301

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг Database/Setup.sql:

```
CREATE DATABASE [Illustre];
GO

USE [Illustre];
GO

-- Roles:
-- SuperAdmin = 0
-- Editor = 1
-- User = 2

CREATE TABLE [Accounts] (
   [Id]           INT           IDENTITY(1,1),
   [Email]        VARCHAR(320)  NOT NULL,
   [Salt]         CHAR(32)      NOT NULL,
   [PasswordHash] CHAR(64)      NOT NULL,
   [Username]     NVARCHAR(MAX) NOT NULL,
   [Role]         INT           NOT NULL,
   [SessionGuid]  CHAR(32)      NULL,
   [LastLogin]    DATETIME      NULL,
   [IsActive]     BIT           NOT NULL      CONSTRAINT
[DF_Account_IsActive] DEFAULT 1,

   CONSTRAINT [PK_Account]          PRIMARY KEY ([Id]),
   CONSTRAINT [CHK_Account_Role]    CHECK       ([Role] IN
(0, 1, 2)),
   CONSTRAINT [UQ_Account_Email]    UNIQUE      ([Email]),
   CONSTRAINT [UQ_Account_SessionGuid] UNIQUE   ([SessionGuid])
);
GO

CREATE TABLE [Tags] (
   [Id]       INT          IDENTITY(1,1),
   [Title]    VARCHAR(900) NOT NULL,
   [IsActive] BIT          NOT NULL      CONSTRAINT [DF_Tag_IsActive]
DEFAULT 1,

   CONSTRAINT [PK_Tag]      PRIMARY KEY ([Id]),
   CONSTRAINT [UQ_Tag_Title] UNIQUE     ([Title])
);
```

```
GO

CREATE TABLE [Images] (
   [Id]         INT           IDENTITY(1,1),
   [Title]      VARCHAR(900)    NOT NULL,
   [IsActive]   BIT           NOT NULL         CONSTRAINT
[DF_Image_IsActive]    DEFAULT    1,

   CONSTRAINT    [PK_Image]    PRIMARY KEY    ([Id])
);
GO

CREATE TABLE [ImageProperties] (
   [Id]         INT    IDENTITY(1,1),
   [ImageId]    INT    NOT NULL,
   [TagId]      INT    NOT NULL,
   [IsActive]   BIT    NOT NULL         CONSTRAINT
[DF_ImageProperty_IsActive]    DEFAULT    1,

   CONSTRAINT    [PK_ImageProperty]          PRIMARY KEY    ([Id]),
   CONSTRAINT    [FK_ImageProperty_ImageId]    FOREIGN KEY    ([ImageId])
REFERENCES    [Images]    ([Id]),
   CONSTRAINT    [FK_ImageProperty_TagId]     FOREIGN KEY    ([TagId])
REFERENCES    [Tags]     ([Id]),
   CONSTRAINT    [UQ_ImageProperty_Ids]       UNIQUE        ([ImageId],
[TagId])
);
GO

CREATE TABLE [Reactions] (
   [Id]         INT    IDENTITY(1,1),
   [AccountId]    INT    NOT NULL,
   [ImageId]     INT    NOT NULL,
   [IsLiked]     BIT    NOT NULL,
   [IsActive]    BIT    NOT NULL         CONSTRAINT    [DF_Reaction_IsActive]
DEFAULT    1,

   CONSTRAINT    [PK_Reaction]            PRIMARY KEY    ([Id]),
   CONSTRAINT    [FK_Reaction_AccountId]    FOREIGN KEY    ([AccountId])
REFERENCES    [Accounts]    ([Id]),
   CONSTRAINT    [FK_Reaction_ImageId]     FOREIGN KEY    ([ImageId])
REFERENCES    [Images]     ([Id]),
        CONSTRAINT    [UQ_Reaction_Ids]        UNIQUE        ([AccountId],
[ImageId])
```

```
);
GO

-- SuperAdmin password: =8K4Kbzt44,Wvv
INSERT INTO [Accounts] (
    [Email],
    [Salt],
    [PasswordHash],
    [Username],
    [Role]
)
VALUES (
    'superadmin@example.com',
    '476F95CB4B4A27F4F08853C327EFD808',

'B22C85A69EE37B7C5702753E0BD0E5A5AAADF9B21B4DD40C6B3097962A188D8A',
    'SuperAdmin',
    0
);
GO
```

