БАКАЛАВРСЬКА РОБОТА

БР.КІ–14.00.00.000 ПЗ

Група КІ-21-1

Паламаренко Назар

2025

Івано-Франківський Національний Технічний Університет Нафти і Газу
Факультет Інформаційних Технологій
Кафедра комп'ютерних систем і мереж

Паламаренко Назар Романович

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Розробка web-додатку редагування зображень на основі декомпозиції об'єктів засобами нейронної мережі Yolo8

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Паламаренко Н. Р.
(підпис, прізвище та ініціали здобувача)

Науковий керівник Гарасимів Т.Г., *асистент*
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри КСМ

д.т.н., професор Мельничук С. І.
(посада) (підпис) (дата) (прізвище та ініціали)

м. Івано-Франківськ – 2025

Івано-Франківський Національний Технічний Університет Нафти і Газу

Факультет Інформаційних Технологій

Кафедра Комп'ютерних систем і мереж

Освітній рівень Бакалавр

Спеціальність 123 Комп’ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри

Мельничук С. І.

" 05 " *травня* 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Паламаренку Назару Романовичу

(прізвище, ім’я, по батькові)

1. Тема роботи Розробка web-додатку редагування зображень на основі декомпозиції об’єктів засобами нейронної мережі Yolo8

керівник роботи Гарасимів Тарас Григорович, асистент

затверджені наказом закладу вищої освіти від "05" травня 2025 року №275/7

2. Термін подання студентом роботи 12 червня 2025 року

3. Вихідні дані до роботи Методичні вказівки, технічна література

4. Зміст пояснювальної записки: 1 Огляд аналогів, опис вхідних даних, відображення даних та кінцева постановка задачі; 2 Розробка структури бази даних, розробка UML діаграм; 3 Розробка функціоналу веб-додатку;

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання – 29 січня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Пошук інформативних аналогів, теоретична підготовка до виконання бакалаврської роботи	06.03.2025 – 15.03.2025	виконав
2	Написання коду	16.03.2025 – 15.04.2025	виконав
3	Розробка діаграм UML	05.04.2025 – 15.04.2025	виконав
4	Оформлення пояснювальної записки	11.03.2025 – 31.05.2025	виконав

Студент _____ Паламаренко Н. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гарасимів Т.Г.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська робота присвячена розробці web-додатку для реалізації редагування зображення на основі декомпозиції об’єктів засобами нейронної мережі Yolo8.

У першій частині роботи наведено огляд існуючих рішень, описано вхідні дані, способи їхнього представлення та сформульовано постановку задачі.

У другій частині описано проектування таблиць, розробку структури бази даних та UML-діаграм класів і послідовностей.

У третій частині подано настанову користувача, описано реалізацію веб-додатку.

Для виконання завдання використано мову програмування Python з фреймворком FastAPI для серверної частини, бібліотеку React для клієнтської та реляційну СУБД MariaDB, розробка проводилася в середовищі Visual Studio Code.

Ключові слова: ВЕБ-ДОДАТОК, НЕЙРОННА МЕРЕЖА, YOLO8, ДЕКОМПОЗИЦІЯ ОБ’ЄКТІВ, БАЗА ДАНИХ, СЕРВЕР, КЛІЄНТ, КОРИСТУВАЧ.

ANNOTATION

The bachelor's thesis is devoted to the development of a web application for implementing image editing based on object decomposition using the Yolo8 neural network.

The first part of the work provides an overview of existing solutions, describes the input data, ways of representing them, and formulates the problem statement.

The second part describes the design of tables, development of the database structure and UML diagrams of classes and sequences.

In the third part, the user's manual is presented, the implementation of the web application is described.

To accomplish the task, the Python programming language with the FastAPI framework for the server side, the React library for the client side, and the MariaDB relational database were used, the development was carried out in the Visual Studio Code environment.

Keywords: WEB APPLICATION, NEURAL NETWORK, YOLO8, OBJECT DECOMPOSITION, DATABASE, SERVER, CLIENT, USER.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Web API – реалізація REST-сервісів через HTTP-ендпоінти на стороні сервера.

API (Application Programming Interface) – програмний інтерфейс для взаємодії різних компонентів або сервісів.

REST – архітектурний стиль взаємодії клієнт-сервер із використанням HTTP-методів та ресурсів.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту, основа комунікації в веб.

CRUD – базові операції з даними: Create (створення), Read (читання), Update (оновлення), Delete (видалення).

БД – База Даних.

СУБД – Система Управління Базами Даних.

DL (Deep learn) – глибоке навчання, підгалузь машинного навчання, що використовує багаторівневі нейронні мережі для автоматичного виділення ознак.

Yolo8 (You Only Look Once) – сучасна модель DL для обробки зображень.

ВСТУП

Актуальність – у сучасному цифровому середовищі стрімко зростає потреба у зручному та зрозумілому редакторі зображень, особливо в умовах швидко зростання та інтегрування в наше життя соціальних мереж. Звичайні інструменти для редагування, наприклад Photoshop, вимагають час та знання для його застосування. Натомість web-додатки на основі комп'ютерного зору та моделей DL, відкривають можливість редагувати зображення без особливих навичок. Найбільш актуальні зараз задачі декомпозиції зображення на окремі об'єкти з можливістю подальшої взаємодії з ними, наприклад видалення або заміна. Для реалізації даного функціоналу використовують сучасні архітектури моделей DL, наприклад Yolo8, яка застосовується для швидкого виявлення об'єктів. Також використовуються інструменти, такі як lama-cleaner, щоб після видалення або заміни об'єкта не змінювався фон зображення.

Для реалізації такого web-додатку, була реалізована REST архітектура для забезпечення гнучкості та масштабованості додатку та фронтендом на основі React, для зберігання даних була реалізована у реляційній СУБД MariaDB.

Об'єкт дослідження – процес автоматичного виявлення та редагування об'єктів на зображення використовуючи методи DL та комп'ютерного зору.

Предмет дослідження – алгоритми DL для знаходження об'єктів, способи декомпозиції об'єктів, інтеграція їх у web-додаток використовуючи архітектуру REST API та React, збереження даних у MariaDB і реалізації інтуїтивно зрозумілого інтерфейсу для роботи з зображеннями.

Мета і завдання роботи – метою бакалаврської роботи є розробка web-додатку редагування зображень на основі декомпозиції об'єктів засобами нейронної мережі Yolo8. Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати вже існуючі рішення для автоматичного редагування зображень;

– ознайомитись з архітектурою та принципами роботи DL моделі Yolo8;

– спроектувати архітектуру REST API для взаємодії з обробки зображень;

– налагодити збереження даних та взаємодію з реляційною СУБД;

– реалізувати бекенд-сервіс на основі FastAPI, який оброблятиме CRUD-операції та функції завантаження зображення, виявлення об’єктів, редагування та забезпечуватиме автентифікацію й авторизацію користувачів;

– створити фронтенд частину додатку на базі React;

– виконати тестування працездатності, зручності інтерфейсу та навантаження з метою перевірки стійкості системи.

Методи дослідження – у роботі реалізовано поетапний підхід до розробки веб-додатку, що включає аналіз вимог, вивчення аналогічних рішень у сфері редагування зображень, побудову логічної моделі даних із ER-діаграми, створення RESTful API на базі FastAPI, а також проєктування клієнтської частини із застосуванням React для інтерактивної роботи з елементами інтерфейсу. Основу функціональності обробки зображень становить інтеграція нейронної мережі Yolo8 для визначення об’єктів.

Практичне значення – розроблений web-додаток надає користувачам можливість взаємодіяти зі зручним інструментом для інтелектуального редагування зображень на основі автоматичного розпізнавання об'єктів. Використовуючи інтегровану нейронну модель Yolo8, web-сервіс надає виявлення об’єктів на зображенні з мінімальною похибкою, що дозволяє здійснювати подальше редагування об’єктів. REST–архітектура бекенду забезпечує гнучкість між взаємодією клієнтською частиною та сервером. Використання FastAPI та MariaDB гарантує безпеку, а фронтенд на базі React забезпечує зручність користування.

Таким чином, розроблений web-застосунок поєднує в собі сучасні досягнення в галузях AI, web-розробки та UX-дизайну, відповідає вимогам до гнучких і зручних цифрових рішень, і може використовуватись як у сфері цифрового дизайну, так і як персональний інструмент для швидкого редагування зображень.

1 ОГЛЯД ВЕБ-ДОДАТКІВ АНАЛОГІВ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Пояснення алгоритму роботи нейронних моделей для знаходження об’єктів

У рамках бакалаврської роботи об’єктом дослідження є інформаційна система комп’ютерного зору, яка реалізує автоматичне виявлення, ідентифікацію та подальшу декомпозицію, за допомогою методів глибокого навчання. Відповідно до класифікації ДСТУ ISO/IEC 12207:2018 "Інформаційні технології. Комплекс стандартів на автоматизовані системи" [1], ця система належить до інтелектуальних інформаційно–аналітичних систем, що забезпечують повний цикл обробки інформації – від збору до формування висновків для прийняття рішень. Система реалізує підхід, притаманний системам підтримки прийняття рішень (СППР): вона автоматично вилучає релевантні ознаки з цифрових зображень, формує структуровані дані, придатні для подальшої інтерпретації, та використовує глибокі нейронні мережі для адаптивного аналіз, здатного до навчання на нових даних без необхідності ручного втручання.

У контексті ДСТУ 2226–93 [2] ця система також класифікується як автоматизована система обробки зображень із спеціальними функціями розпізнавання об’єктів. За класифікацією типів прийняття рішень, що ґрунтується на рівні структурованості задач, система працює з слабко структурованими задачами. Вона не потребує чіткого формального опису правил або рішень і генерує результат на основі виявлення закономірностей у даних. Таким чином, система має евристичний характер прийняття рішень, що є типовим для інтелектуальних інформаційних систем на базі машинного навчання. Рішення формуються на основі великої вибірки навчальних даних без жорстко заданого алгоритму, а детекція об’єктів здійснюється із застосуванням ймовірного аналізу (confidence score) та процедури Non–Maximum Suppression

(NMS) [3], що дозволяє фільтрувати надмірно схожі передбачення. Завдяки здатності до генералізації – адаптації до нових даних – система демонструє ознаки статичного прийняття рішень або нечіткої логіки.

Алгоритми функціонування інформаційної системи охоплює такі ключові етапи:

1. Обробка вхідного зображення

Вхідне зображення нормалізується, наприклад масштабується та перетворюється у тензор, придатний до подачі на вхід згорткової нейронної (CNN).[4,5]

2. Етап вилучення інформаційних ознак [4,5,6,7]. Зображення проходить через згорткові шари, які поступово виділяють:

– низькорівневі ознаки (краї, контури, текстури);

– ознаки середнього рівня (частини об’єктів);

– високорівневі ознаки (цілі об’єкти).

3. Локалізація об’єктів [4,5]. Після виявлення ознак мережа виконує локалізацію, тобто визначає координати об’єктів. Існує два основні підходи:

– region Proposal: генерується область, де може знаходитись об’єкт;

– grid-based detection: зображення поділяється на сітку, де кожна клітинка відповідає за передбачення.

4. Прогнозування класів та координат. Модель на кожному область видає:

– координати рамок у вигляді: x, y, width, height;

– перевірка приналежності до певного класу;

Даний етап відповідає за виведення результатів з нейронної мережі.

5. Обробка результатів. Оскільки модель для одного об’єкту може передбачати декілька рамок застосовується:

– non-Maximum Suppression (NMS) – відсікання дублікатів при детекції;

– confidence thresholding – видаляє рамки для об'єктів з низькою впевненістю [3].

6. Декомпозиція об’єктів: детекція та сегментація Bounding box є швидким способом виділення об’єкта. Для точної декомпозиції застосовується сегментація:

– semantic segmentation – всі об’єкти класу мають одну маску;

– instance segmentation – кожен об’єкт має окрему маску [5,7].

На рисунку 1.1 чітко видно різницю між детекцією (bounding boxes) та сегментацією (точні обриси).

Рисунок 1.1 – Приклад реалізації класифікації, локалізації, детекції та сегментації [8]

7. Автоматичне вилучення ознак об’єктів для декомпозиції

На відміну від класичних методів, де ознаки формуються вручну, у моделях глибокого навчання (DL) ознаки об’єктів витягуються автоматично [3,4]. Це дає можливість моделі самостійно навчитись відрізняти об’єкти, розпізнавати контури, тощо. Різницю між цими підходами зображено на рисунку. 1.2 та рис. 1.3.

