Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: Розробка методів і програмного забезпечення для класифікації графічних зображень з використанням технологій .NET, WPF, WCF та Entity Framework

Виконав: студент II курсу
групи 1ПІ-18 м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Трач О. Ю.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д. І.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. КН Арсенюк І. Р.
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – Інженерія програмного забезпечення

УЗГОДЖЕНО
Директор ТОВ «Дельфи»
Бондар Н. П.
"____"_____2019 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
"____"_____2019 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Трачу Олександру Юрійовичу

1. Тема роботи – розробка методів і програмного забезпечення для класифікації графічних зображень з використанням технологій .NET, WPF, WCF та Entity Framework.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "__"_____2019 року №___

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: максимальна роздільна здатність зображення дорівнює 2560 на 1440 пікселів, максимальна кількість класів дорівнює 100, час виконання класифікації не більше 30 секунд.

4. Зміст розрахунково-пояснювальної записки: аналіз сучасного стану вирішення задачі та постановка завдання, розробка методів класифікації зображень, розробка програмного забезпечення, тестування системи, економічна частина.

5. Перелік графічного матеріалу: загальні відомості про роботу, метод класифікації графічних зображень з можливістю віднесення зображення одразу до кількох класів, процес пришвидшеного додавання нового класу, програмна реалізація, економічне обґрунтування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д. І., к.т.н., доц. кафедри ПЗ		
5	Бальзан М.В., к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану вирішення задачі	07.10.2018 – 27.10-2019	Вик.
2	Розробка методів класифікації зображень	28.10.2019 – 8.11.2019	Вик.
3	Розробка програмного забезпечення	9.11.2019 – 20.11.2019	Вик.
4	Тестування програмного забезпечення	21.11.2019 – 30.11.2019	Вик.
5	Економічна частина	1.12.2019 – 8.12.2019	Вик.

Студент _____ **Трач О.Ю.**

(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Катєльніков Д.І.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У магістерській кваліфікаційній роботі «Розробка методів і програмного забезпечення для класифікації графічних зображень з використанням технологій .NET, WPF, WCF та Entity Framework» розроблено метод класифікації графічних зображень з можливістю віднесення зображення одразу до кількох класів та поліпшений метод додавання нового класу.

В ході виконання роботи було проаналізовано існуючі методи класифікації зображень та існуючі аналоги. Обґрунтовано вибір засобів розробки програмного забезпечення: .NET, WPF, WCF, Entity Framework. Розроблено структуру системи, програмні модулі, протестовано правильність роботи системи.

ABSTRACT

In master's qualification thesis "Development of methods and software for image classification with use of .NET, WPF, WCF and Entity Framework technologies" the image classification method with ability of relating the image to several classes simultaneously and enhanced method of adding new class were developed.

During work conduction existing methods of image classification were analyzed. Means of software development were justified: .NET, WPF, WCF, Entity Framework. The structure of the system and program modules were developed, the correctness of system functioning was tested.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сьогодні великої популярності набувають електронні сховища інформаційних ресурсів. Такі сховища надають користувачам можливість зберігати у них інформацію у різних форматах та шукати її найбільш ефективними та простими методами. Одним з найбільш розповсюджених видів таких ресурсів є графічні зображення. Вони дають змогу представити інформацію у найбільш зручному для людського сприйняття вигляді.

Для ефективного пошуку інформації в електронному сховищі вона повинна бути класифікована. Класифікація графічних зображень може бути реалізована багатьма способами, проте найбільш зручним для користувача є додавання до зображень міток, що відповідають категоріям, до яких належать зображення.

Одним з ключових напрямків науки та техніки є полегшення та автоматизація людської діяльності. До такої діяльності можна віднести і класифікацію образів. Завдяки значному розвитку теоретичних та практичних аспектів у галузі штучного інтелекту доцільним є їх залучення для автоматизації класифікації зображень.

Хоча методи класифікації зображень і набули певного розвитку, вони є далекими від досконалості та потребують створення поліпшених варіантів. Крім того, реалізація таких методів у прикладному програмному забезпеченні ще не стала поширеною та потребує розвитку.

Таким чином, питання створення методів та програмного забезпечення для класифікації графічних зображень є актуальним.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є розширення функціональних можливостей процесу класифікації за рахунок додавання

можливості віднесення зображення одразу до кількох класів та пришвидшення налаштування системи класифікації за рахунок пришвидшення процесу додавання нового класу.

Основними завданнями дослідження є:

- провести аналіз існуючих методів і засобів для класифікації графічних зображень з метою визначення шляхів їх покращення;
- запропонувати нові:
    - методи класифікації графічних зображень з можливістю віднесення зображення одразу до кількох класів;
    - методи пришвидшеного додавання нового класу до нейромережі;
- розробити програмний засіб на основі запропонованих методів;
- провести тестування розробленого програмного засобу.

Об’єкт дослідження – процес класифікації образів.

Предмет дослідження – методи та засоби класифікації графічних зображень.

Методи дослідження. У процесі дослідження використовувались теорія штучного інтелекту, теорія машинного навчання, теорія розпізнавання образів, теорія комп’ютерного зору для розробки методу класифікації зображень, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Вперше запропоновано метод класифікації зображень, особливість якого полягає у застосуванні кластеру нейронних мереж, кожна мережа якого навчена на певний набір класів, що дає можливість відносити зображення одразу до кількох класів.

2. Вдосконалено процес додавання нового класу до нейронної мережі, який відрізняється від аналогів пришвидшеним етапом навчання за рахунок перенавчання окремих мереж у кластері замість перенавчання великої нейронної мережі, що дає можливість швидше додавати новий клас.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень розроблено алгоритми у складі програмного засобу для класифікації графічних зображень, що може бути застосований на підприємствах, де виникає така потреба, наприклад, у студіях веб-дизайну чи компаніях, що розробляють графічні інтерфейси користувача у складі програмного забезпечення.

Впровадження. Впровадження результатів досліджень підтверджуються відповідними актами та використовуються на таких підприємствах і організаціях:

- ТОВ «Дельфи» для організації зберігання графічних зображень при розробці програмного забезпечення.

Особистий внесок здобувача. У роботі «Розробка методів і програмного забезпечення для класифікації графічних зображень з використанням технологій AForge.NET і платформи .NET Framework» автору належать наступні результати: метод класифікації з віднесенням зображення одразу до кількох класів, метод пришвидшеного додавання класу до нейромережі, програмне забезпечення для класифікації графічних зображень.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися на міжнародній Internet-конференції «Електронні інформаційні ресурси: створення, використання, доступ», ВНТУ, Вінниця, 8-9 листопада 2019 року.

Публікації. Основні результати досліджень опубліковано в матеріалах 1 конференції.

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’ятьох розділів, висновків, списку літератури, що містить 30 найменувань, 5 додатків. Робота містить 58 ілюстрацій, 7 таблиць.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ВИРІШЕННЯ ЗАДАЧІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз стану вирішення задачі класифікації графічних зображень

Графічне зображення – двовимірне зображення, яке представлене у цифровому вигляді. В залежності від способу описання, зображення може бути растровим або векторним.

Растрові зображення – це зображення, що представляються як прямокутний двовимірний масив чисел, при цьому кожне число відповідає одному елементу зображення або пікселю. Ці масиви зберігаються у стисненому вигляді. Растрові зображення отримуються з допомогою різних пристроїв, таких як цифрові фотоапарати, цифрові відеокамери, сканери та інші, або синтезуються штучно засобами машинної графіки.

Векторні зображення – це зображення, що отримуються шляхом математичного описання елементарних геометричних об’єктів, які зазвичай називаються примітивами, таких як точки, лінії, криві Безьє, кола, многокутники.

Сьогодні люди майже кожного дня стикаються з зображеннями. Інколи виникає бажання зберегти те чи інше зображення, щоб потім його передивитись або поділитись ним із знайомими. Саме в таких випадках стають у нагоді системи зберігання та класифікації зображень. Такі системи дозволяють зручно та швидко додати зображення до спеціального сховища, яке дає можливість їх зберегти без втрати якості та з можливістю швидко відтворити їх на екрані. Ключовою функцією таких систем є класифікація зображень. Саме завдяки класифікації зображень стає можливим швидкий пошук та систематизація. Системи зберігання та класифікації зображень є популярними з декількох причин:

1. Популярність представлення інформації у вигляді зображень.
2. Наявність інфраструктури для швидкого обміну зображеннями.

3. Потреба зберігати окремі зображення для наявності швидкого доступу до них.

4. Потреба зберігати зображення з прив’язкою до окремих категорій, з якими ці зображення пов’язані, попередньо віднісши їх до цих категорій шляхом класифікації.

Саме ці причини є рушійною силою у створенні нових систем, які є більш досконалими ніж попередні. З кожним етапом розвитку таких систем задаються нові стандарти надійності, швидкодії та ергономічності. Проте неможливо створити ідеальну систему для класифікації зображень, а можливо лише нескінченно покращувати існуючі рішення для такого роду задач та вводити нові, які є більш ефективними, надійними та зручними[1]. Саме такими і є сучасні системи класифікації зображень. Вони є кращими за попередні системи, які були створені для вирішення такого роду задач, проте не є досконалими і деякі аспекти їхньої роботи можуть бути покращені.

Зокрема, до недоліків сучасних систем класифікації зображень можна віднести неможливість віднесення зображення одразу до кількох класів автоматичними засобами. Цей недолік ґрунтується на недосконалості методів автоматичної класифікації, які можуть відносити зображення лише до одного класу. Даний недолік може бути усунено шляхом розробки нових методів класифікації графічних зображень, які матимуть можливість віднести одне зображення одразу до кількох класів.

Окремо потрібно виділити такий недолік автоматичних засобів класифікації, що ґрунтуються на нейронних мережах, як повільний процес додавання нового класу. Він пов’язаний з тим, що коли додається новий клас до нейронної мережі, це змінює її структуру і як наслідок така мережа потребує потворного навчання. Цей недолік може бути усунено шляхом розробки нових методів навчання нейронних мереж, які матимуть більшу швидкість, аніж ті, що існують на даний момент.

Створювана система не матиме вказаних недоліків, а тому її розробка є актуальною.

1.2 Аналіз методів класифікації зображень

Задача класифікації – задача, в якій є множина об’єктів, розділених певним чином на класи. Задана кінцева множина об’єктів, для яких відомо, до яких класів вони відносяться. Ця множина називається вибіркою. Класова належність інших об’єктів є невідомою. Потрібно побудувати алгоритм, що здатний класифікувати довільний об’єкт з початкової множини.

Класифікувати об’єкт – означає, вказати номер або найменування класу, до якого належить даний об’єкт.

Класифікація об’єкта – номер або найменування класу, що видається алгоритмом класифікації в результаті його застосування до даного конкретного об’єкта.

Існують наступні типи класів:

- двокласова класифікація(найбільш простий у технічному відношенні випадок, який слугує основою для вирішення більш складних задач);
- багатокласова класифікація(коли число класів досягає багатьох тисяч, задача класифікації стає суттєво складнішою);
- класи, що не перетинаються;
- класи, що перетинаються(об’єкт може одночасно відноситися до кількох класів);
- нечіткі класи(потрібно визначати ступінь належності об’єкта кожному з класів, зазвичай це дійсне число від 0 до 1).

Класифікація зображень є окремим випадком класифікації образів.

Сьогодні існує декілька основних методів класифікації образів. До них належать:

1. Метод перебору виду об’єкта під різними кутами, масштабами, зміщеннями, тощо.

2. Метод знаходження контуру об’єкта та дослідження його властивостей(зв’язність, наявність кутів, тощо).

3. Метод використання нейронних мереж.

Метод перебору виду об’єкта потребує великої обчислювальної потужності та, як результат, не є досить розповсюдженим.

Метод знаходження контуру об’єкта та дослідження його властивостей також потребує великої обчислювальної потужності та, як результат, не є досить розповсюдженим.

Метод використання нейронних мереж потребує малої обчислювальної потужності та завдяки цьому є досить розповсюдженим. Топологія мережі, яка використовується для класифікації, характеризується тим, що кількість нейронів у вихідному шарі, як правило, дорівнює кількості класів, що визначаються. При цьому встановлюється відповідність між виходом нейронної мережі та класом, який він представляє. Коли мережі надається певний образ, на одному з її виходів повинна з’явитися ознака того, що образ належить цьому класу. В той же час на інших виходах повинна бути ознака того, що даному класу не належить. Якщо на двох чи більше виходах є ознака належності до класу, вважається, що мережа «не впевнена» у своїй відповіді. Однак, цей метод має декілька недоліків.

Перший недолік методу класифікації за допомогою нейронної мережі полягає у тому, що за допомогою цього методу дані можна віднести до одного класу одночасно. Цей недолік ґрунтується на принципах функціонування нейронних мереж. Однак, при класифікації зображень можуть виникати ситуації, коли зображення потрібно віднести одразу до кількох класів.

Другий недолік методу класифікації за допомогою нейронної мережі полягає у тому, що при великому розмірі мережі додавання нового класу потребує багато часу. Цей недолік ґрунтується на тому факті, що додавання нового класу до нейронної мережі змінює її структуру і, як наслідок, нейронна мережа потребує повторного навчання.

Отже, існуючі методи не є досконалими, а тому актуальною є розробка нових методів класифікації зображень, які будуть позбавлені недоліків існуючих методів.

1.3 Порівняльний аналіз аналогів

Сьогодні існує декілька систем для зберігання та класифікації зображень, які можна віднести до аналогів розроблюваної системи. Ці система володіють як певними перевагами, так і певними недоліками, які будуть розглянуті далі.

iCloud for Windows – додаток для операційної системи Windows, який дозволяє зберігати зображення у хмарному сховищі для подальшого доступу до них(рисунок 1.1). Додаток був розроблений компанією Apple та є частиною інфраструктури сервісів, які є у складі екосистеми цієї компанії[2].

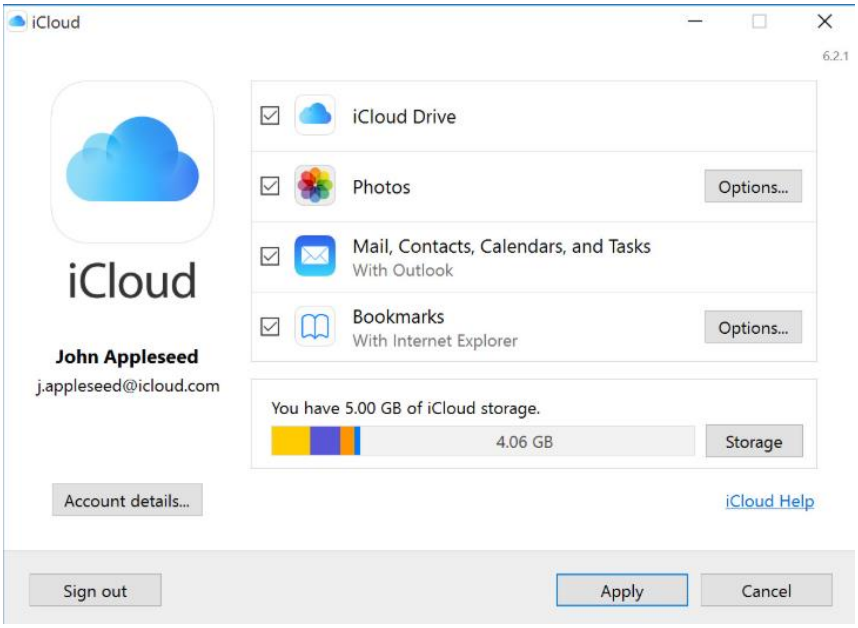


Рисунок 1.1 – Головне вікно додатку “iCloud for Windows”

Недоліком цього додатку є відсутність можливості автоматично класифікувати зображення(це можна зробити лише вручну), а також неможливість віднести зображення одразу до кількох категорій без дублювання даних.

OneDrive – хмарне сховище, створене у серпні 2007 року та кероване компанією Microsoft(рисунок 1.2). Дозволяє зберігати зображення та надає подальший доступ до них[3].

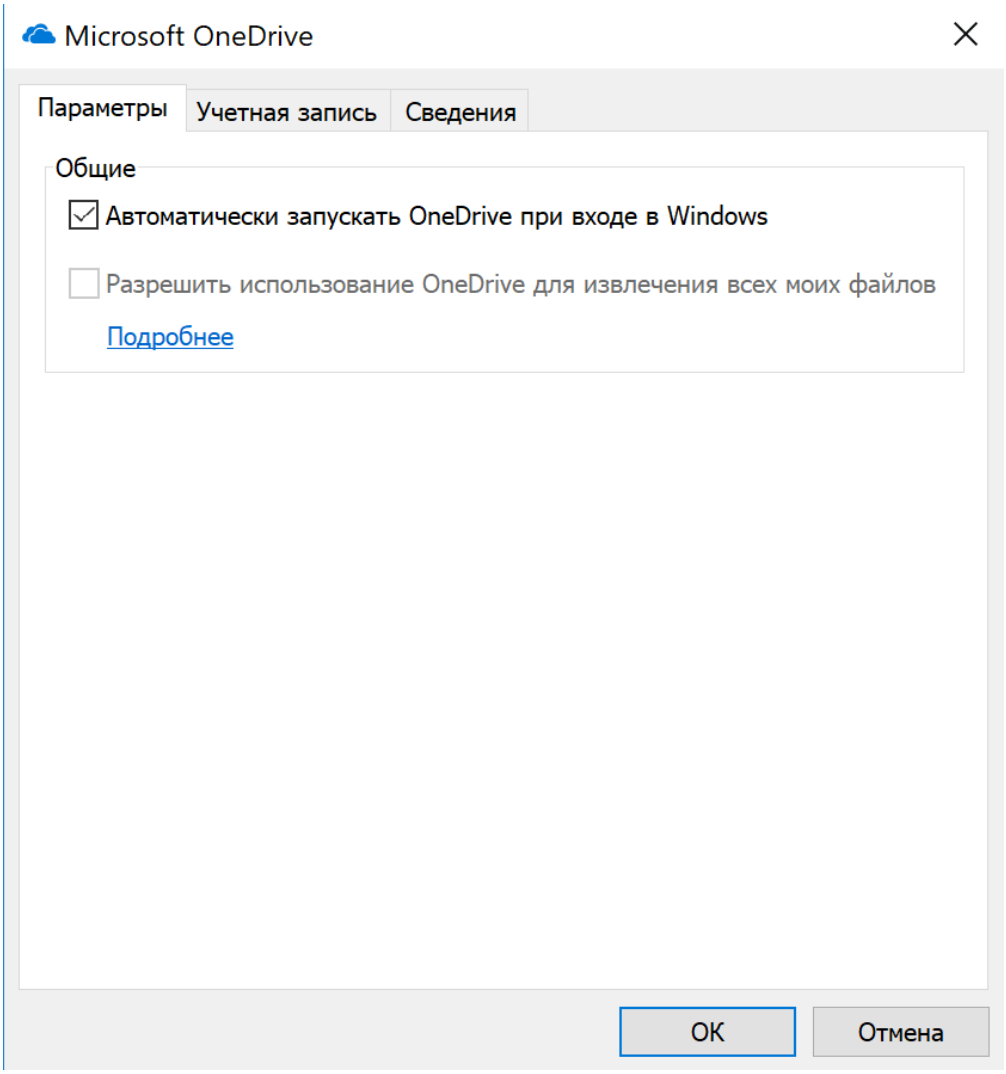


Рисунок 1.2 – Вікно програми “OneDrive”

Недоліком цього додатку є відсутність можливості автоматично класифікувати зображення(це можна зробити лише вручну), а також неможливість віднести зображення одразу до кількох категорій без дублювання даних.

Google Диск – файловий хостинг, який дозволяє зберігати зображення(рисунок 1.3). Розроблений компанією Google та має клієнтське програмне забезпечення для таких платформ як Windows, iOS, Android та інших[4].

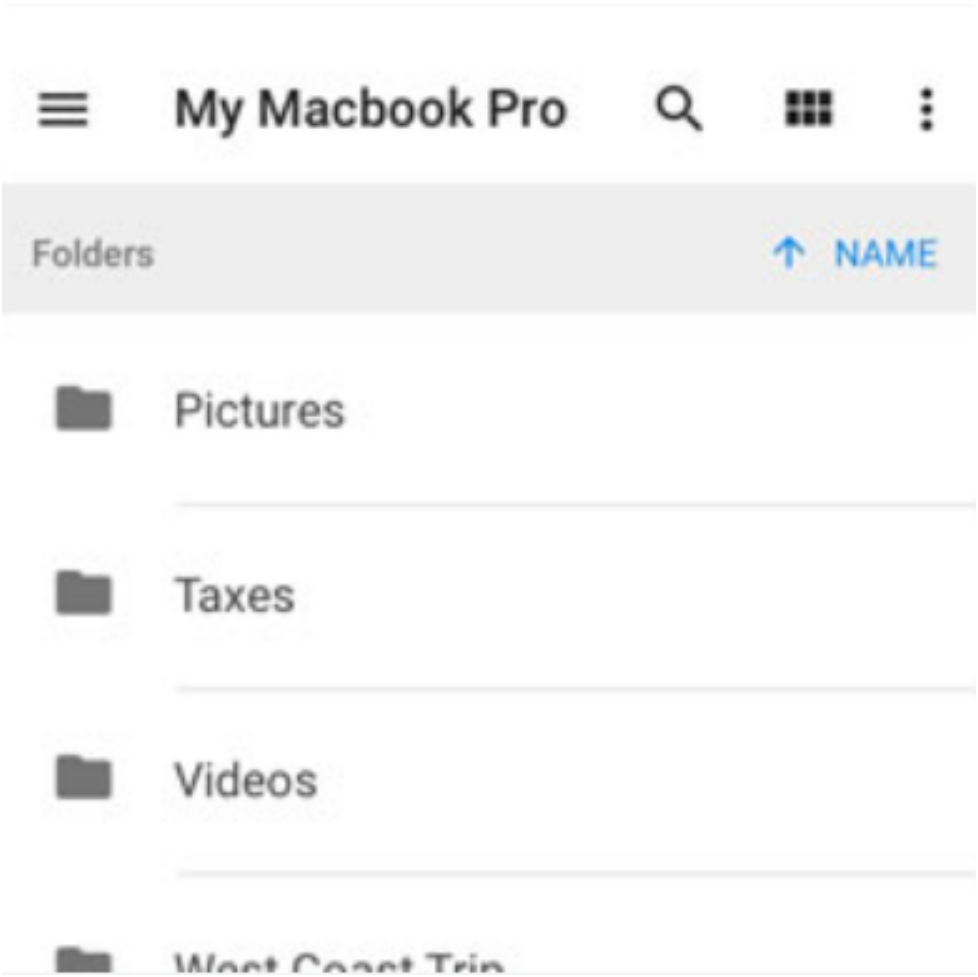


Рисунок 1.3 – версія додатку “Google Диск” для iOS

Недоліком цього додатку є відсутність можливості автоматично класифікувати зображення(це можна зробити лише вручну), а також неможливість віднести зображення одразу до кількох категорій без дублювання даних.

На основі проведеного аналізу аналогів було встановлено, що їх основними недоліками є відсутність можливості автоматично класифікувати зображення, а також неможливість віднести зображення одразу до кількох категорій без дублювання даних.

Для усунення цих недоліків було прийнято рішення розробити власну систему для класифікації графічних зображень, яка є актуальною та не матиме недоліків, які мають наведені аналоги.

1.4 Аналіз та обґрунтування вибору форматів зображень

Сьогодні існує доволі багато форматів, у яких можна зберігати зображення. Одними з найбільш відомих є BMP, GIF, JPEG та PNG.

Існує декілька факторів, які потрібно розглядати при виборі формату. До них належать стиснення та глибина кольору.

Є два типи стиснення: без втрат та з втратами. Стиснення без втрат означає, що зображення робиться меншим, але без втрати якості. Стиснення з втратами означає, що зображення робиться меншим, але з втратами якості.

Є також різні глибини кольорів: індексований колір та дійсний колір. Індексований колір означає, що зображення може зберігати лише обмежену кількість кольорів (зазвичай 256) у спеціальній таблиці. Дійсний колір означає, що можна зберігати тисячі кольорів, які не є заздалегідь визначеними.

BMP – формат зберігання растрових зображень, розроблений компанією Microsoft[5]. Є старим форматом. Зберігає зображення без втрат, проте майже не стискає його. Це означає, що зберігання у BMP призводить до дуже великого розміру файлу. Крім того, формат може мати як індексований колір, так і дійсний колір.

GIF – популярний растровий формат графічних зображень[6]. Він використовує стиснення без втрат. Розміри файлів є набагато меншими за BMP. Проте формат може зберігати лише індексований колір. Це означає, що може бути не більше 256 кольорів. Зображення у цьому форматі також можуть бути анімованими та мати прозорість.

JPEG – один з популярних растрових графічних форматів, що застосовується для зберігання фотозображень та подібних їм зображень[7]. Формат був розроблений, щоб зробити деталізовані фотографічні зображення якомога меншими шляхом прибирання інформації, яку людське око не помітить. В результаті це формат із втратами(рисунок 1.4). Він зберігає колір у дійсному форматі.

Рисунок 1.4 – Зображення у форматі JPEG зі ступенем стиснення, який збільшується від лівої до правої частини

PNG – растровий формат зберігання графічної інформації[8]. Він поєднує стиснення без втрат із дійсним кольором. Крім того, існує підтримка прозорості(рисунок 1.5). Зображення у цьому форматі мають дуже високу якість.

Рисунок 1.5 – Демонстрація прозорості у PNG

Отже, враховуючи особливості вказаних форматів було вирішено обрати PNG для зберігання зображень. Він має можливість задавати прозорість зображення та забезпечує високу якість зображення.

1.5 Постановка завдання

Метою дипломної роботи є розробка методів та програмного забезпечення для класифікації графічних зображень. Для цього необхідно:

1. Провести аналіз сучасного стану питання та обґрунтування задачі.
2. Розробити методи класифікації зображень:
   2.1 Розробити метод класифікації зображень з можливістю віднесення зображення одразу до кількох класів.
   2.2 Розробити метод пришвидшеного додавання нового класу до нейронної мережі.
3. Розробити систему:
   3.1 Розробити загальну структуру системи.
   3.2 Розробити структуру бази даних.
   3.3 Розробити структуру частини системи з графічним інтерфейсом.
   3.4 Розробити структуру інтерфейсу користувача.
   3.5 Розробити алгоритми роботи системи.
   3.6 Обґрунтувати вибір мови розробки системи.
   3.7 Обґрунтувати вибір середовища розробки системи.
   3.8 Розробити схему класів та компонентів системи.
   3.9 Написати програмний код для вирішення поставлених задач.
4. Провести тестування системи:
   4.1 Описати методики тестування.
   4.2 Розробити інструкцію тестування системи.
   4.3 Розробити інструкцію користувача системи.

1.6 Висновки

Отже, в даному розділі було проаналізовано сучасний стан вирішення задачі. Крім того, було проведено аналіз аналогів, на основі якого було доведено актуальність поточної розробки. Також була виконана постановка задач для виконання в ході кваліфікаційної роботи.

2 РОЗРОБКА МЕТОДІВ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

2.1 Розробка методу класифікації зображень з можливістю віднесення зображення одразу до кількох класів

Для досягнення здатності класифікації зображень з можливістю віднесення зображення одразу до кількох класів доцільно використати технологію нейронних мереж. Ця технологія потребує помірної обчислювальної потужності. Проте ця технологія має значний недолік: при використанні нейронної мережі для класифікації даних неможливо забезпечити здатність мережі відносити дані одразу до кількох класів.

Цей недолік можливо подолати шляхом застосування кластера нейронних мереж. Кожна нейронна мережа такого кластера відповідає за певний набір класів, які є взаємовиключними(рисунок 2.1).

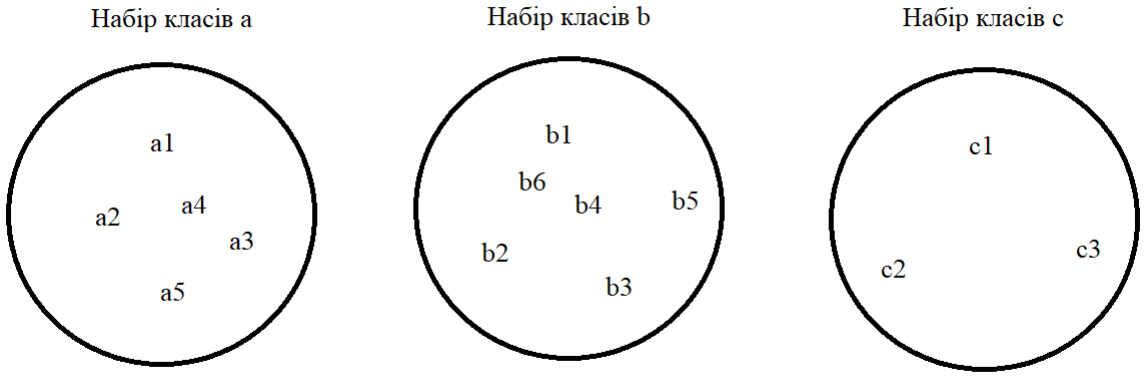


Рисунок 2.1 – Схематичне зображення наборів класів

Кожний набір класів містить класи, до яких може належати зображення, проте не одночасно до двох з набору. Так, зображення може належати до класів a1, b2, c3, може належати до a5, c3, може належати до b4 чи не належати до жодного класу. Наприклад, якщо набір класів a містить класи для класифікації літаків, набір класів b для класифікації танків, а набір c для класифікації вертольотів, то зображення, на якому зображений танк поруч з вертольотом

буде віднесене до класу з набору b(наприклад, b3 – важкий танк) та до класу з набору c(наприклад, c1 – транспортний вертоліт) і не віднесене до жодного класу з набору a, адже не містить жодного літака.

Кожна нейронна мережа містить вихідні нейрони, які відповідають класам(рисунок 2.2). При оцінці належності зображення до одного з класів на одному з вихідних нейронів з’явиться ознака належності, а на інших ознака неналежності. Якщо ж зображення не належить до жодного з класів, на які навчена конкретна мережа, то усі вихідні нейрони міститимуть ознаку неналежності.