Лістинг Illustre/Illustre/Controllers/AccountController.cs:

```
using Data;
using Data.Contracts.Accounts;
using Data.Entities;
using Illustre.Models;
using Microsoft.AspNetCore.Mvc;
using Service;
using System.Diagnostics;

namespace Illustre.Controllers;

public class AccountController : CommonController
{
    public AccountController(AccountService accountService)
        : base(accountService) { }

    [HttpGet]
    public async Task<IActionResult> Index(SignInRequest request)
    {
        return await ExecuteRedirect(
```

```
        null,
        async (request) =>
        {
            var dto = request as SignInRequest;
            return View(dto);
        },
        request);
}

[HttpPost]
public async Task<IActionResult> SignIn(SignInRequest request)
{
    return await ExecuteRedirect(
        null,
        async (request) =>
        {
            var dto = request as SignInRequest;

            if (ModelState.IsValid)
            {
                var response = await _accountService.TrySignIn(dto!);

                if (response != null)
                {
                    SetCookies(response);

                    var result = RedirectAuthenticated(response.Role);
                    if (result != null)
                    {
                        return result;
                    }
                }
            }

            return Redirect("/Account/Index?isFirstAttempt=false");
        },
        request);
}

[HttpGet]
public async Task<IActionResult> SignUp(SignUpRequest request)
{
    return await ExecuteRedirect(
        null,
```

```
        async (request) =>
        {
            var dto = request as SignUpRequest;
            return View(dto);
        },
        request);
}

[HttpPost]
public async Task<IActionResult> Register(SignUpRequest request)
{
    return await ExecuteRedirect(
        null,
        async (request) =>
        {
            var dto = request as SignUpRequest;
            if (ModelState.IsValid)
            {
                var response = await _accountService.TrySignUp(dto!);

                if (response != null)
                {
                    SetCookies(response);

                    var result = RedirectAuthenticated(response.Role);
                    if (result != null)
                    {
                        return result;
                    }
                }
            }

            return Redirect("/Account/SignUp?isFirstAttempt=false");
        },
        request);
}

[HttpGet]
public async Task<IActionResult> LogOut()
{
    return await ExecuteRedirect(
        () =>
        {
            RemoveCookies();
```

```
            return Redirect(IndexRedirect);
        },
        async (NoParameters) =>
        {
            return Redirect(IndexRedirect);
        },
        NoParameters);
    }

    [HttpGet]
    public async Task<IActionResult> Account()
    {
        return await Execute(
            Array.Empty<Role>(),
            NoParameters,
            async (NoParameters) =>
            {
                var array = NoParameters as object[];
                var cookie = array![CookieIndex] as string;

                var account = await _accountService.TryGetAccount(cookie!);
                return View(account);
            },
            true);
    }

    [HttpPost]
    public async Task<IActionResult> UpdateAccount(UpdateAccountRequest
request)
    {
        return await Execute(
            Array.Empty<Role>(),
            request,
            async (request) =>
            {
                var array = request as object[];
                var dto = array![DtoIndex] as UpdateAccountRequest;
                var cookie = array[CookieIndex] as string;

                var result = await _accountService
                    .TryUpdateAccount(cookie!, dto!);

                if (result == null)
                {
```

```
                return Redirect("/Account/Account?isFirstAttempt=false");
            }
            else
            {
                var options = new CookieOptions()
                {
                    Expires = DateTime.UtcNow.AddDays(1),
                    Domain = Request.Host.Host,
                };

                Response.Cookies.Append(
                    ConstantsHelper.UsernameCookie,
                    result.Username,
                    options);

                return View("Account", result);
            }
        },
        true);
    }

    [HttpGet]
    public async Task<IActionResult> ManageEditors
        (ManageAccountsRequest request)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin },
            request,
            async (request) =>
            {
                var dto = request as ManageAccountsRequest;
                dto!.AccountsData = await _accountService
                    .GetEditors(dto.Skip, dto.SearchPattern);
                return View(dto);
            });
    }

    [HttpGet]
    public async Task<IActionResult> ManageUsers(ManageAccountsRequest
request)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin },
            request,
```

```
            async (request) =>
            {
                var dto = request as ManageAccountsRequest;
                dto!.AccountsData = await _accountService
                    .GetUsers(dto.Skip, dto.SearchPattern);
                return View(dto);
            });
    }

    [HttpPost]
    public async Task<IActionResult> AddAccount(AddAccountRequest request)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin },
            request,
            async (request) =>
            {
                var dto = request as AddAccountRequest;
                var result = false.ToString().ToLower();
                var action = dto!.Role == Role.Editor ?
                            "ManageEditors" :
                            "ManageUsers";

                if (ModelState.IsValid)
                {
                    result = (await _accountService.TryAddAccount(dto!))
                        .ToString()
                        .ToLower();
                }

                return Redirect($"/Account/{action}?isFirstAttempt={result}");
            });
    }

    [HttpPost]
    public async Task<IActionResult> UpdateAccountById(ManageAccountModel
model)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin },
            model,
            async (model) =>
            {
                var dto = model as ManageAccountModel;
                var result = false.ToString().ToLower();
```

```
            var action = dto!.Role == Role.Editor ?
                    "ManageEditors" :
                    "ManageUsers";

            if (ModelState.IsValid)
            {
               result = (await _accountService.TryUpdateAccountById(dto!))
               .ToString()
               .ToLower();
            }

            return Redirect($"/Account/{action}?isFirstAttempt={result}");
         });
   }
}
```