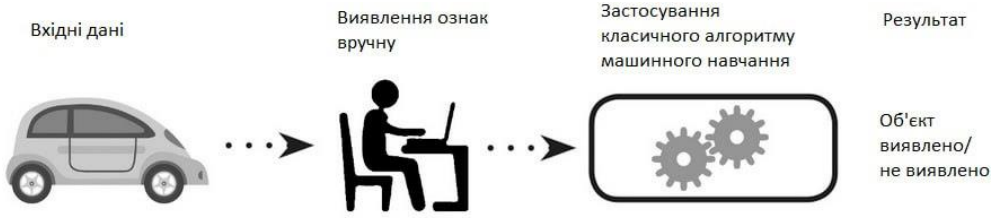

Рисунок 1.3 – Принцип роботи класичних алгоритмів детекції об’єктів [9]

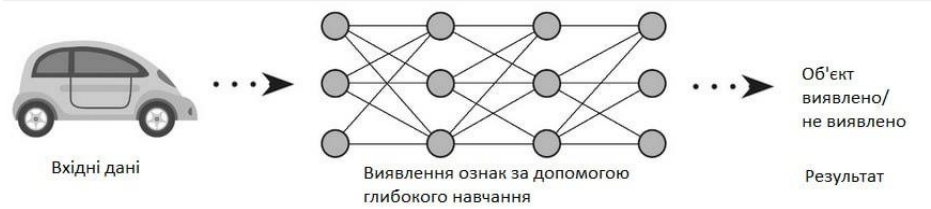

Рисунок 1.2 – Принцип роботи методів детекції об’єктів на основі глибокого навчання [9]

8. Навчання нейронної моделі. Навчання здійснюється на основі функцій втрати, які комбінують декілька функцій втрат, також застосовують алгоритми оптимізації:

– втрата координат: L1/L2 loss;

– втрата класифікації: Cross-Entropy [3,4].

Найпопулярніші алгоритми оптимізації для задач знаходження об’єктів – SGD або Adam на великих датасетах, щоб забезпечити точність системи [10]. Під час розробки були враховані нормативні документи, які регламентують якість програмного забезпечення, його функціональність, надійність, і життєвий цикл (зокрема, ДСТУ ISO/IEC 25010:2016 [11]). Крім того, були використані релзультати сучасних досліджень у сфері комп’ютерного зору й глибокого навчання [3–10, 12–13], а також документація до практичних інструментів, таких як ultralytics, FastAPI та SQLAlchemy [14–16]. Розроблена інформаційна система не лише відповідає сучасним вимогам до інтелектуальних систем підтримки прийняття рішень, а й демонструє повну інтеграцію циклу оборобки інформації – від вхідного зображення до формалізованих ознак об’єктів, які можуть бути вбудовані в інший аналітичні модулі.

1.2 Виділення інформативних атрибутів на основі огляду веб-додатків аналогів

Для того щоб з’ясувати, які атрибути потрібно передбачити у базі даних та в інтерфейсі, розглянемо кілька прикладів реалізації редакторів зображень.

Найбільш найбільш схожий веб-додаток – Cleanup.pictures [17]. На рисунку 1.4 та рисунку 1.5 зображений вигляд сайту та функціонал, який нас цікавить.

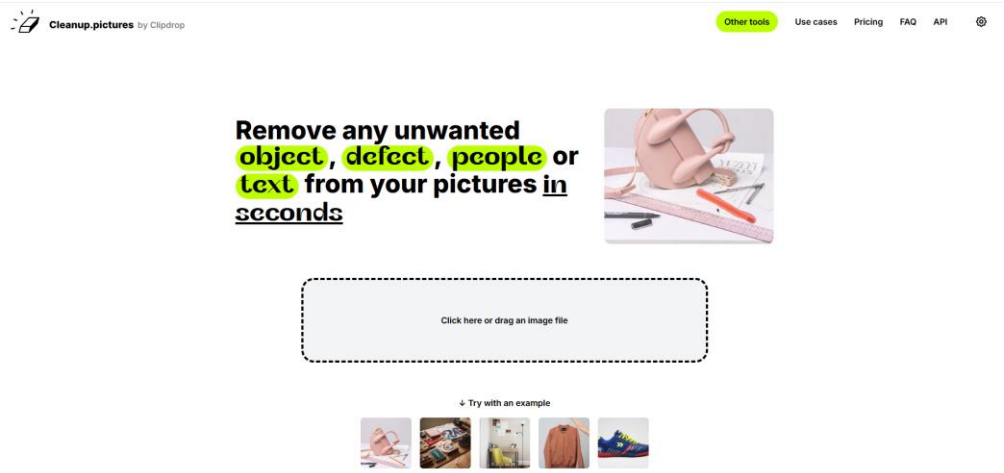

Рисунок 1.4 – Зовнішній вигляд сайту

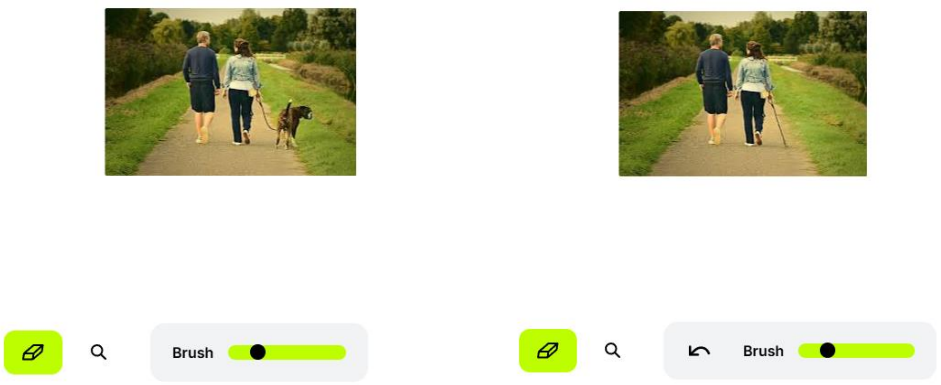

Рисунок 1.5 – Приклад функціоналу видалення

Функціонал даного веб-додатку надає можливість редагування об’єктів шляхом видалення їх з зображення, хоча функція видалення, яка теж базується на інструменті lama-cleaner працює на цьому веб-додатку зовсім по-іншому, даючи можливість власноруч виділяти область для видалення. В цьому підході є, як і плюси так і мінуси, наприклад видалення працює значно швидше, тому що, немає декомпозиції об’єктів, відповідно витрачається менше ресурсів сервера, але власноручне виділення області, може бути і мінусом, якщо на зображенні багато об’єктів, є шанс видалити зайве. Також цей веб-додаток не надає функції заміни об’єктів, але надає інші функції, наприклад видалення тексту з зображень або генерація зображень по тексту, проте ці функції мають обмеження в безкоштовному варіанті.

Інший приклад – Pixlr [18], який є аналогом Photoshop, але як веб-додаток. На рисунку 1.6 та рисунку 1.7 зображений вигляд сайту та функціонал.

Рисунок 1.6 – Зовнішній вигляд сайту

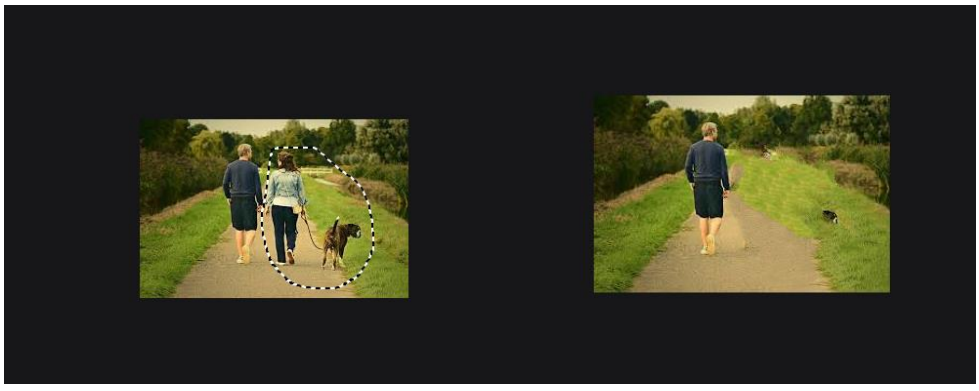

Рисунок 1.7 – Приклад функціоналу видалення

Цей веб-додаток надає мінімальний функціонал Photoshop, але з інтегрованим нейронними мережами. В цьому додатку видалення об’єкта працює гірше, ніж на попередньому додатку, але він має багато інструментів, які базуються на нейромережах, крім видалення об’єктів, наприклад заміна лиць або видалення фону. Проте, для повного користування функціоналом сайту, потрібно придбати підписку, з безкоштовних функцій доступні лише базові функції Photoshop та видалення об’єктів, з ручним виділенням області.

У результаті проведеного аналізу можна зробити висновок, що обидва розглянуті аналоги, реалізують функції для редагування зображень, використовуючи як основу нейромережі та схожі інструменти, проте в них є певні недоліки та обмеження. Перший веб-додаток робить акцент на швидкості редагування за рахунок ручного виділення області, але при такому підході є ризик видалити або пошкодити інший об’єкт. Другий веб-додаток надає ширший набір інструментів та можливостей, включаючи заміну облич або видалення фону, однак якісь видалення об’єктів значно гірша ніж на першому аналогу. Крім того, ці два веб-додатка мають обмеження в безкоштовних версіях, також функціонал облікових записів вкрай обмежена і використовується тільки для покупки платних версій. Це вказує на актуальність розробки гнучкішого рішення використовуючи автоматизоване виділення і взаємодію з об’єктами.

1.3 Вхідні дані для опрацювання

Виходячи з аналізу принципів робити алгоритмів нейронних мереж та підходів до декомпозиції зображень в розділs 1.1 та функціональних особливостей аналогічних веб-додатків в розділі 1.2, можна визначити ключові типи вхідних даних, з якими буде працювати цей web-додаток. Цей додаток орієнтований на автоматичне виявлення об’єктів та інтерактивну взаємодію з ними, для редагування.

Основні типи вхідних даних

1. Зображення

– Формат: .jpeg, .jpg, .png;

– Джерело: локальне завантаження через інтерфейс;

– Приклад: test_image.jpeg.

Після завантаження зображення миттєво передається нейронній мережі для обробки.

2. Результати детекції

– Bounding boxes: виділяє об’єкти прямокутниками по координатах;

– Клас: назви об’єктів всередині bounding boxes (наприклад "dog ", " person");

– Confidence: значення впевненості в класі та/або об’єкті у процентах.

Результати детекції автоматично виводяться на зображенні у вигляді інтерактивних рамок.

3. Дії користувача

– Видалення об’єкта: при натискання на рамку користувачу на вибір дається можливість видалити об’єкт, просто натиснувши на відповідну кнопку, після цього об’єкт зникає з зображення, та повторно запускає детекцію;

– Заміна об’єкта: при натисканні на рамку користувачу на вибір дається можливість замінити об’єкт, натиснувши на відповідну кнопку та вибрати об’єкт, на який, ви хочете замінити, після цього повторно запускає детекцію.

Приклад вхідних даних при створені користувача:

– Ім’я користувача: Nazar;

– Пошта: nazarpalamarenko1@gmail.com;

– Пароль: 158138158r.

1.4 Постановка задачі

Враховуючи все описане в попередніх підрозділах, структура розроблюваного web-додатку орієнтована на простоту для користувача та ефективність редагування зображень:

Головна сторінка:

– Форма для завантаження зображення, яка дозволяє користувачу завантажити зображення зі свого пристрою;

– Після завантаження автоматично запускає детекцію об’єктів за допомогою DL моделі;

– Результат детекції має бути відображений на зображенні у вигляді інтерактивних bounding boxes;

Інтерактивна робота з об’єктами:

– Користувач має мати можливісьб натиснути на об’єкти всередині bounding boxes;

– Після цього повинно з’являться дві кнопки:

– Видалити – після натискання на кнопку об’єкт видалятиметься зі сцени, зображення повинне оновиться, після цього повинна запуститься повторна детекція;

– Замінити – після натискання на кнопку користувач буде мати можливість обрати об’єкт, який бажає замінити та об’єкт на який він буде заміняти, замінений об’єкт повинен автоматично масштабується до потрібних розмірів, після цього повинна запуститься повторна детекція;

– Після редагування зображення, користувач повинен мати змогу завантажити його, натиснувши на відповідну кнопку.

Авторизація та реєстрація:

– Повинна бути реалізована базова система аутентифікації;

– Реєстрація нового користувача, має мати наступні поля: введенням імені користувача, пошти, пароля;

– Також повинна бути авторизація із перевіркою JWT-токена для безпечного доступу.

Особистий кабінет:

– Створення інтерфейсу для завантаження або зміни аватарки;

– В подальшому передбачити вивід інформації про користувача та вкладку "Історія" де буде виведене інформація про редагування зображень.

2 ОБҐРУНТУВАННЯ ВИБОРУ МОДЕЛІ ТА РОЗРОБКА СТРУКТУРИ БАЗИ ДАНИХ ТА UML ДІАГРАМ

2.1 Обґрунтування вибору моделі

При розробці web-додатку для автоматичного редагування зображень ключовим технічним елементом є вибір нейронної моделі, яка здатна ефективно та з мінімальними вимогами виявляти об’єкти на зображенні. Правильний вибір архітектури безпосередньо впливає на швидкість, точність, ресурсоємність та якість інтеграції у веб-додаток.

Після аналізу основних архітектур моделей для детекції об’єктів, таких як R-CNN, SSD, RetinaNet та попередніх версій YOLO, було прийняте рішення зупинитись на YOLOv8, тому що на даний момент, це найбільш оптимізована модель з серії YOLO, також на цей вибір вплинув фактор простої інтеграції моделі в проект та роботи з іншими інструментами для сегментації зображення.

Порівняємо основні характеристики описаних моделей у таблиці 2.1.