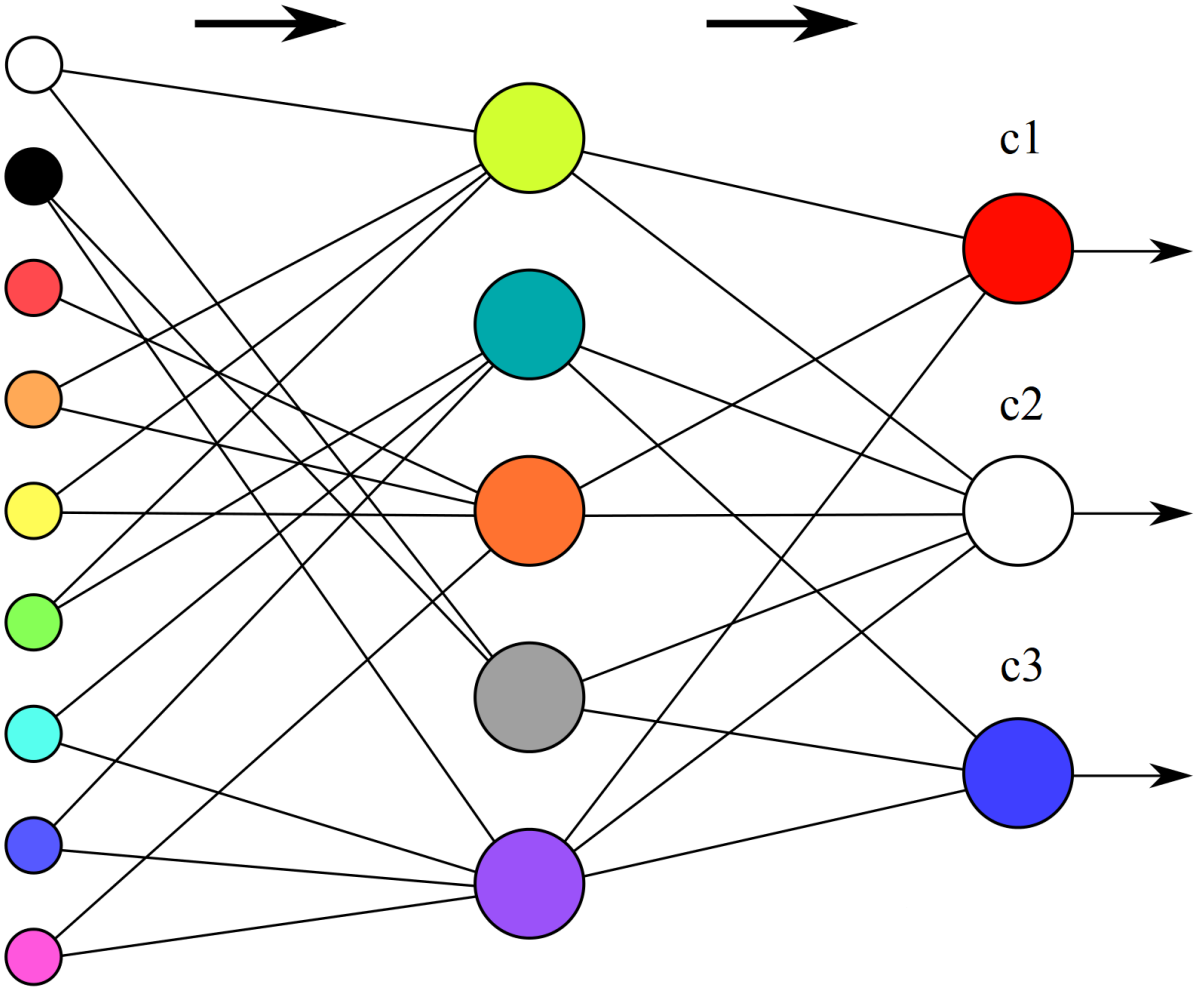


Рисунок 2.2 – Схематичне зображення нейронної мережі для класифікації

При об’єднанні кількох мереж для класифікації у кластер отримуємо структуру, здатну класифікувати зображення з віднесенням одразу до кількох класів(рисунок 2.3).

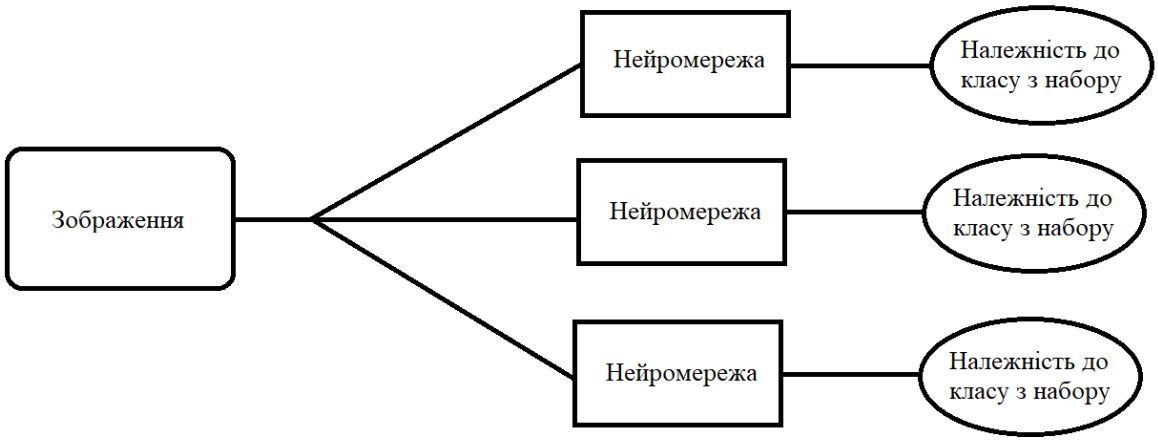


Рисунок 2.3 – Схематичне зображення процесу класифікації зображення з використанням кластеру нейронних мереж

При використанні такого кластеру зображення поступає на вхідні нейрони кожної зі штучних нейронних мереж, що містяться в ньому. Далі кожна нейронна мережа виконує класифікацію зображення в межах класів, на які вона навчена. Результат класифікації(належність до одного з класів або неналежність до жодного з них) подається на вихідні нейрони.

Результат класифікації зображення кластером нейронних мереж формується з результатів класифікації кожної нейронної мережі, яка входить до кластеру. Наприклад, якщо подати на вхід кластеру зображення, що містить мобільний телефон та ручку, нейронні мережі, які містять класифікацію телефонів та канцелярських виробів, дадуть відповідні результати, а інші мережі не дадуть жодного результату. В результаті зображення буде класифіковане як те, що належить до категорій ручок та мобільних телефонів.

2.2 Розробка методу пришвидшеного додавання нового класу до нейронної мережі

Нейронні мережі, що використовуються для класифікації мають недолік: при додаванні нового класу мережа потребує повторного навчання. Цей недолік ґрунтується на тому, що при додаванні нового класу додаються нові нейрони. Таким чином змінюється структура нейронної мережі і, як результат, вона потребує повторного навчання. Повторне навчання є тим довшим, чим більшим є набір класів, який містить нейронна мережа. Під час класифікації зображень множина класів є доволі великою, а тому додавання нового класу до нейронної мережі є довгим за рахунок повільного процесу повторного навчання через великі розміри нейронної мережі(рисунок 2.4).

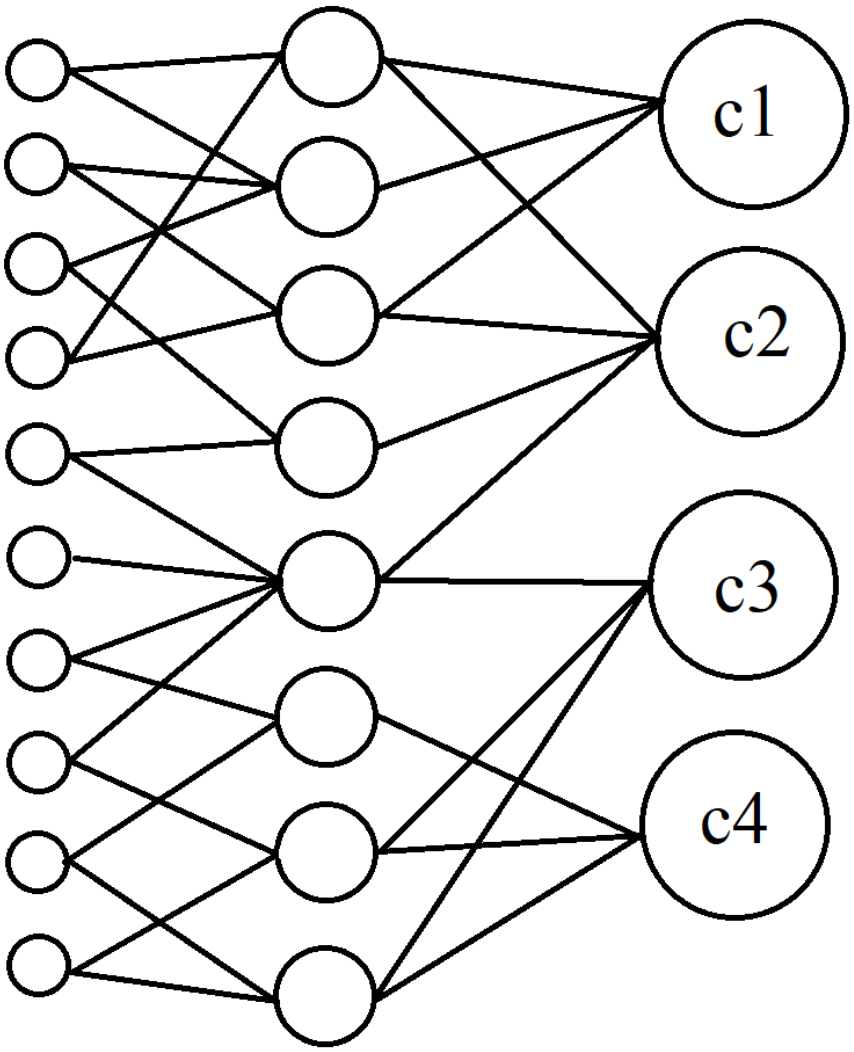


Рисунок 2.4 – Схематичне зображення великої нейронної мережі

Цей недолік можливо усунути шляхом розділення великої нейронної мережі на декілька менших нейронних мереж з наступним їх об’єднанням у кластер(рисунок 2.5).

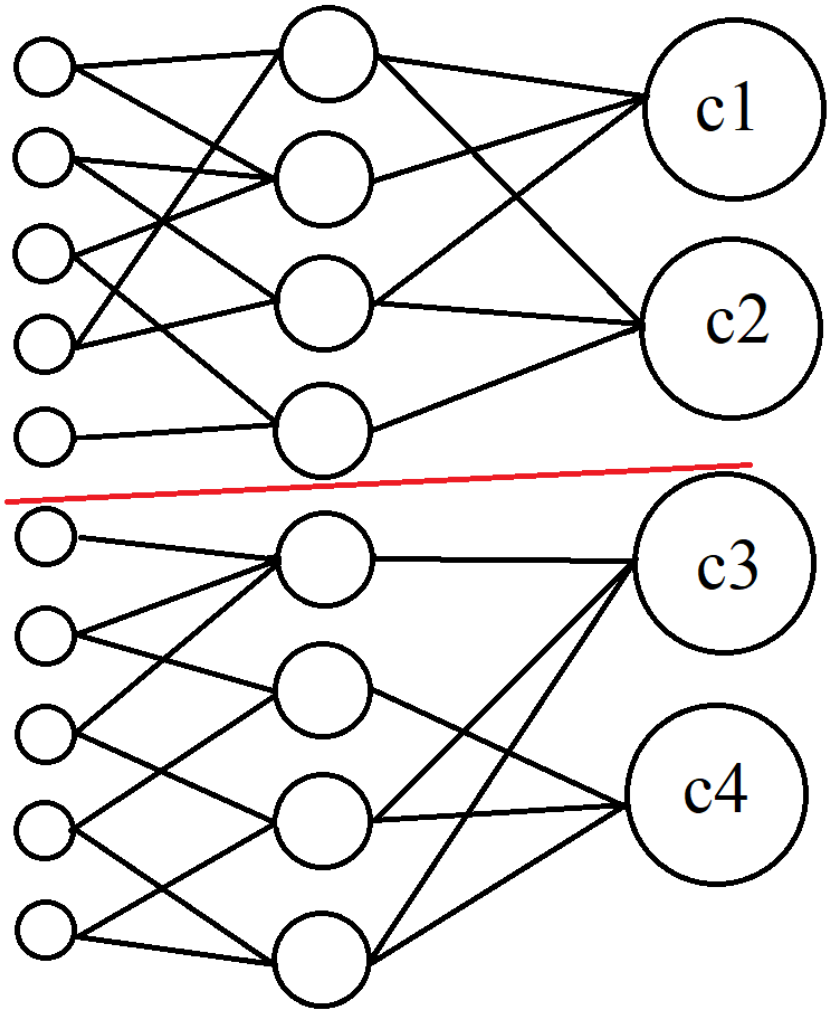


Рисунок 2.5 – Схематичне зображення великої нейронної мережі, розділеної на менші мережі

Таким чином процес додавання нового класу зводиться до додавання класу до невеликої нейронної мережі з кластеру та подальшого її перенавчання, яке є швидшим, ніж перенавчання великої нейронної мережі(рисунок 2.6).

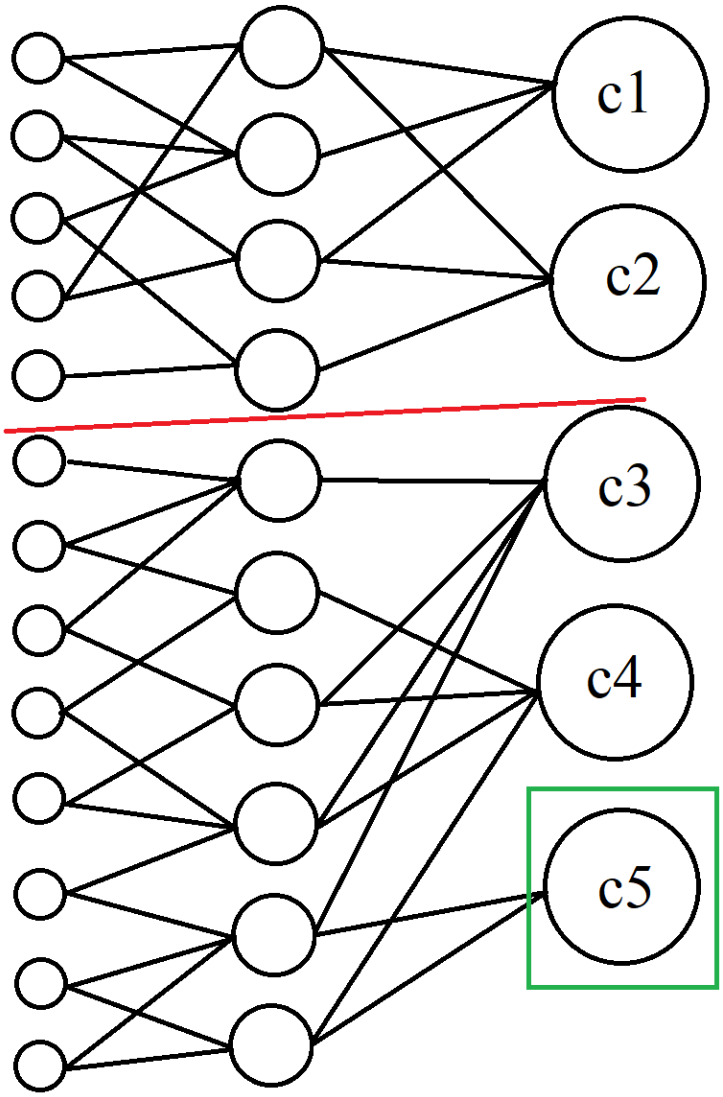


Рисунок 2.6 – Схематичне зображення менших нейронних мереж з доданим класом

Отже, процес додавання нового класу пришвидшується завдяки перенавчанню невеликої нейронної мережі, до якої було додано новий клас, в той час як інші мережі у кластері не змінюються.

2.3 Висновки

Отже, було розроблено метод класифікації зображень з можливістю віднесення зображення одразу до кількох класів. Крім того, було розроблено метод пришвидшеного додавання нового класу до нейронної мережі.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка загальної структури системи

Для створення системи було вирішено застосувати таку загальну архітектуру, що з базою даних, яка зберігає зображення, взаємодіє сервіс. Цей сервіс отримує повідомлення від бібліотеки, яка знаходиться на стороні клієнта. У свою чергу бібліотека взаємодіє з додатком, що має графічний інтерфейс користувача. Загальна структура системи зображена на рисунку 3.1.

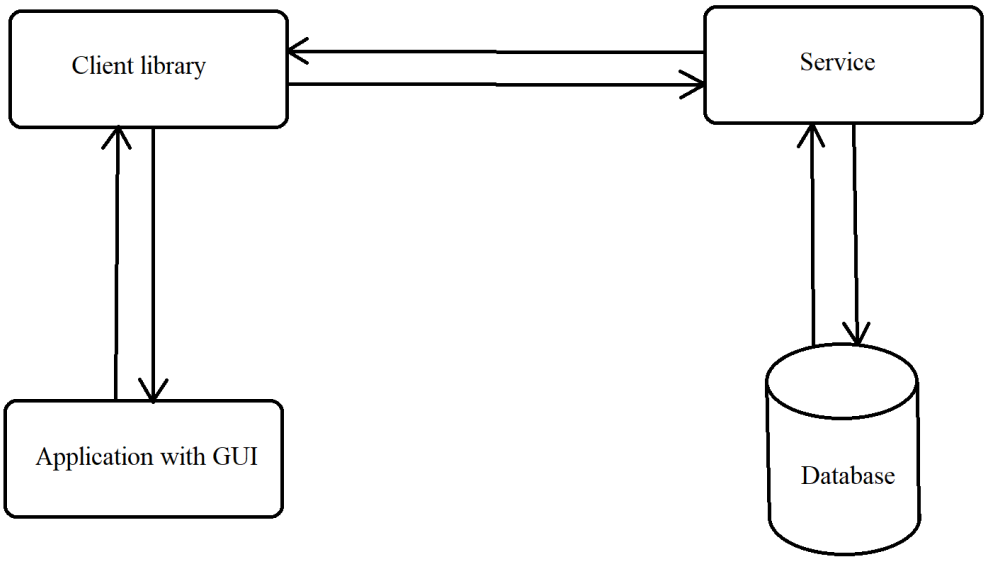


Рисунок 3.1 – Загальна структура системи

Така архітектура дає декілька переваг. По-перше, розміщення бази даних та сервісу на окремій машині дозволяє централізувати зберігання зображень. Як результат, декілька клієнтів може працювати зі сховищем зображень. Наприклад, один дизайнер може створити зображення, зберегти його до бази даних, а інший може його переглянути і дати якусь оцінку роботі першого. Крім того, таке розділення додає надійності системі, адже якщо машина з клієнтом вийде з ладу, можна буде використовувати іншу. Для більшої надійності можна розмістити сервіс та сервер бази даних на різних машинах.

По-друге, використання окремої бібліотеки для взаємодії із сервісом дає змогу написання різних версій додатків з графічним інтерфейсом користувача без необхідності копіювати код роботи із сервісом.

3.2 Розробка структури бази даних

Для того, щоб розробити структуру бази даних, потрібно виконати декілька дій.

По-перше, потрібно розробити універсальне відношення. Універсальним називається відношення проектованої бази даних, яке містить у собі атрибути, що представляють інтерес і мають структуру, у якій кожний кортеж складається з атомарних та непустих значень[9].

Атрибут – це характеристика певного інформаційного об’єкта бази даних.

З використанням універсального відношення пов’язані три специфічні проблеми:

1. Проблема, обумовлена необхідністю включення нових кортежів до бази даних.

2. Проблема, пов’язана з оновленням бази даних.

3. Проблема, обумовлена необхідністю видалення кортежів з бази даних.

Ці проблеми називаються аномаліями вставки, оновлення та видалення, оскільки під аномалією розуміють відхилення від норми.

До універсального відношення потрібно внести атрибути, що описують наступні інформаційні об’єкти: зображення, мітка.

Перерахуємо атрибути вищеназваних об’єктів:

1. Зображення(Ідентифікатор зображення, назва зображення, вміст зображення).

2. Мітка(Ідентифікатор мітки, назва мітки).

У таблиці 3.1 наведено перелік атрибутів універсального відношення.

Таблиця 3.1

Перелік атрибутів універсального відношення

№	Назва атрибута	Ім’я поля	Коментар
1	Ідентифікатор зображення	intId	Ідентифікатор зображення
2	Назва зображення	nvcImageName	Назва зображення
3	Вміст зображення	vbImageContent	Вміст зображення
4	Ідентифікатор мітки	intId	Ідентифікатор мітки
5	Назва мітки	nvcTagName	Назва мітки

Оскільки всі перераховані в таблиці атрибути є незалежними, тобто значення одних з них не можуть бути обчислені за значеннями інших, то всі вони можуть бути включеними в склад універсального відношення[10].

Універсальне відношення R(Ідентифікатор зображення, назва зображення, вміст зображення, ідентифікатор мітки, назва мітки). Ступінь універсальної множини – 5.

По-друге, потрібно розробити ER-модель предметної області. Модель «сутність-зв’язок» (ER-модель) – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. ER-модель – це мета-модель даних, тобто засіб опису моделей даних. ER-модель є одною з найпростіших візуальних моделей. Вона дозволяє осягнути структуру об’єкта «крупними мазками», в загальних рисах.

Сутність – це збірне поняття, деяка абстракція реально існуючого об’єкта, процесу, явища чи деякого уявлення про об’єкт. Хоча термін «сутність» найбільш вживаний, потрібно розрізняти поняття «типу сутності» та «екземпляру сутності». Поняття «тип сутності» відноситься до набору однорідних особистостей, предметів, подій або ідей, виступаючих як ціле. «Екземпляр сутності» відноситься до конкретної речі в наборі. Наприклад, типом сутності може бути МІСТО, а екземпляром – Київ, Львів і т. д.

Для представлення структури й обмеження реального світу використовуються моделі типу «суть-зв'язок» або ER-моделі.

Для кожної суті необхідно вибрати атрибут чи сукупність атрибутів, що однозначно ідентифікують її і можуть виступати ключем визначеного відношення. Перерахуємо суті, які використовуються в заданій предметній області та визначимо ключі (атрибут чи групу атрибутів), які однозначно ідентифікують екземпляри даних сутей[11]:

1. Зображення(Ідентифікатор зображення).

2. Мітка(Ідентифікатор мітки).

Зв’язок сутностей «Зображення-Мітка» характеризується типом N:N та класом належності «Необов’язковий» з обох сторін. Одне зображення може мати багато міток, а мітка може мати багато зображень. Клас належності вказує на те, що зображення може мати або не мати мітки, а мітка може мати або не мати зображення[12]. Діаграму зображено на рисунку 3.2.

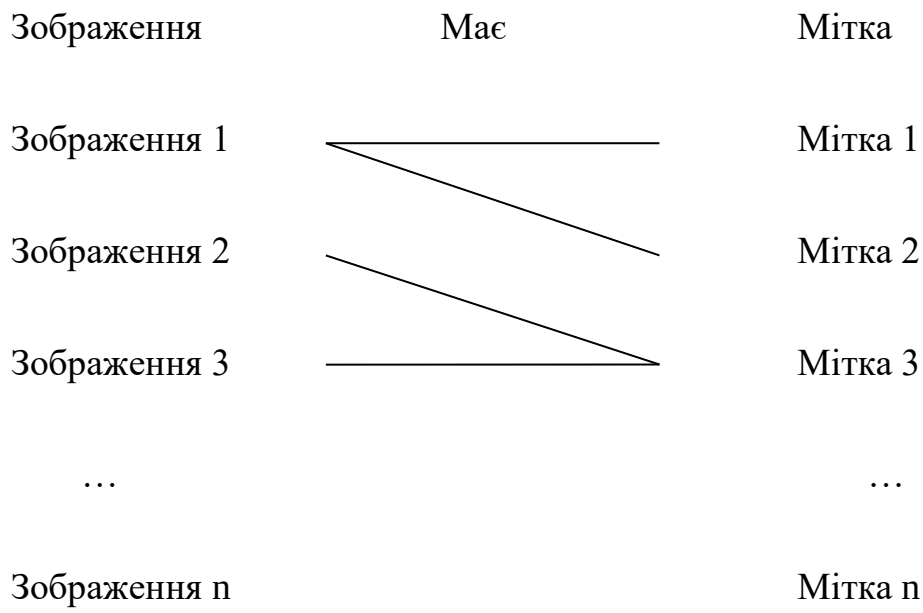


Рисунок 3.2 – ER-діаграма екземплярів сутностей «Зображення-Мітка»

Модель ER-діаграм екземплярів сутностей представлено у таблиці 3.2.

Таблиця 3.2

Зв’язки екземплярів сутностей ER-моделі бази даних

Ім’я суті 1	Ім’я суті 2	Тип зв’язку	Ім’я зв’язку	Клас належності
Зображення	Мітка	N:N	Має	Необов.; Необов.

На основі даних таблиці 3.2 можна побудувати ER-діаграму типів предметної області (рис. 3.3).

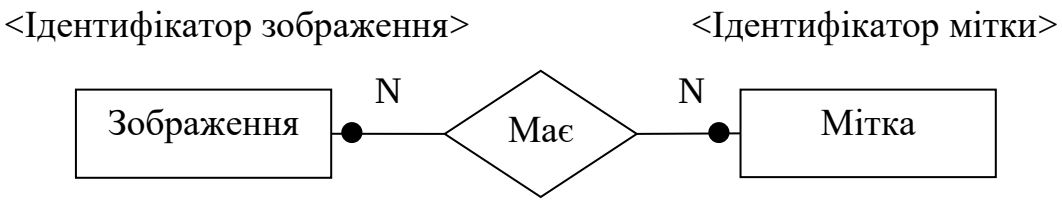


Рисунок 3.3 – ER-діаграма типів предметної області

Дана діаграма повністю відображає зв’язки між типами сутностей.

По-третє, потрібно спроектувати нормалізовані відношення. Нормалізація – це процес організації інформації в базі даних, що включає створення таблиць і встановлення відносин між ними відповідно до правил, які забезпечують захист даних і роблять базу даних більш гнучкою, усуваючи надмірність і неузгоджені залежності[13].

Надмірність даних призводить до непродуктивного витрачання вільного місця на диску і ускладнює обслуговування баз даних. Наприклад, якщо дані, що зберігаються в кількох місцях, буде потрібно змінити, в них доведеться внести одні і ті ж зміни у всіх цих місцях.

Відношення знаходиться в першій нормальній формі (1НФ) тоді і тільки тоді, коли в будь-якому допустимому значенні відношення кожен його кортеж містить тільки одне значення для кожного з атрибутів.

Відношення знаходиться в другій нормальній формі (2НФ) тоді і тільки тоді, коли воно знаходиться в першій нормальній формі і кожен неключовий атрибут функціонально повно залежить від її потенційного ключа.

Відношення знаходиться в третій нормальній формі (3НФ) тоді і тільки тоді, коли воно знаходиться в другій нормальній формі, і відсутні транзитивні функціональні залежності неключових атрибутів від ключових.

Потрібно зазначити, щоб транзитивні залежності є загалом коректними, але в зв'язку з надлишковістю їх використання у процесі проектування не потрібне. В зв'язку з цим транзитивні залежності потрібно виключати з набору перед початком проектування.

Відношення знаходиться в нормальній формі Бойса-Кодда, якщо між ключами-кандидатами немає функціональної залежності.

Проводимо нормалізацію за алгоритмом та визначеннями нормальних форм. Універсальне відношення R розбивається на такі відношення:

R1(<Ідентифікатор зображення>, назва зображення, вміст зображення);

R2(<Ідентифікатор мітки>, назва мітки);

Оскільки кортежі не повторюються, всі атрибути атомарні (1НФ), всі ключі прості (2НФ), відсутні транзитивні залежності (3НФ), немає функціональних залежностей між ключами-кандидатами, то всі відношення знаходяться у НФБК.

По-четверте, потрібно одержати попередні відношення методом «сутність-зв’язок». Зв’язки будуються за певними правилами залежно від класу належності відношень та типу зв’язку:

1. Якщо ступінь бінарного зв’язку дорівнює 1:1 і клас належності обох сутностей є обов’язковим, то потрібно лише одне відношення, первинним ключем якого може бути ключ будь-якого з двох відношень[14].

2. Якщо ступінь бінарного зв’язку дорівнює 1:1 і клас належності однієї сутності є обов’язковим («О»), а другої не обов’язковим («НО»), то необхідно побудувати два відношення. На кожну сутність потрібно виділити одне відношення, при цьому ключ сутності повинен слугувати первинним ключем

для відповідного відношення. Крім того ключ сутності, для якої клас належності є «НО», додається в якості атрибута у відношення, що виділене для сутності з обов’язковим класом належності.

3. Якщо ступінь бінарного зв’язку дорівнює 1:1 і клас належності обох сутностей є «НО», то необхідно використовувати три відношення: по одному для кожної сутності, ключі яких слугуватимуть в якості первинних у відповідних відношеннях та одного відношення для зв’язку. Відношення, що виділяється для зв’язку буде мати по одному ключу сутності від кожної сутності.

4. Якщо ступінь бінарного зв’язку дорівнює 1:N та клас належності N-зв’язної сутності є «О», то достатнім є використання двох відношень по одному на кожну сутність за умови, що ключ кожної сутності слугує в якості первинного ключа для відповідного відношення. Додатково ключ однозв'язної сутності повинен бути добавлений, як атрибут у відношення, що відводиться для N-зв`язної сутності.

5. Якщо ступінь бінарного зв'язку 1:N і клас належності багатозв’язної сутності є необов'язковим, то необхідно формувати три відношення: по одному для кожної сутності з відповідними ключами і одне для зв'язку аналогічно правилу №3.

6. Якщо ступінь бінарного зв'язку M:N, то для зберігання даних потрібні три відношення: по одному для кожної суті, причому ключ кожної суті служить як первинний ключ для відповідного відношення, та одного відношення для зв'язку. Зв'язок повинен мати серед своїх атрибутів ключі кожної зі зв'язних сутей.

Згідно з рисунком 3.3, усі відношення відповідають правилу 6.

Згідно з цим правилом, використовуючи множину атрибутів з таблиці 3.1 та ER-діаграму типів з рисунку 3.3, можна представити попередні відношення, що подані у таблиці 3.3.

Таблиця 3.3

Попередні відношення предметної області

Ім’я зв’язку	Правило	Попередні відношення	Додаткові атрибути
Має	6	R1(Ідентифікатор зображення)	Назва зображення, вміст зображення
		R2(Ідентифікатор мітки)	Назва мітки

Після застосування правила 6:

1. R1(<Ідентифікатор зображення>, назва зображення, вміст зображення).

2. R2(<Ідентифікатор мітки>, назва мітки).

3. R3(<Ідентифікатор зображення, ідентифікатор мітки>).

Отже, попередні відношення відповідають правилам методу «суть-зв’язок».

По-п’яте, потрібно нормалізувати відношення методом декомпозиції. Метод декомпозиції часто використовується для проектування великих баз даних.

Метод декомпозиції має наступний алгоритм:

1. Створення універсального відношення бази даних.

2. Визначення способів і типів зв’язку між атрибутами універсального відношення.

3. Визначення того, чи знаходиться розглянуте відношення в НФБК. Якщо так, проектування закінчується, якщо ні – відношення розбивається на два відношення.

4. Повторення кроків 2 та 3 для кожного нового відношення, отриманого в результаті декомпозиції.

При виконанні декомпозиції зберігається множина вихідних функціональних залежностей між атрибутами і виконується зворотність. Зворотність означає можливість відновлення вихідної схеми. Функціональні

залежності відображають зв'язки між атрибутами, які властиві реальному об'єкту[15].

Згідно з алгоритмом декомпозиції попереднє універсальне відношення виглядає так: R (<Ідентифікатор зображення, ідентифікатор мітки, ідентифікатор зображення, ідентифікатор мітки>, назва зображення, вміст зображення, назва мітки).