Лістинг Illustre/Illustre/Controllers/CommonController.cs:

```
using Data;
using Data.Contracts.Accounts;
using Data.Entities;
using Microsoft.AspNetCore.Mvc;
using Service;

namespace Illustre.Controllers;

public abstract class CommonController : Controller
{
   public const string SuperAdminMenuRedirect = "/Main/SuperAdminMenu";

   public const string EditorMenuRedirect = "/Main/EditorMenu";

   public const string UserMenuRedirect = "/Main/UserMenu";

   public const string IndexRedirect = "/Account/Index";

   public const string NoParameters = "";

   public const int DtoIndex = 0;

   public const int CookieIndex = 1;

   protected readonly AccountService _accountService;

   public CommonController(AccountService accountService)
```

```
{
    _accountService = accountService;
}

protected async Task<IActionResult?> TryRedirect()
{
    if (Request.Cookies
            .TryGetValue(ConstantsHelper.SessionCookie, out var cookie))
    {
        var role = await _accountService.TryGetRoleBySessionGuid(cookie!);
        if (role != null)
        {
            var result = RedirectAuthenticated(role!.Value);
            if (result != null)
            {
                return result;
            }
        }
    }

    return null;
}

protected RedirectResult? RedirectAuthenticated(Role role)
{
    switch (role)
    {
        case Role.SuperAdmin:
            {
                return Redirect(SuperAdminMenuRedirect);
            }
        case Role.Editor:
            {
                return Redirect(EditorMenuRedirect);
            }
        case Role.User:
            {
                return Redirect(UserMenuRedirect);
            }
    }

    return null;
}
```

```
protected void SetCookies(SignInResponse response)
{
    var options = new CookieOptions()
    {
        Expires = response.Expires,
        Domain = Request.Host.Host,
    };

    Response.Cookies.Append(
        ConstantsHelper.SessionCookie,
        response.SessionGuid,
        options);

    Response.Cookies.Append(
        ConstantsHelper.UsernameCookie,
        response.Username,
        options);

    Response.Cookies.Append(
        ConstantsHelper.RoleCookie,
        response.Role.ToString(),
        options);
}

protected void RemoveCookies()
{
    Response.Cookies.Delete(ConstantsHelper.SessionCookie);
    Response.Cookies.Delete(ConstantsHelper.UsernameCookie);
    Response.Cookies.Delete(ConstantsHelper.RoleCookie);
}

protected async Task<IActionResult> Execute(
    Role[] allowedRoles,
    object dto,
    Func<object, Task<IActionResult>> action,
    bool injectCookie = false)
{
    if (Request.Cookies
            .TryGetValue(ConstantsHelper.SessionCookie, out var cookie) &&
        await CheckRoles(allowedRoles, cookie!))
    {
        var parameter = dto;

        if (injectCookie)
```

```
            {
                var array = new object[] { dto, cookie! };
                parameter = array;
            }

            return await action(parameter);
        }

        return Redirect(IndexRedirect);
    }

    protected async Task<IActionResult> ExecuteRedirect(
        Func<IActionResult>? redirectNotNullAction,
        Func<object, Task<IActionResult>> redirectNullAction,
        object redirectNullParameter)
    {
        var redirect = await TryRedirect();

        if (redirect != null)
        {
            return redirectNotNullAction is not null ?
                  redirectNotNullAction() :
                  redirect;
        }

        return await redirectNullAction(redirectNullParameter);
    }

    private async Task<bool> CheckRoles(
        Role[] allowedRoles,
        string cookie)
    {
        var role = await _accountService.TryGetRoleBySessionGuid(cookie);

        return role is not null &&
             (allowedRoles.Count() == 0 ||
              allowedRoles.Contains(role.Value));
    }
}
```