Таблиця 2.1 – Порівняння основних характеристик

Модель	Швидкість	Точність	Підтримка сегментації	Складність інтегрування
Faster R-CNN [4]	низька	висока	відсутня	складна
SSD [19]	висока	середня	відсутня	помірна
RetinaNet [4]	середня	висока	відсутня	складна
YOLOv3 [5]	висока	середня	відсутня	проста
YOLOv8 [5]	висока	висока	так	дуже проста

Тепер варто розписати основні переваги моделі YOLOv8 в межах цього проекту:

1. Універсальність

YOLOv8 підтримує декілька режимів роботи: детекція, сегментація (YOLOv8-Seg) та класифікація, що робить її універсальною для роботи з зображенням.

2. Висока точність

Завдяки anchor-free архітектурі, YOLOv8 досягає високих значень середніх значеннь середньої точності (mAP) – набагато краще ніж попередні версії, наприклад YOLOv3 [5].

3. Швидкість роботи

Модель оптимізована для швидкого інференсу, що дозволяє моделі працювати в режимі реального часу, що є критично важливим для інтерактивного web-додатку.

4. Підтримка сегментації

У режимі YOLOv8-Seg [20] модель повертає піксельну маску об’єкта, що дає можливість точного редагування, на відміну від моделей, які працюють тільки з bounding boxes.

5. Зручність інтеграції

Моделі YOLO, мають відкрите Python API, яка легко підключається через бібліотеку ultralytics [14].

Саме ці фактори вплинули на вибір даної моделі, тому що вона підтримує все необхідне та не вимагає високих обчислювальних ресурсів, відповідно буде менше навантаження на сервер.

2.2 Розробка таблиць бази даних

Потрібно розробити базу даних забезпечивши третю форму нормалізації (3НФ). [21]

У кожної основній таблиці присутній атрибут id, який є ідентифікатором запису в таблиці та виконує роль первинного ключа. У зв’язувальних таблицях присутні ідентифікатори записів з таблиць, які зв’язуються. Ці ідентифікатори утворюють композитний первинний ключ.

Users – таблиця користувачів. Містить інформацію про кожен обліковий запис користувача. Атрибут Id – первинний ключ. Зовнішні ключі відсутні. Таблиця перебуває в третій нормальній формі (3 НФ), оскільки всі її неключові

атрибути функціонально залежать від первинного ключа Id, а між самими неключовими атрибутами немає транзитивних залежностей.

Структуру таблиці Users подано у таблиці 2.2.

Таблиця 2.2 – Структура таблиці Users

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	INT	11	-	Містить ідентифікатор користувача
	username	VARCHAR	50	-	Ім’я користувача
	password_hash	VARCHAR	255	-	Хешований пароль
	created_at	TIMESTAMP	-	-	Дата створення облікового запису
	email	VARCHAR	100	-	Електронна пошта
	is_banned	BOOLEAN	1	☑	Стан блокування
	avatar_url	VARCHAR	100	☑	Шлях до аватарки

Admins – таблиця адміністраторів. Містить облікові записи адміністраторів, які мають доступ до журналу активності. Атрибут Id – первинний ключ. Зовнішні ключі відсутні. Таблиця перебуває в третій нормальній формі (3 НФ), оскільки всі її неключові атрибути функціонально залежать від первинного ключа Id, а між самими неключовими атрибутами немає транзитивних залежностей.

Структуру таблиці Admins подано у таблиці 2.3.

Таблиця 2.3 – Структура таблиці Admins

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	INT	11	-	Містить ідентифікатор адміністратора
	username	VARCHAR	50	-	Ім’я адміністратора

Кінець таблиці 2.3

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
	password_hash	VARCHAR	255	-	Хешований пароль
	created_at	TIMESTAMP	-	-	Дата створення облікового запису
	email	VARCHAR	100	-	Електронна пошта

Images – таблиця зображень. Містить інформацію про зображення, які були завантажені в систему користувачем або адміністраторами.Атрибут Id – первинний ключ. Зовнішні ключі user_id, admin_id – посилання на таблиці users і admin. Таблиця відповідає третій нормальній формі (3 НФ), оскільки всі неключові атрибути функціонально залежать лише від первинного ключа Id, а також між ними відсутні транзитивні залежності.

Структуру таблиці Images подано у таблиці 2.4.

Таблиця 2.4 – Структура таблиці Images

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	INT	11	-	Містить ідентифікатор зображення
	filename	VARCHAR	255	-	Містить назву файлу
	uploaded_at	TIMESTAMP	-	-	Містить дату та час завантаження
	user_id	INT	11	☑	Містить посилання на користувача
	admin_id	INT	11	☑	Містить посилання на адміністратора
	file_path	VARCHAR	255	-	Містить шлях до файлу

Detections – таблиця з інформацією про детекції. Містить інформацію про координати, клас об’єкта та рівень впевненості, які були отримані в результаті роботи моделі. Атрибут id – первинний ключ. Зовнішній ключ image_id – посилання на таблиці Images. Таблиця відповідає третій нормальній формі (3 НФ), оскільки всі неключові атрибути функціонально залежать лише від первинного ключа Id, а також між ними відсутні транзитивні залежності.

Структуру таблиці Detections подано у таблиці 2.5.

Таблиця 2.5 – Структура таблиці Detections

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	INT	11	-	Містить ідентифікатор детекції
	image_id	INT	11	-	Містить посилання на зображення
	x1	INT	11	-	Містить x1 координату
	y1	INT	11	-	Містить y1 координату
	x2	INT	11	-	Містить x2 координату
	y2	INT	1	-	Містить y1 координату
	detected_class	VARCHAR	100	-	Містить клас виявленого об’єкту
	confidence	FLOAT		-	Містить рівень впевненості моделі

Logs – таблиця журналу дій. Містить інформацію про дії, виконані користувачами або адміністраторами. Атрибут Id – первинний ключ. Зовнішні ключі: user_id та admin_id – посилання на таблиці users і admin. Ця таблиця відповідає третій нормальній формі (3НФ), оскільки всі її неключові атрибути

функціонально залежать лише від первинного ключа Id, а між собою не мають транзитивних залежностей.

Структуру таблиці Logs подано у таблиці 2.6.

Таблиця 2.6 – Структура таблиці Logs

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	INT	11	-	Містить ідентифікатор події
	user_id	INT	11	-	Містить посилання на користувача
	admin_id	INT	11	-	Містить посилання на адміністратора
	action	VARCHAR	255	-	Містить назву дії
	level	VARCHAR	50	-	Містить рівень важливості
	timestamp	TIMESTAMP		-	Містить дату і час події
	details	VARCHAR	45	-	Містить додаткову інформацію

2.3 Розробка структури бази даних

Для розробки web-додатку для декомпозиції на основі нейронної мережі Yolo8 необхідно створити діаграму реляційної бази даних у третій формі нормалізації (3НФ) [22].

Отже, у базі даних diplomna було створено п’ять таблиць:

- users – основна таблиця користувачів системи
- admins – таблиця облікових записів адміністраторів
- images – таблиця для зберігання зображень, які завантажили
- detections – таблиця для зберігання результатів моделі Yolo8
- logs – таблиця для зберігання журналу дій

Між таблицями встановлено зв’язок "один–до–багатьох". Також у розробці використовуватиметься інструмент Alembic, який є стандартним засобом керування схемами баз даних у середовищі Python при роботі з ORM SQLAlchemy. Всі таблиці створюються SQLAlchemy-модель, а зміни фіксуються за допомогою міграції Alembic.

Для генерації наочної діаграми сутностей та їх зв’язків використано вбудований інструмент моделювання даних у середовищі MySQL Workbench. Розроблена діаграма демонструє назви таблиць, перелік полів із зазначенням типів даних і спрямовані стрілки, що позначають зовнішні ключі та типи зв’язків.

Структуру розробленої бази даних наведено на рисунку 2.1.

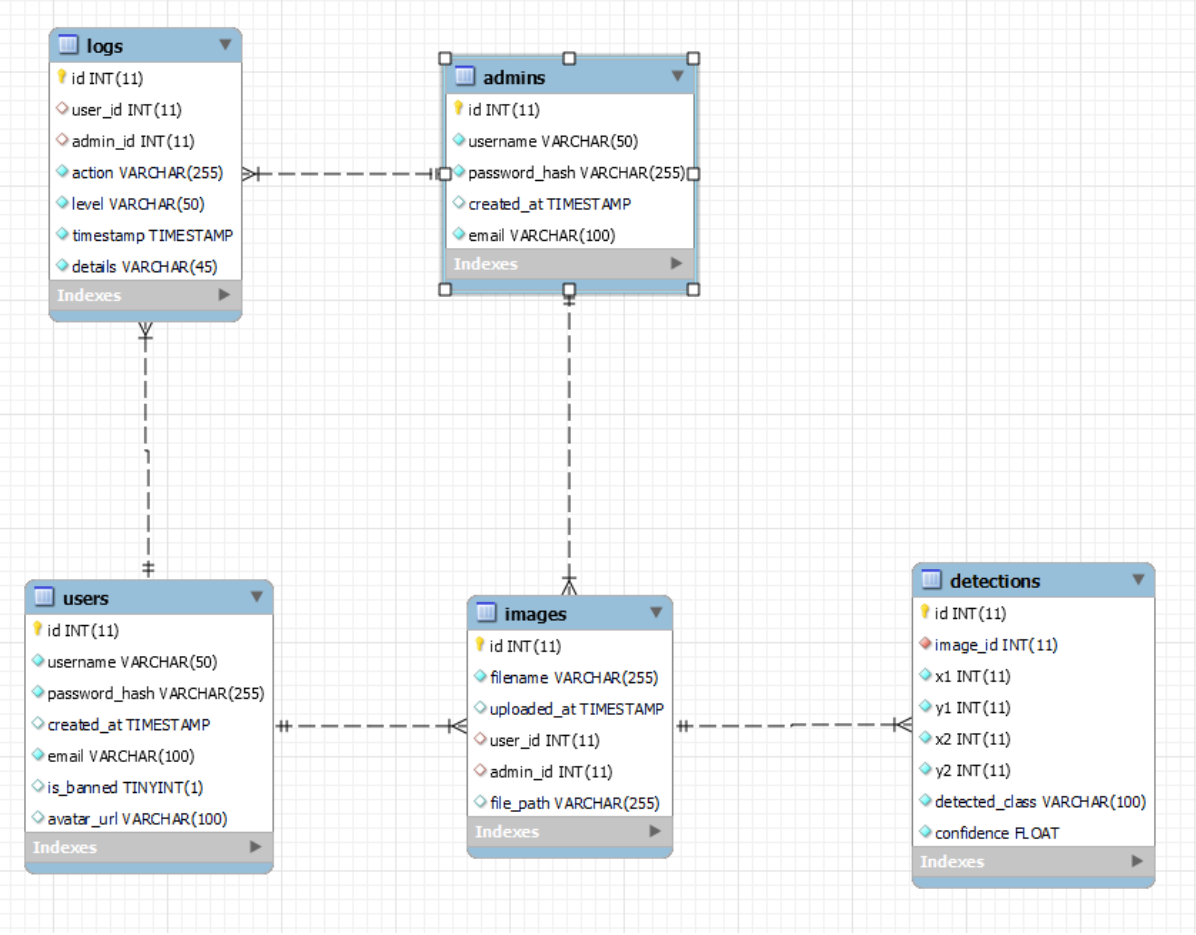

Рисунок 2.1 – Структура бази даних diplomna

Структура бази даних системи редагування зображень (MySQL ERD).

Діаграма бази даних є критично важливим етапом проєкту, оскільки вона дозволяє всебічно оцінити взаємозв'язки даних і уникнути надлишковості. Завдяки застосуванню третьої форми нормалізації забезпечується відсутність транзитивних та часткових залежностей між атрибутами, що гарантує цілісність і консистентність даних.

2.4 Розробка UML діаграм взаємодії з користувачем

Основними складовими діаграм є учасник (користувач) та прецедент (варіант). Побудова діаграм виконана з використанням онлайн-сервісу Visual Paradigm [23].

На рисунку 2.2 подано діаграму прецедентів.