Після декомпозиції попереднього універсального відношення отримаємо відношення R1 і R2:

1. R1(<Ідентифікатор зображення>, назва зображення, вміст зображення).

2. R2(<Ідентифікатор мітки, ідентифікатор зображення, ідентифікатор мітки >, назва мітки).

Відношення R1 знаходиться в НФБК. Проводимо декомпозицію відношення R2:

1. R3(<Ідентифікатор мітки>, назва мітки).

2. R4(<Ідентифікатор зображення, ідентифікатор мітки>).

Функціональні залежності між атрибутами універсального відношення зображені на рисунку 3.4.

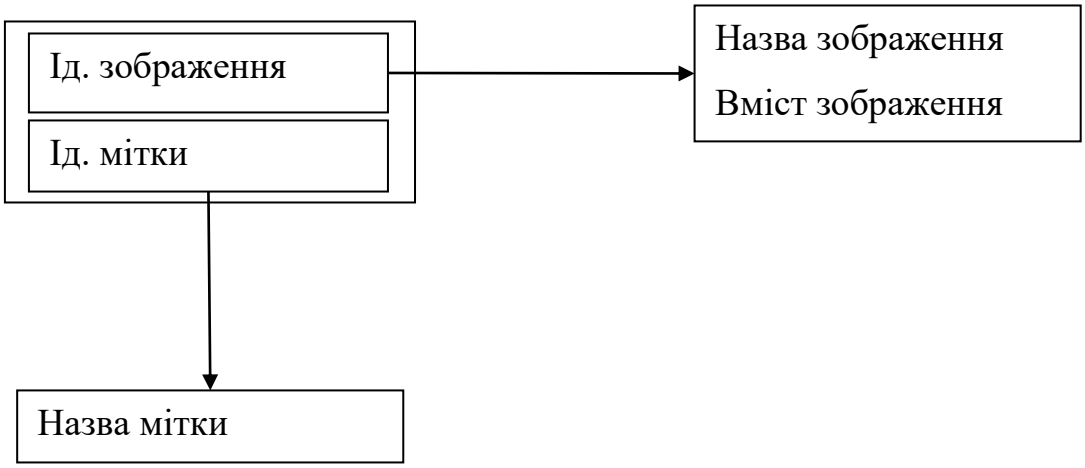


Рисунок 3.4 – Діаграма функціональної залежності

Отже, відношення, що були отримані в результаті декомпозиції відповідають НФБК та мають мінімальний набір функціональних залежностей.

По-шосте, потрібно оцінити спроектовані НФБК-відношення. Після отримання кінцевих відношень, можна зробити такі висновки:

1. Кожна функціональна залежність не повторюється більше одного разу.

2. Набір функціональних залежностей є мінімальним.

Аналіз відношень показує, що не можна вказати серед них жодного, всі атрибути якого були б підмножиною атрибутів іншого відношення. Крім того, неможливо об’єднати будь-які два відношення так, щоб у результаті були отримані всі атрибути третього відношення. Тобто, жодне з відношень не є надлишковим, що свідчить про правильність проведення проектування[16].

Після застосування методу декомпозиції отримано такі кінцеві відношення:

1. R1(<Ідентифікатор зображення>, назва зображення, вміст зображення).

2. R3((<Ідентифікатор мітки>, назва мітки).

3. R4(<Ідентифікатор зображення, ідентифікатор мітки>).

3.3 Розробка структури додатку з графічним інтерфейсом

При створенні додатку з графічним інтерфейсом було використано архітектурний шаблон MVVM(Model-View-ViewModel). Цей шаблон поділяє систему на три частини: модель, представлення, модель представлення. MVVM використовується для розділення моделі та її представлення, що є необхідним для їх зміни окремо одне від одного. Наприклад, розробник задає логіку роботи з даними, а дизайнер відповідно працює з інтерфейсом користувача[17].

MVVM зручно використовувати замість класичного MVC та йому подібних у випадках, коли у платформі, на якій ведеться розробка, присутнє “зв’язування даних”. У шаблонах проектування MVC/MVP зміни у користувацькому інтерфейсі не впливають безпосередньо на модель, попередньо йдуть через контроллер чи презентер. В таких технологіях як WPF є концепція “зв’язування даних”, яка дозволяє зв’язувати дані з візуальними

елементами в обидві сторони. Отже, при використання цього прийому застосування моделі MVC стає вкрай незручним через те, що прив’язка даних до представлення напряму не вкладається в концепцію MVC/MVP.

Модель – це частина, яка представляє собою логіку роботи з даними та опис фундаментальних даних, що необхідні для роботи додатку[18].

Представлення – це графічний інтерфейс, тобто вікно, кнопки та інше. Представлення підписане на подію зміни значень властивостей або команд, які надаються моделлю представлення. У разі, якщо в моделі представлення змінилася яка-небудь властивість, то вона сповіщає усіх підписників про це, і представлення, у свою чергу, робить запит на отримання оновленого значення з властивості моделі представлення. У разі, якщо користувач впливає на який-небудь елемент інтерфейсу, представлення викликає відповідну команду, надану моделлю представлення.

Модель представлення – це, з однієї сторони, абстракція представлення, а з іншої, надає обгортку для даних з моделі, які підлягають зв’язуванню. Тобто, воно містить модель, яка перетворена на представлення, а також містить у собі команди, якими може користуватися представлення, щоб впливати на модель.

3.4 Розробка структури інтерфейсу користувача

Під час роботи користувача з системою вона буде зосереджена у двох вікнах.

Перше вікно є головним та містить мініатюри зображень, доступних у базі даних. Крім того, вікно має меню, яке дозволяє завантажити файл з диску, завершити роботу додатку, внести зміни до наявних міток та шукати зображення за обраними мітками. Структурна схема головного вікна зображена на рисунку 3.5.

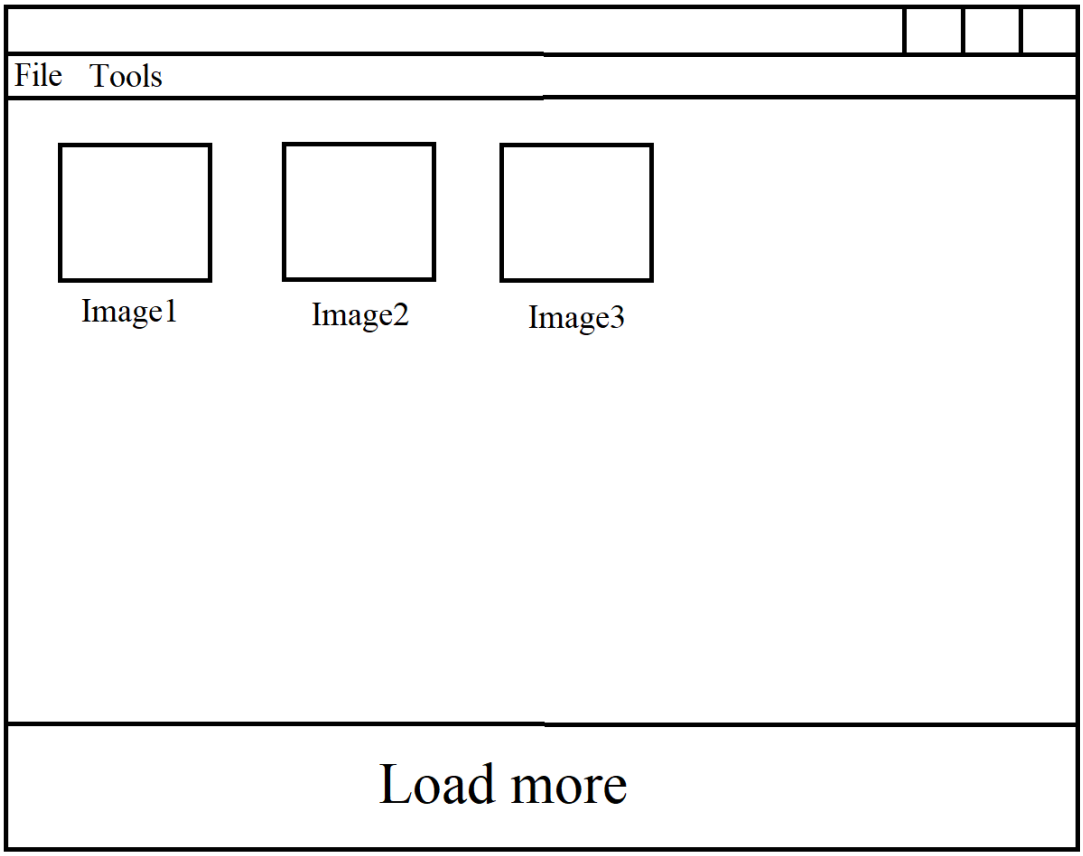


Рисунок 3.5 – Структурна схема головного вікна додатку

Друге вікно з’являється при натисканні користувача на одну з мініатюр та дозволяє проводити різні дії над зображенням. Вікно має меню, яке дозволяє замінити вміст зображення шляхом завантаження іншого файлу з диску. Також воно містить ім’я зображення, яке можна змінити, набір міток зображення, який теж можна змінити, вміст зображення та кнопки в нижній частині, які дозволяють додати зображення до бази даних, оновити або видалити його. Структурна схема вікна для проведення дій над зображенням зображена на рисунку 3.6.

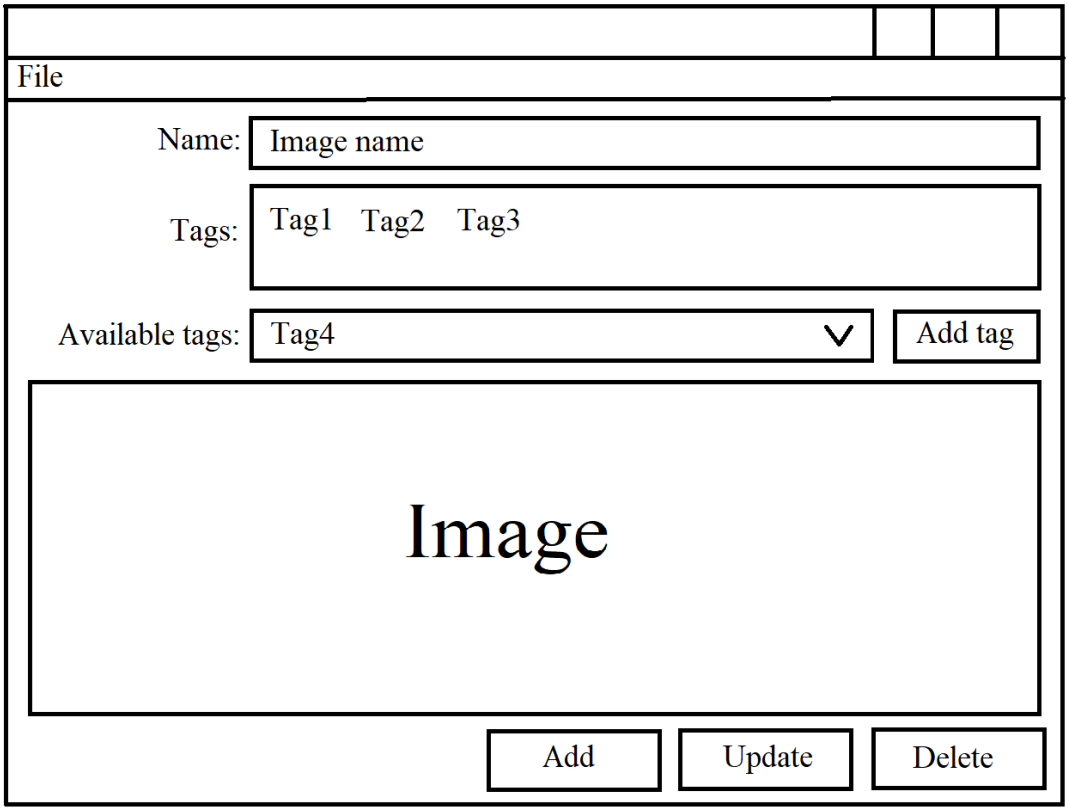


Рисунок 3.6 – Структурна схема вікна для проведення дій над зображенням

Отже, було розроблено структуру інтерфейсу користувача, який дозволить виконувати усі необхідні дії над зображеннями.

3.5 Розробка основних алгоритмів роботи системи

Серед великої кількості алгоритмів у системі слід звернути увагу на декілька основних. До них належать:

1. Алгоритм перевірки двох масивів міток на однаковість.
2. Алгоритм додавання зображення до бази даних.
3. Алгоритм отримання мініатюр з бази даних.

Алгоритм порівняння двох масивів міток використовується у сервісі для порівняння масиву міток, за яким здійснювався пошук від час попереднього запиту, та масивом міток, який містить мітки, за якими потрібно здійснювати пошук під час поточного запиту. Якщо вони співпадають, то повертаються наступні мініатюри з такими мітками, кількість котрих можна налаштувати. Наприклад, можна отримувати по 5 мініатюр або по 10. Якщо ж вони не

співпадають, то робиться висновок, що здійснюється пошук за новим набором міток і повертаються перші 5 або 10 мініатюр (або скільки потрібно) і при наступному запиті повертаються наступні 5 або 10 мініатюр при умові, що набір міток не зміниться.

Сам алгоритм порівняння працює наступним чином:

1. Порівнюється довжина масивів. Якщо вона не однакова, то масиви точно не співпадають. Якщо ж вона є однаковою, то алгоритм продовжує виконання.

2. Далі кожна мітка з першого масиву шукається у другому масиві. Якщо мітка з першого масиву не є присутня у другому, то вони не однакові. Якщо кожна мітка з першого масиву присутня у другому, то вони є однаковими.

Блок-схему алгоритму подано у додатку В.

Алгоритм додавання зображення до бази даних використовується для створення нових записів у базі даних. Користувач може відкрити зображення з файлу на жорстокому диску, змінити його назву та додати мітки. Після цих дій можна додати зображення до бази даних. Саме в цей момент починається алгоритм додавання зображення до бази даних.

Алгоритм бере дані про зображення та передає їх до клієнтської бібліотеки, яка передає ці дані до сервісу, який у свою чергу зберігає їх до бази даних.

Блок-схему алгоритму зображено на рисунку 3.7.

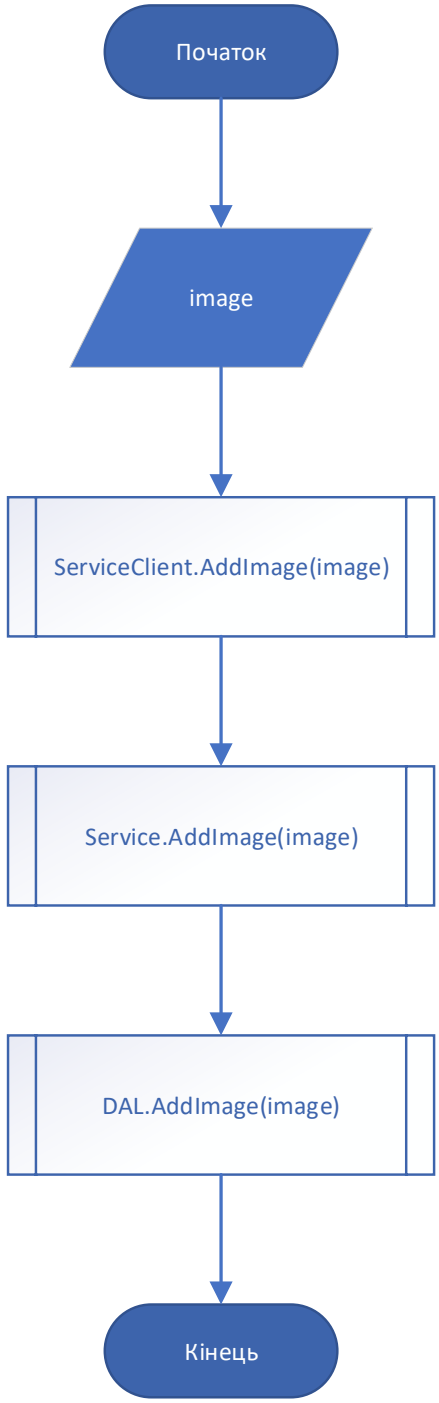


Рисунок 3.7 – Блок-схема алгоритму додавання зображення до бази даних

Алгоритм отримання мініатюр з бази даних використовується для отримання з бази даних мініатюр у заданій кількості та заданого розміру. Він починає свою роботу коли користувач бажає завантажити більше мініатюр.

Алгоритм бере дані, які визначають кількість мініатюр, розмір мініатюри та чи потрібно повертати мініатюри з початку або продовжити з місця, де повернення мініатюр завершилося минулого разу.

Блок-схему алгоритму зображено на рисунку 3.8.

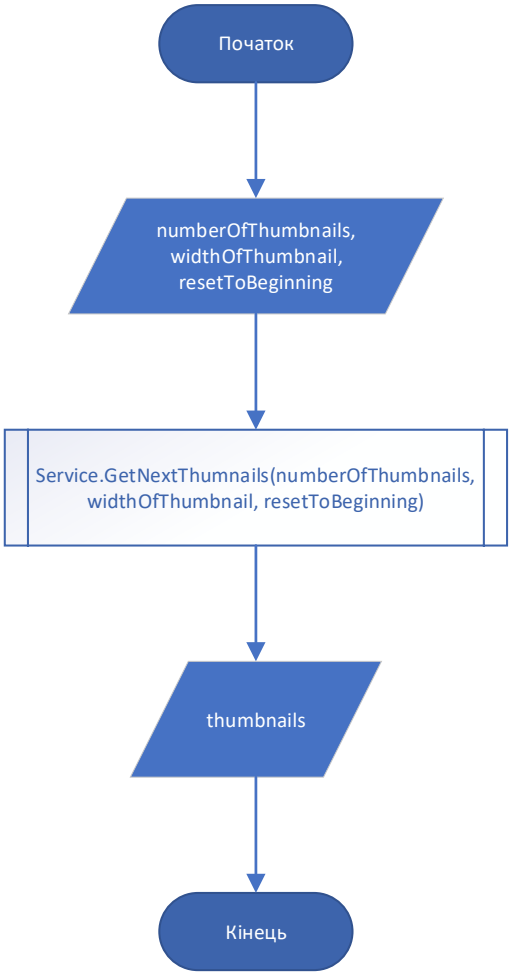


Рисунок 3.8 – Блок-схема алгоритму отримання мініатюр з бази даних

Отже, було створено основні алгоритми роботи системи, які складають основу її функціональності.

3.6 Варіантний аналіз і обґрунтування вибору програмних засобів вирішення задач дипломної роботи

Серед багатьох технологій, які сьогодні доступні розробникам програмного забезпечення, для поточного проекту потрібно обрати СКБД, технологію зв’язку клієнта та сервера, а також технологію створення інтерфейсу користувача. Серед різноманіття доступних варіантів для подальшого вибору були обрані наступні:

1. Windows Presentation Foundation.

2. Windows Forms.

3. Windows Communication Foundation.

4. .NET Remoting.

5. Microsoft SQL Server.

6. MySQL.

Windows Presentation Foundation – це система для побудови клієнтських додатків Windows з візуально привабливими можливостями взаємодії з користувачем, графічна підсистема у складі .NET Framework, яка використовую мову XAML[19]. З допомогою Windows Presentation Foundation можна створювати широкий спектр як автономних додатків, так і додатків, які запускаються у браузері. В основі WPF лежить векторна система візуалізації, яка не залежить від роздільної здатності пристрою виводу і створена з урахуванням можливостей сучасного графічного обладнання. Графічною технологією, що лежить в основі WPF є DirectX, на відміну від Windows Forms, де використовується GDI/GDI+. Особливостями цієї системи є гнучкість у створенні зовнішнього вигляду інтерфейсу користувача, можливість описувати інтерфейс користувача у декларативній манері за допомогою мови XAML, наявність технології “зв’язування даних”, яка дозволяє блискавично оновлювати стан інтерфейсу користувача у відповідь на зміну властивостей, до яких виконується прив’язування.

Windows Forms – це інтерфейс програмування додатків, який відповідає за графічний інтерфейс користувача та є частиною .NET Framework[20]. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Особливістю цієї технології є простота у використання, але при цьому їй не вистачає гнучкості та декларативної манери опису інтерфейсу користувача.

Windows Communication Foundation – це середовище виконання та набір інтерфейсів програмування додатків у складі .NET Framework для побудови з’єднаних, сервісноорієнтованих додатків[21]. Особливостями технології є можливість опису конфігурації у XML файлах з налаштуваннями, простота

використання, гнучкість, яка дозволяє змінювати багато налаштувань за допомогою файлів конфігурації без потреби перекомпілювати програму. Крім того, дана технологія дозволяє комунікувати системам не лише під керуванням ОС Windows, а й інших ОС.

.NET Remoting – це компонент, створений компанією Microsoft. API для міжпроцесної взаємодії[22]. Реалізація протоколу SOAP від Microsoft. Головною особливістю даної технології є її застарілість на сьогоднішній день та прихід на зміну їй WCF.

Microsoft SQL Server – це система керування реляційними базами даних, розроблена компанією Microsoft[23]. Основна мова запитів, яка використовується – T-SQL. Особливостями цього програмного забезпечення є тісна інтеграція з ОС Windows та власна мова запитів T-SQL, яка була доповнена процедурними розширеннями, що збільшило її можливості.

MySQL – це вільна реляційна система керування базами даних. Розробку та підтримку проводить компанія Oracle[24]. Основною особливістю цього програмного забезпечення є вільне розповсюдження.

Отже, в результаті аналізу особливостей технологій, які було розглянуто, прийнято рішення обрати для розробки програмних модулів додатку WPF, WCF та Microsoft SQL Server.

3.7 Вибір середовища програмування

Для розробки програмного забезпечення, потрібно обрати інтегроване середовище розробки (IDE − Integrated Development Environment). IDE – це програма, що допомагає розробнику створювати нове програмне забезпечення та вдосконалювати вже існуюче. Для порівняння було обрано Microsoft Visual Studio, Visual Studio Code та JetBrains Rider.

Microsoft Visual Studio – це серія продуктів фірми Microsoft, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з

підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight. Перевагами є інтуїтивно зрозумілий інтерфейс, підсвічування синтаксису, покрокове виконання коду, вбудована підтримка Git. До недоліків можна віднести відсутність підказки про потребу підключення необхідних для використання окремих класів бібліотек .

Visual Stduio Code – це засіб для створення, редагування та зневадження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X. Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковаге рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Перевагою є невеликий розмір програми. Недоліком є відсутність багатьох функцій повноцінних IDE.

JetBrains Rider – це кросплатформне інтегроване середовище розробки програмного забезпечення для платформи .NET, розроблюване компанією JetBrains. Підтримуються мови програмування C#, VB.NET та F#. Проект було анонсовано у січні 2016 року. У його основі лежить інший продукт JetBrains – ReSharper. Середовище підтримує платформи .NET Framework, .NET Core та Mono. Працює на операційних системах Windows, MacOS, Linux. Перевагою є кросплатформність. Недоліком є невідшліфованість нового продукту.

Після дослідження переваг та недоліків наведених вище IDE було прийнято рішення використовувати для розробки Microsoft Visual Studio. Дана програма має багато зручних функцій, які значно полегшують процес розробки та роблять його швидшим.

3.8 Схема класів та компонентів програмних модулів

Діаграма класів UML – це діаграма, на якій зображуються класи, які відповідають основним категоріям сутностей предметної області. Клас містить ім’я, набір атрибутів та операцій. Атрибути описують внутрішній стан об’єкта та можуть мати область видимості, яка може бути загальнодоступною, захищеною або закритою[25]. Загальнодоступна область видимості дозволяє отримати доступ до атрибута з будь-якого класу. Захищена область видимості надає доступ до атрибута з класу, який містить цей атрибут, та з дочірніх класів. Закрита область видимості робить атрибут доступним тільки з класу, який його містить. Операції описують поведінку об’єкта і також можуть мати область видимості. Класи можуть бути пов’язані між собою різними типами зв’язків, до яких належать асоціація, залежність, агрегація, композиція, узагальнення та реалізація. Асоціація описує зв’язок між класами, які концептуально взаємодіють один з одним. Залежність виникає тоді, коли один клас використовує інший. Агрегація виникає коли декілька класів утворюють структуру “частина(частини)-ціле”. Композиція є підвидом агрегації, який характеризується тим, що компонент може належати лише єдиному цілому. Узагальнення – це тип зв’язку між батьківським та дочірнім класом. Реалізація – це зв’язок між інтерфейсом та класом, який його реалізує.

Діаграма класів предметної області зображена на рисунку 3.9.

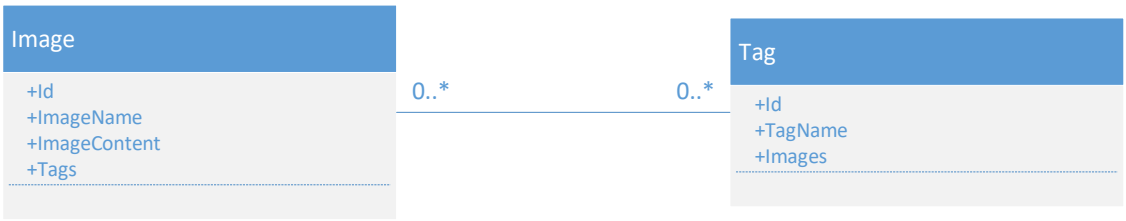


Рисунок 3.9 – Діаграма класів предметної області

Діаграма компонентів UML – це діаграма, на якій зображуються компоненти програмного забезпечення. За допомогою цієї діаграми клієнти можуть побачити структуру завершеної системи, а розробники можуть уявити

собі структуру майбутньої роботи[26]. Компоненти можна буде використовувати багаторазово.

Діаграма компонентів системи зображена на рисунку 3.10.

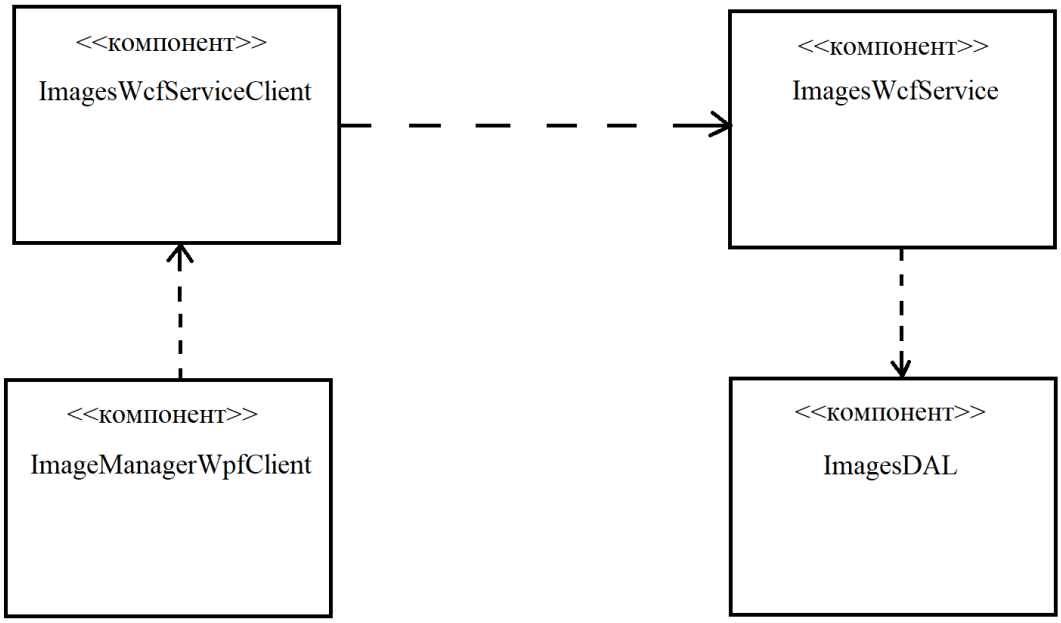


Рисунок 3.10 – Діаграма компонентів системи

На діаграмі компонентів зображено загальну структуру системи.

Отже, було створено діаграми класів та компонентів, які дозволяють зрозуміти загальну структуру системи.

3.9 Реалізація розроблених методів класифікації

Для реалізації розроблених методів класифікації зображень було вирішено використати AForge.NET.

AForge.NET – це фреймворк комп’ютерного зору та штучного інтелекту, розроблений для .NET Framework. Фреймворк включає такі особливості, як комп’ютерний зір, обробка зображень та відео, бібліотека штучних нейронних мереж, генетичні алгоритми, нечітка логіка, машинне навчання.

Для програмної реалізації кластера нейронних мереж були застосовані простори імен AForge.Neuro та AForge.Neuro.Learning.

Простір імен AForge.Neuro містить інтерфейси та класи для обчислень, пов’язаних з нейронними мережами.

Простір імен AForge.Neuro.Learning містить інтерфейси та класи для навчання нейронів та нейронних мереж. Цей простір імен містить класи як для навчання за учителем, так і для навчання без учителя.

Для реалізації кластера нейронних мереж було використано класи ActivationNetwork та PerceptronLearning.

Екземпляри класу ActivationNetwork представляють собою нейронні мережі у кластері, який в свою чергу представлений класом List<ActivationNetwork>.

Навчання мереж відбувається за допомогою набору екземплярів класу PerceptronLearning. Набір представлений класом List<PerceptronLearning>. Кожен екземпляр з набору відповідає одній мережі з кластеру нейронних мереж.

На початку процесу класифікації зображення перетворюється на двовимірний масив чисел, кожне з яких представляє собою дані про колір окремого пікселя зображення.

Наступним етапом є створення задач(представлених класом Task) для виконання класифікації кожною мережею в окремому потоці. Двовимірний масив чисел(double[][]) перетворюється на одновимірний масив(double[]) та передається до кожної нейронної мережі і після цього задачі, що містять у собі виконання методу ActivationNetwork.Compute, починають виконуватись.

В результаті усі задачі завершуються та значення, що повертаються методом Compute, використовуються для призначення міток зображенню, адже ці значення представлять собою набори результатів на вихідних нейронах мережі.

Під час призначення міток зображенням вручну, відбувається навчання нейронних мереж із застосуванням набору екземплярів класу PerceptronLearning, що був описаний раніше.

3.10 Розробка програмних модулів системи

Під час створення програмних модулів системи було отримано файли з вихідним кодом. Загальна структура файлів зображена на рисунку 3.11.