Лістинг Illustre/Illustre/Controllers/MainController.cs:

```
using Data.Entities;
using Microsoft.AspNetCore.Mvc;
```

```
using Service;

namespace Illustre.Controllers;

public class MainController : CommonController
{
   public MainController(
      AccountService accountService)
      : base(accountService) { }

   [HttpGet]
   public async Task<IActionResult> SuperAdminMenu()
   {
      return await Execute(
         new Role[] { Role.SuperAdmin },
         NoParameters,
         async (NoParameters) =>
         {
            return View();
         });
   }

   [HttpGet]
   public async Task<IActionResult> EditorMenu()
   {
      return await Execute(
         new Role[] { Role.Editor },
         NoParameters,
         async (NoParameters) =>
         {
            return View();
         });
   }

   [HttpGet]
   public async Task<IActionResult> UserMenu()
   {
      return await Execute(
         new Role[] { Role.User },
         NoParameters,
         async (NoParameters) =>
         {
            return Redirect("/Media/Image");
         });
```

```
    }
}
```

Лістинг Illustre/Illustre/Controllers/MediaController.cs:

```
using Data.Contracts.Media;
using Data.Entities;
using Microsoft.AspNetCore.Mvc;
using Service;

namespace Illustre.Controllers;

public class MediaController : CommonController
{
    private readonly MediaService _mediaService;
    public MediaController(
        AccountService accountService,
        MediaService mediaService)
        : base(accountService)
    {
        _mediaService = mediaService;
    }

    [HttpGet]
    public async Task<IActionResult> ManageTags(ManageTagsRequest request)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin, Role.Editor },
            request,
            async (request) =>
            {
                var dto = request as ManageTagsRequest;
                dto!.TagsData = await _mediaService
                    .GetTags(dto.Skip, dto.SearchPattern);
                return View(dto);
            });
    }

    [HttpPost]
    public async Task<IActionResult> AddTag(AddTagRequest request)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin, Role.Editor },
            request,
```

```
        async (request) =>
        {
            var dto = request as AddTagRequest;
            var result = false.ToString().ToLower();

            if (ModelState.IsValid)
            {
                result = (await _mediaService.TryAddTag(dto!))
                    .ToString()
                    .ToLower();
            }

            var redirect = dto!.Action + $"?isFirstAttempt={result}";

            if (dto.ImageId is not null)
            {
                redirect += $"&imageId={dto.ImageId}";
            }

            return Redirect(redirect);
        });
}

[HttpPost]
public async Task<IActionResult> UpdateTagById(ManageTagModel model)
{
    return await Execute(
        new Role[] { Role.SuperAdmin, Role.Editor },
        model,
        async (model) =>
        {
            var dto = model as ManageTagModel;
            var result = false.ToString().ToLower();

            if (ModelState.IsValid)
            {
                result = (await _mediaService.TryUpdateTagById(dto!))
                    .ToString()
                    .ToLower();
            }

            return Redirect(GetManageTagsRedirect(result));
        });
}
```

```
    [HttpGet]
    public async Task<IActionResult> ManageImages(ManageImagesRequest
request)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin, Role.Editor },
            request,
            async (request) =>
            {
                var dto = request as ManageImagesRequest;
                dto!.ImagesData = await _mediaService
                    .GetImages(dto.Skip, dto.SearchPattern);
                return View(dto);
            });
    }

    [HttpPost]
    public async Task<IActionResult> AddImage(AddImageRequest request)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin, Role.Editor },
            request,
            async (request) =>