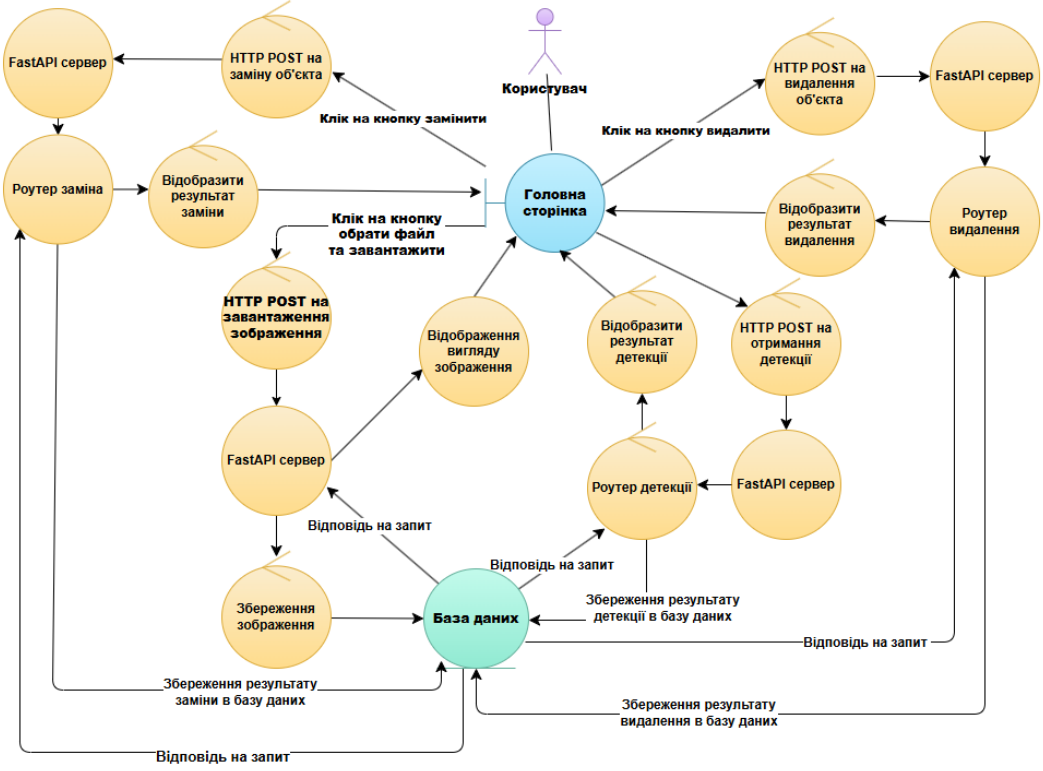

Рисунок 2.2 – Діаграма прецедентів

На рисунку 2.2 представлено діаграму прецедентів, що відображає основні сценарії взаємодії користувача з веб-додатком для редагування зображень на

основі результатів детекції об’єктів. У ролі учасника виступає кінцевий користувач системи, який має можливість завантажити зображення, виконати над ним операції та отримати оброблений результат. Після переходу на головну сторінку веб-додатку користувач здійснює завантаження файлу, що передається на сервер за допомогою HTTP POST-запиту, обробляється сервером FastAPI та зберігається у базі даних. Далі виконується автоматична детекція об’єктів, результат якої також фіксується у базі даних та повертається користувачу у вигляді зображення з рамками (bounding boxes).

Після отримання результату користувач має можливість взаємодіяти з конкретними об’єктами: натиснувши на рамку, він може обрати варіант видалення або заміни. У випадку видалення система надсилає новий HTTP POST-запит до відповідного роутера, сервер обробляє зображення, видаляє обраний об’єкт і оновлює результат. Якщо користувач обирає заміну, необхідно попередньо завантажити новий файл. До моменту завантаження кнопка підтвердження заміни залишається неактивною. Після вибору файлу та натискання кнопки «Замінити» сервер виконує оновлення вмісту зображення, результат зберігається у базі даних і повертається оновлене зображення.

На рисунку 2.3 подано діаграму послідовностей, яка ілюструє повний життєвий цикл взаємодії користувача з системою: від завантаження зображення до отримання результату детекції та редагування зображення.

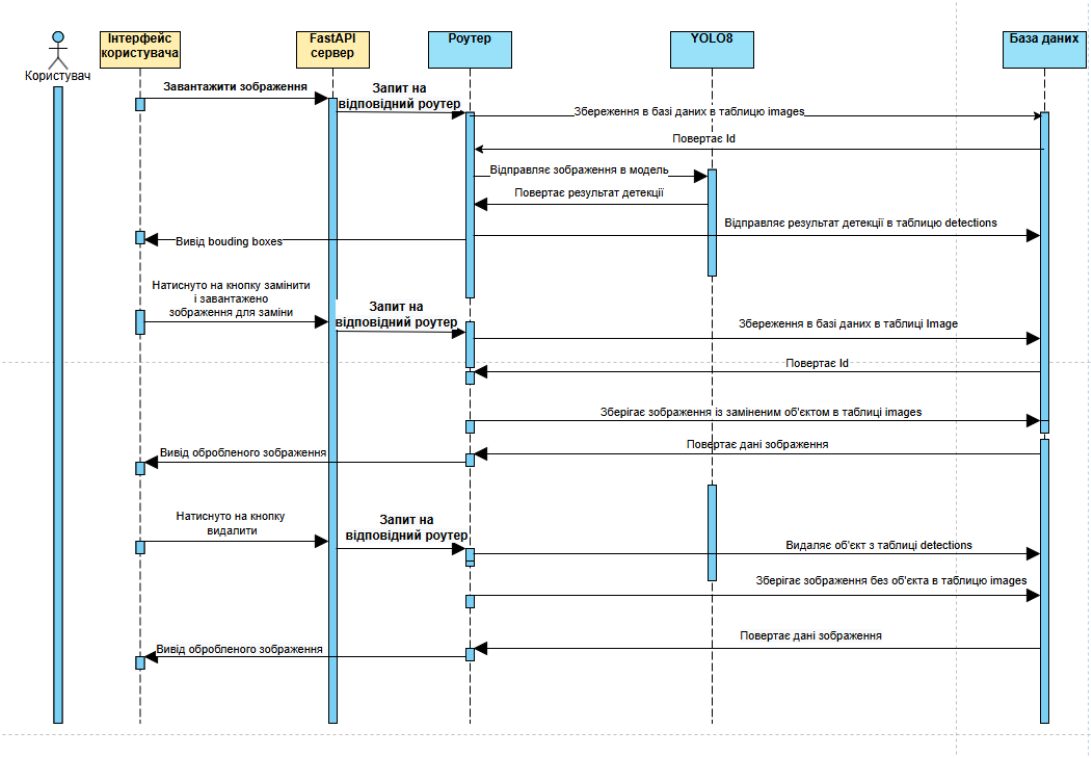

Рисунок 2.3 – Діаграма послідовностей

Основні елементи діаграми:

– Користувач ініціює запит, зокрема авторизацію, завантаження зображення, взаємодія з об’єктами, вибір дії;

– Веб-інтерфейс (Frontend) формує запит до серверної частини та відображає на відповіді;

– Серверна частина (FastAPI) обробляє запити, зберігає інформацію у базу даних та взаємодіє з моделлю YOLOv8;

– YOLOv8 здійснює детекцію об’єктів та повторну обробку після редагування;

– База даних (БД) зберігає зображення, результат детекції, логування дій користувача.

3 РОЗРОБКА ФУНКЦІОНАЛУ ДОДАТКУ

3.1 Настанова користувача

Цей пункт призначений для ознайомлення з основними функціями та можливостями нашого сервісу, а також початковими діями, які здійснює користувач для декомпозиції зображення та детекції.

Головна сторінка: На головній сторінці розміщена форма для завантаження, що складається з кнопок "Обрати файл" та "Завантажити". Після натискання на кнопку "Обрати файл" та вибору зображення, користувач повинен натиснути на кнопку "Завантажити", після цього запуститься детекція. Головну сторінку зображено на рисунку 3.1.

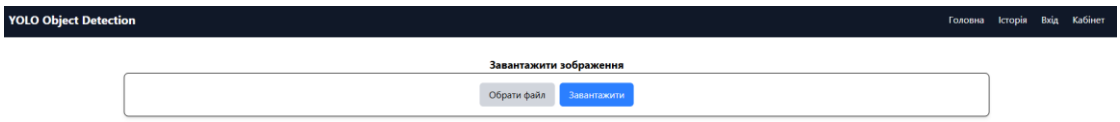

Рисунок 3.1 – Головна сторінка

Головна сторінка з результатом детекції: Після завантаження зображення користувача, відображається результат детекції у вигляді списоку об’єктів, які знаходяться на зображенні. Нижче знаходиться саме зображення з нанесеними bounding boxes, які позначаються об’єкти на сцені. Під зображенням знаходиться кнопка "Завантажити зображення", яка дозволяє завантажити оброблений результат зображення. Головну сторінку з результатами детекції зображено на рисунку 3.2.

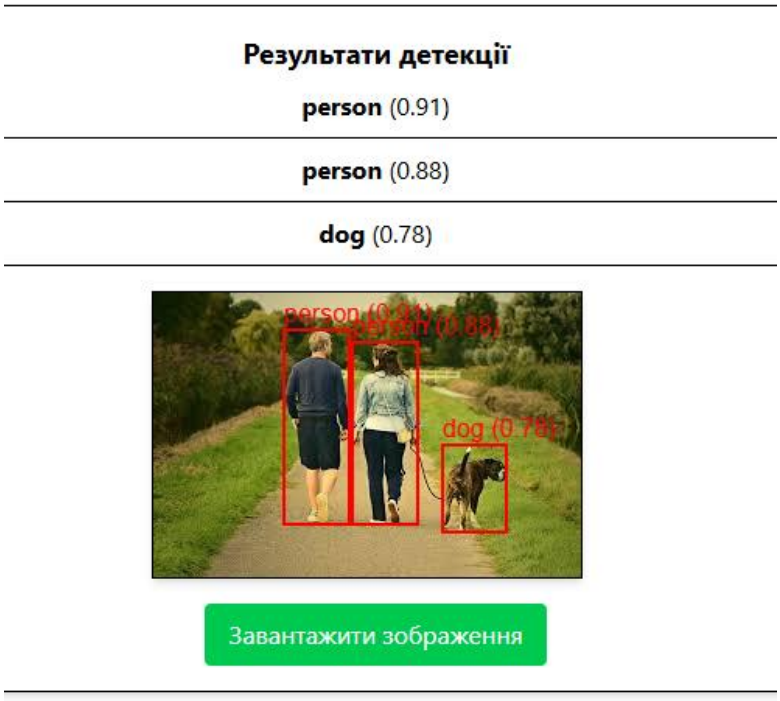

Рисунок 3.2 – Головна сторінка з результатом детекції

Головна сторінка з кнопами: Після детекції об’єктів, користувач має змогу натиснути на будь-який із прямокутників (bounding boxes), після чого в інтерфейсі з’являються дві кнопки – "Видалити об’єкт" та "Замінити об’єкт". При цьому кнопка "Замінити об’єкт" залишається неактивною доти, доки користувач не обере локальний файл для заміни, скориставшись кнопкою "Обрати файл". Це допомагає запобігти небажаним замінам об’єктів та підвищує зручність інтерфейсу. Головну сторінку з кнопками "Видалити об’єкт" та "Замінити об’єкт" зображено на рисунку 3.3.

Рисунок 3.3 – Головна сторінка з кнопками "Видалити об’єкт" та "Замінити об’єкт"

Після того як користувач обрав об’єкт і натиснув кнопку «Видалити об’єкт», відповідний запит надсилається на сервер, де запускається процедура обробки зображення. Система видаляє вибраний об’єкт з використанням інструментів нейронного заповнення (inpainting), формує нове зображення без цього об’єкта і повертає його клієнту. Одразу після цього запускається повторна детекція для оновленого зображення, в результаті чого на ньому будуть згенеровані нові рамки для виявлених об’єктів. Такий підхід гарантує інтерактивність редагування та збереження актуальності результатів аналізу. Головну сторінку з оновленим зображенням після успішного видалення об’єкта наведено на рисунку 3.4.

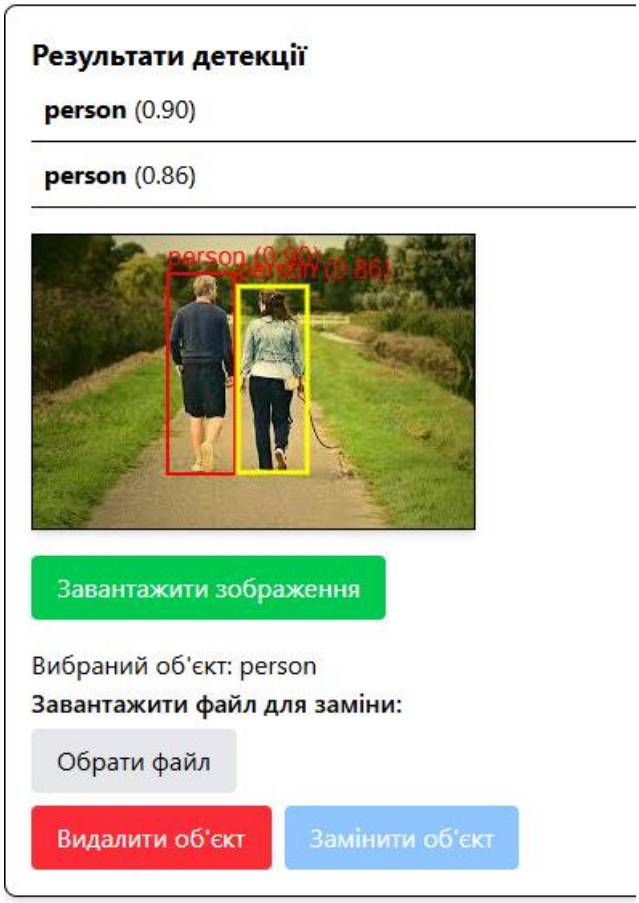

Рисунок 3.4 – Головна сторінка з результатом видаленого об’єкта

Головна сторінка з результатом редагування об’єкта: Після вибору об’єкта, який користувач буде заміняти, потрібно натиснути на кнопку "Обрати файл" та завантажити об’єкт, який буде використаний для заміни. Після цього кнопка "Замінити об’єкт" стане активною, і користувачу потрібно натиснути на неї для заміни об’єкта, в кінцевому результаті на сайт повернеться оброблене зображення із заміненим об’єктом та заново запуститься детекції. Головну сторінку з результатом заміненого об’єкта зображено на рисунку 3.5.