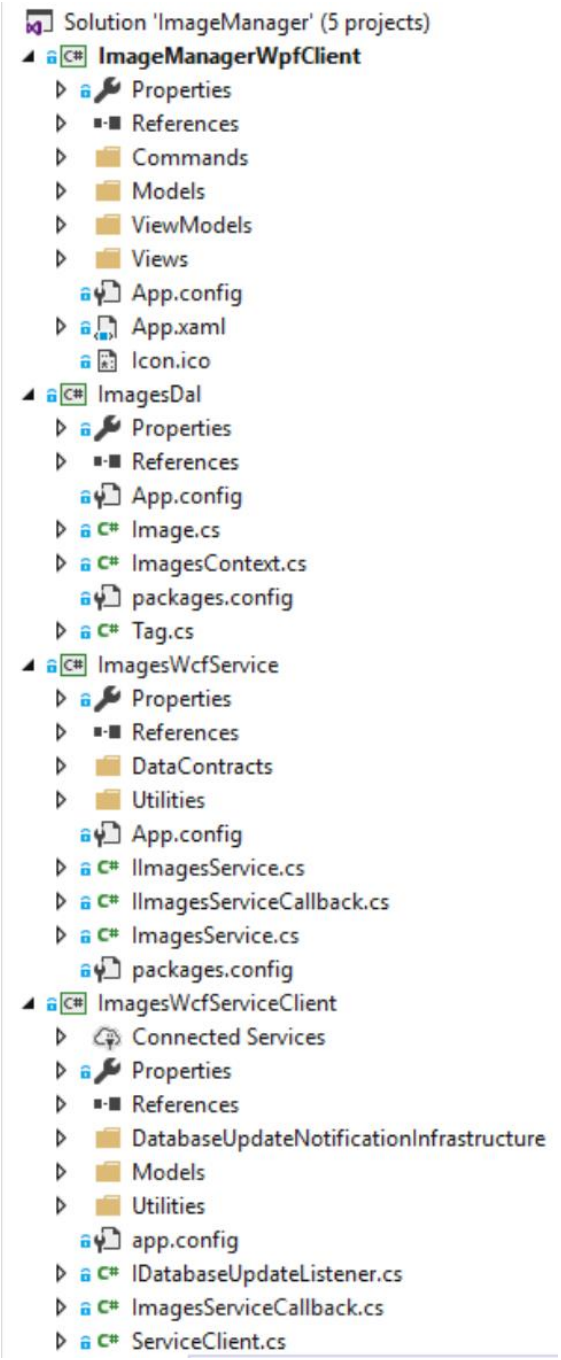


Рисунок 3.11 – Загальна структура файлів з вихідним кодом проекту

Почнемо розгляд файлів з вихідним кодом з проекту ImagesDAL(рисунок 3.12). Цей проект представляє собою бібліотеку, яка відповідає за взаємодію з базою даних.

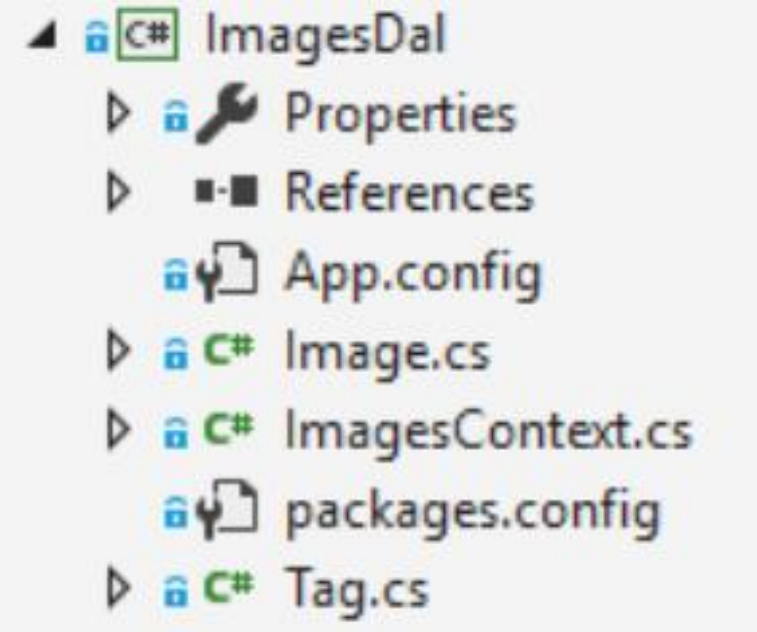


Рисунок 3.12 – Проект ImagesDAL

Проект використовує технологію Entity Framework для операцій з базою даних. До файлів з кодом належать файли, які представляють класи предметної області та файл з класом-контекстом, який надає можливості по роботі з базою даних.

Розглянемо файл Image.cs(рисунок 3.13). Він містить клас Image, який відповідає зображенню та таблиці tblImage у базі даних. Клас властивості, які відповідають полям таблиці. Властивість Id відповідає полю intId та представляє первинний ключ, який є автоінкрементованим цілим числом. Властивість ImageName відповідає полю nvcImageName та представляє назву зображення. Властивість ImageContent відповідає полю vbImageContent та представляє вміст зображення. Властивість Tags є навігаційною. Навігаційні властивості потрібні для зв’язку між класами, який відповідає зв’язку між таблицями.

```
namespace ImagesDal
{
    using System;
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    using System.ComponentModel.DataAnnotations.Schema;
    using System.Data.Entity.Spatial;

    [Table("tblImage")]
    public partial class Image
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
        public Image()
        {
            Tags = new HashSet<Tag>();
        }

        [Key]
        [Column("intId")]
        public int Id { get; set; }

        [Required]
        [StringLength(100)]
        [Column("nvcImageName")]
        public string ImageName { get; set; }

        [Required]
        [Column("vbImageContent")]
        public byte[] ImageContent { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<Tag> Tags { get; set; }
    }
}
```

Рисунок 3.13 – Вміст файлу Image.cs

Далі розглянемо файл Tag.cs(рисунок 3.14). Він містить клас Tag, який відповідає мітці та таблиці tblTag у базі даних. Властивість Id відповідає полю intId та представляє первинний ключ, який є автоінкрементованим цілим числом. Властивість TagName відповідає полю nvcTagName та представляє назву мітки. Властивість Images є навігаційною.

```
namespace ImagesDal
{
    using System;
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    using System.ComponentModel.DataAnnotations.Schema;
    using System.Data.Entity.Spatial;

    [Table("tblTag")]
    public partial class Tag
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
        public Tag()
        {
            Images = new HashSet<Image>();
        }

        [Key]
        [Column("intId")]
        public int Id { get; set; }

        [Required]
        [StringLength(100)]
        [Column("nvcTagName")]
        public string TagName { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<Image> Images { get; set; }
    }
}
```

Рисунок 3.14 – Вміст файлу Tag.cs

Нарешті розглянемо файл ImagesContext.cs(рисунок 3.15). Він містить клас ImagesContext, який слугує для взаємодії з базою даних.

```
namespace ImagesDal
{
    using System;
    using System.Data.Entity;
    using System.ComponentModel.DataAnnotations.Schema;
    using System.Linq;

    public partial class ImagesContext : DbContext
    {
        public ImagesContext()
            : base("name=ImagesContextConnection")
        {
        }

        public virtual DbSet<Image> Images { get; set; }
        public virtual DbSet<Tag> Tags { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Image>()
                .HasMany(e => e.Tags)
                .WithMany(e => e.Images)
                .Map(m => m.ToTable("tblTagImage").MapLeftKey("intImageId").MapRightKey("intTagId"));
        }
    }
}
```

Рисунок 3.15 – Вміст файлу ImagesContext.cs

Наступним розглянемо проект ImagesWcfService(рисунок 3.16). Цей проект представляє собою WCF-сервіс, який взаємодіє з клієнтами.

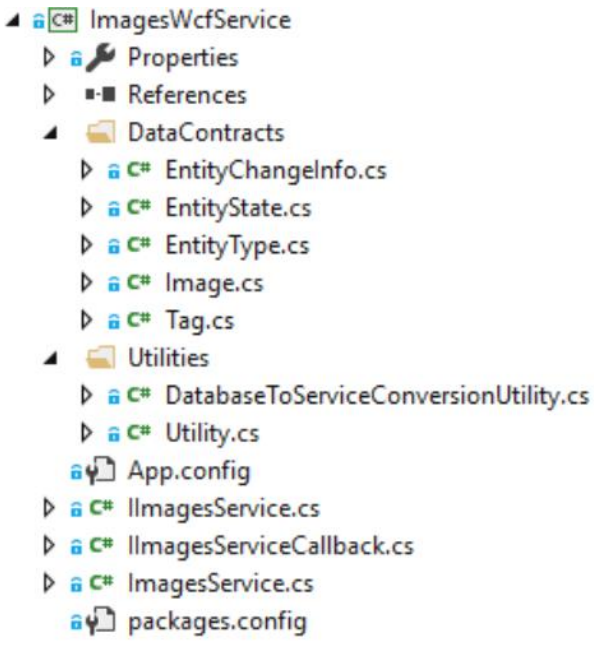


Рисунок 3.16 – Проект ImagesWcfService

Розглянемо файл IImagesService.cs(рисунок 3.17). Він містить інтерфейс IImagesService, який представляє контракт сервісу. Контракт сервісу визначає операції, які сервіс може виконувати та типи даних, якими він може оперувати. В даному випадку контракт визначає, що сервіс працює з клієнтами в режимі сесії та використовує операції зворотного виклику(сервіс може викликати операції клієнта), які визначені інтерфейсом IImagesServiceCallback. Контракт визначає операції для початку і завершення сесії, операції для отримання мініатюр без міток та з відповідними мітками, операції отримання мініатюри та повно розмірного зображення, отримання усіх міток та окремої мітки, а також операції додавання, оновлення, видалення зображень та міток.

```
namespace ImagesWcfService
{
    [ServiceContract(CallbackContract = typeof(IImagesServiceCallback), SessionMode = SessionMode.Required)]
    public interface IImagesService
    {
        [OperationContract(IsOneWay = true, IsInitiating=true, IsTerminating = false)]
        void Subscribe();

        [OperationContract(IsInitiating = false, IsTerminating = false)]
        Image[] GetNextThumbnails(int numberOfThumbnails, int widthOfThumbnail, bool resetToBeginning);

        [OperationContract(IsInitiating = false, IsTerminating = false)]
        Image[] GetNextThumbnailsWithSuchTags(int numberOfThumbnails, int widthOfThumbnail, Tag[] tags, bool resetToBeginning);

        [OperationContract(IsInitiating = false, IsTerminating = false)]
        Image GetThumbnail(int widthOfThumbnail, int id);

        [OperationContract(IsInitiating = false, IsTerminating = false)]
        Image GetFullSizeImage(int id);

        [OperationContract(IsInitiating = false, IsTerminating = false)]
        Tag[] GetAllTags();

        [OperationContract(IsInitiating = false, IsTerminating = false)]
        Tag GetTag(int id);

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void AddImage(Image image);

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void UpdateImage(Image image);

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void DeleteImage(int id);

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void AddTag(Tag tag);

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void UpdateTag(Tag tag);

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
        void DeleteTag(int id);

        [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = true)]
        void Unsubscribe();
    }
}
```

Рисунок 3.17 – Вміст файлу IImagesService.cs

Наступним розглянемо файл ImagesService.cs(рисунок 3.18). Він містить клас ImagesService, який представляє собою реалізацію інтерфейсу IImagesService. Це означає, що цей клас реалізує операції, які визначені контрактом і коли клієнт викликає операцію сервісу, саме цей клас відповідає за її виконання. Крім реалізації операцій, які визначені у контракті, клас місить функціонал, який використовується для інформування усіх клієнтів про зміни у базі даних. Це досягається за допомогою зберігання ідентифікаторів сесій клієнтів та об’єктів, які дозволяють викликати операції на клієнті, а також за допомогою методу, який при змінах у базі даних викликає операцію клієнта для повідомлення про ці зміни.

```
namespace ImagesWcfService
{
    public class ImagesService : IImagesService
    {
        private static Dictionary<string, IImagesServiceCallback> _clients = new Dictionary<string, IImagesServiceCallback>();
        private static object _clientsSyncObject = new object();

        private int _numberOfSentThumbnails;

        private Tag[] _previousTagsToSearchBy = new Tag[] { };
        private int _numberOfSentThumbnailsWithSpecifiedTags;

        public void Subscribe()...

        public Image[] GetNextThumbnails(int numberOfThumbnails, int widthOfThumbnail, bool resetToBeginning)...

        public Image[] GetNextThumbnailsWithSuchTags(int numberOfThumbnails, int widthOfThumbnail, Tag[] tags, bool resetToBeginning)...

        public Image GetThumbnail(int widthOfThumbnail, int id)...

        public Image GetFullSizeImage(int id)...


        public Tag[] GetAllTags()...

        public Tag GetTag(int id)...

        public void AddImage(Image image)...

        public void UpdateImage(Image image)...

        public void DeleteImage(int id)...

        public void AddTag(Tag tag)...

        public void UpdateTag(Tag tag)...

        public void DeleteTag(int id)...

        private int SaveChanges(ImagesDal.ImagesContext imagesContext)...

        public void Unsubscribe()...

        private void NotifyOtherClientsAboutDatabaseUpdate(EntityChangeInfo entityChangeInfo)...
    }
}
```

Рисунок 3.18 – Вміст файлу ImagesService.cs

Далі розглянемо файл IImagesServiceCallback.cs(рисунок 3.19). Він містить інтерфейс IImagesServiceCallback, який описує операції клієнта, які можуть бути викликані сервісом. В даному випадку це лише одна операція, яка використовується сервісом для того, щоб повідомити клієнт про зміни у базі даних.

```
namespace ImagesWcfService
{
    public interface IImagesServiceCallback
    {
        [OperationContract(IsOneWay = true)]
        void NotifyAboutDatabaseUpdate(EntityChangeInfo entityChangeInfo);
    }
}
```

Рисунок 3.19 – Вміст файлу IImagesServiceCallback.cs

Розглянемо вміст папки DataContracts, яка містить типи даних, які використовує сервіс при обміні повідомленнями з клієнтом.

Спочатку розглянемо вміст файлу Image.cs(рисунок 3.20). Він містить клас Image, який використовується для передачі даних про зображення між клієнтом та сервісом.

```
namespace ImagesWcfService
{
    [DataContract]
    public class Image
    {
        [DataMember]
        public int Id { get; set; }
        [DataMember]
        public string ImageName { get; set; }
        [DataMember]
        public byte[] ImageContent { get; set; }
        [DataMember]
        public Tag[] Tags { get; set; }
    }
}
```

Рисунок 3.20 – Вміст файлу Image.cs

Тепер розглянемо вміст файлу Tag.cs(рисунок 3.21). Він містить клас Tag, який використовується для передачі даних про мітку між клієнтом та сервісом.

```
namespace ImagesWcfService
{
    [DataContract]
    public class Tag
    {
        [DataMember]
        public int Id { get; set; }
        [DataMember]
        public string TagName { get; set; }
    }
}
```

Рисунок 3.21 – Вміст файлу Tag.cs

Далі розглянемо вміст файлу EntityChangeInfo.cs(рисунок 3.22). Він містить клас EntityChangeInfo, який представляє дані про зміни у базі даних. Він містить властивості для передачі ідентифікатора сутності, типу сутності та стану сутності, яка відповідає запису у базі даних.

```
namespace ImagesWcfService
{
    [DataContract]
    public class EntityChangeInfo
    {
        [DataMember]
        public int EntityId { get; set; }
        [DataMember]
        public EntityType EntityType { get; set; }
        [DataMember]
        public EntityState EntityState { get; set; }

        public EntityChangeInfo(int entityId, EntityType entityType, EntityState entityState)
        {
            EntityId = entityId;
            EntityType = entityType;
            EntityState = entityState;
        }
    }
}
```

Рисунок 3.22 – Вміст файлу EntityChangeInfo.cs

Розглянемо вміст файлу EntityType.cs(рисунок 3.23). Він містить перелічуваний тип даних, який використовується у класі EntityChangeInfo для вказування типу сутності(зображення або мітка), яка змінилася у бази даних.

```
namespace ImagesWcfService
{
    public enum EntityType
    {
        Image,
        Tag
    }
}
```

Рисунок 3.23 – Вміст файлу EntityType.cs

Нарешті розглянемо вміст файлу EntityType.cs(рисунок 3.24). Він містить перелічуваний тип даних, який використовується у класі EntityChangeInfo для вказування стану сутності(додана, змінена, видалена), яка змінилася у базі даних.

```
namespace ImagesWcfService
{
    public enum EntityState
    {
        Added,
        Modified,
        Deleted
    }
}
```

Рисунок 3.24 – Вміст файлу EntityState.cs

Нарешті розглянемо вміст папки Utilities, яка містить файли зі статичними класами, які виконують допоміжну роль у роботі сервісу.

Розглянемо вміст файлу DatabaseToServiceConversionUtility.cs(рисунок 3.25). Він містить статичний клас DatabaseToServiceConversionUtility, який містить методи для перетворення типів, які відповідають записам у базі даних на типи для передачі даних.

```
namespace ImagesWcfService.Utilities
{
    static class DatabaseToServiceConversionUtility
    {
        public static Image[] CreateThumnailsToSendToClient(List<ImagesDal.Image> imagesFromDatabase, int widthOfThumbnail)...
        public static Image CreateThumbnailToSendToClient(ImagesDal.Image imageFromDatabase, int widthOfThumbnail)...
        public static byte[] CreateThumbnailContent(byte[] imageContent, int widthOfThumbnail)...
        public static Tag[] CreateTagsToSendToClient(List<ImagesDal.Tag> tagsFromDatabase)...
        public static Tag CreateTagToSendToClient(ImagesDal.Tag tagFromDatabase)...
    }
}
```

Рисунок 3.25 – Вміст файлу DatabaseToServiceConversionUtility.cs

Тепер розглянемо вміст файлу Utitlity.cs(рисунок 3.26). Він містить статичний клас Utility, який містить метод для порівняння на однаковість двох масивів міток.

```
namespace ImagesWcfService.Utilities
{
    static class Utility
    {
        public static bool TagArraysAreEqual(Tag[] firstTagArray, Tag[] secondTagArray)
        {
            if (firstTagArray.Length != secondTagArray.Length)
            {
                return false;
            }

            for (int i = 0; i < firstTagArray.Length; i++)
            {
                bool isTagFromFirstArrayPresentInSecondArray = false;

                for (int j = 0; j < secondTagArray.Length; j++)
                {
                    if (firstTagArray[i].Id == secondTagArray[j].Id)
                    {
                        isTagFromFirstArrayPresentInSecondArray = true;
                        break;
                    }
                }

                if (!isTagFromFirstArrayPresentInSecondArray)
                {
                    return false;
                }
            }

            return true;
        }
    }
}
```

Рисунок 3.26 – Вміст файлу Utitlity.cs

Далі розглянемо проект ImagesWcfServiceClient(рисунок 3.27). Цей проект представляє собою клієнтську бібліотеку, яка дозволяє додатку з графічним інтерфейсом взаємодіяти з базою даних.

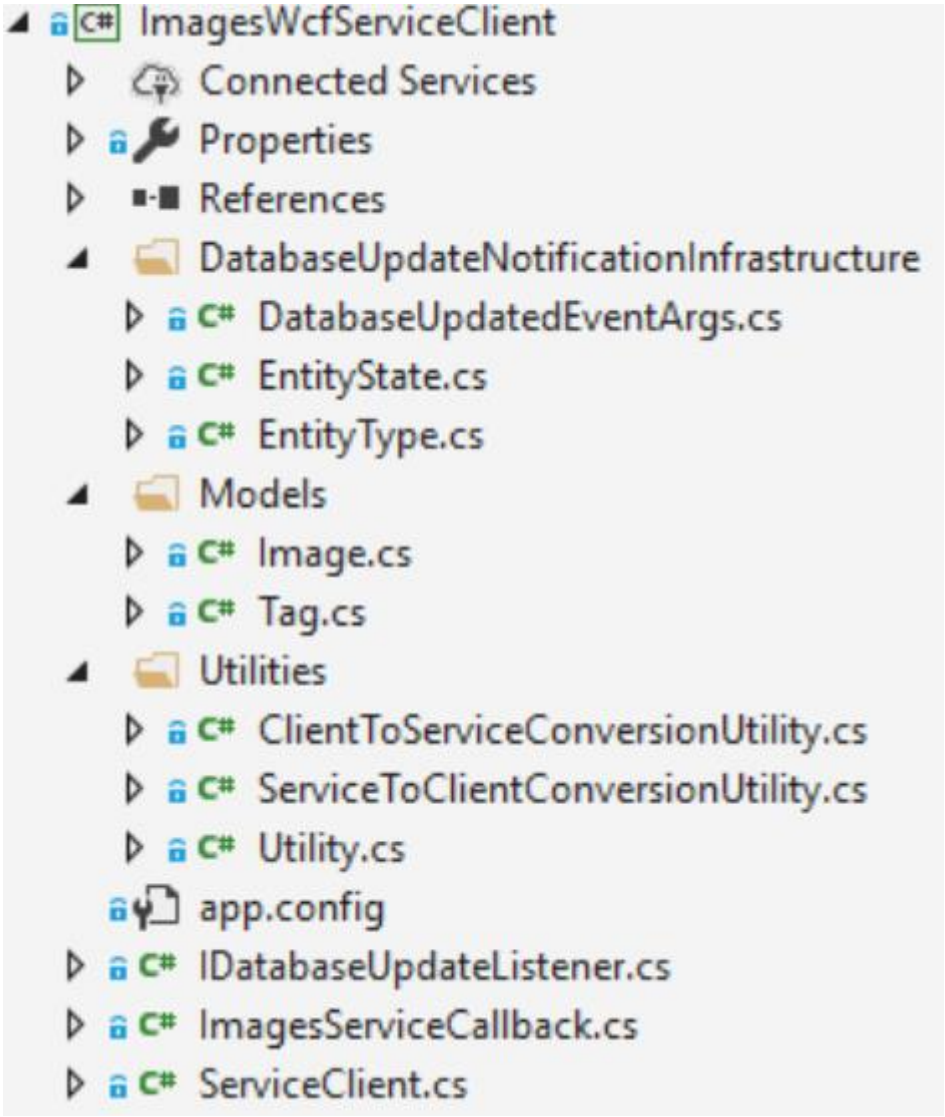


Рисунок 3.27 – Проект ImagesWcfServiceClient

Розглянемо файл ServiceClient.cs(рисунок 3.28). Він містить клас ServiceClient, який обгортку для автозгенерованого класу клієнта. Він додає валідацію в роботу клієнта.

```
namespace ImagesWcfServiceClient
{
    public class ServiceClient : IDisposable
    {
        private const int NUMBER_OF_THUMBNAILS = 10;
        private const int WIDTH_OF_THUMBNAIL = 100;

        private ImagesWcfServiceReference.ImagesServiceClient _client;

        private bool _receivedAllAvailableThumbnails = false;
        private bool _receivedAllAvailableThumbnailsWithSuchTags = false;
        private List<Tag> _previousTagsToSearchBy = new List<Tag>();

        public ServiceClient(IDatabaseUpdateListener listener)...

        public List<Image> GetNextThumbnails(bool resetToBeginning)...

        public List<Image> GetNextThumbnailsWithSuchTags(List<Tag> tags, bool resetToBeginning)...

        public Image GetThumbnail(int id)...

        public Image GetFullSizeImage(int id)...

        public List<Tag> GetAllTags()...

        public Tag GetTag(int id)...

        public void AddImage(Image image)...

        public void UpdateImage(Image image)...

        public void DeleteImage(Image image)...

        public void AddTag(Tag tag)...

        public void UpdateTag(Tag tag)...

        public void DeleteTag(Tag tag)...

        private bool _disposed = false;

        public void Dispose()...

        private void CleanUp(bool disposing)...

        ~ServiceClient()...
    }
}
```

Рисунок 3.28 – Вміст файлу ServiceClient.cs

Розглянемо файл ImagesServiceCallback.cs(рисунок 3.29). Він містить клас ImagesServiceCallback, який є реалізацією інтерфейсу IImagesServiceCallback. Цей клас відповідає за обробку виклику операції клієнта сервером. Він містить подію, яка повідомляє об’єкт-обробник цієї події про зміни у базі даних.

```
namespace ImagesWcfServiceClient.ImagesWcfServiceReference
{
    class ImagesServiceCallback : IImagesServiceCallback
    {
        public event EventHandler<DatabaseUpdatedEventArgs> DatabaseUpdated;

        public ImagesServiceCallback(IDatabaseUpdateListener listener)
        {
            DatabaseUpdated += listener.ImagesServiceCallback_DatabaseUpdated;
        }

        public void NotifyAboutDatabaseUpdate(EntityChangeInfo entityChangeInfo)
        {
            OnDatabaseUpdated(new DatabaseUpdatedEventArgs(entityChangeInfo.EntityId, entityChangeInfo.EntityType, entityChangeInfo.EntityState));
        }

        protected virtual void OnDatabaseUpdated(DatabaseUpdatedEventArgs e)
        {
            DatabaseUpdated?.Invoke(this, e);
        }
    }
}
```

Рисунок 3.29 – Вміст файлу ImagesServiceCallback.cs

Далі розглянемо файл IDatabaseUpdateListener.cs(рисунок 3.30). Він містить інтерфейс IDatabaseUpdateListener, який визначає обробник події зміни бази даних та який повинен бути реалізований класом, який буде реагувати на цю подію.

```
namespace ImagesWcfServiceClient
{
    public interface IDatabaseUpdateListener
    {
        void ImagesServiceCallback_DatabaseUpdated(object sender, DatabaseUpdatedEventArgs e);
    }
}
```

Рисунок 3.30 – Вміст файлу IDatabaseUpdateListener.cs

Тепер розглянемо коротко вміст папок, які знаходяться у цьому проекті.

Папка DatabaseUpdateNotificationInfrastructure містить типи даних для повідомлення про зміни у базі даних.

Папка Models містить типи для передачі даних про зображення та мітки програмі, яка використовує цю бібліотеку.

Папка Utilities містить типи даних, які виконують допоміжну роль у роботі бібліотеки.

Нарешті розглянемо проект ImageManagerWpfClient(рисунок 3.31). Цей проект представляє собою додаток з графічним інтерфейсом користувача, який використовує клієнтську бібліотеку для взаємодії з сервісом.

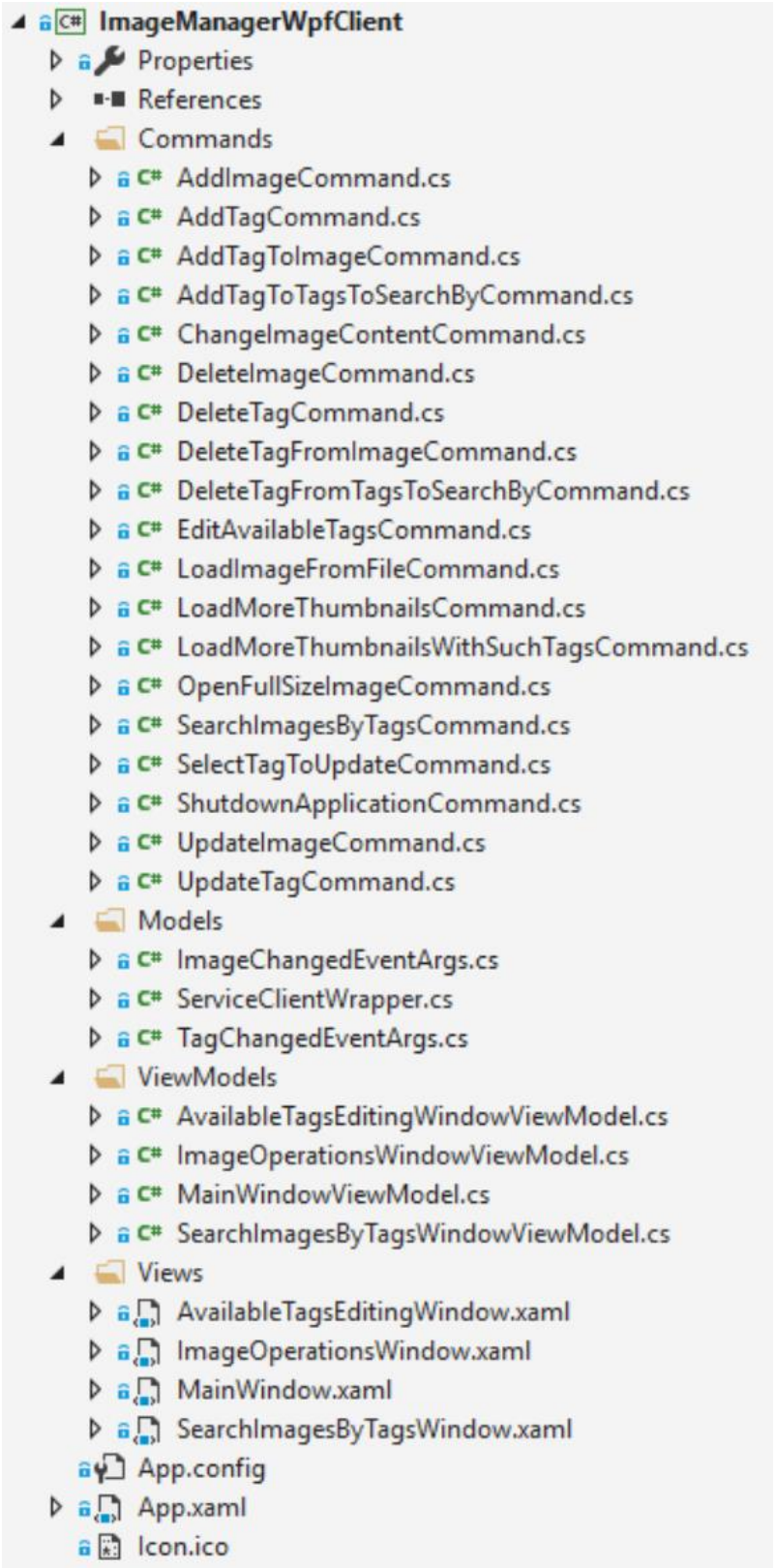


Рисунок 3.31 – Проект ImageManagerWpfClient

Розглянемо коротко вміст цього проекту.

Файл Icon.ico містить значок додатку, який відображатиметься вгорі зліва у кожному вікні та у панелі задач.

Файл App.xaml містить ресурси мовою розмітки XAML, які визначають загальний зовнішній вигляд для елементів інтерфейсу користувача.

Папка Views містить файли, які представляють вікна додатку та визначають їх зовнішній вигляд.

Папка ViewModels містить файли, які представляють моделі представлень для вікон(які є представленнями). Моделі представлень мають дані для відображення у інтерфейсі користувача та команди для взаємодії користувача із даними додатку.

Папка Commands містить файли, які представляють команди, що визначають функціонал доступний користувачеві для маніпуляції з даними додатку.

Папка Models містить файли, які містять типи даних, що представляють дані для використання у додатку.

3.11 Висновки

Отже, було розроблено загальну структуру системи, структуру бази даних, структуру додатку з графічним інтерфейсом, структуру інтерфейсу користувача, основні алгоритми роботи системи, проведено варіантний аналіз та обґрунтовано вибір програмних засобів для вирішення задач роботи, під час якого було обрано такі технології, як Windows Presentation Foundation, Windows Communication Foundation, Microsoft SQL Server, які найбільш відповідають вимогам, які ставлять задачі поточного проекту. Крім того, було проведено аналіз інтегрованих середовищ розробки, серед яких було обрано найкраще для поточних умов. Також було створено діаграми класів та компонентів системи. Було проведено розробку програмних модулів системи та розроблено вихідний код.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Методики тестування

Тестування програмного забезпечення – це процес дослідження, випробування програмного продукту, що має метою перевірку відповідності між реальною поведінкою програми та її очікуваною поведінкою на кінцевому наборі тестів, обраних визначеним чином[27].