            {
                var dto = request as AddImageRequest;
                var result = false.ToString().ToLower();

                if (ModelState.IsValid)
                {
                    result = (await _mediaService.TryAddImage(dto!))
                        .ToString()
                        .ToLower();
                }

                return Redirect(GetManageImagesRedirect(result));
            });
    }

    [HttpPost]
    public async Task<IActionResult> UpdateImageById(ManageImageModel model)
    {
        return await Execute(
            new Role[] { Role.SuperAdmin, Role.Editor },
            model,
```

```
        async (model) =>
        {
            var dto = model as ManageImageModel;
            var result = false.ToString().ToLower();

            if (ModelState.IsValid)
            {
                result = (await _mediaService.TryUpdateImageById(dto!))
                    .ToString()
                    .ToLower();
            }

            return Redirect(GetManageImagesRedirect(result));
        });
}

[HttpGet]
public async Task<IActionResult> EditTags(EditTagsRequest request)
{
    return await Execute(
        new Role[] { Role.SuperAdmin, Role.Editor },
        request,
        async (request) =>
        {
            var dto = request as EditTagsRequest;
            dto!.TagsData = await _mediaService
            .GetEditableTags(dto.Skip, dto.SearchPattern, dto.ImageId ?? 0);
            return View(dto);
        });
}

[HttpPost]
public async Task<IActionResult> EditTagById(EditTagModel model)
{
    return await Execute(
        new Role[] { Role.SuperAdmin, Role.Editor },
        model,
        async (model) =>
        {
            var dto = model as EditTagModel;
            var result = false.ToString().ToLower();

            if (ModelState.IsValid)
            {
```

```
            result = (await _mediaService.TryEditTagById(dto!))
                .ToString()
                .ToLower();
        }

        return Redirect(GetEditTagsRedirect(result, dto!.ImageId));
    });
}

[HttpGet]
public async Task<IActionResult> EditImages(EditImagesRequest request)
{
    return await Execute(
        new Role[] { Role.SuperAdmin, Role.Editor },
        request,
        async (request) =>
        {
            var dto = request as EditImagesRequest;
            dto!.ImagesData = await _mediaService
            .GetEditableImages(dto.Skip, dto.SearchPattern, dto.TagId ?? 0);
            return View(dto);
        });
}

[HttpPost]
public async Task<IActionResult> EditImageById(EditImageModel model)
{
    return await Execute(
        new Role[] { Role.SuperAdmin, Role.Editor },
        model,
        async (model) =>
        {
            var dto = model as EditImageModel;
            var result = false.ToString().ToLower();

            if (ModelState.IsValid)
            {
                result = (await _mediaService.TryEditImageById(dto!))
                    .ToString()
                    .ToLower();
            }

            return Redirect(GetEditImagesRedirect(result, dto!.TagId));
        });
```

```
}

[HttpGet]
public async Task<IActionResult> Image()
{
    return await Execute(
        new Role[] { Role.User },
        NoParameters,
        async (NoParameters) =>
        {
            var array = NoParameters as object[];
            var cookie = array![CookieIndex] as string;
            var userId = await _accountService
                .TryGetAccountIdBySessionGuid(cookie!);

            if (userId is not null)
            {
                var image = await _mediaService.GetNextImage(userId!.Value);
                return View(image);
            }
            else
            {
                return RedirectToAction("Account", "Account");
            }
        },
        true);
}

[HttpGet]
public async Task<IActionResult> GetImage(int imageId)
{
    return await Execute(
        new Role[] { Role.User },
        imageId,
        async (model) =>
        {
            var array = model as object[];