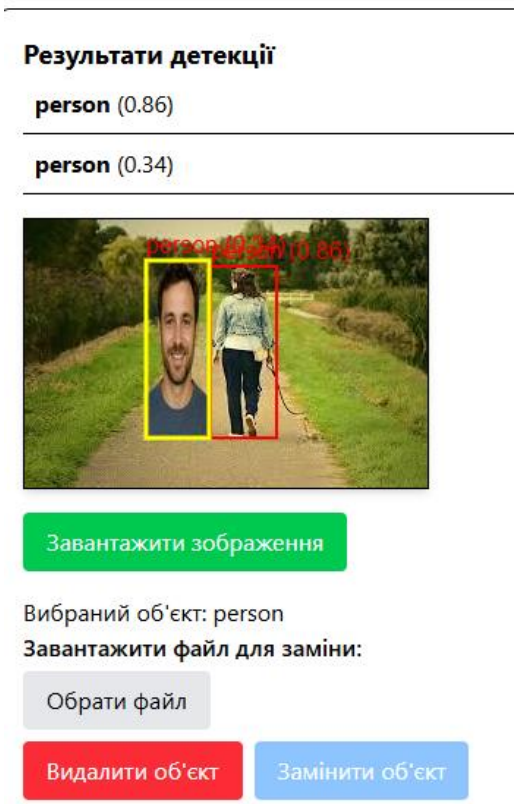

Рисунок 3.5 – Головна сторінка з результатом заміненого об’єкта

3.2 Розробка функціоналу веб-додатку

Розпочато розробку веб-сервісу, послуговуючись недоліками аналогічними веб-сайтами, які були розглянуті в пункті 1.3. Перелік відповідних джерел наведений у списку літератури [1-2].

Проект має REST-орієнтовану архітерктуру з чітким розділенням на фронтенд та бекедн частини. На серверній частині, який реалізований на FastAPI, основна логіка розміщується в структурованих папках і має такий вигляд:

- routers – містить модулі з REST-маршрутами, які відповідають за обробку запитів.
- schemas – містить Pydantic-схеми для валідації даниїх.
- models – містить SQLAlchemy-моделі бази даних.
- services – реалізує бізнес-логіку, зокрема взаємодію з моделлю YOLOv8, видалення та заміна об’єктів.

– database – конфігурація підключення до MariaDB.

– auth – окрема логіка для авторизації.

У файлі main.py виконує роль точки входу у застосунок та підключає маршрути, CORS. Файлову структуру бекенду подано на рисунку 3.6.

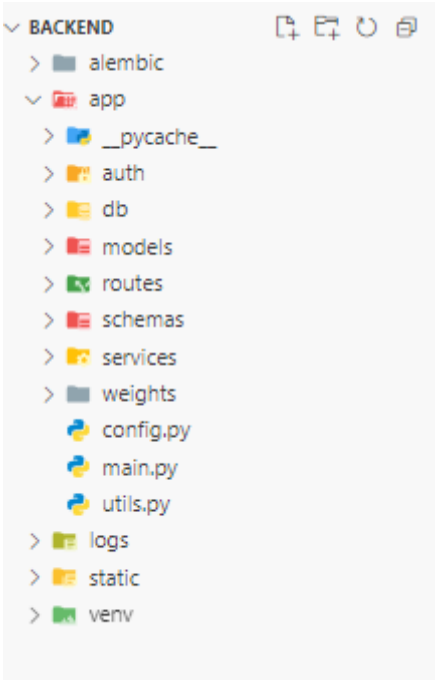

Рисунок 3.6 – Файлова структура бекенду

У рамках реалізації клієнтської частини web-додатку було створену окрему структуру проекту на основі фреймворку React та Tailwind, яка дозволяє забезпечити швидкий рендер, гнучкість інтерфейсу та зручну взаємодію з REST API бекендом. Основна структура фронтенду складається з таких структурованих папках:

– components – містить візуальні компоненти інтерфейсу.

– pages – окремі сторінки веб-додатку, наприклад авторизація, головна сторінка і тощо.

– api – модуль для взаємодії з сервером, наприклад завантаження зображення.

У центрі клієнтської частини знаходиться файл App.jsx, який виконує роль основного компонента-обгортки для всього веб-додатку. Саме з нього відбувається рендер усіх сторінок та компонентів. Файлову структуру фронтенду подано на рисунку 3.7.

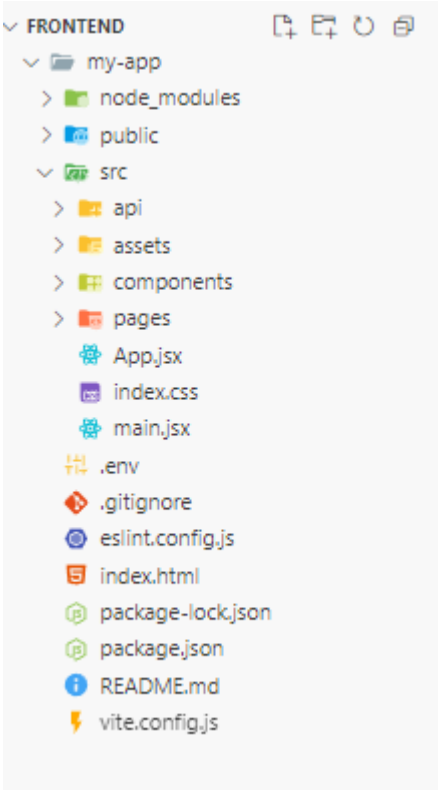

Рисунок 3.7 – Файлова структура фронтенду

Після запуску web-додатку користувач автоматично буде перенаправлений на сторінку авторизації. Ця сторінка є відправною точкою для роботи на цьому web-додатку, оскільки тільки після проходження авторизації користувач отримає доступ до основного функціоналу.

Для створення облікового запису передбачене окрема сторінка, доступна через посилання "Create an Account". Форма реєстрації містить такі поля: Ім’я користувача, електронна пошта, пароль та підтвердження пароля. Сторінка створення акаунта зображена на рисунку 3.8.

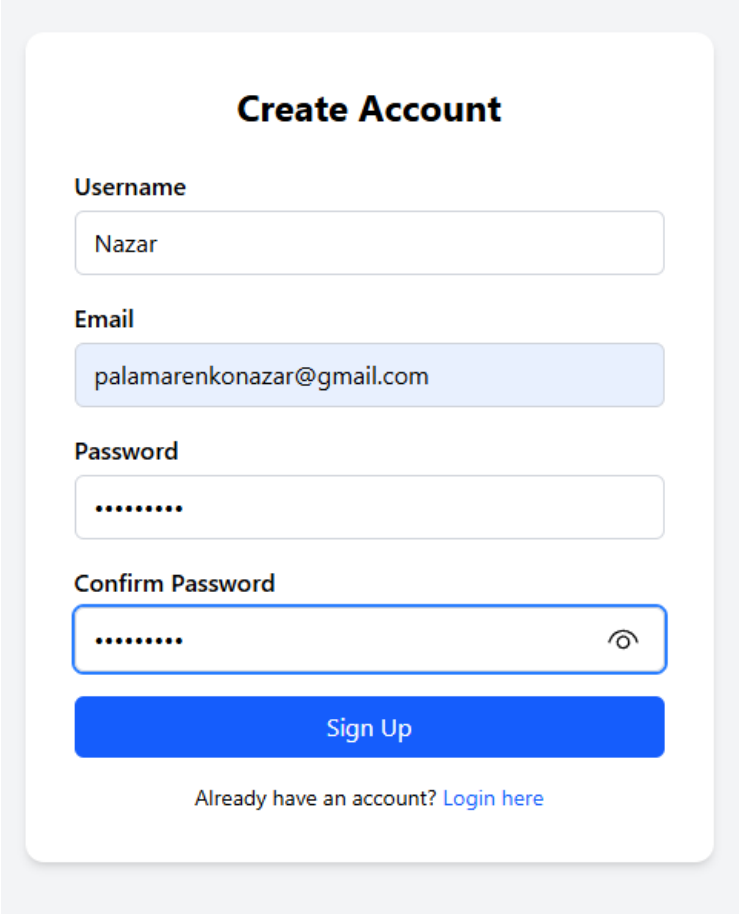

Рисунок 3.8 – Сторінка створення акаунта

Після заповнення форми і натискання на кнопку "Sign Up" спрацьовує функція handleSubmit, яка відправляє дані користувача на сервер за допомогою HTTP – запиту. Код форми представлений нижче:

```
  <form onSubmit={handleSubmit}>
    <input type="text" value={username} onChange={...} required />
    <input type="email" value={email} onChange={...} required />
    <input type="password" value={password} onChange={...} required
/>
    <input type="password" value={confirmPassword} onChange={...}
required />
    <button type="submit">Sign Up</button>
  </form>
```

На серверній частині запит обробляється маршрутом /auth/signup, що реалізований на FastAPI. У тілі запиту надходять всі дані з форми у вигляді Pydantic – схеми. Після отримання цих даних генерується JWT – токен, який генерується під час підтвердження листа:

```
@router.post('/signup')
async def signup(signup_args: SignUpArgs, db: AsyncSession = Depends(get_async_db)):
    token = generate_signup_confirmation_token(signup_args)
    await send_confirmation_email(token, signup_args)
```

Такий підхід дозволяє підтвердити автентичність email – адреси та запобігає реєстрації неіснуючих облікових записів.

Після створення нового акаунта на електронну пошту надсилається лист підтвердження, яке містить унікальний JWT – токен. На рисунку 3.9 зображений лист підтвердження.

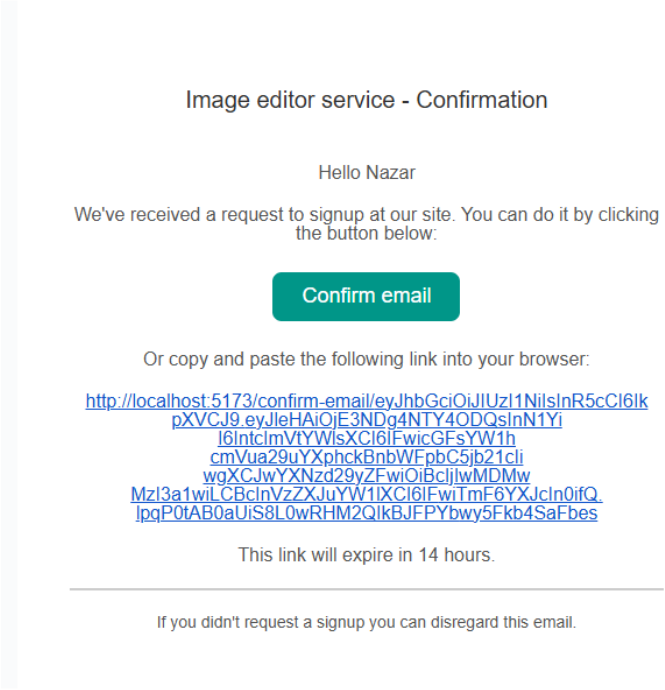

Рисунок 3.9 – Лист підтвердження для підтвердження пошти

Користувач натискає на кнопку "Confirm email", яка переадресовує його на сайт, і токен передається на сервер через маршрут /signup–confirmation. Після цього токен перевіряється функцією validate_signup_confirmation_token, яка розшифровує та підтверджує його достовірність. Якщо користувач із такою електронною поштою вже існує, сервер повертає повідомлення, що email вже підтверджено. Якщо ж користувача немає, він створюється у базі даних за допомогою функції create_user, після чого генерується JWT – токен доступу, який повертає клієнту для подальшої автентифікації. Такий механізм гарантує, що акаунт буде активований тільки після підтвердження пошти, захищаючи. Код роутера для підтвердження електронної пошти:

```
@router.post('/signup-confirmation')
async def signup_confirmation(token: str, db: AsyncSession = Depends(get_async_db)):
    args = validate_signup_confirmation_token(token)
    if await user_exists(db, args.email):
        raise HTTPException(status_code=200, detail="Email has been already confirmed")
    new_user = await create_user(db, UserCreate(**args.model_dump()))
    access_token = create_access_token(data={"sub": args.username, "role": 'user'})
    return {"access_token": access_token, "token_type": "bearer"}
```

Після підтвердження пошти, користувач може увійти на сайт через сторінку авторизації, яка представляє собою класичну форму з двома полями: електронна пошта та пароль. При натисканні на "Login" виконується функція handleLogin, яка відправляє дані форми на сервер. На рисунку 3.10 зображена сторінка авторизації.