Існує декілька ознак, за якими прийнято проводити класифікацію видів тестування. Зазвичай виділяють наступні:

1. За об’єктом тестування.
   1.1 Функціональне тестування.
   1.2 Тестування продуктивності.
      1.2.1 Навантажувальне тестування.
      1.2.2 Стрес-тестування.
      1.2.3 Тестування стабільності.
   1.3 Конфігураційне тестування.
   1.4 Юзабіліті-тестування.
   1.5 Тестування інтерфейсу користувача.
   1.6 Тестування безпеки.
   1.7 Тестування локалізації.
   1.8 Тестування сумісності.
2. За знанням внутрішньої будови системи.
   2.1 Тестування чорної скрині.
   2.2 Тестування білої скрині.
   2.3 Тестування сірої скрині.
3. За ступенем автоматизації.
   3.1 Ручне тестування.
   3.2 Автоматизоване тестування.
   3.3 Напівавтоматизоване тестування.
4. За ступенем ізольованості.

4.1 Тестування компонентів.

4.2 Інтеграційне тестування.

4.3 Системне тестування.

5. За часом проведення тестування.

5.1 Альфа-тестування.

5.1.1 Димове тестування.

5.1.2 Тестування нової функції.

5.1.3 Підтверджуюче тестування.

5.1.4 Регресивне тестування.

5.1.5 Приймальне тестування.

5.2 Бета-тестування.

6. За ознакою позитивності сценаріїв.

6.1 Позитивне тестування.

6.2 Негативне тестування.

7. За ступенем підготовленості до тестування.

7.1 Тестування за документацією.

7.2 Інтуїтивне тестування.

Розглянемо тестування білої та чорної скрині. В залежності від доступу розробника тестів до вихідного коду програми, що тестується, розрізняють «тестування білої скрині» та «тестування чорної скрині».

При тестуванні білої скрині розробник тесту має доступ до вихідного коду програм та може писати код, який пов’язаний з бібліотеками програмного забезпечення, що тестується[28]. Це типово для компонентного тестування, при якому тестуються лише окремі частини системи. Воно забезпечує те, що компоненти конструкції є працездатними та стійкими, до певного ступеню.

При тестуванні чорної скрині тестувальник має доступ до програми тільки через ті ж інтерфейси, що й замовник чи користувач, або через зовнішні інтерфейси, які дозволяють іншому комп’ютеру або процесу підключитися до системи для тестування[29]. Наприклад, компонент, що тестує, може віртуально натискати клавіші або кнопки миші в програмі, що тестується, за

допомогою механізму взаємодії процесів. Як правило, тестування чорної скрині ведеться з використанням специфікацій або інших документів, що описують вимоги до системи.

4.2 Тестування системи

Тестування програмного модуля проводиться за допомогою методики «чорної скрині». Вона базується на використанні шаблонів тестування(тест-кейсів)[30]. Це означає, що буде створено декілька тест-кейсів для перевірки правильності роботи основних функцій додатку.

Тест-кейс №1. Додавання зображення.

Кроки:

1. Запустити додаток.
2. Обрати пункт меню «File», у ньому обрати «Load image from file…».
3. Обрати файл із зображенням.
4. У вікні, що з’явилося натиснути кнопку «Add».

Очікувані результати зображено на рисунку 4.1.

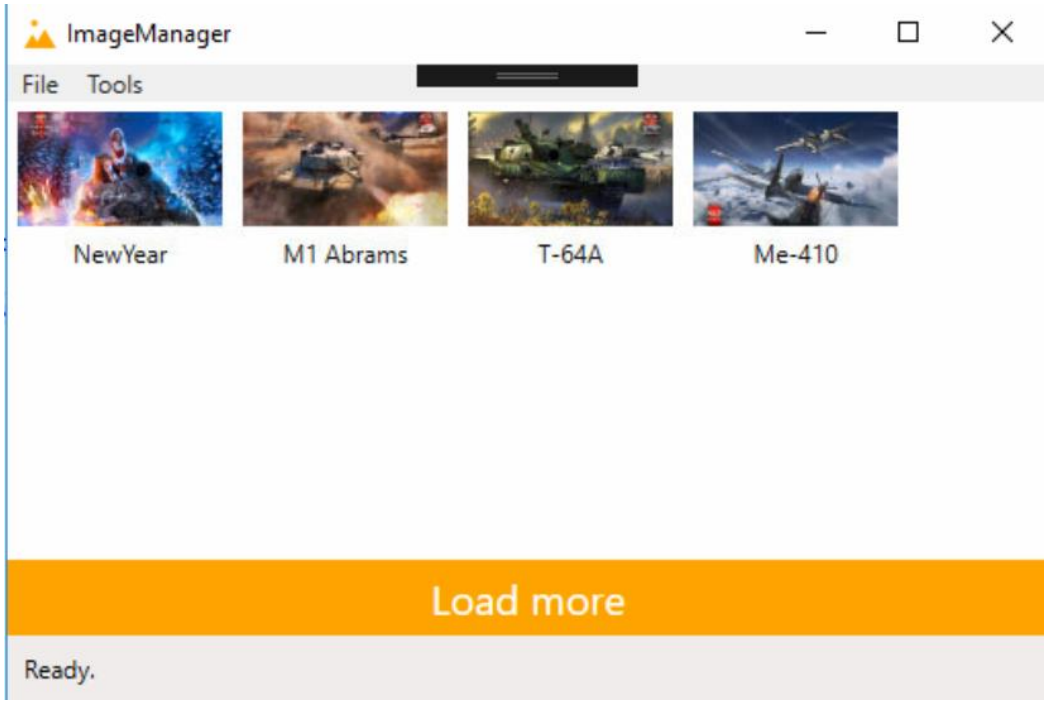


Рисунок 4.1 – Очікуваний результат додавання зображення

Результат тест-кейсу №1: виконано.

Тест-кейс №2. Оновлення зображення.

Кроки:

1. Запустити додаток.

2. Натиснути кнопку «Load more».

3. Обрати зображення для оновлення шляхом натискання на нього.

4. У вікні, що з’явилося змінити назву зображення.

5. Натиснути кнопку «Update».

Очікувані результати зображено на рисунку 4.2.

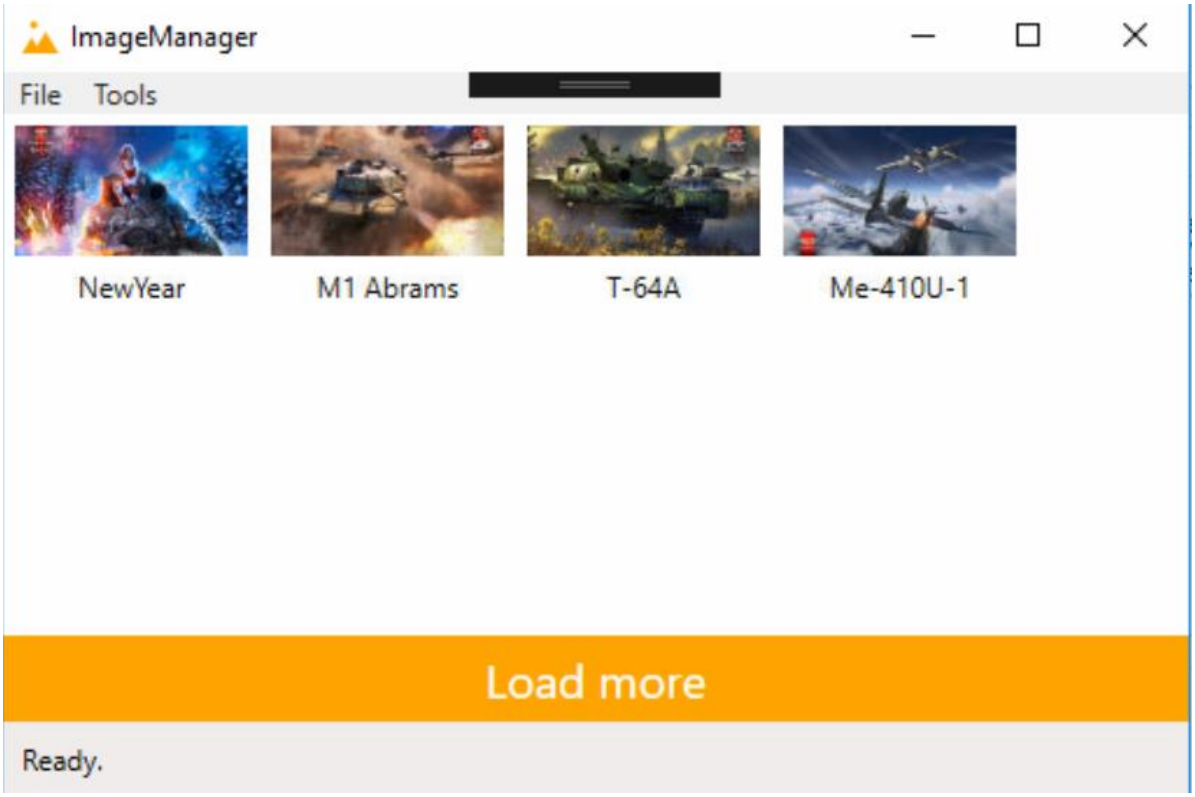


Рисунок 4.2 – Очікуваний результат оновлення зображення

Результат тест-кейсу №2: виконано.

Тест-кейс №3. Видалення зображення.

Кроки:

1. Запустити додаток.

2. Натиснути кнопку «Load more».

3. Обрати зображення для оновлення шляхом натискання на нього.

4. У вікні, що з’явилося натиснути кнопку «Delete».

Очікувані результати зображено на рисунку 4.3.

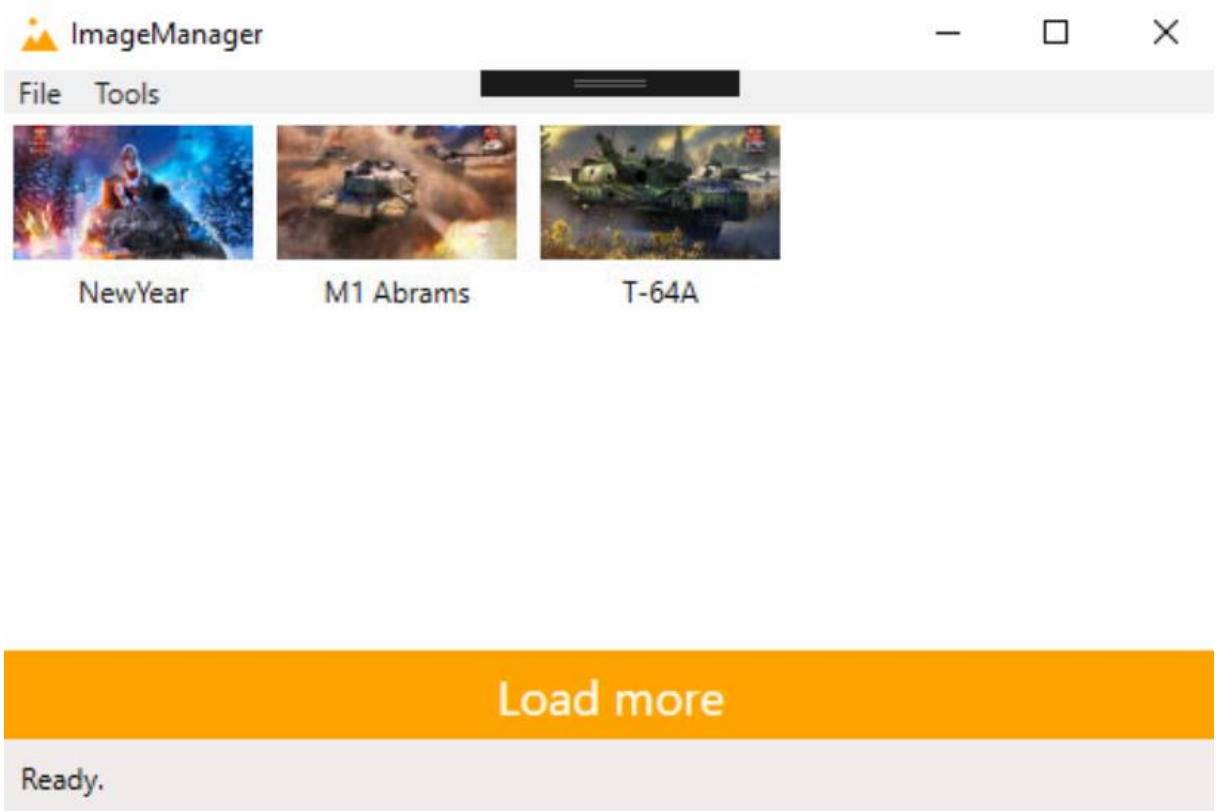


Рисунок 4.3 – Очікуваний результат видалення зображення

Результат тест-кейсу №3: виконано.

Тест-кейс №4. Додавання мітки.

Кроки:

1. Запустити додаток.

2. У пункті меню «Tools» обрати пункт підменю «Edit available tags…».

3. У вікні, що з’явилося, у полі поруч з кнопкою «Add tag» ввести назву мітки, яку потрібно додати.

4. Натиснути кнопку «Add tag».

Очікувані результати зображено на рисунку 4.4.

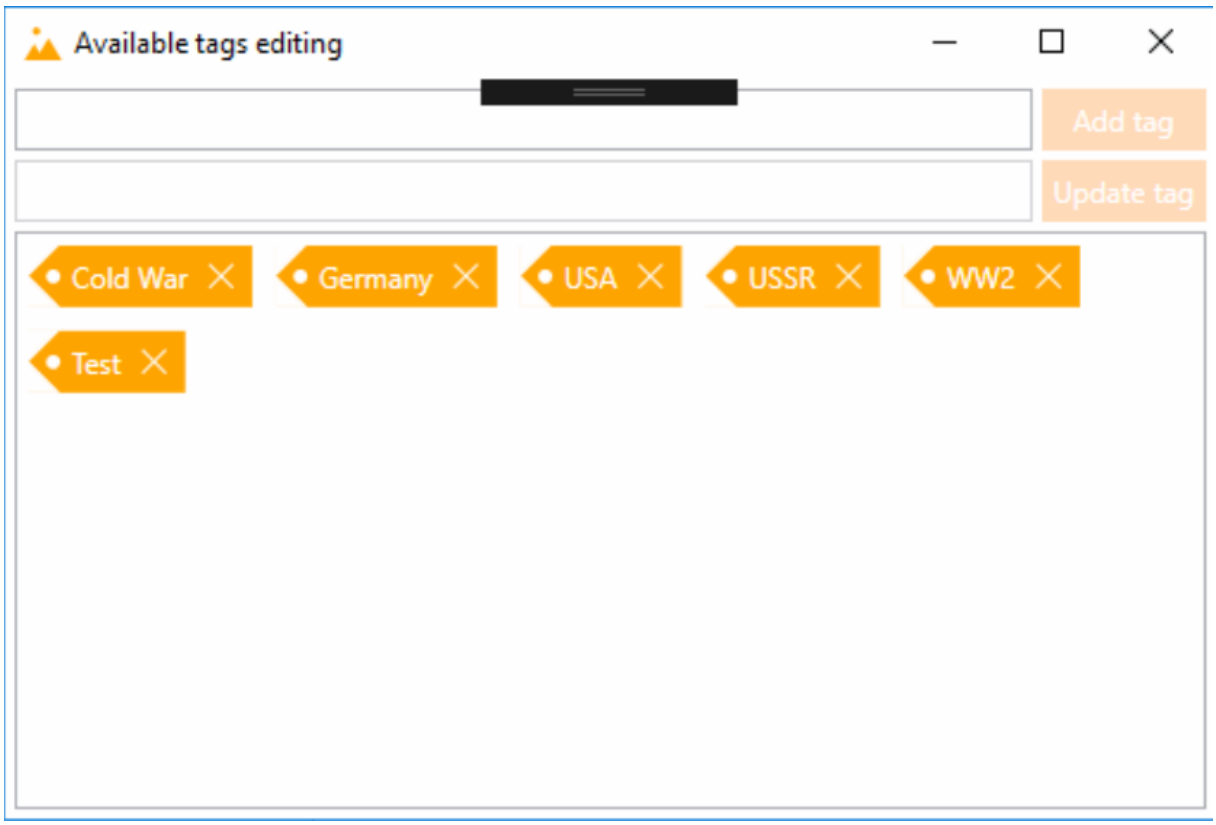


Рисунок 4.4 – Очікуваний результат додавання мітки

Результат тест-кейсу №4: виконано.

Тест-кейс №5. Оновлення мітки.

Кроки:

1. Запустити додаток.

2. У пункті меню «Tools» обрати пункт підменю «Edit available tags…».

3. У вікні, що з’явилося, у полі поруч з кнопкою «Update tag» змінити назву мітки, яку потрібно оновити.

4. Натиснути кнопку «Update tag».

Очікувані результати зображено на рисунку 4.5.

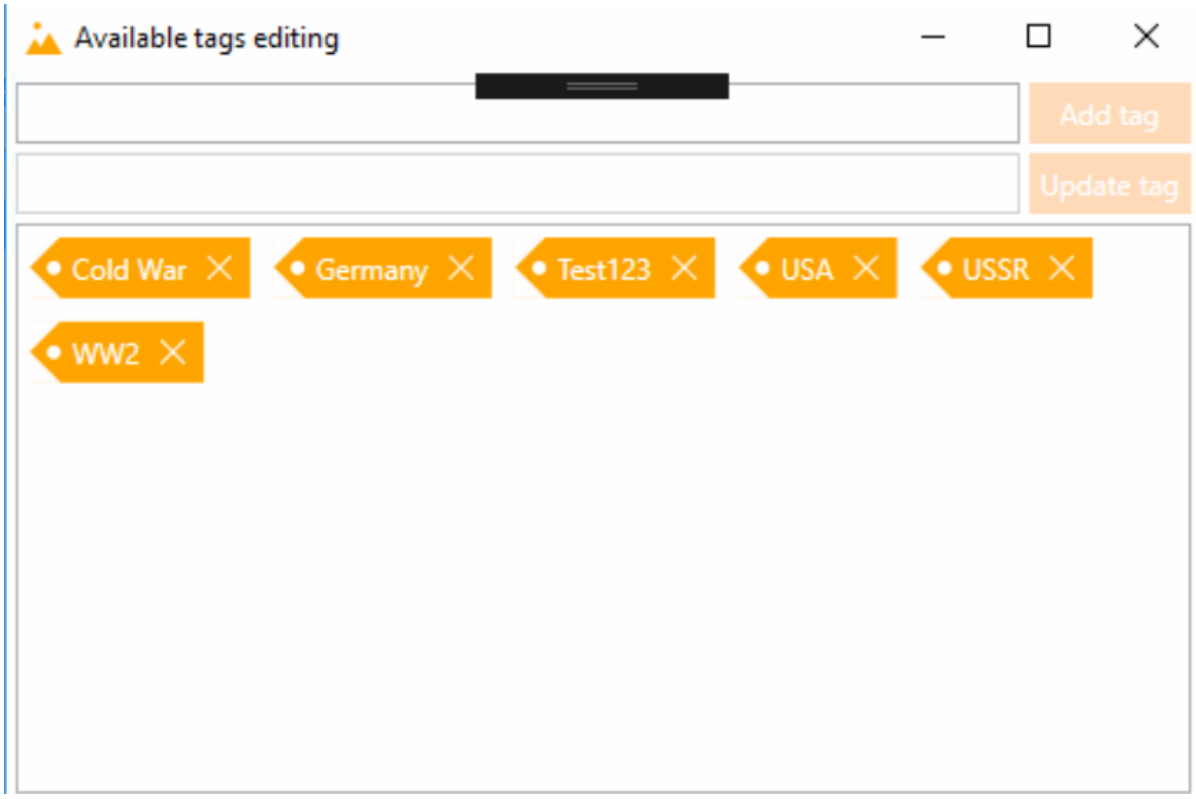


Рисунок 4.5 – Очікуваний результат оновлення мітки

Результат тест-кейсу №5: виконано.

Тест-кейс №6. Видалення мітки.

Кроки:

1. Запустити додаток.

2. У пункті меню «Tools» обрати пункт підменю «Edit available tags…».

3. У вікні, що з’явилося, натиснути на зображення хрестика на мітці, яку потрібно видалити.

Очікувані результати зображено на рисунку 4.6.

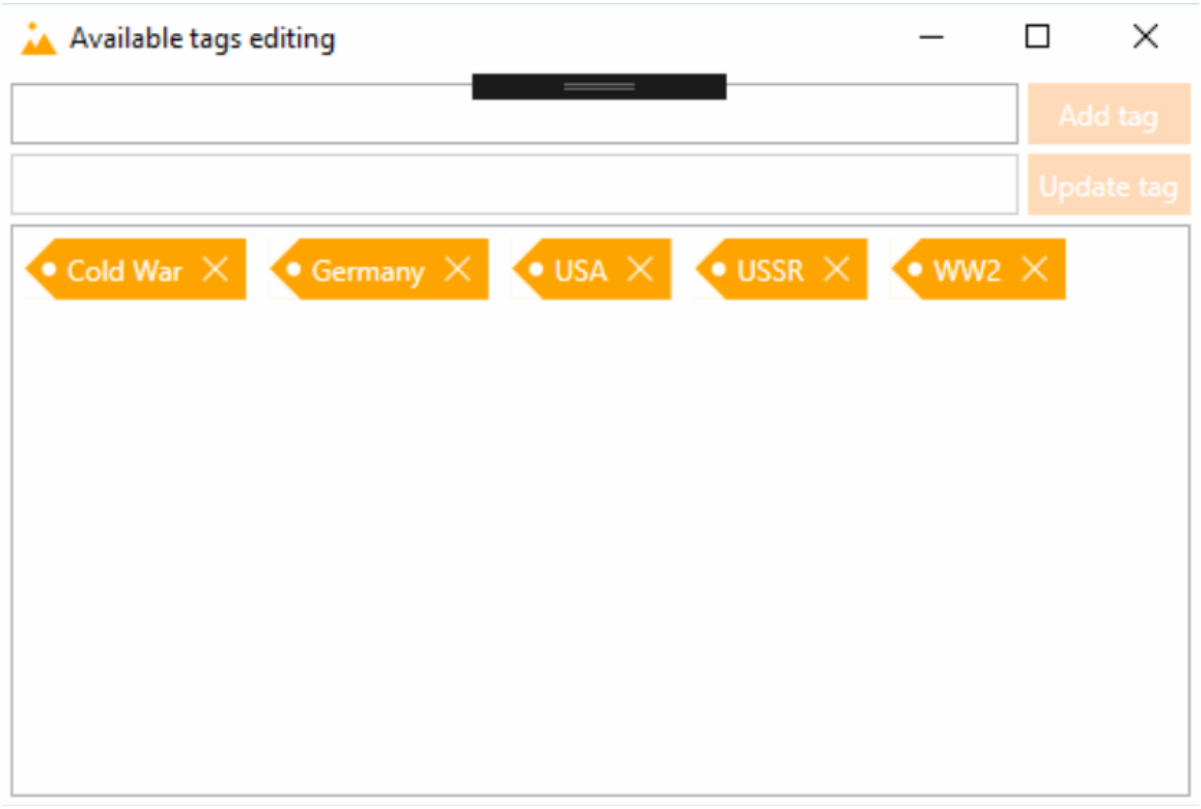


Рисунок 4.6 – Очікуваний результат видалення мітки

Результат тест-кейсу №6: виконано.

Тест-кейс №7. Додавання мітки до зображення.

Кроки:

1. Запустити додаток.

2. Натиснути кнопку «Load more».

3. Натиснути зображення, до якого потрібно додати мітку.

4. У вікні, що з’явилося, обрати з випадаючого списку(поряд з міткою «Available tags») мітку, яку треба додати до зображення.

5. Натиснути кнопку «Add tag».

Очікувані результати зображено на рисунку 4.7.

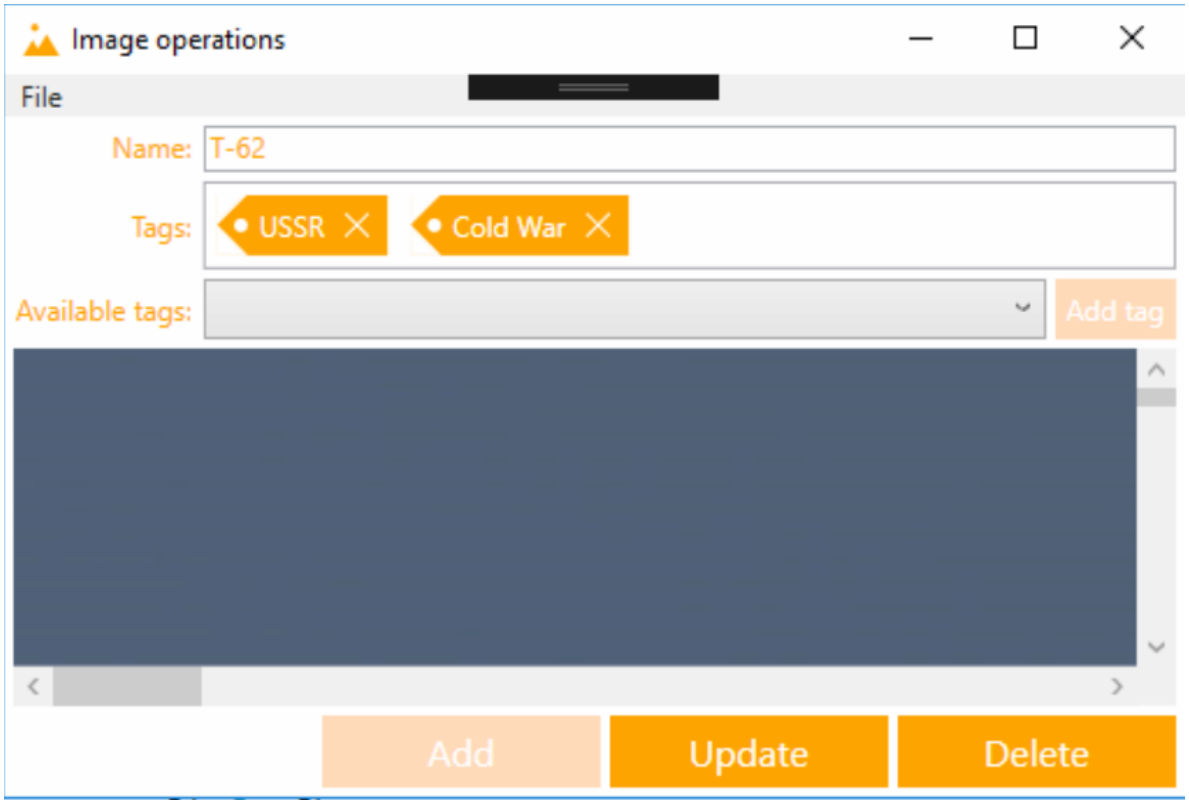


Рисунок 4.7 – Очікуваний результат додавання мітки до зображення

Результат тест-кейсу №7: виконано.

Тест-кейс №8. Видалення мітки у зображення.

Кроки:

1. Запустити додаток.

2. Натиснути кнопку «Load more».

3. Натиснути зображення, до якого потрібно додати мітку.

4. У вікні, що з’явилося, натиснути на зображення хрестика на мітці, яку потрібно видалити.

Очікувані результати зображено на рисунку 4.8.

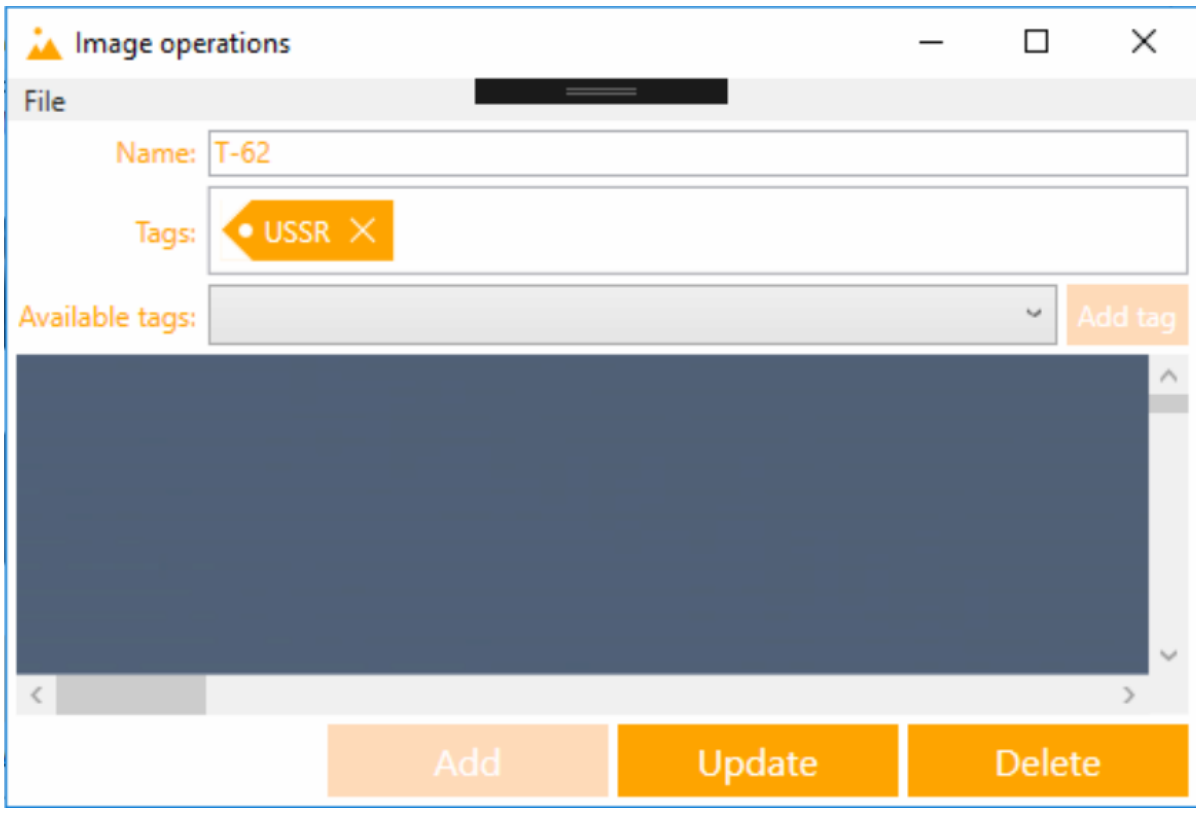


Рисунок 4.8 – Очікуваний результат видалення мітки у зображення

Результат тест-кейсу №8: виконано.

Виходячи з результатів тест-кейсів, тестування системи пройшло успішно.

4.3 Інструкція користувача системи

Для того, щоб почати роботу із системою, потрібно запустити додаток з графічним інтерфейсом. Для цього потрібно двічі клацнути лівою кнопкою миші на виконуваний файл додатку аби на ярлик додатку на робочому столі, якщо його було попередньо створено.

Для того, щоб переглянути зображення, які знаходяться у базі даних, потрібно запустити додаток та натиснути на кнопку «Load more»(рисунок 4.9). Під час завантаження кнопка змінить колір на більш блідий та стане неактивною. Це зроблено для того, щоб попередити повторні натискання на неї під час завантаження, які можуть створити у користувача ілюзію того, що система не відповідає на його дії. Крім того, для більшої інформативності інтерфейсу присутня панель статусу, яка інформує користувача про поточний стан додатку з графічним інтерфейсом. Коли він готовий до завантаження мініатюр зображень, панель статусу містить напис «Ready.». Коли усі доступні мініатюри зображень будуть завантажені, статус зміниться на «All available thumbnails were loaded.».