            var cookie = array![CookieIndex] as string;
            var imageId = (int)array[DtoIndex];
            var userId = await _accountService
                .TryGetAccountIdBySessionGuid(cookie!);

            var result = await _mediaService.GetImage(
                userId!.Value,
```

```
            imageId);

        return View("Image", result);
    },
    true);
}

[HttpGet]
public async Task<IActionResult> SetReaction(SetReactionModel model)
{
    return await Execute(
        new Role[] { Role.User },
        model,
        async (model) =>
        {
            var array = model as object[];
            var cookie = array![CookieIndex] as string;
            var dto = array[DtoIndex] as SetReactionModel;
            var userId = await _accountService
                .TryGetAccountIdBySessionGuid(cookie!);

            if (userId is not null)
            {
                dto!.UserId = userId!.Value;

                await _mediaService.SetReaction(dto);

                return RedirectToAction("Image", "Media");
            }
            else
            {
                return RedirectToAction("Account", "Account");
            }
        },
        true);
}

[HttpGet]
public async Task<IActionResult> Search(SearchModel model)
{
    return await Execute(
        new Role[] { Role.User },
        model,
        async (model) =>
```

```
        {
            var array = model as object[];
            var cookie = array![CookieIndex] as string;
            var dto = array[DtoIndex] as SearchModel;
            var userId = await _accountService
                .TryGetAccountIdBySessionGuid(cookie!);

            if (userId is not null)
            {
                dto!.UserId = userId!.Value;

                var result = await _mediaService.GetImagePreviews(dto);

                result.SearchModel = dto;

                return View(result);
            }
            else
            {
                return RedirectToAction("Account", "Account");
            }
        },
        true);
}

[HttpGet]
public async Task<IActionResult> GetTags(int? skip)
{
    return await Execute(
        new Role[] { Role.User },
        skip,
        async (skip) =>
        {
            var array = skip as object[];
            var dto = array[DtoIndex] as int?;
            var cookie = array![CookieIndex] as string;
            var userId = await _accountService
                .TryGetAccountIdBySessionGuid(cookie!);

            var result = await _mediaService
                .GetTagsPreviews(userId!.Value, dto);

            result.SearchModel = new SearchModel()
            {
```

```
                Skip = dto ?? 0,
            };

            return View(result);
        },
        true);
    }

    private string GetManageTagsRedirect(string isFirstAttempt)
    {
        return $"/Media/ManageTags?isFirstAttempt={isFirstAttempt}";
    }

    private string GetManageImagesRedirect(string isFirstAttempt)
    {
        return $"/Media/ManageImages?isFirstAttempt={isFirstAttempt}";
    }

    private string GetEditTagsRedirect(string isFirstAttempt, int imageId)
    {
        return $"/Media/EditTags?isFirstAttempt={isFirstAttempt}&imageId="
            + imageId;
    }

    private string GetEditImagesRedirect(string isFirstAttempt, int tagId)
    {
        return $"/Media/EditImages?isFirstAttempt={isFirstAttempt}&tagId="
            + tagId;
    }
}
```

Решта файлів є наявною на магнітному носії.

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
"Розробка веб-додатку крос-платформеної системи пошуку зображень з використанням технології ASP.NET"
студентки групи 121-18-2 Дробот Анни Ярославівни

доц. Л. В. Касьяненко

ДОДАТОК В

Перелік файлів на диску

ПЕРЕЛІК ДОКУМЕНТІВ НА МАГНІТНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Дробот.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Дробот.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Дробот.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Дробот.ppt	Презентація кваліфікаційної роботи