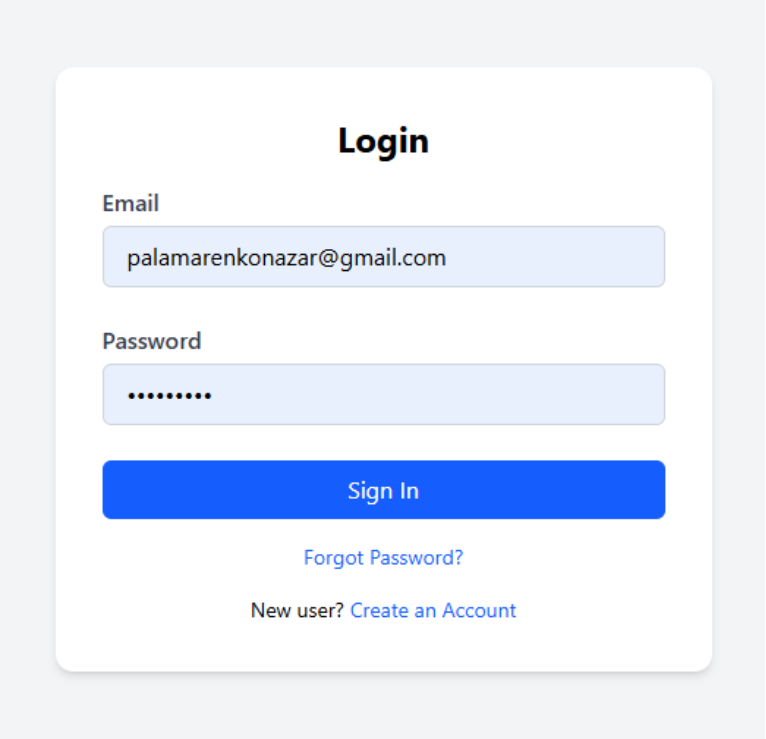

Рисунок 3.10 – Сторінка авторизації при вході на сайт

Код представлення форми для авторизації:

```
  <form onSubmit={handleLogin}>
    <input type="email" value={email} onChange={...} required />
    <input type="password" value={password} onChange={...} required
/>
    <button type="submit">Login</button>
  </form>
```

Дані з форми надсилаються на сервер методом POST на маршрут /login, де їх приймає роутер і перевіряє достовірність через функцію authenticate_user, яка шукає користувача за email і перевіряє пароль. Якщо дані не коректні, сервер повертає 400 з повідомленням про неправильне ім’я або пароль. Якщо аутентифікація пройшла успішно, генерується JWT – токен доступу, в який записується ім’я користувача та його роль, цей токен повертає клієнту і зберігається у localStorage для подальшої авторизації в системі. Код роутера:

```
@router.post("/login")
async def login_for_access_token(args: SignInArgs,db: AsyncSession = Depends(get_async_db)):
    auth_result = await authenticate_user(db, args.email, args.password)
    if not auth_result:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = auth_result["user"]
    role = auth_result["role"]
    access_token = create_access_token(data={"sub": user.username, "role": role})
```

Якщо користувач забув пароль, він може перейти на сторінку "Forgot Password?", де потрібно ввести електронну пошту для отримання посилання на відновлення доступу. Ця сторінка містить просту форму з полем для email і кнопку "Send Reset Link", при натисканні якої виконується функція handleSubmit, що відсилає запит на сервер. Сторінка для відновлення пароля зображена на рисунку 3.11.

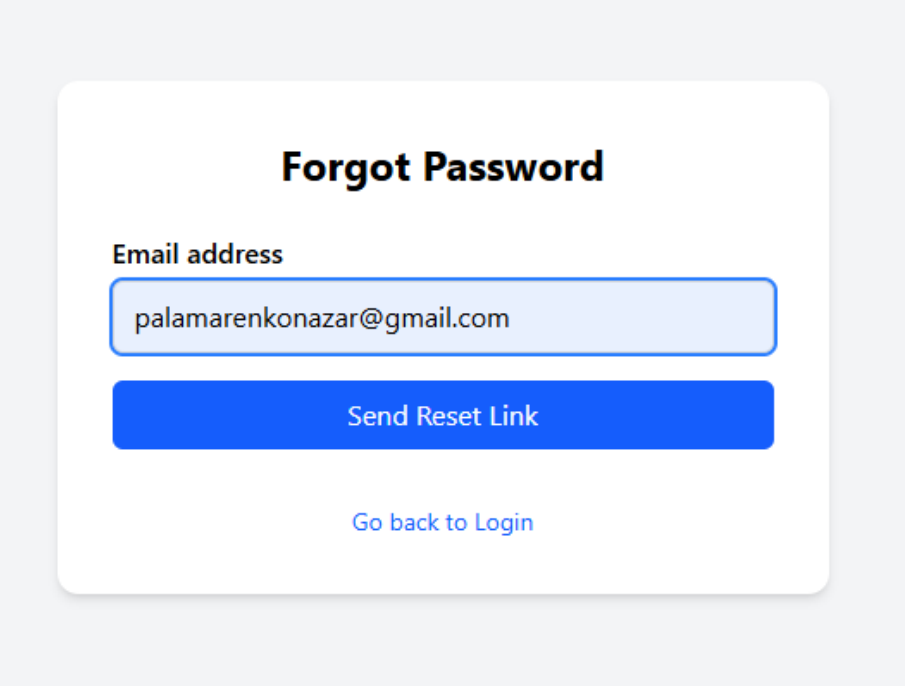

Рисунок 3.11 – Сторінка для відновлення пароля

Код представлення форми для відновлення пароля:

```
<form onSubmit={handleSubmit}>
  <input type="email" value={email} onChange={...} required />
  <button type="submit">Send Reset Link</button>
</form>
```

Сервер отримує запит на маршрут /reset–password, де відбувається обробка нового пароля. При цьому електронна пошта користувача підтверджується через JWT – токен, який був надісланий у електронному листі для скидання пароля і передається як залежність у параметрі email за допомогою функції validate_password_reset_token. Далі за цим email знаходиться користувач у базі даних, і його пароль оновлюється через функцію update_user_password з новим паролем, який передається в параметрі new_password. Такий підхід гарантує безпеку процедури скидання пароля, оскільки новий пароль приймається і змінюється тільки після підтвердження користувача через унікальне посилання, надіслане на його електронну пошту. Код роутера:

```
@router.patch("/reset-password")
async def reset_password(new_password: str, email: EmailStr =
Depends(validate_password_reset_token), db: AsyncSession =
Depends(get_async_db)):
    user = await get_user_by_email(db, email)
    await update_user_password(db, user,
ChangePassword(new_password=new_password))
```

Після введення електронної пошти на неї приходить лист підтвердження для скидання пароля. Це посилання містить унікальний JWT – токен, який використовується для перевірки достовірності запиту на заміну пароля. Лист підтвердження зображений на рисунку 3.12

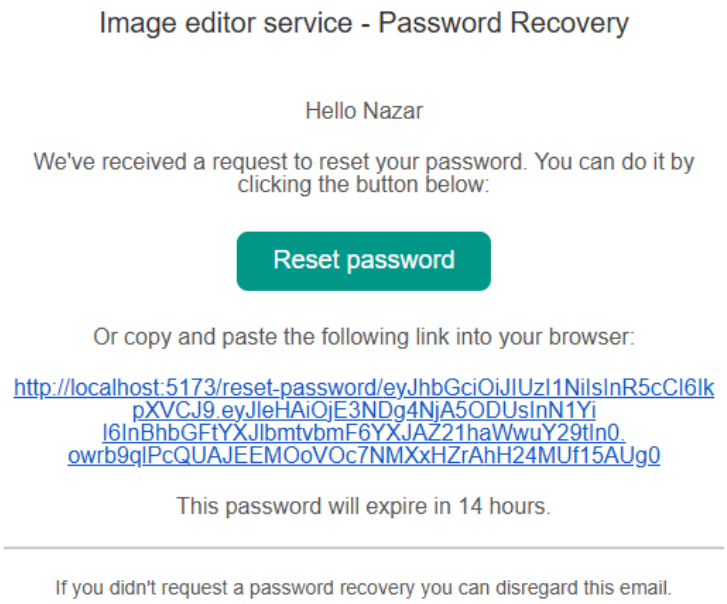

Рисунок 3.12 – Лист підтвердження для скидання пароля

На серверу маршрут /reset-password приймає новий пароль і електронну пошту, яка підтверджується через функцію validate_password_reset_token, що перевіряє токен із посилання. Далі за підтвердженою електронною адресою знаходиться відповідний користувач в базі даних, і його пароль оновлюється за допомогою функції update_user_password, яка приймає новий пароль у вигляді об’єкта ChangePassword. Такий механізм забезпечує безпечну зміну пароля лише після підтвердження на електнонній пошті користувача. Код роутера:

```
@router.patch("/reset-password")
async def reset_password(new_password: str,
email: EmailStr = Depends(validate_password_reset_token),
 db: AsyncSession = Depends(get_async_db)):
    user = await get_user_by_email(db, email)
    await update_user_password(db, user,
ChangePassword(new_password=new_password))
```

Після авторизації користувач потрапляє на головну сторінк web–додатку, де відображаються кнопки "Обрати файл" та "Завантажити". Для завантажити зображення користувач нискає на кнопку "Обрати файл" і вибирає потрібний файл зі свого пристрою, після чого клікає на кнопку "Завантажити". Після цього зображення надсилається на сервер, де відбувається його обробка – детекція об’єктів із накладеними bounding boxes. Результати детекції повертаються на фронтенд, де за допомогою Canvas API відбувається малювання прямокутників навколо знайдених об’єктів із підписами класів і впевноності. Після успішної обробки на сторінці з’являється додаткова кнопка "Завантаження зображення", що дозволяє користувачеві завантажити оброблене зображення з накладеними bounding boxes. Приклад взаємодії з цим функціонал зображено на рисунку 3.13.

Рисунок 3.13 – Приклад взаємодії з функціоналом web-додатку

На фронтенді для завантаження зображення використовується форма, що містить прихований елемент <input type ="file">, кнопки для вибору файлу та відправлення запиту на сервер, а також кнопка для завантаження обробленого зображення. Код малювання bounding boxes реалізований через контекст canvas: спочатку зображення малюється на полотні, а потім поверх нього за допомогою методу strokeRect створюються прямокутники та підписи з іменами класів та рівнем впевненості.

Код форми для завантаження:

```
    <div className="p-4 border rounded-lg bg-white shadow-md">
      <input type="file" ref={fileInputRef} style={{ display:
"none" }} accept="image/*" />
      <button onClick={...} className="bg-gray-300 px-4 py-2
rounded-md mr-2">
        Обрати файл
      </button>
      <button onClick={...} className="bg-blue-500 text-white px-4
py-2 rounded-md">
        Завантажити
      </button>
    </div>
```

Код для малювання bounding boxes на Canvas:

```
ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
ctx.strokeStyle = "red";
ctx.lineWidth = 2;
ctx.font = "16px Arial";
ctx.fillStyle = "red";
detections.forEach(({ x1, y1, x2, y2, detected_class, confidence })
=> {
  ctx.strokeRect(x1, y1, x2 - x1, y2 - y1);
```

```
  ctx.fillText(`${detected_class} (${confidence.toFixed(2)})`, x1,
y1 - 5);
});
```

На бекенді запити обробляють кілька роутерів. Перший роутер /upload_file приймає файл зображення, зберігає його через функцію save_uploaded_file, яка повертає його ідентифікатор. Другий роутер /detect/{id} отримує ідентифікатор зображення, завантажує файл і виконує детекцію об’єків за допомогою нейронної моделі YOLO. Якщо під час обробки виникає помилка, сервер повертає HTTP – помилку 500 із повідомлення. Для кожного знайденого об’єкта створюється запис у базі даних із координатам bounding boxes, класом об’єкта та рівнем впевненості. Крім того, логуються дії користувача із зазначенням кількості виявлених об’єктів і назви файлу. Цей комплексний підхід забезпечує зручну взаємодію користувача з функцією завантаження та обробки зображень у web-додатку.

Код роутера для завантаження зображень:

```
@router.post("/upload_file")
async def upload_file(
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
    image=awaitsave_uploaded_file(file, current_user.id, db)
    return {"image_id": image.id}
```

Код роутера для детекції об’єктів:

```
@router.post("/detect/{id}")
async def detect_objects(id: int, db: AsyncSession =
Depends(get_async_db)):
    image = await get_image(db, id)
```

```
        try:
            results = await yolo.detect_objects(image.file_path)
        detection_data = DetectionCreate(
                image_id=image.id,
                x1=obj["x1"],
                y1=obj["y1"],
                x2=obj["x2"],
                y2=obj["y2"],
                detected_class=detected_class,
                confidence=obj.get("confidence", 0.0)
            )
    detection=awaitcreate_detection(db,detection_data)
    detections.append(detection)
```

Після завершення процесу детекції об’єктів на зображенні користувач має можливість взаємодіяти з кожним із них. Для цього достатньо натиснути на будь-який bounding box на зображенні. Після кліку система визначає, який саме об’єкт вибрано, та відображає додаткові елементи керування. Зокрема, користувач бачить інформацію про клас вибраного об’єкту, а також отримує доступ до трьох кнопок: "Обрати файл", "Видалити об’єкт" та "Замінити об’єкт". Результат взаємодії з bounding boxes зображений на рисунку 3.14.

Рисунок 3.14 – Результат взаємодії з bounding boxes

Код, що реалізує цей функціонал, складається з елементів <canvas>, на якому відображається зображення з bounding boxes, та умовного блоку з кнопками, який з'являється після вибору об'єкта:

```
  <canvas ref={canvasRef} onClick={handleCanvasClick}
classNamе="border my-4" />
  {selectedObject && (
    <div>
      <p>Вибраний об'єкт: {selectedObject.detected_class}</p>
      <label classNamе="inline-block bg-gray-200 px-3 py-1
rounded cursor-pointer">
        Обрати файл
        <input type="file" accept="image/*" onChange={(e) =>
setReplacementFile(e.target.files[0])} classNamе="hidden" />
      </label>
```

```
        <button onClick={handleRemoveObject} className="bg-red-500
text-white px-3 py-1 rounded ml-2">Видалити</button>
        <button                          onClick={handleReplaceObject}
disabled={!replacementFile} className="bg-blue-500 text-white px-
3 py-1 rounded ml-2">Замінити</button>
      </div>
    )}
```

У разі натискання користувачем на кнопку "Видалити об’єкт" відбувається запит на роуте /remove, який обробляє зображення шляхом видалення вибраного об’єкта. Цей процес реалізується шляхом передачі координат bounding box до відповідного роутера, який виконує видалення. Результат видалення об’єкта зображено на рисунку 3.15.