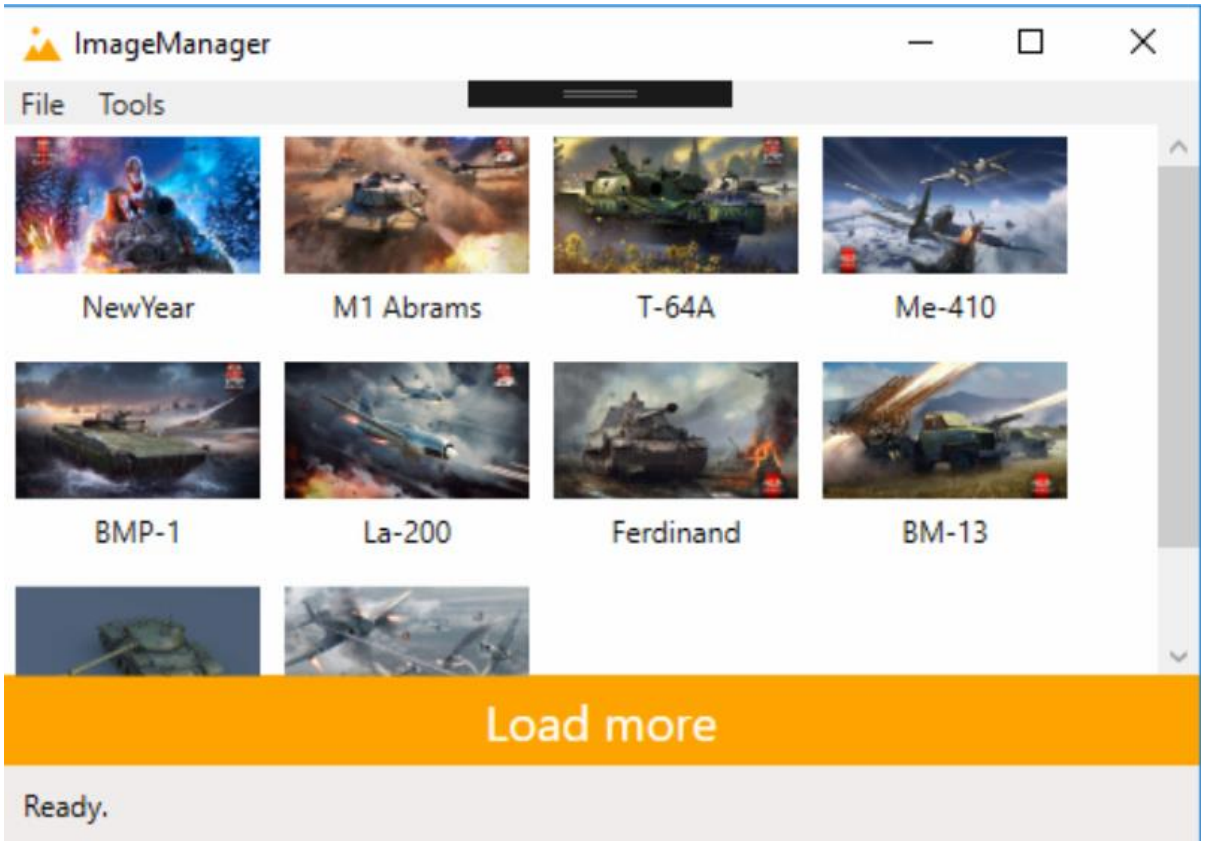


Рисунок 4.9 – Головне вікно

Для того, щоб додати зображення до бази даних, потрібно спочатку відкрити діалог вибору файлу(рисунок 4.10). Для цього потрібно запустити додаток з графічним інтерфейсом користувача. Потім потрібно у верхній частині головного вікна знайти пункт меню «File». Потім потрібно навести курсором на цей пункт меню. З’явиться підменю. У ньому потрібно обрати пункт «Load image from file…». Після цього з’явиться діалог для вибору файлу зображення.

На вибір файлу накладено обмеження, щоб користувач міг обрати лише файл, який представляє собою зображення, а не який-небудь інший файл.

Діалог дозволяє обрати файл з будь-якої директорії.

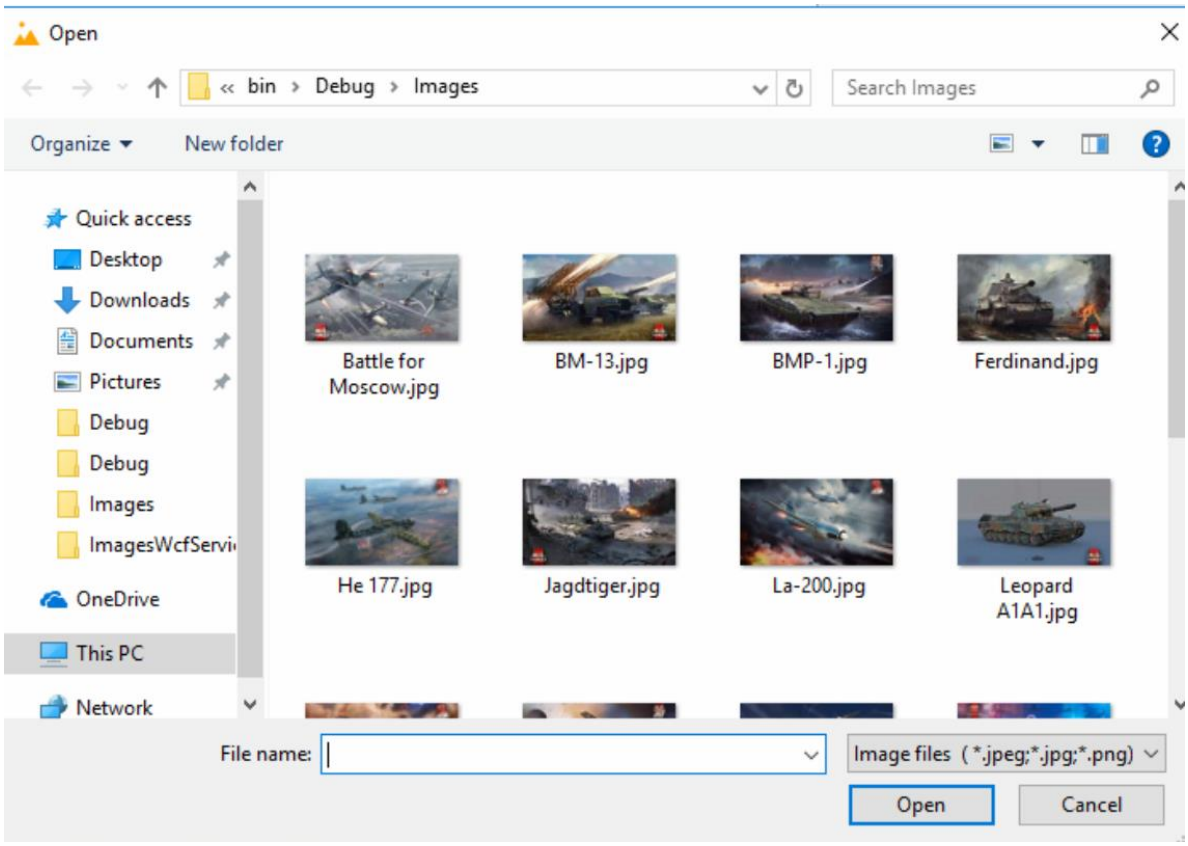


Рисунок 4.10 – Діалог вибору файлу для відкриття

Після вибору файлу зображення у відповідному діалозі відкриється вікно для проведення дій над зображенням(рисунок 4.11). Воно містить кнопки для основних дій. Крім того, можна змінити назву зображення. Вона міститься у окремому полі вводу та відповідає назві файлу, з якого воно було завантажене. Проте, як вище було зазначено, користувач може обрати будь-яке ім’я для зображення, яке не є пустим та є меншим за 100 символів. Також можна додати мітки до зображення. Для цього потрібно обрати мітку з випадаючого списку та натиснути кнопку «Add tag». Після цього обрана мітка з’явиться у області під назвою зображення. Якщо користувач передумає додавати мітку, він може її видалити у зображення. Для цього потрібно натиснути на зображення хрестика на мітці.

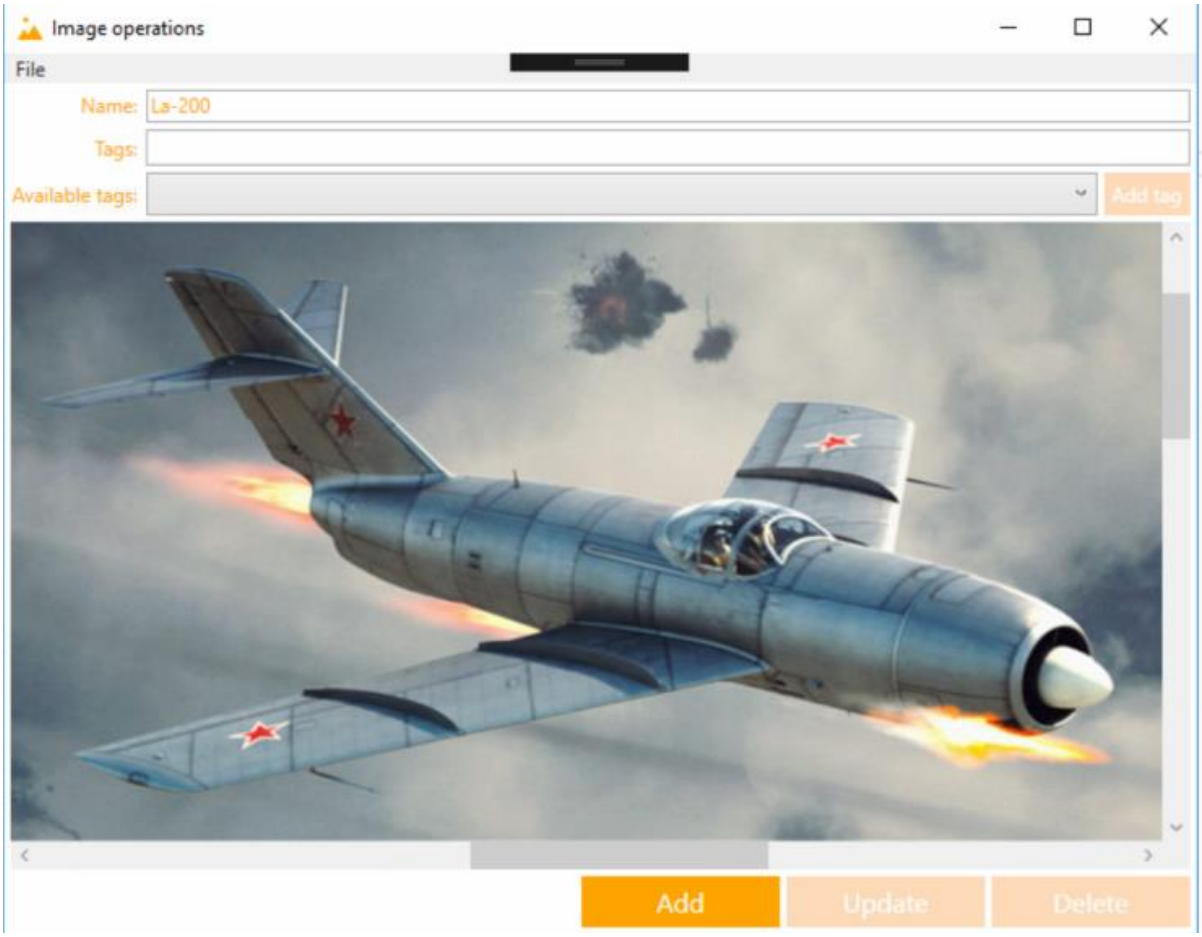


Рисунок 4.11 – Вікно для проведення дій над зображенням

Щоб відредагувати доступні мітки потрібно запустити додаток. Після цього потрібно знайти пункт меню «Tools». У підменю цього пункту потрібно обрати пункт «Edit available tags…». Після натискання на цей пункт меню з’явиться вікно для редагування доступних міток(рисунок 4.12). Це вікно дозволяє користувачеві додавати мітки, оновлювати їх та видаляти.

Щоб додати мітку потрібно ввести її назву у поле поряд із кнопкою «Add tag» та натиснути на неї.

Щоб оновити мітку потрібно натиснути на неї. Після цього її назва з’явиться у полі поряд з кнопкою «Update tag». Після зміни назви мітки можна натиснути цю кнопку і мітку буде оновлено.

Щоб видалити мітку потрібно натиснути на зображення у вигляді хрестика на мітці.

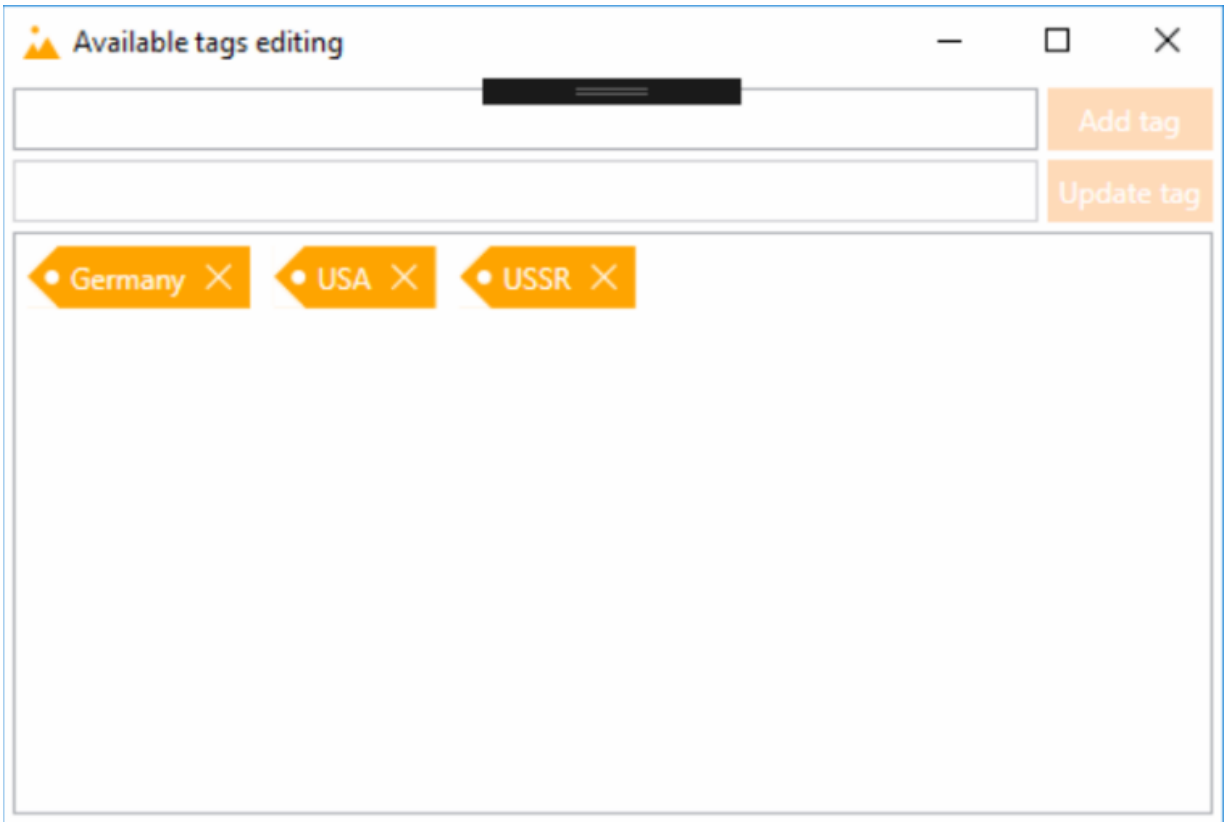


Рисунок 4.12 – Вікно для редагування міток

Для того, щоб шукати зображення за мітками потрібно відкрити відповідне вікно(рисунок 4.13). Для цього потрібно виконати наступні кроки:

1. Запустити додаток.

2. Навести курсором на пункт меню «Tools».

3. У підменю, що з’явилося обрати пункт «Search images by tags».

Вікно, що з’явиться дає користувачеві шукати зображення за мітками. Для цього потрібно додати мітки, за якими проводитиметься пошук. Щоб зробити це потрібно з випадючого списку обрати мітку та натиснути на кнопку «Add tag». Можна також прибирати мітки, за якими проводитиметься пошук. Для цього потрібно натиснути на зображення хрестика на мітці.

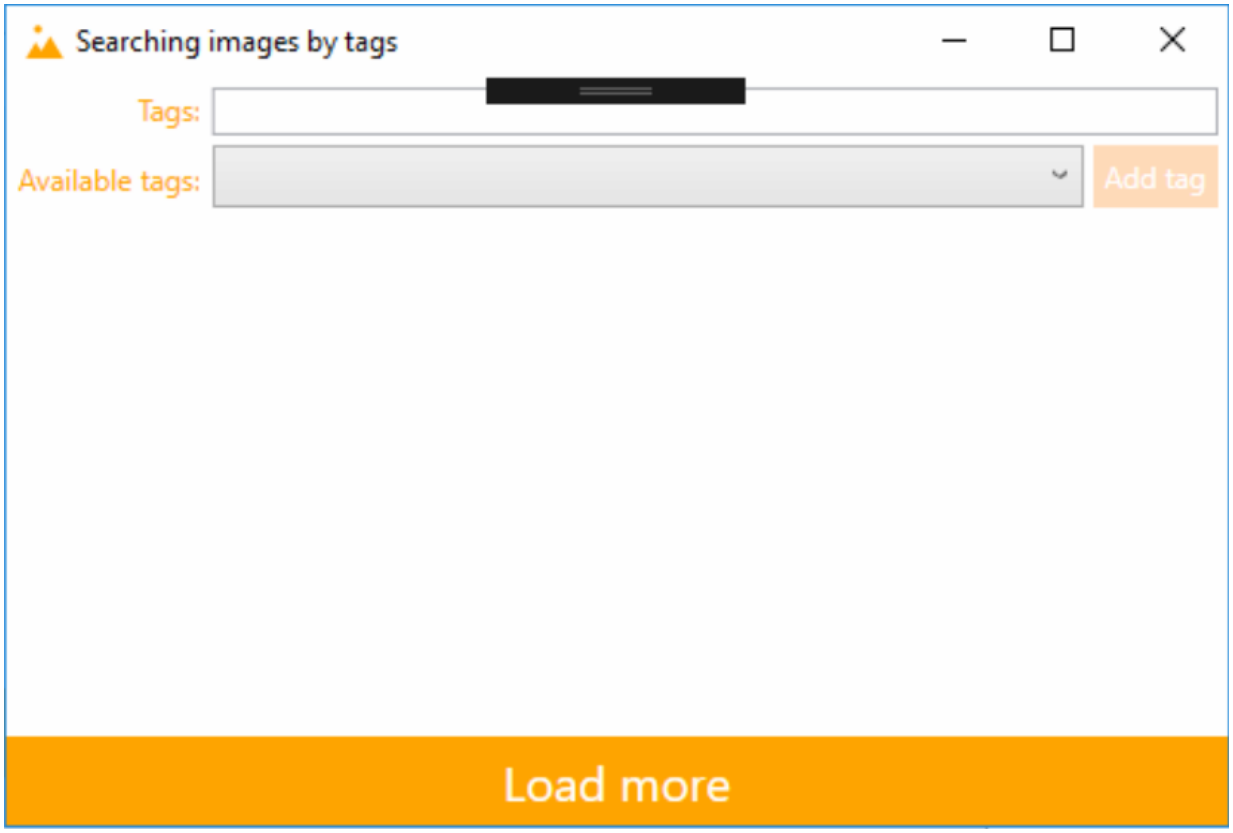


Рисунок 4.13 – Вікно для пошуку зображень за мітками

Після додавання міток для пошуку можна натиснути на кнопку «Load more». Це завантажить порцію мініатюр, які мають серед своїх міток ті, за якими проводиться пошук(рисунок 4.14). Кнопку «Load more» можна натискати до тих пір, поки не будуть завантажені мініатюри усіх зображень з мітками, за якими ведеться пошук.

Після зміни набору міток, за якими потрібно проводити пошук, завантаження мініатюр починається спочатку.

При натисканні на мініатюру відкривається вікно дій над зображеннями(як і в головному вікні).

Панель статусу у нижній частині вікна слугує для повідомлення користувача про стан додатку.

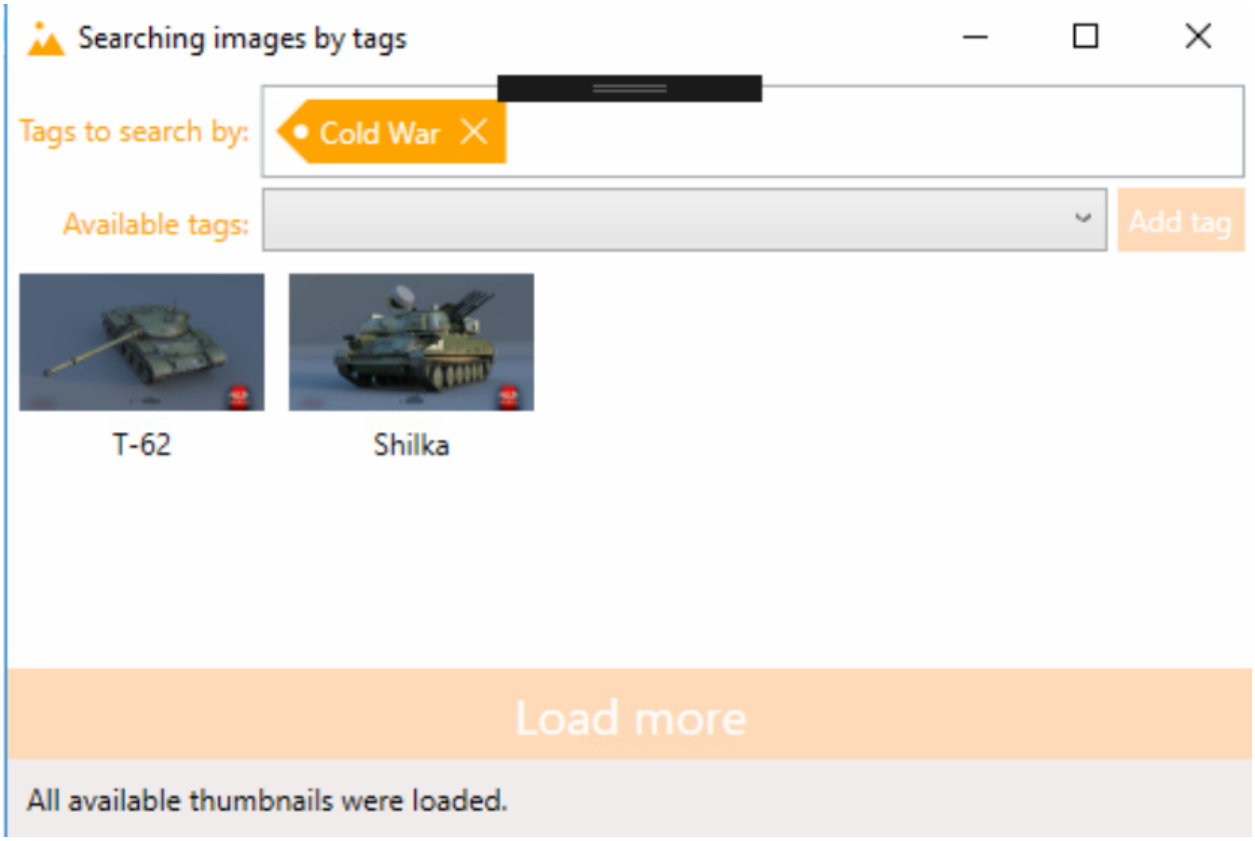


Рисунок 4.14 – Вікно пошуку за мітками із результатом пошуку

Для того, щоб завершити роботу із системою, потрібно у головному вікні натиснути на стандартну кнопку «Закрити» у правому верхньому куті або обрати пункт меню «File» та натиснути на пункт підменю «Close»(рисунок 4.15).

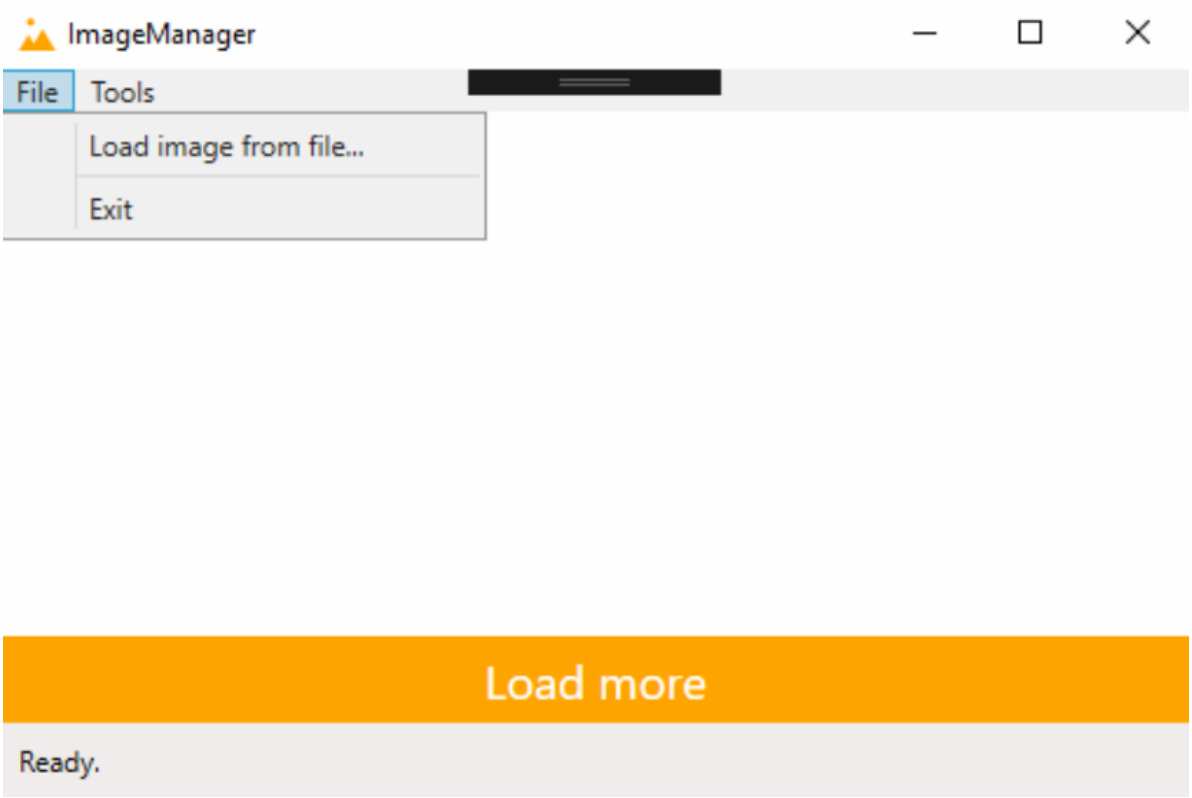


Рисунок 4.15 – Пункт меню для завершення роботи із системою

Отже, було створено інструкцію користувача, яка дозволяє досягнути бажаних результатів при роботі з системою.

4.4 Висновки

Отже, під час написання розділу було розглянуто методики тестування та обрано серед них таку, яка відповідає умовам тестування програмних модулів поточного проекту. Крім, того було проведене тестування основних функцій програмних модулів системи, яке показало, що функціонал працює коректно. Також було розроблено інструкцію користувача.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Катєльніков Д.І. та Арсенюк І.Р.

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1

Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Експерт 1	2. Експерт 2
	Бали, виставлені експертами:	
1	4	4
2	3	3
3	3	4
4	4	3
5	3	4
6	4	4
7	3	3
8	4	4
9	4	4
10	4	3
11	3	4
12	3	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 44$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 43{,}5$	

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько-технологічної роботи

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де

М – місячний посадовий оклад конкретного розробника;

$Т_р$ – кількість робочих днів у місяці, $Т_р$ = 22дні;

t – число днів роботи розробника, t = 45 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2

Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	5500	250	7	1750
Інженер-програміст	3500	159,09	45	7159,05
Всього:				8909,05

Розрахуємо додаткову заробітну плату:

$$З_{дод}= 0,1\cdot 8909,05=890,9 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати(формула 5.2):

$$Н_{зп}= (З_о + З_р)\cdot \frac{\beta}{100}, \quad (5.2)$$

$$Н_{зп}= (8909,05+890,9)\cdot \frac{36,3}{100} = 3557,38 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за формулою 5.3:

$$А=\frac{Ц\cdot Н_а}{100}\cdot \frac{Т}{12}, \quad (5.3)$$

де

Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.3

Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань,грн
Персональний комп’ютер	10000	25	3	625
Всього:				625

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою 5.4:

$$K = \sum_1^n \text{Н}_\text{і} \cdot \text{Ц}_\text{і} \cdot \text{К}_\text{і}, \quad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.4

Витрати на комплектуючі, що були використані для розробки ПЗ

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	180	1	180
Пачка паперу	уп.	120	1	120
Ручка	шт.	10	1	10
Всього з урахуванням транспортних витрат				341

Витрати на силову електроенергію розраховуються за формулою 5.5:

$$\text{В}_\text{е}=\text{В}\cdot\text{П}\cdot\text{Ф}\cdot\text{К}_\text{п}, \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,7 грн/кВт);

П – установлена потужність комп’ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп’ютера (Ф=150 год.);

$\text{К}_\text{п}$ – коефіцієнт використання потужності ($\text{К}_\text{п}< 1$, $\text{К}_\text{п} = 0{,}8$).

$$\text{В}_\text{е}= 1{,}7 \cdot 0{,}6 \cdot 150 \cdot 0{,}8 = 122{,}4 \text{ (грн.)}$$

Розрахуємо інші витрати $\text{В}_\text{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто(формула 5.6):

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1,5 * (8909,05 + 890,9) = 14699,92 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

В = 8909,05+890,9+3557,38+625+341+122,4+14699,92= 29145,65 (грн.)

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою 5.7:

$$В_{заг} = \frac{В_{ін}}{\alpha} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{29145,65}{1} = 29145,65$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою 5.8:

$$ЗВ = \frac{В_{заг}}{\beta} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{29145,65}{0,9} = 32384,05 \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою 5.9:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \Delta N)_i \quad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

Δ N – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшаться на 35 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 35 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 150 користувачів, протягом другого року – на 125 користувачів, протягом третього року – 100 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 500 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 250 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого прибутку $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 35 \cdot 500 + (250 + 35) \cdot 150 = 60250 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 35 \cdot 500 + (250 + 35) \cdot (150 + 125) = 95875 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 35 \cdot 500 + (250 + 35) \cdot (150 + 125 + 100) = 124375 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $E_{абс}$ вкладених інвестицій розраховується за формулою 5.10:

$$E_{абс} = (ПП - PV), \quad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

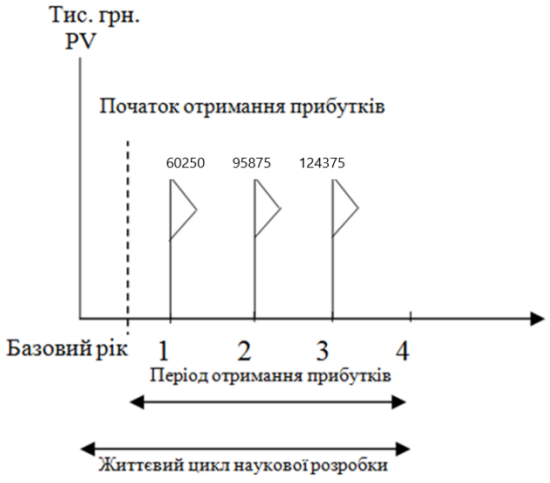


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою 5.11:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{32384,05}{(1+0,1)^0} + \frac{60250}{(1+0,1)^2} + \frac{95875}{(1+0,1)^3} + \frac{124375}{(1+0,1)^4} = 239159,52 \text{ (грн.)}$$

Тоді розрахуємо $E_{абс}$:

$$Е_{абс} = 239159,52 - 32384,05 = 206775,47 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою 5.12:

$$Е_в = \sqrt[T]{1 + \frac{Е_{абс}}{PV}} - 1 \quad (5.12)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$Е_в = \sqrt[3]{1 + \frac{206775,47}{32384,05}} - 1 = 0,94 \text{ або } 94\ \%$$

Далі, розраховану величина $Е_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою 5.13:

$$\tau = d + f, \tag{5.13}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0,2 + 0,1 = 0,3$$

Оскільки $Е_в$ = 94% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $Т_{ок}$ розраховується за формулою 5.14:

$$Т_{ок} = \frac{1}{Е_в} \tag{5.14}$$

$$T_{ок} = \frac{1}{0,94} = 1,06 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

5.5 Висновки

Отже, було проведену оцінку комерційного потенціалу розробки, в ході якої було проведено аналіз за дванадцятьма критеріями та п’ятибальною шкалою, та з участю висококваліфікованих експертів, підсумком якої є висновок про високий комерційний потенціал.

Також було виконано прогнозування витрат на виконання науково-дослідної роботи, в результаті якого було встановлено, що сума усіх витрат становитиме близько тридцятьох двох тисяч гривень. Було розраховано заробітну плату розробників. Також було розраховано амортизаційні витрати для програмного забезпечення. Крім того, було розраховано витрати на електроенергію. Окремо було розраховано інші витрати та проведено загальний обрахунок усіх вказаних статей витрат.

Крім того, було виконано прогнозування комерційних ефектів від реалізації результатів розробки, результатом якого є висновок про те, що чистий прибуток за перший рік становитиме близько шістдесятьох тисяч гривень, за другий – близько дев’яноста п’яти тисяч гривень, за третій – близько ста двадцяти чотирьох тисяч гривень.

Також було проведено розрахунок ефективності вкладених інвестицій та оцінку періоду їх окупності.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи було проведено аналіз сучасного стану вирішення задачі класифікації зображень. Під час аналізу було доведено актуальність створення нових методів та програмного забезпечення. Було проаналізовано методи класифікації графічних зображень та обґрунтовано необхідність створення нових, адже існуючі мають певні недоліки. Було здійснено аналіз аналогів серед програмного забезпечення, який виявив їх недоліки та показав необхідність створення нового ПЗ.

Було вперше запропоновано метод класифікації зображень, особливість якого полягає у застосуванні кластеру нейронних мереж, кожна мережа якого навчена на певний набір класів, що дає можливість відносити зображення одразу до кількох класів. Також вдосконалено процес додавання нового класу до нейронної мережі, який відрізняється від аналогів пришвидшеним етапом навчання за рахунок перенавчання окремих мереж у кластері замість перенавчання великої нейронної мережі, що дає можливість швидше додавати новий клас.

Розроблено програмне забезпечення для класифікації зображень, в ході якого було розроблено як структуру усієї системи, так і структуру основних її складових частин. Технологіями для реалізації системи було обрано .NET, WPF, WCF та Entity Framework. Також було розроблено вихідний код програмної системи та описано усі основні його елементи.

Також було проведено тестування системи, в ході якого було обрано методику тестування, розроблено тест-кейси та створено інструкцію користувача.

Крім того, було проведено економічне обґрунтування розробки, в ході якого було проаналізовано комерційний потенціал розробки, розраховано витрати на створення програмного забезпечення та обраховано прибутки. В результаті розробку було визнано економічно доцільною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Майданюк В.П. Людино-машинна взаємодія. ВНТУ, 2004. 255 с.

2. iCloud for Windows. URL: https://support.apple.com/en-us/HT204283 (Last accessed: 24.11.19).

3. OneDrive. URL: https://onedrive.live.com/about/uk-ua/ (Last accessed: 25.11.19).

4. Google Диск. URL: https://www.google.com/intl/ru_ALL/drive/ (дата звернення: 26.11.19).

5. BMP. URL: https://ru.wikipedia.org/wiki/BMP (Last accessed: 27.11.19).

6. GIF. URL: https://ru.wikipedia.org/wiki/GIF (Last accessed: 28.11.19).

7. JPEG. URL: https://ru.wikipedia.org/wiki/JPEG (Last accessed: 15.10.19).

8. PNG. URL: https://ru.wikipedia.org/wiki/PNG (Last accessed: 16.10.19).

9. Тиори Т. Проектирование структур баз данных. Москва, 1985. 287 с.

10. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. Москва, 2000. 1127 с.

11. Мейер Д. Теория реляционных баз данных: Пер. с англ.. Москва, 1987. 185 с.

12. Мейер Д. Теория реляционных баз данных: Пер. с англ.. Москва, 1987. 215 с.

13. Хомоненко А.Д. Базы данных: Учебник. Санкт-Петербург, 2000. 433 с.

14. Дейт К. Введение в системы баз данных. Москва, 2001. 1023 с.

15. Фиайли К. SQL: Руководство по изучению языка. Москва, 2003. 456 с.

16. Дрибас В.П. Реляционные модели данных. Москва, 1992. 192 с.

17. MVVM. URL: https://ru.wikipedia.org/wiki/Model-View-ViewModel (дата звернення: 25.10.19).

18. Троелсен Е. C# 6.0 and the .NET 4.6 Framework Seventh edition. Apress, 2015. 1215 с.

19. WPF. URL: https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation (Last accessed: 20.10.19).

20. Windows Forms.URL: https://ru.wikipedia.org/wiki/Windows_Forms (Last accessed: 24.10.19).

21. WCF. URL: https://ru.wikipedia.org/wiki/Windows_Communication_Foundation (Last accessed: 20.10.19).

22. .NET Remoting. URL: https://ru.wikipedia.org/wiki/.NET_Remoting (Last accessed: 10.11.19)

23. Microsoft SQL Server. URL: https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F_%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F_Microsoft_SQL_Server (дата звернення: 15.11.19).

24. MySQL. URL: https://ru.wikipedia.org/wiki/MySQL (Last accessed: 10.11.19)

25. Шмуллер Д. UML за 24 години. Київ, 2005. 64с.

26. Шмуллер Д. UML за 24 години. Київ, 2005. 177с.

27. Тестування ПЗ. URL: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F (дата звернення: 12.11.19).

28. Тестування білої скрині. URL: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B1%D0%B5%D0%BB%D0%BE%D0%B3%D0%BE_%D1%8F%D1%89%D0%B8%D0%BA%D0%B0 (дата звернення: 15.11.19).

29. Тестування чорної скрині. URL: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D0%B8_%D1%87%D1%91%D1%80%D0%BD%D0%BE%D0%B3%D0%BE_%D1%8F%D1%89%D0%B8%D0%BA%D0%B0 (дата звернення: 16.11.19).

30. Тест-кейс. URL: https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F (дата звернення: 16.11.19).

ДОДАТКИ

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

УЗГОДЖЕНО
Директор ТОВ «Дельфи» Бондар Н. П.
"____"_____2019 р.

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"____"_____2019 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методів і програмного забезпечення для класифікації графічних зображень з використанням технологій .NET, WPF, WCF та Entity Framework»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. Д.І. Катєльніков
"____"_____2019 р.
Виконав:
_____студент гр.1ПІ-18м О.Ю. Трач
"____"_____2019 р.

Вінниця – 2019 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмного забезпечення для класифікації графічних зображень з використанням технологій .NET, WPF, WCF та Entity Framework».

Галузь застосування – системи зберігання та обробки інформації.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є розширення функціональних можливостей процесу класифікації за рахунок додавання можливості віднесення зображення одразу до кількох класів та пришвидшення налаштування системи класифікації за рахунок пришвидшення процесу додавання нового класу.

Призначення роботи – розробка методів і засобів класифікації графічних зображень.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Трач О.Ю., Катєльніков Д.І. Розробка методів і програмного забезпечення для класифікації графічних зображень з використанням технологій AForge.NET і платформи .NET Framework на Міжнародній Internet-конференції «Електронні інформаційні ресурси: створення, використання, доступ», м. Вінниця, с. 259-264, 2019.

2. Аркадьев А.Г., Браверман Э.М. Обучение машины классификации объектов. Москва, 1971.

3. Горбань А.Н. Обучение нейронных сетей. Москва, 1990. 160 с.

4. Аркадьев А.Г., Браверман Э. М. Обучение машины распознаванию образов. Москва, 1964.

5. Технічні вимоги

Вихідні дані до роботи: максимальна роздільна здатність зображення дорівнює 2560 на 1440 пікселів, максимальна кількість класів дорівнює 100, час виконання класифікації не більше 30 секунд

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до МКР;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз сучасного стану вирішення задачі	
2	Розробка методів класифікації зображень	
3	Розробка програмного забезпечення	
4	Тестування програмного забезпечення	
5	Економічна частина	

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б АКТ ВПРОВАДЖЕННЯ

ДОДАТОК В БЛОК-СХЕМИ АЛГОРИТМІВ

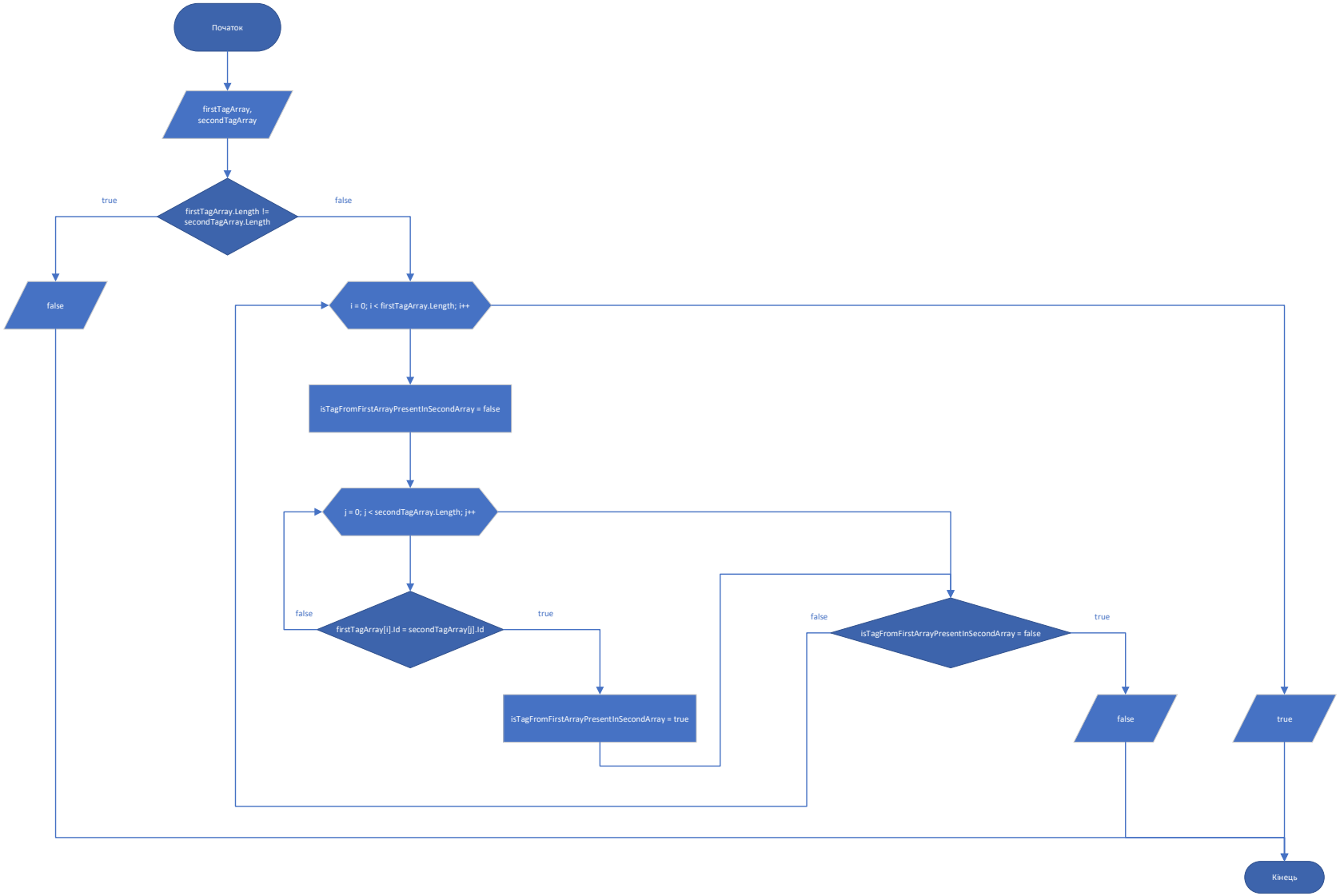


Рисунок В.1 – Блок-схема алгоритму порівняння двох масивів міток на однаковість

ДОДАТОК Г ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д.т.н., професор _____ О. Н. Романюк

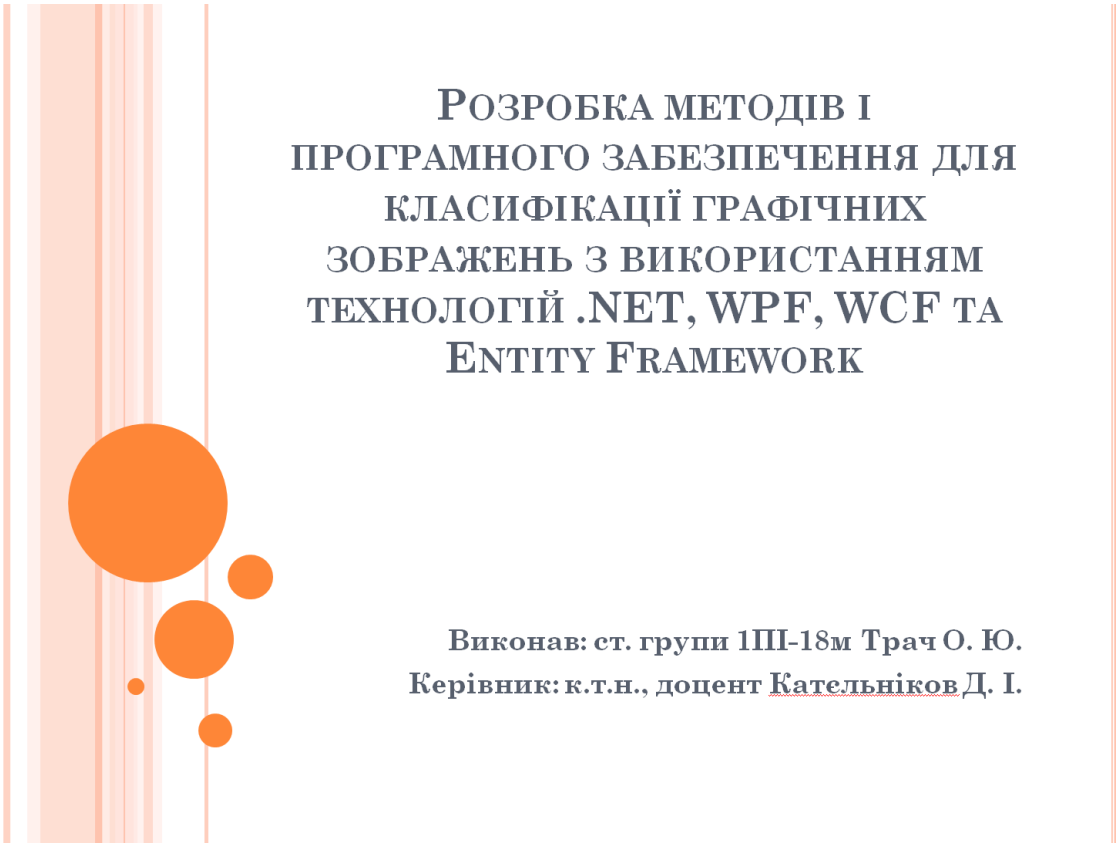
Науковий керівник, к.т.н., доцент _____ Д. І. Катєльніков

Рецензент, к.т.н., доцент _____ І. Р. Арсенюк

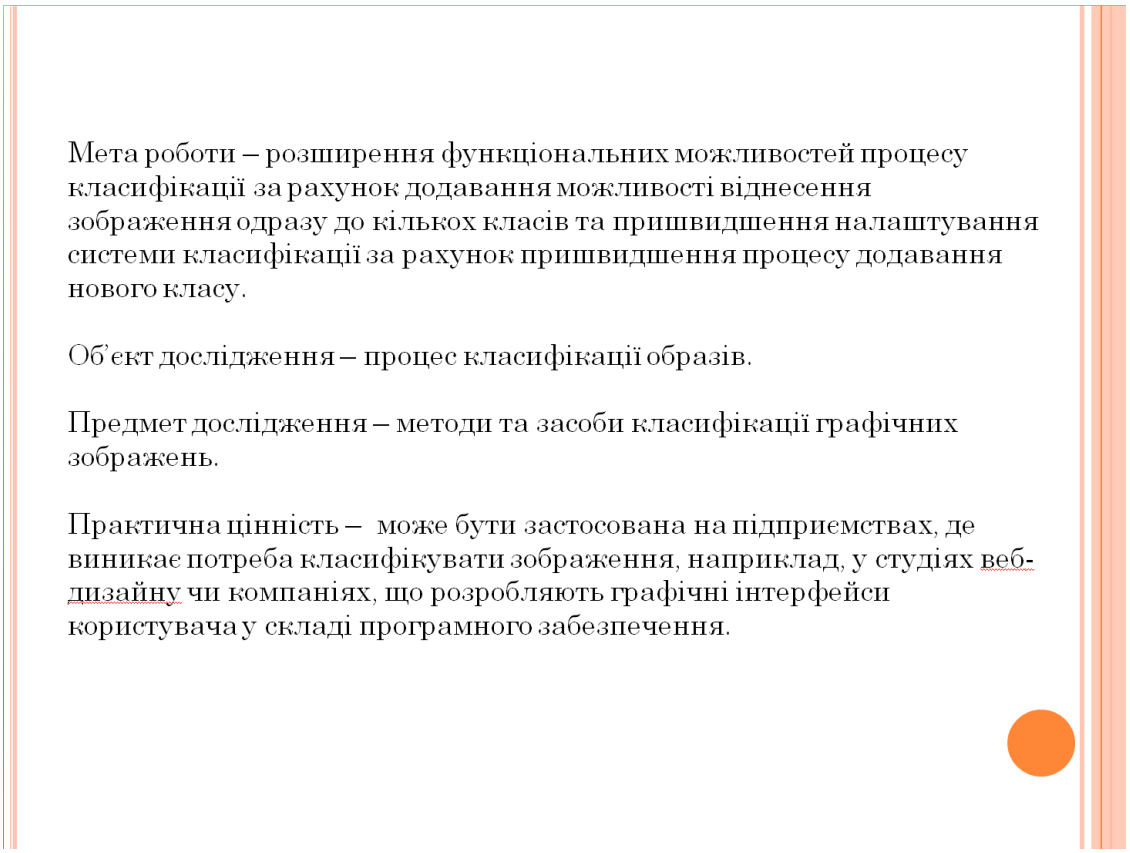
Нормоконтроль, к.т.н., доцент _____ Д. І. Катєльніков

Виконавець, студент групи 1ПІ-18м _____ О. Ю. Трач

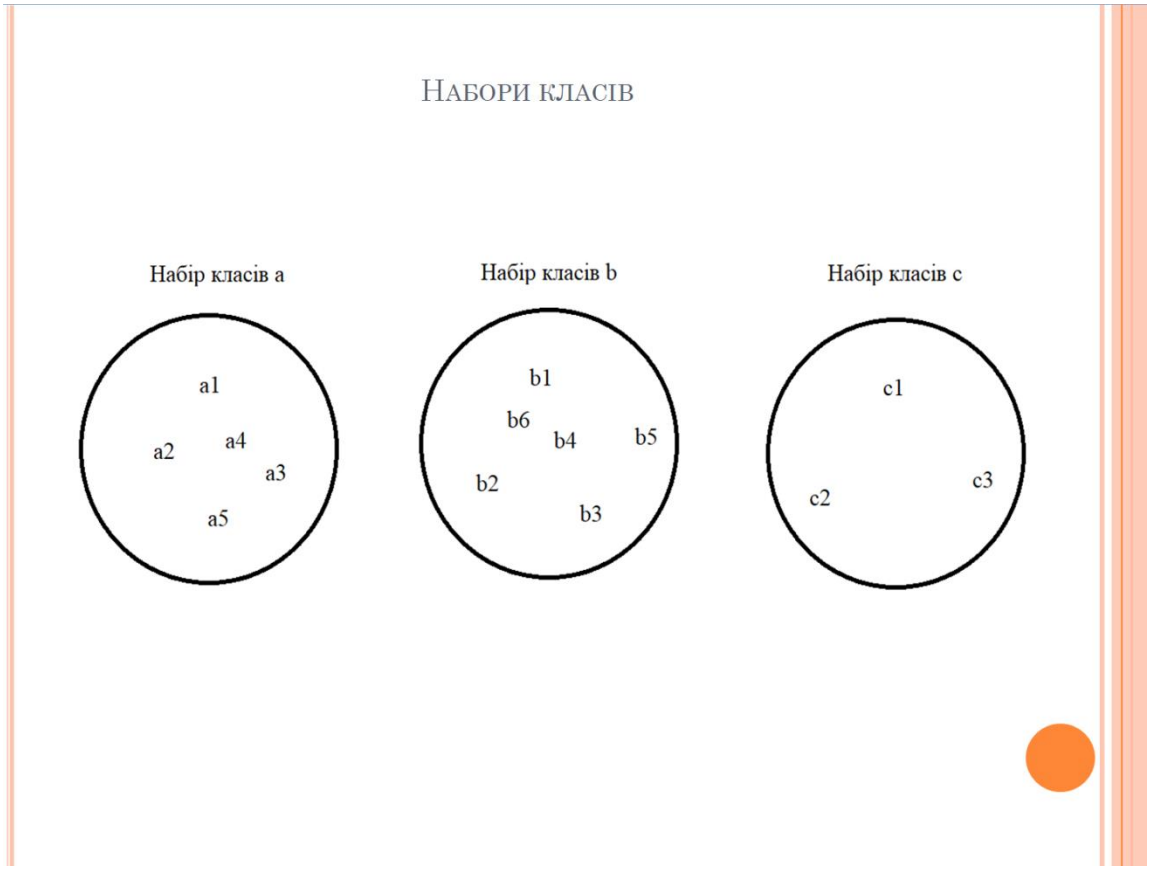
Плакат 1 – Тема роботи, автор та керівник



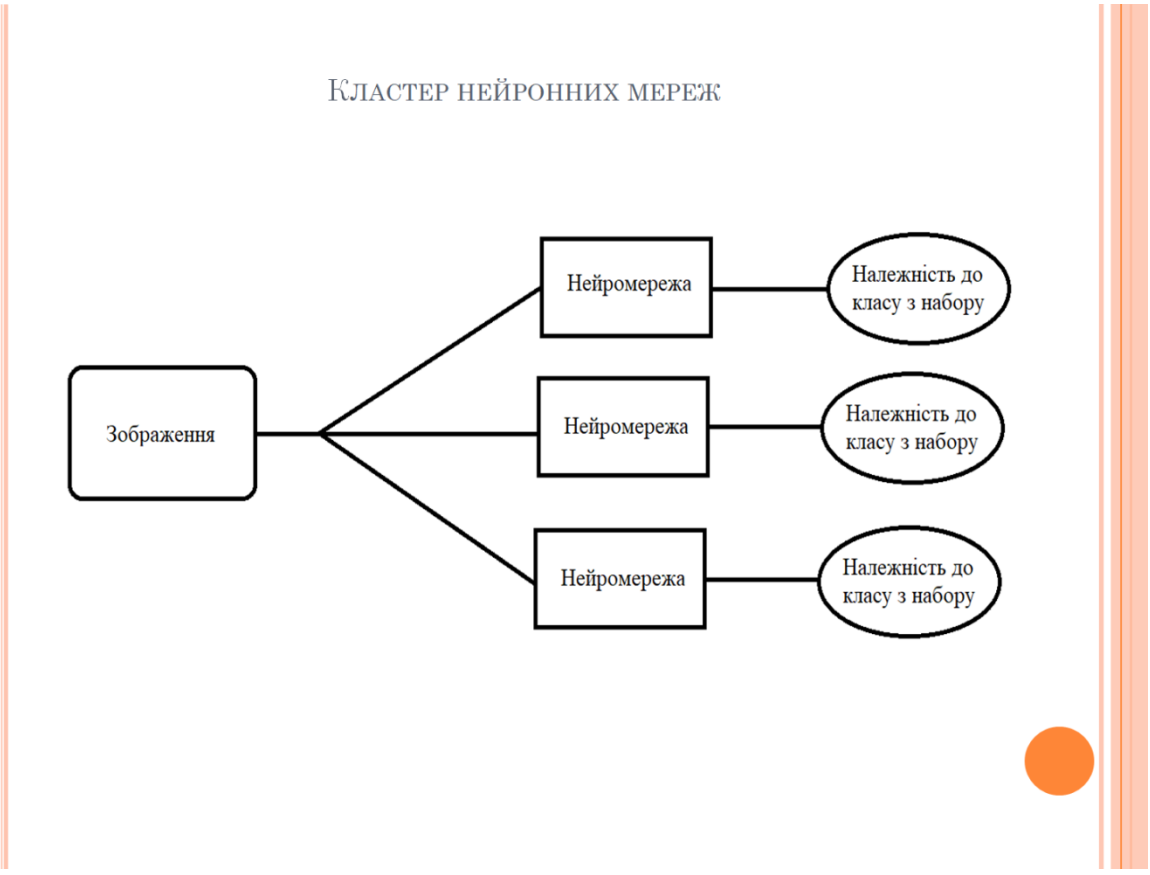
Плакат 2 – Загальна інформація про роботу