Рисунок 3.15 – Результат видалення об’єкта

Фрагмент коду, що відповідає за обробку натискання кнопки:

```
const handleRemoveObject = async () => {
  if (!selectedObject) return;
  try {
    const res = await removeObject(imageId, selectedObject);
    if (res.data?.updated_image_id) {
      setImageId(res.data.updated_image_id);
      setSelectedObject(null);
      setReplacementFile(null);
    }
  } catch (error) {
    console.error("Помилка під час видалення об'єкта:",
error.response?.data);
```

Серверна логіка спочатку перевіряє, чи існує зображення в базі даних. Якщо запис не знайдено, користувачу повертається повідомлення з кодом помилки 404. Якщо зображення існує, відбувається обробка: видалення об’єкта з зображення за допомогою алгоритиму заповнення за допомогою інструмента LaMa. У разі виненкнення помилки під час обробки система повертає повідомлення про помилку з кодом 500. Така перевірка та обробока помилок дозволяють забезпечити стабільність роботи системи й повідомити користувача про будь-які збої. Після успішної обробки користувачу повертається оновлене зображення, яке автоматично виводиться на клієнтську частину. Одразу після цього повторно запускається детекція об’єктів. Код роутера, який відповідає за видалення об’єктів з зображення :

```
@router.post("/remove/")
async def remove_object_route(
    data: YoloBase,
    current_user:User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
```

```
):
    x1, y1, x2, y2 = data.x1, data.y1, data.x2, data.y2
    try:
        modified_image_path = await asyncio.to_thread(
            remove_object_lama,
            image.file_path,
            {"x1": x1, "y1": y1, "x2": x2, "y2": y2}
        )
    delete_query = text("""
        DELETE d
        FROM detections d
        WHERE d.image_id = :image_id
          AND d.x1 = :x1 AND d.y1 = :y1
          AND d.x2 = :x2 AND d.y2 = :y2
    """)
    await db.commit()
```

Якщо користувач хоче замінити об’єкт на зображенні, йому спочатку необхідно натиснути на "Обрати файл" і вибрати зображення для заміни. До моменту вибору файлу кнопка "Замінити об’єкт" залишається неактивною, що реалізовано через перевірку наявності файлу. Після вибору нового зображення кнопка активується, і користувач може здійснити заміну. Результат заміни об’єкта зображено на рисунку 3.16

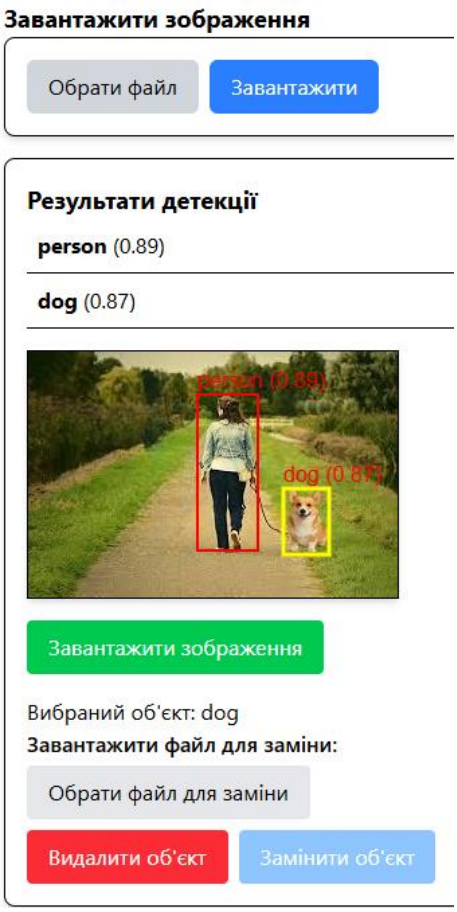

Рисунок 3.16 – Результат заміни об’єкта

Фрагмент коду, який відповідає за цю логіку:

```
const handleReplaceObject = async () => {
  if (!selectedObject || !replacementFile) return;
  try {
    const formData = new FormData();
    formData.append("file", replacementFile);
    const res = await replaceObject(formData, imageId,
selectedObject);
    if (res.data?.processed_image) {
      setImageId(res.data.processed_image);
      setSelectedObject(null);
      setReplacementFile(null);
<button
  onClick={handleReplaceObject}
```

```
        disabled={!replacementFile}
        className={`px-4 py-2 rounded text-white ${
      <button>
        Замінити об'єкт
      </button>
```

Після натискання на кпоку "Замінити об’єкт" відправляється запит на бекенд за маршрутом /replace, до якого передаються координати обраного об’єкта та новий файл, що має його замінити. Серверна логіка починається з перевірки наявності запису зображення в базі даних. Якщо зображення не знайдено, система поверта помилку з кодом 400. У випадку успішного знаходження зображення та переданого об’єкта для заміни, сервер виконує обробку: старий об’єкт видаляється з вказаних координат, а на його місце вставлється новий. Код роутера для замінити об’кта:

```
    @router.post("/replace/")
    async def replace_object_route(
        data: YoloReplace,
        current_user:User = Depends(get_current_user),
        db: AsyncSession = Depends(get_async_db)
    ):
        image = await get_image(db,data.image_id)
        x1, y1, x2, y2 = data.x1, data.y1, data.x2, data.y2
        replacement_file = await
get_image(db,data.replacement_id)
        replacement_path = replacement_file.file_path
        image_path = image.file_path
        try:
            modified_image_path = await asyncio.to_thread(
                replace_object_lama,
                image_path,
                {"x1": x1, "y1": y1, "x2": x2, "y2": y2},
                replacement_path
```

```
            )
        modified_image = await
save_uploaded_file_by_url(modified_image_path,current_user.id,db)
```

Після завершення редагування зображення користувач має можливість завантажити його на свій пристрій, натиснувши кнопку "Завантажити зображення". При цьому з відредагованого зображення видаляються всі bounding boxes, що були накладені під час процесу детекції та редагування. На рисунку 3.17 результат завантаженого відредагованого зображення.

Рисунок 3.17 – Результат завантаженого зображення.

В цьому фрагменті коду реалізований функціонал завантаження зображення. Спочатку перевіряється, чи обрано файл для завантаження, і якщо ні – користувач отримує повідомлення з проханням вибрати зображення. Обраний файл упаковується у формат FormData і надсилається на сервер через асинхронний запит. У разі успішного отримання відповіді сервер повертає унікальний ідентифікатор зображення, який зберігається як у стані компонента, так і в локальному сховищі браузера для подальшого використання. Якщо виникає помилка, користувач отримує відповідне повідомлення. Фрагмент коду, що відповідає за цю логіку:

```
  const handleUpload = async () => {
    if (!fileInputRef.current?.files[0]) return alert("Виберіть
зображення!");
    const formData = new FormData();
    formData.append("file", fileInputRef.current.files[0]);
    try {
      setLoading(true);
      const res = await uploadImage(formData);
      const newId = res.data.image_id;
      setImageId(newId);
      localStorage.setItem("imageId", newId);
    } catch (error) {
      alert("Не вдалося завантажити зображення.");
    } finally {
      setLoading(false);
    }
  };
```

Реалізовано особовий кабінет, у якому користувач має можливість додати або змінити аватарку, написнувши на кнопку "Вибір файл" та обравши відповідне зображення з пристрою, а потім зберегти зміни, натиснувши на кнопку Зберегти аватарку". Крім того, передбачена функція видалення облікового запису через кнопку "Видалити акаунт" із подальшим підтвердженням дії. Збереження аватарки здійснюється у локальному сховищі браузері. При завантаженні сторінки відбувається запит до API для отримання профілю користувача, після чого аватарка відображається на сторінці. Також користувач може легко замінити аватарку, просто вибравиш інше зображення. Сторінка особового кабінетру зображена на рисунку 3.18.

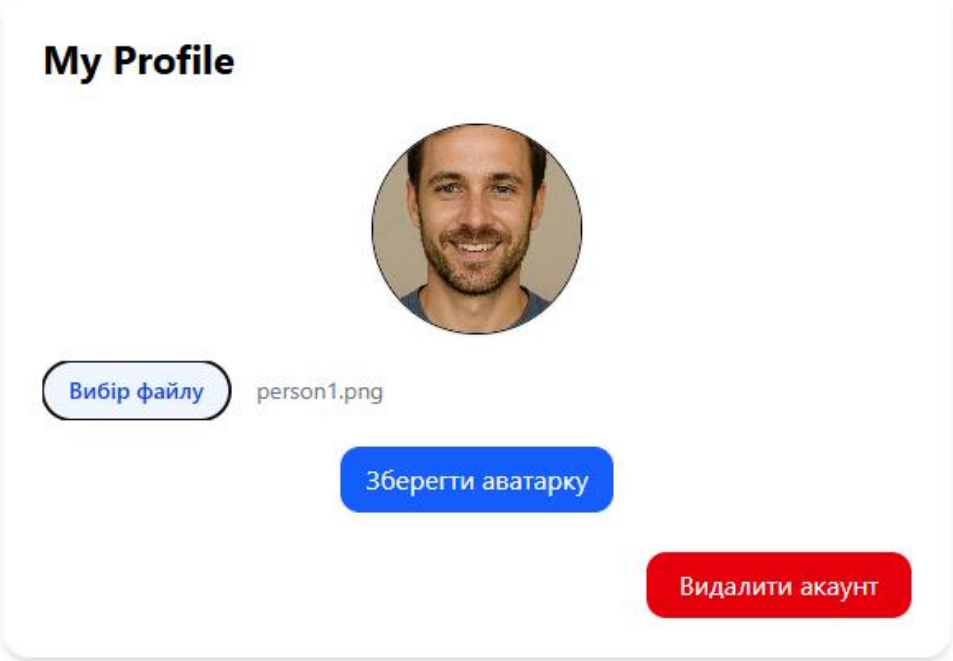

Рисунок 3.18 – Сторінка особового кабінетру.

У фрагменті коду показано логіку завантаження та збереження аватарки. Після вибору файлу виконується асинхронний запит на сервер для оновлення аватарки. Після успішного завантаження оновленний URL аватарки зберігається у профілі та локальному сховищу для подальшого відображенням:

```
const handleAvatarChange = async (e) => {
    const file = e.target.files[0];
    if (file) {
      const updated = await uploadAvatar(file);
      setProfile((prev) => ({
        ...prev,
        avatar_url: avatarWithTimestamp,
      }));
 localStorage.setItem("avatar_url", avatarWithTimestamp);
    }
  };
```

Серверна частина реалізована через роутер /me/avater, який приймає файл аватарки, зберігає його на сервері з унікальним іменем, оновлює шлях до аватарки у базі даних. Такий підхід забезпечує збереження і відображення актуальної аватарки користувача в особовому кабінеті. Код роутера:

```
@router.post("/me/avatar")
async def upload_avatar(
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
        filename = f"{current_user.id}_{file.filename}"
        file_path = os.path.join(UPLOAD_DIR, filename)
        with open(file_path, "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)
   current_user.avatar_url = f"/static/avatars/{filename}"
        await db.commit()
```

Користувач має можливість видалити свій акаунт, натиснувши на кнопку "Видалити акаунт". Після цього система відкриває спливаюче вінко підтвердження, де користувач повинен підтвердити своє рішення або скасувати дію. Якщо користувач підтверджеє видалення акаутна, відправляється асинхронний запит на сервер, який виконує процедуру облікового запиту з бази даних. Після успішного видалення акаутна з локального сховища видаляються всі дані, пов’язані з авторизацією, зокрема токен доступу та збереження аватарки, а користувач автоматично перенаправляється на сторінку авторизації. На рисунку 3.19 зображений процес видалення акаунта.