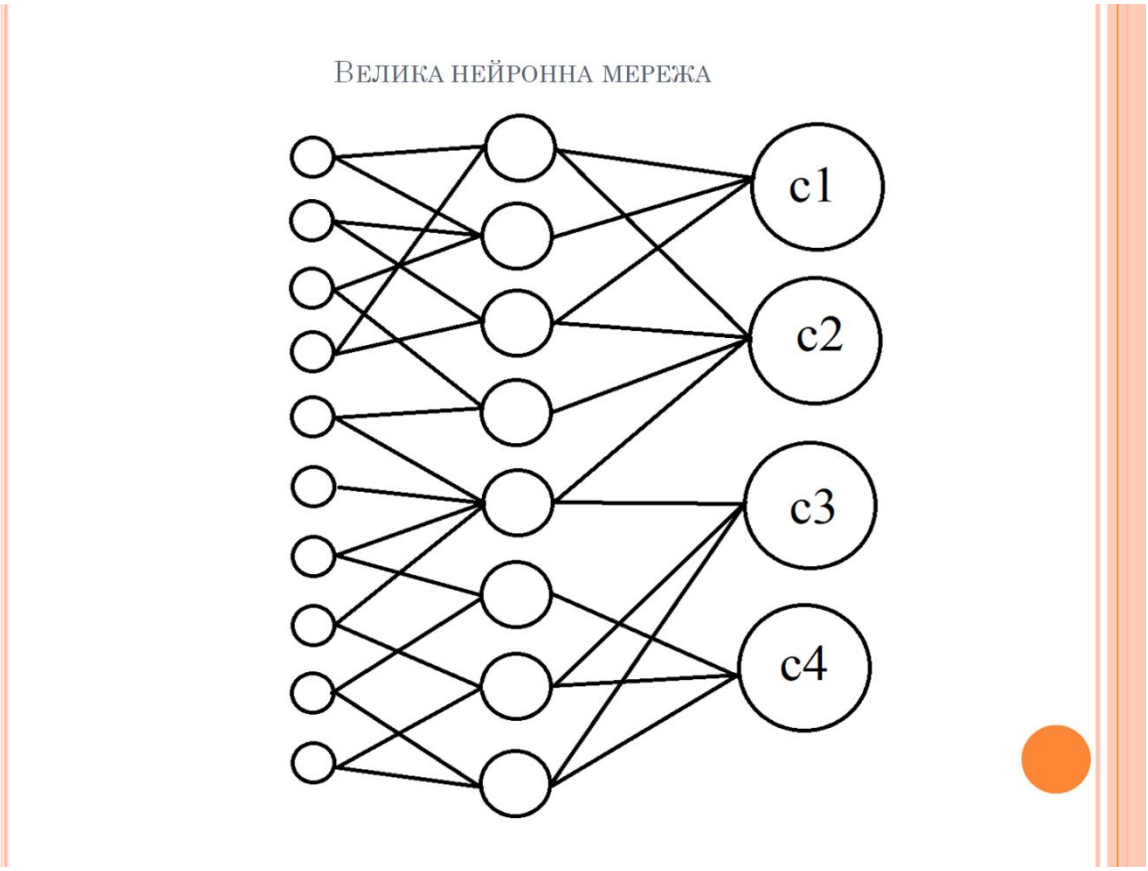
Плакат 3 – Набори класів



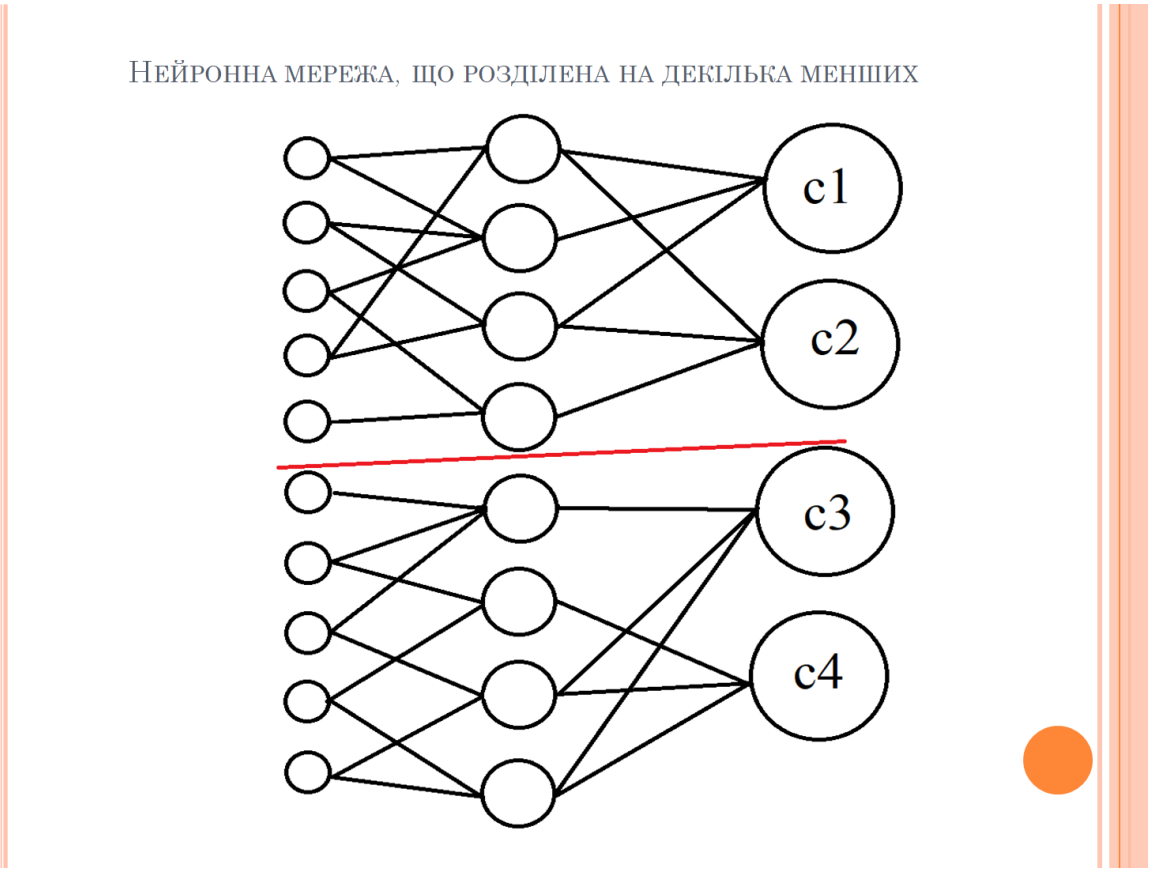
Плакат 4 – Кластер нейронних мереж



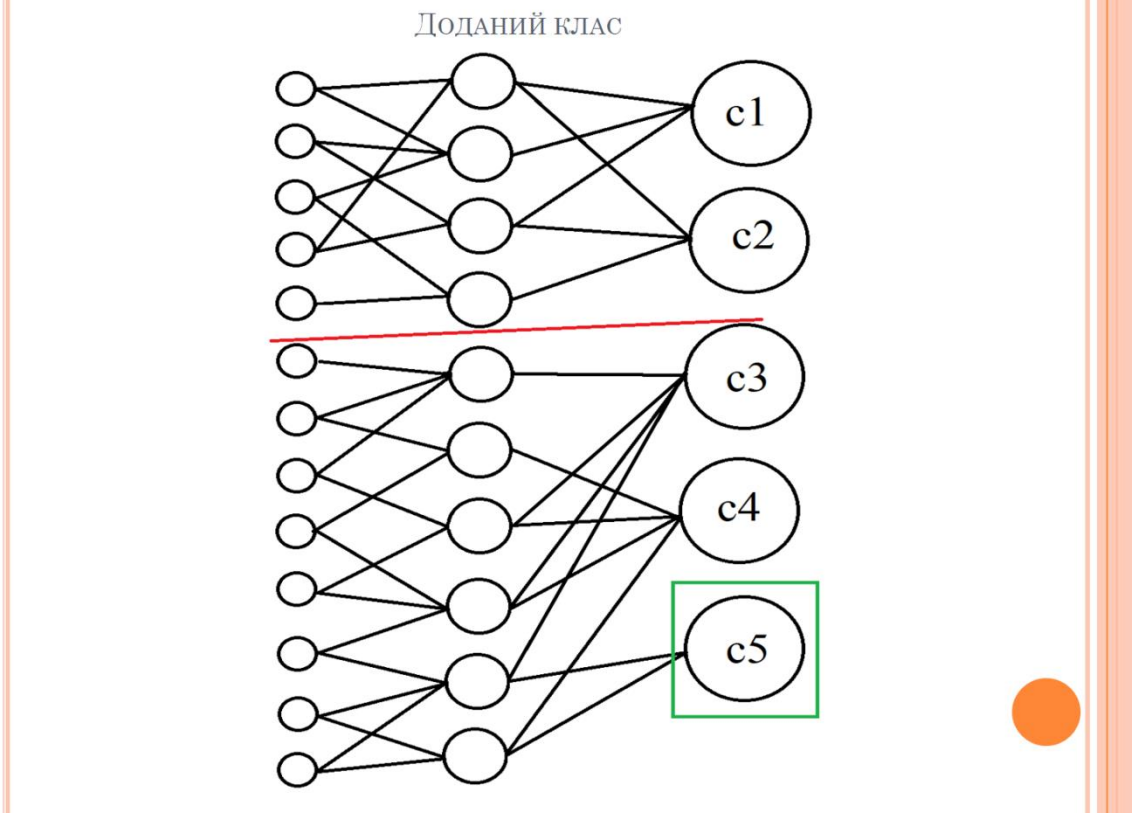
Плакат 5 – Велика нейронна мережа



Плакат 6 – Нейронна мережа, що розділена на декілька менших



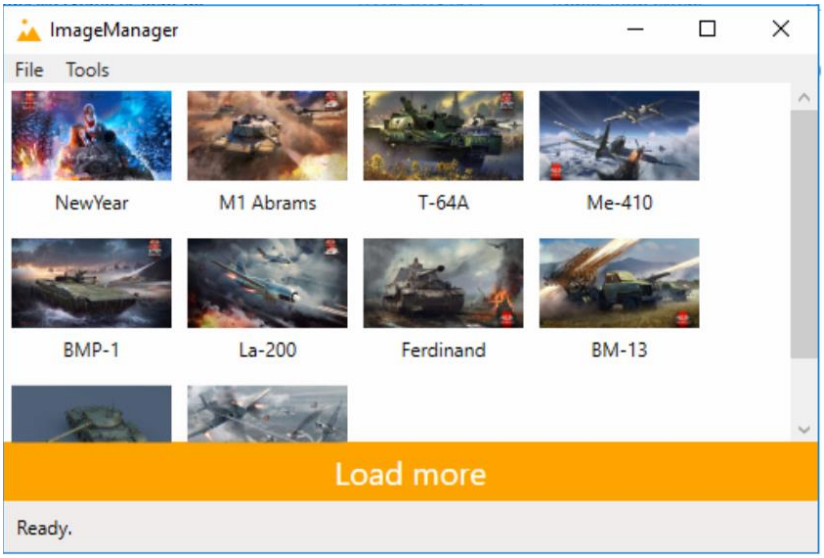
Плакат 7 – Доданий клас



Плакат 8 – Загальна структура системи

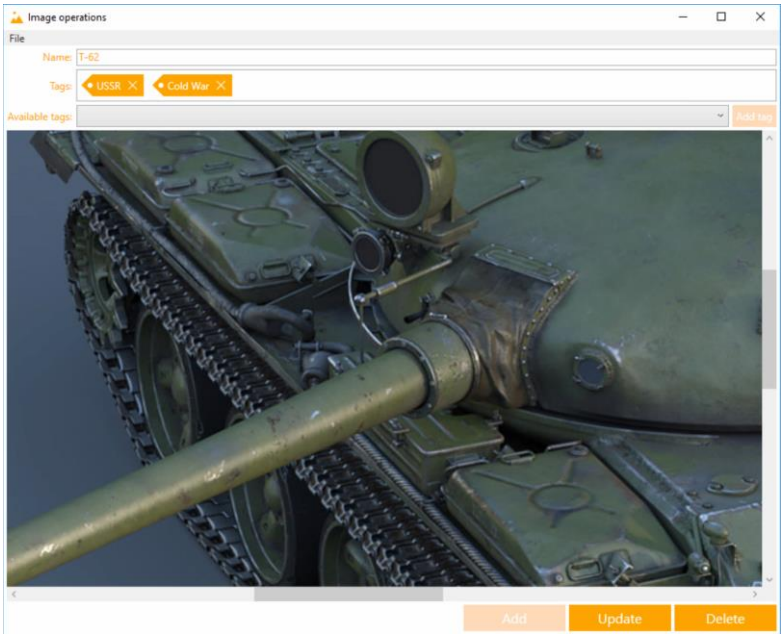
Плакат 9 – Головне вікно додатку

Головне вікно додатку

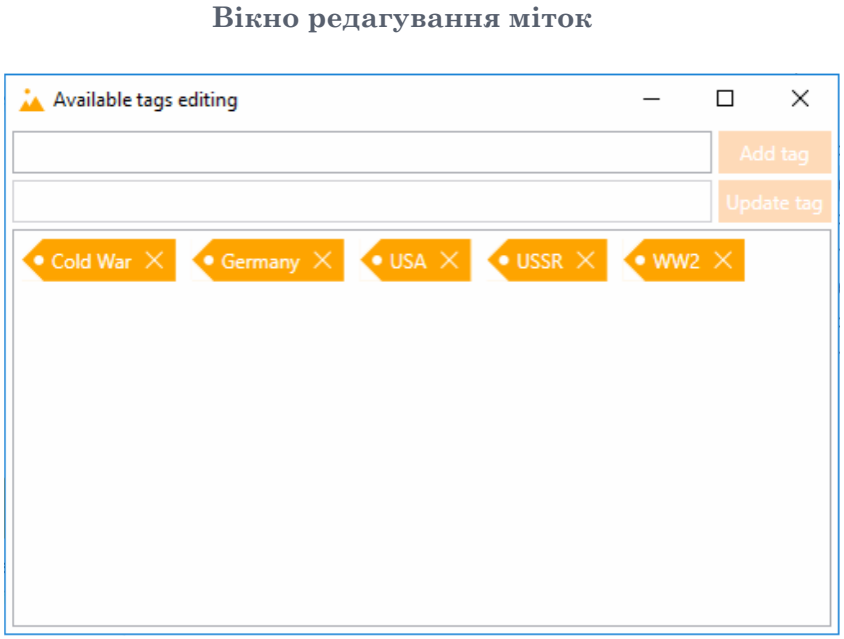


Плакат 10 – Вікно дій над зображенням

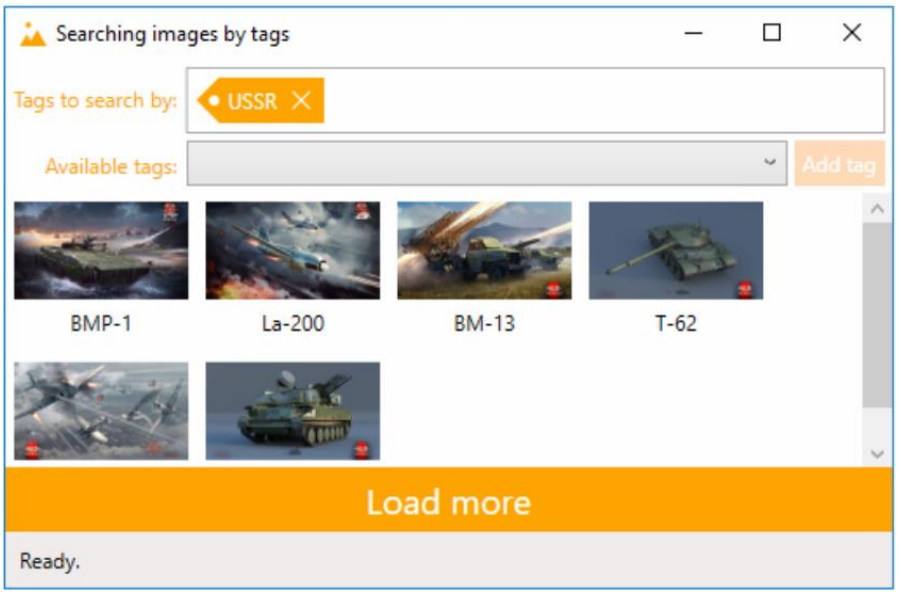
Вікно дій над зображенням



Плакат 11 – Вікно редагування міток



Плакат 12 – Вікно пошуку зображень за мітками



Плакат 13 – Економічне обґрунтування

Економічне обґрунтування

Прибуток за перший рік становитиме 60250 грн., за другий – 95875 грн., за третій – 124375 грн..

Абсолютна ефективність становить 206775 грн..

Відносна ефективність становить 94%.

Термін окупності становить 1,06 року.

Плакат 14 – Дякую за увагу

Дякую за увагу!

ДОДАТОК Ґ ЛІСТИНГ ВИХІДНОГО КОДУ СИСТЕМИ

Лістинг файлу ImagesService.cs

```
using System;
using System.Collections.Generic;
using System.Collections.Concurrent;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using ImagesWcfService.Utilities;
using System.Data.Entity.Infrastructure;

namespace ImagesWcfService
{
    public class ImagesService : IImagesService
    {
        private static Dictionary<string, IImagesServiceCallback> _clients = new
Dictionary<string, IImagesServiceCallback>();
        private static object _clientsSyncObject = new object();

        private int _numberOfSentThumbnails;

        private Tag[] _previousTagsToSearchBy = new Tag[] { };
        private int _numberOfSentThumbnailsWithSpecifiedTags;

        public void Subscribe()
        {
            lock (_clientsSyncObject)
```

```
            {
                _clients[OperationContext.Current.SessionId] =
OperationContext.Current.GetCallbackChannel<IImagesServiceCallback>();
            }
        }

        public Image[] GetNextThumbnails(int numberOfThumbnails, int
widthOfThumbnail, bool resetToBeginning)
        {
            using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
            {
                if (resetToBeginning)
                {
                    _numberOfSentThumbnails = 0;
                }

                List<ImagesDal.Image> imagesFromDatabase =
imagesContext.Images.OrderBy(image =>
image.Id).Skip(_numberOfSentThumbnails).Take(numberOfThumbnails).ToList();
                _numberOfSentThumbnails += imagesFromDatabase.Count;

                return
DatabaseToServiceConversionUtility.CreateThumnailsToSendToClient(imagesFrom
Database, widthOfThumbnail);
            }
        }

        public Image[] GetNextThumbnailsWithSuchTags(int
numberOfThumbnails, int widthOfThumbnail, Tag[] tags, bool resetToBeginning)
```

```
        {
            using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
            {
                if (resetToBeginning)
                {
                    _numberOfSentThumbnailsWithSpecifiedTags = 0;
                }

                if (!Utility.TagArraysAreEqual(_previousTagsToSearchBy, tags))
                {
                    _previousTagsToSearchBy = new Tag[tags.Length];
                    for (int i = 0; i < tags.Length; i++)
                    {
                        _previousTagsToSearchBy[i] = tags[i];
                    }

                    _numberOfSentThumbnailsWithSpecifiedTags = 0;
                }

                int[] ids = new int[tags.Length];
                for (int i = 0; i < tags.Length; i++)
                {
                    ids[i] = tags[i].Id;
                }

                IQueryable<ImagesDal.Tag>          tagsToSearchBy          =
imagesContext.Tags.Where(tagToSearchBy => ids.Contains(tagToSearchBy.Id));
```

```
                IQueryable<ImagesDal.Image> queryResult =
imagesContext.Images.Where(image =>
image.Tags.Intersect(tagsToSearchBy).Count() == tagsToSearchBy.Count());

                List<ImagesDal.Image> imagesFromDatabase =
queryResult.OrderBy(image =>
image.Id).Skip(_numberOfSentThumbnailsWithSpecifiedTags).Take(numberOfThu
mbnails).ToList();
                _numberOfSentThumbnailsWithSpecifiedTags +=
imagesFromDatabase.Count;

                return
DatabaseToServiceConversionUtility.CreateThumnailsToSendToClient(imagesFrom
Database, widthOfThumbnail);
            }
        }

        public Image GetThumbnail(int widthOfThumbnail, int id)
        {
            using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
            {
                ImagesDal.Image imageFromDatabase =
imagesContext.Images.Find(id);
                if (imageFromDatabase == null)
                {
                    return null;
                }
                else
                {
```

```
                return
DatabaseToServiceConversionUtility.CreateThumbnailToSendToClient(imageFrom
Database, widthOfThumbnail);
            }
        }
    }

    public Image GetFullSizeImage(int id)
    {
        using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
        {
            ImagesDal.Image imageFromDatabase =
imagesContext.Images.Find(id);
            if (imageFromDatabase == null)
            {
                return null;
            }
            else
            {
                return new Image()
                {
                    Id = imageFromDatabase.Id,
                    ImageName = imageFromDatabase.ImageName,
                    ImageContent = imageFromDatabase.ImageContent,
                    Tags =
DatabaseToServiceConversionUtility.CreateTagsToSendToClient(new
List<ImagesDal.Tag>(imageFromDatabase.Tags))
                };
            }
```

```
        }
    }


    public Tag[] GetAllTags()
    {
        using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
        {
            return
DatabaseToServiceConversionUtility.CreateTagsToSendToClient(imagesContext.Ta
gs.ToList());
        }
    }

    public Tag GetTag(int id)
    {
        using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
        {
            ImagesDal.Tag tagFromDatabase = imagesContext.Tags.Find(id);
            if (tagFromDatabase == null)
            {
                return null;
            }
            else
            {
                return
DatabaseToServiceConversionUtility.CreateTagToSendToClient(tagFromDatabase);
            }
```

```
        }
    }

    public void AddImage(Image image)
    {
        using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
        {
            ImagesDal.Image imageToAdd = new ImagesDal.Image()
            {
                ImageName = image.ImageName,
                ImageContent = image.ImageContent,
            };
            foreach (Tag tag in image.Tags)
            {
                ImagesDal.Tag tagToAddToImage =
imagesContext.Tags.Find(tag.Id);
                if (tagToAddToImage != null)
                {
                    imageToAdd.Tags.Add(tagToAddToImage);
                }
            }

            imagesContext.Images.Add(imageToAdd);
            if (SaveChanges(imagesContext) != 0)
            {
                NotifyOtherClientsAboutDatabaseUpdate(new
EntityChangeInfo(imageToAdd.Id, EntityType.Image, EntityState.Added));
            }
        }
```

```
        }

        public void UpdateImage(Image image)
        {
            using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
            {
                ImagesDal.Image imageToUpdate =
imagesContext.Images.Find(image.Id);

                if (imageToUpdate != null)
                {
                    imageToUpdate.ImageName = image.ImageName;
                    imageToUpdate.ImageContent = image.ImageContent;
                    imageToUpdate.Tags.Clear();
                    foreach (Tag tag in image.Tags)
                    {
                        ImagesDal.Tag tagToAddToImage =
imagesContext.Tags.Find(tag.Id);
                        if (tagToAddToImage != null)
                        {
                            imageToUpdate.Tags.Add(tagToAddToImage);
                        }
                    }

                    if (SaveChanges(imagesContext) != 0)
                    {
                        NotifyOtherClientsAboutDatabaseUpdate(new
EntityChangeInfo(imageToUpdate.Id, EntityType.Image, EntityState.Modified));
                    }
```

```
            }
        }
    }

    public void DeleteImage(int id)
    {
        using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
        {
            imagesContext.Entry(new ImagesDal.Image() { Id = id }).State =
System.Data.Entity.EntityState.Deleted;
            if (SaveChanges(imagesContext) != 0)
            {
                NotifyOtherClientsAboutDatabaseUpdate(new
EntityChangeInfo(id, EntityType.Image, EntityState.Deleted));
            }
        }
    }

    public void AddTag(Tag tag)
    {
        using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
        {
            ImagesDal.Tag tagToAdd = new ImagesDal.Tag()
            {
                TagName = tag.TagName
            };

            imagesContext.Tags.Add(tagToAdd);
```

```
            if (SaveChanges(imagesContext) != 0)
            {
                NotifyOtherClientsAboutDatabaseUpdate(new
EntityChangeInfo(tagToAdd.Id, EntityType.Tag, EntityState.Added));
            }
        }
    }

    public void UpdateTag(Tag tag)
    {
        using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
        {
            ImagesDal.Tag tagToUpdate = imagesContext.Tags.Find(tag.Id);

            if (tagToUpdate != null)
            {
                tagToUpdate.TagName = tag.TagName;

                imagesContext.Entry(tagToUpdate).State =
System.Data.Entity.EntityState.Modified;
                if (SaveChanges(imagesContext) != 0)
                {
                    NotifyOtherClientsAboutDatabaseUpdate(new
EntityChangeInfo(tagToUpdate.Id, EntityType.Tag, EntityState.Modified));
                }
            }
        }
    }
```

```
        public void DeleteTag(int id)
        {
            using (ImagesDal.ImagesContext imagesContext = new
ImagesDal.ImagesContext())
            {
                imagesContext.Entry(new ImagesDal.Tag() { Id = id }).State =
System.Data.Entity.EntityState.Deleted;
                if (SaveChanges(imagesContext) != 0)
                {
                    NotifyOtherClientsAboutDatabaseUpdate(new
EntityChangeInfo(id, EntityType.Tag, EntityState.Deleted));
                }
            }
        }

        private int SaveChanges(ImagesDal.ImagesContext imagesContext)
        {
            try
            {
                return imagesContext.SaveChanges();
            }
            catch (DbUpdateConcurrencyException)
            {
                return 0;
            }
            catch (Exception)
            {
                throw;
            }
        }
```

```
        public void Unsubscribe()
        {
            lock (_clientsSyncObject)
            {
                _clients.Remove(OperationContext.Current.SessionId);
            }
        }

        private void NotifyOtherClientsAboutDatabaseUpdate(EntityChangeInfo
entityChangeInfo)
        {
            lock (_clientsSyncObject)
            {
                List<string> clientsToRemove = new List<string>();

                foreach (var client in _clients)
                {
                    if (((ICommunicationObject) client.Value).State ==
CommunicationState.Opened)
                    {
                        try
                        {
                            client.Value.NotifyAboutDatabaseUpdate(entityChangeInfo);
                        }
                        catch (Exception)
                        {
                            clientsToRemove.Add(client.Key);
                        }
                    }
```

```
                else
                {
                    clientsToRemove.Add(client.Key);
                }
            }

            foreach (string client in clientsToRemove)
            {
                _clients.Remove(client);
            }
        }
    }
}
```

Лістинг файлу ServiceClient.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using ImagesWcfServiceClient.Utilities;
using ImagesWcfServiceClient.Models;

namespace ImagesWcfServiceClient
{
    public class ServiceClient : IDisposable
    {
```

```
        private const int NUMBER_OF_THUMBNAILS = 10;
        private const int WIDTH_OF_THUMBNAIL = 100;

        private ImagesWcfServiceReference.ImagesServiceClient _client;

        private bool _receivedAllAvailableThumbnails = false;
        private bool _receivedAllAvailableThumbnailsWithSuchTags = false;
        private List<Tag> _previousTagsToSearchBy = new List<Tag>();

        public ServiceClient(IDatabaseUpdateListener listener)
        {
            _client = new ImagesWcfServiceReference.ImagesServiceClient(new
InstanceContext(new
ImagesWcfServiceReference.ImagesServiceCallback(listener)));
            _client.Open();
            _client.Subscribe();
        }

        public List<Image> GetNextThumbnails(bool resetToBeginning)
        {
            if (resetToBeginning)
            {
                _receivedAllAvailableThumbnails = false;
            }

            if (!_receivedAllAvailableThumbnails)
            {
                List<Image>                thumbnails                =
ServiceToClientConversionUtility.CreateImages(_client.GetNextThumbnails(NUMB
ER_OF_THUMBNAILS, WIDTH_OF_THUMBNAIL, resetToBeginning));
```

```
            if (thumbnails.Count == 0)
            {
                _receivedAllAvailableThumbnails = true;
            }

            return thumbnails;
        }
        else
        {
            return new List<Image>();
        }
    }

    public List<Image> GetNextThumbnailsWithSuchTags(List<Tag> tags,
bool resetToBeginning)
    {
        if (resetToBeginning)
        {
            _receivedAllAvailableThumbnailsWithSuchTags = false;
        }

        if
(!Utility.CheckWhetherTagCollectionsAreEqual(_previousTagsToSearchBy, tags))
        {
            _previousTagsToSearchBy.Clear();
            _previousTagsToSearchBy.AddRange(tags);

            _receivedAllAvailableThumbnailsWithSuchTags = false;
        }
```

```
            if (tags != null && tags.Count > 0)
            {
                if (!_receivedAllAvailableThumbnailsWithSuchTags)
                {
                    List<Image> thumbnails = ServiceToClientConversionUtility.CreateImages(_client.GetNextThumbnailsWithSuchTags(NUMBER_OF_THUMBNAILS, WIDTH_OF_THUMBNAIL, ClientToServiceConversionUtility.CreateTagsToSendToService(tags), resetToBeginning));

                    if (thumbnails.Count == 0)
                    {
                        _receivedAllAvailableThumbnailsWithSuchTags = true;
                    }

                    return thumbnails;
                }
                else
                {
                    return new List<Image>();
                }
            }
            else
            {
                throw new ArgumentException("List of tags must not be null and must not be empty.");
            }
        }
```

```
        public Image GetThumbnail(int id)
        {
            ImagesWcfServiceReference.Image thumbnailFromService =
_client.GetThumbnail(WIDTH_OF_THUMBNAIL, id);
            if (thumbnailFromService == null)
            {
                return null;
            }
            else
            {
                return
ServiceToClientConversionUtility.CreateImage(thumbnailFromService);
            }
        }

        public Image GetFullSizeImage(int id)
        {
            ImagesWcfServiceReference.Image imageFromService =
_client.GetFullSizeImage(id);
            if (imageFromService == null)
            {
                return null;
            }
            else
            {
                return
ServiceToClientConversionUtility.CreateImage(imageFromService);
            }
        }
```

```
        public List<Tag> GetAllTags()
        {
            return
ServiceToClientConversionUtility.CreateTags(_client.GetAllTags());
        }

        public Tag GetTag(int id)
        {
            ImagesWcfServiceReference.Tag tagFromService = _client.GetTag(id);
            if (tagFromService == null)
            {
                return null;
            }
            else
            {
                return
ServiceToClientConversionUtility.CreateTag(tagFromService);
            }
        }

        public void AddImage(Image image)
        {

_client.AddImage(ClientToServiceConversionUtility.CreateImageToSendToService(
image));
        }

        public void UpdateImage(Image image)
        {
```

```
_client.UpdateImage(ClientToServiceConversionUtility.CreateImageToSendToServi
ce(image));
            }

            public void DeleteImage(Image image)
            {
               _client.DeleteImage(image.Id);
            }

            public void AddTag(Tag tag)
            {

_client.AddTag(ClientToServiceConversionUtility.CreateTagToSendToService(tag))
;
            }

            public void UpdateTag(Tag tag)
            {

_client.UpdateTag(ClientToServiceConversionUtility.CreateTagToSendToService(ta
g));
            }

            public void DeleteTag(Tag tag)
            {
               _client.DeleteTag(tag.Id);
            }

            private bool _disposed = false;
```

```
        public void Dispose()
        {
            CleanUp(true);
            GC.SuppressFinalize(this);
        }

        private void CleanUp(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    _client.Unsubscribe();
                    _client.Close();
                }
            }
            _disposed = true;
        }

        ~ServiceClient()
        {
            CleanUp(false);
        }
    }
}
```

Лістинг файлу App.xaml

```
<Application x:Class="ImageManagerWpfClient.App"
```

```
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:ImageManagerWpfClient"
        StartupUri="Views\MainWindow.xaml" Exit="Application_Exit">
    <Application.Resources>

        <Style x:Key="ActionButtonStyle" TargetType="Button">

            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
                        <Label                    Background="{TemplateBinding
Property=Background}"
                               Foreground="{TemplateBinding Property=Foreground}"
                               HorizontalContentAlignment="Center"
                               Content="{TemplateBinding Property=Content}"/>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>

            <Setter Property="Background" Value="Orange"/>
            <Setter Property="Foreground" Value="White"/>

            <Style.Triggers>

                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Background" Value="DarkOrange"/>
                </Trigger>

                <Trigger Property="IsPressed" Value="True">
```

```
                <Setter Property="Background" Value="OrangeRed"/>
            </Trigger>

            <Trigger Property="IsEnabled" Value="False">
                <Setter Property="Background" Value="PeachPuff"/>
            </Trigger>

        </Style.Triggers>
    </Style>

    <Style x:Key="LoadMoreButtonStyle" TargetType="Button"
BasedOn="{StaticResource ActionButtonStyle}">
        <Setter Property="FontSize" Value="20"/>
        <Setter Property="Content" Value="Load more"/>
    </Style>

    <Style x:Key="ThumbnailsItemsControlStyle"
TargetType="ItemsControl">

        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate>
                    <ScrollViewer HorizontalScrollBarVisibility="Disabled"
VerticalScrollBarVisibility="Auto">
                        <ItemsPresenter/>
                    </ScrollViewer>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
```

```
                <Setter Property="ItemsPanel">
                    <Setter.Value>
                        <ItemsPanelTemplate>
                            <WrapPanel/>
                        </ItemsPanelTemplate>
                    </Setter.Value>
                </Setter>

                <Setter Property="ItemContainerStyle">
                    <Setter.Value>
                        <Style>
                            <Setter Property="Control.Margin" Value="5"/>
                        </Style>
                    </Setter.Value>
                </Setter>

                <Setter Property="ItemTemplate">
                    <Setter.Value>
                        <DataTemplate>
                            <Button ToolTip="{Binding Path=ImageName}"
                                    CommandParameter="{Binding}"
                                    Command="{Binding
Path=DataContext.OpenFullSizeImageCommand,
                                                RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType=Window}}">
                                <Button.Template>
                                    <ControlTemplate>
                                        <StackPanel>
                                            <Image Width="100" Source="{Binding
Path=ImageContent}"/>
```

```
                        <Label                                  Width="100"
HorizontalContentAlignment="Center" Content="{Binding Path=ImageName}"/>
                    </StackPanel>
                </ControlTemplate>
            </Button.Template>
        </Button>
    </DataTemplate>
</Setter.Value>
</Setter>

</Style>

<Style x:Key="LabelStyle" TargetType="Label">
    <Setter Property="Foreground" Value="Orange"/>
    <Setter Property="HorizontalContentAlignment" Value="Right"/>
    <Setter Property="VerticalContentAlignment" Value="Center"/>
</Style>

<Style x:Key="TextBoxStyle" TargetType="TextBox">
    <Setter Property="Foreground" Value="Orange"/>
</Style>

<Style  x:Key="ErrorTemplateForTextBoxStyle"  TargetType="TextBox"
BasedOn="{StaticResource TextBoxStyle}">
    <Style.Triggers>
        <Trigger Property="Validation.HasError" Value="True">
            <Setter Property="ToolTip"
                Value="{Binding                    RelativeSource={x:Static
RelativeSource.Self},
                Path=(Validation.Errors)[0].ErrorContent}"/>
```

```
            </Trigger>
          </Style.Triggers>
        </Style>

    </Application.Resources>
</Application>
```