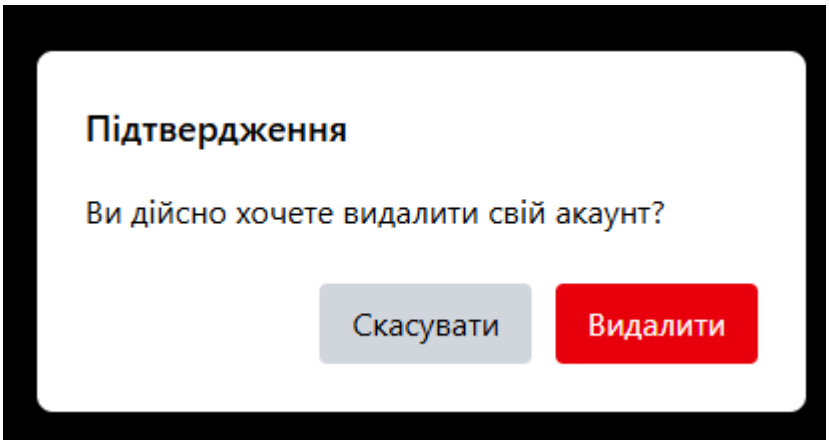

Рисунок 3.19 – Процес видалення акаунта

У клієнтській частині реалізована логіка підтвердження дії видалення акаутна через модальне вікно. Користувач має змогу або скасувати дію, натиснувши на кнопку "Скасувати", або підтвердити її, натиснувши на кнопку "Видалити". При підтвердженні виконується виклик функції, яка відповідає за відправку запиту на сервер, очищення локального сховища та перенаправлення користувача. Код фрагменту видалення облікового запису:

```
const confirmDelete = async () => {
    await deleteAccount();
    localStorage.removeItem("token");
    localStorage.removeItem("token_type");
    localStorage.removeItem("avatar_url");
    navigate("/login");
  };
          <div className="bg-white p-6 rounded-lg shadow-lg max-w-
sm w-full">
                      <h3   className="text-lg   font-semibold   mb-
4">Підтвердження</h3>
               <p  className="mb-6">Ви  дійсно  хочете  видалити  свій
акаунт?</p>
            <div className="flex justify-end space-x-3">
              <button
                onClick={() => setShowConfirm(false)}
```

```
                 className="px-4 py-2 bg-gray-300 rounded hover:bg-
gray-400"
               >
                 Скасувати
               <button
                 onClick={confirmDelete}
                  className="px-4 py-2 bg-red-600 text-white rounded
hover:bg-red-700"
                 Видалити
               </button>
             </div>
```

На серверній частині реалізований роутер за маршрутом /delete-account, який через механізм залежностей отримує поточного автентифікованого користувача та видаляє його з бази даних. У разі успішного видалення повертається повідомлення про усіх, а у випадку помилки відповідний код – 400 і повідомлення про невдалу спробу видалення. Код роутера для видалення акаутна:

```
@router.delete("/delete-account")
async def delete_my_account(
    current_user: User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
    try:
        await db.delete(current_user)
        await db.commit()
        return {"detail": "Account deleted successfully"}
    except Exception as e:
            raise  HTTPException(status_code=400,  detail="Account
deletion failed")
```

Повний код знаходиться в додатку Б.

ВИСНОВКИ

У першому розділі бакалаврської роботи, проведено постановку задачі, подано теоретичні дані щодо завдання, пояснення алгоритмів нейронних моделей для знаходження об’єктів, розписані принципи декомпозиції, огляд вже існуючих аналогів, правила щодо запису вхідних даних та постановка задачі.

У другому розділі бакалаврської роботи, обрано нейронну модель на основі критеріїв швикості, точності, підтримки сегментації та простоти інтегрування, спроектовано та розроблено базу даних, UML діаграми use-case для представлення можливих варіантів взаємодії користувача з web-додатком та діаграма послідовностей виконання внутрішніх процесів.

У третьому розділі бакалаврської роботи, на основі другого розділу розроблено web-додаток із використанням FastAPI для розробки REST архітектури, React для створення інтерфейсу користувача, а також підключено модель YOLOv8 та інструмент LaMa Cleaner для декомпозиції об’єктів. Також наведено основні частни коду, відровідно до описаного функціоналу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ДСТУ ISO/IEC/IEEE 12207:2018 (ISO/IEC/IEEE 12207:2017, IDT). Інженерія систем і програмних засобів. Процеси життєвого циклу програмних засобів. На заміну ДСТУ ISO/IEC 12207:2016 ; чинний від 2018-08-15. Вид. офіц. Київ : УкрНДНЦ, 2018. 145 с.

2. ДСТУ 2226-93. Автоматизовані системи. Терміни та визначення. Чинний від 1994-07-01. Вид. офіц. Київ : УкрНДНЦ, 1994. 91 с.

3. On Object Detection Metrics With Worked Example | Towards Data Science. *Towards Data Science*. URL: https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e/ (дата звернення: 15.03.2025).

4. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / S. Ren та ін. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017. Т. 39, № 6. С. 1137–1149. URL: https://doi.org/10.1109/tpami.2016.2577031 (дата звернення: 15.03.2025). –

5. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon та ін. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, м. Las Vegas, NV, USA, 27–30 черв. 2016 р. 2016. URL: https://doi.org/10.1109/cvpr.2016.91 (дата звернення: 15.03.2025).

6. Practical Deep Learning for Cloud and Mobile: Hands-On Computer Vision Projects Using Python, Keras and TensorFlow. O'Reilly Media, Incorporated, 2019. 350 с. (дата звернення: 15.03.2025).

7. Survey on neural architecture search / L. Tang та ін. Journal of Image and Graphics. 2021. Т. 26, № 2. С. 245–264. URL: https://doi.org/10.11834/jig.200202

8. Khan A. I., Al-Habsi S. Machine Learning in Computer Vision. Procedia Computer Science. 2020. Т. 167. С. 1444–1451. URL: https://doi.org/10.1016/j.procs.2020.03.355 (дата звернення: 18.03.2025).

9. Elgendy M. 1 Welcome to computer vision · Deep Learning for Vision Systems. Deep Learning for Vision Systems. URL: https://livebook. manning.com/book/deep-learning-for-vision-systems (дата звернення: 19.03.2025).

10. Szeliski R. Computer Vision. Cham: Springer International Publishing, 2022. URL: https://doi.org/10.1007/978-3-030-34372-9 (дата звернення: 19.03.2025).

11. ДСТУ ISO/IEC 25010:2016 (ISO/IEC 25010:2011, IDT). Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів. – На заміну ДСТУ ISO/IEC 25010:2015 ; чинний від 2018-01-01. – Вид. офіц. – Київ : УкрНДНЦ, 2018. – 32 с. (дата звернення: 19.03.2025).

12. A Survey on Evolutionary Neural Architecture Search / Y. Liu та ін. IEEE Transactions on Neural Networks and Learning Systems. 2021. С. 1–21. URL: https://doi.org/10.1109/tnnls.2021.3100554 (дата звернення: 19.03.2025).

13. Indoor Visual Positioning Aided by CNN-Based Image Retrieval: Training-Free, 3D Modeling-Free / Y. Chen та ін. *Sensors*. 2018. Т. 18, № 8. С. 2692. URL: https://doi.org/10.3390/s18082692 (дата звернення: 19.03.2025).

14. Ultralytics. Home. Home - Ultralytics YOLO Docs. URL: https://docs. ultralytics.com/ (дата звернення: 19.03.2025).

15. About HTTPS - FastAPI. URL: https://fastapi.tiangolo.com/deployment /https/ (дата звернення: 19.03.2025).

16. SQLAlchemy Documentation – SQLAlchemy 2.0 Documentation. URL: https://docs.sqlalchemy.org/en/20 (дата звернення19.03.2025).

17. Cleanup.pictures - Remove objects, people, text and defects from any picture for free. URL: https://cleanup.pictures (дата звернення: 20.03.2025).

18. Free Online AI Photo Editor, Image Generator & Design tool. Pixlr.com - Creative AI suite. URL: https://pixlr.com/ (дата звернення: 20.03.2025).

19. Focal Loss for Dense Object Detection / T.-Y. Lin та ін. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020. Т. 42, № 2. С. 318–327. URL: https://doi.org/10.1109/tpami.2018.2858826 (дата звернення: 03.04.2025).

20. Sapkota R., Ahmed D., Karkee M. Comparing YOLOv8 and Mask R-CNN for instance segmentation in complex orchard environments. Artificial Intelligence in Agriculture. 2024. Т. 13. С. 84–99. URL: https://doi.org/10.1016/j.aiia.2024.07.001 (дата звернення: 05.04.2025).

21. Documentation - MariaDB.org. URL: https://mariadb.org/documentation/ (дата звернення: 09.04.2025).

22. Normal Forms in DBMS - GeeksforGeeks. URL: https://www.geeksforgeeks.org/normal-forms-in-dbms/ (дата звернення: 10.04.2025).

23. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/ (дата звернення: 10.04.2025).

24. Rich feature hierarchies for accurate object detection and semantic segmentation / A. Y. Virasova та ін. Radioengineering. 2021. С. 115–126. URL: https://doi.org/10.18127/j00338486-202109-11 (дата звернення: 17.04.2025).

ДОДАТКИ

ДОДАТОК А

Код роботи з базою даних

database.py

```
from sqlalchemy.ext.asyncio import AsyncEngine, AsyncSession,
create_async_engine
from sqlalchemy.orm import declarative_base
from sqlalchemy.ext.asyncio import async_sessionmaker
import os
DATABASE_URL = "mysql+aiomysql://root:localhost@localhost:3306/diplomna"
engine: AsyncEngine = create_async_engine(DATABASE_URL, echo=True)
AsyncSessionLocal = async_sessionmaker(bind=engine, expire_on_commit=False)
Base = declarative_base()
async def get_async_db():
    async with AsyncSessionLocal() as session:
        yield session
```

model.py

```
from sqlalchemy import Column, Integer, String, Text, Float, ForeignKey, TIMESTAMP
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func
from app.db.database import Base
from sqlalchemy import Boolean
class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    username = Column(String(50), unique=True, nullable=False)
    password_hash = Column(String(255), nullable=False)
                created_at       =       Column(TIMESTAMP,       nullable=True,
server_default=func.current_timestamp())
    email = Column(String(100), unique=True, nullable=False)
    avatar_url = Column(String, nullable=True)
 is_banned = Column(Boolean, default=False)
    images = relationship("Image", back_populates="user")
    logs = relationship("Log", back_populates="user")
class Admin(Base):
    __tablename__ = "admins"
  id = Column(Integer, primary_key=True, index=True)
```

```
username = Column(String(50), unique=True, nullable=False)
password_hash = Column(String(255), nullable=False)
                created_at       =       Column(TIMESTAMP,       nullable=True,
server_default=func.current_timestamp())
    email = Column(String(100), unique=True, nullable=False)
    images = relationship("Image", back_populates="admin")
    logs = relationship("Log", back_populates="admin")
class Image(Base):
    __tablename__ = "images"
    id = Column(Integer, primary_key=True, autoincrement=True)
    filename = Column(String(255), nullable=False)
    file_path = Column(String(255), unique=True)
    uploaded_at = Column(TIMESTAMP, server_default=func.current_timestamp())
    user_id = Column(Integer, ForeignKey("users.id", ondelete="SET NULL"))
    admin_id = Column(Integer, ForeignKey("admins.id", ondelete="SET NULL"))
    user = relationship("User", back_populates="images")
    admin = relationship("Admin", back_populates="images")
    detections = relationship("Detection", back_populates="image")
class Detection(Base):
    __tablename__ = "detections"
    id = Column(Integer, primary_key=True, index=True)
      image_id  =  Column(Integer,  ForeignKey("images.id",  ondelete="CASCADE"),
nullable=False)
    x1 = Column(Integer, nullable=False)
    y1 = Column(Integer, nullable=False)
    x2 = Column(Integer, nullable=False)
    y2 = Column(Integer, nullable=False)
    detected_class = Column(String(100), nullable=False)
    confidence = Column(Float, nullable=False)
    image = relationship("Image", back_populates="detections")
class Log(Base):
    __tablename__ = "logs"
    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    admin_id = Column(Integer, ForeignKey("admins.id"), nullable=False)
```

```
action = Column(String(255), nullable=False)
    level = Column(String(50), nullable=False, default="INFO")
timestamp = Column(TIMESTAMP, server_default=func.current_timestamp())
    details = Column(String(50), nullable=False)
    admin = relationship("Admin", back_populates="logs")
    user = relationship("User", back_populates="logs")
```

init_db

```
import asyncio
from app.db.database import engine, Base
async def init_db():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)
if __name__ == "__main__":
    asyncio.run(init_db())
```

ДОДАТОК Б

Вміст файлів проекту

yolo_service.py

```
import asyncio

import os

import torch

import cv2

import numpy as np

from ultralytics import YOLO

class YOLOService:

    def __init__(self, model_path: str):

        self.model = YOLO(model_path)

        self.output_dir = "static/processed_images/"

        os.makedirs(self.output_dir, exist_ok=True)

        self.class_names = self.model.model.names if hasattr(self.model, 'model') else self.model.names

    async def detect_objects(self, image_path: str):

        results = await asyncio.to_thread(self.model, image_path)

   detections = []

        for result in results:

 for box, cls, conf in zip(result.boxes.xyxy, result.boxes.cls, result.boxes.conf):

                x1, y1, x2, y2 = map(int, box[:4])

                class_id = int(cls.item())

                    detected_class = self.class_names[class_id] if class_id in self.class_names else "unknown"

                detections.append({

                    "x1": x1,

                    "y1": y1,

                    "x2": x2,

                    "y2": y2,

                    "detected_class": detected_class,

                    "confidence": float(conf.item())

                })

            return {"filename": os.path.basename(image_path), "detections": detections}
```

```
async def replace_object(self, image_name: str, target_box: dict, replacement_image_path: str):

        image_path = os.path.join(self.output_dir, image_name)

        image = await asyncio.to_thread(cv2.imread, image_path)

 replacement = await asyncio.to_thread(cv2.imread, replacement_image_path)

         x1, y1, x2, y2 = target_box["x1"], target_box["y1"], target_box["x2"], target_box["y2"]

          replacement_resized = await asyncio.to_thread(cv2.resize, replacement, (x2 - x1, y2 - y1))

        image[y1:y2, x1:x2] = replacement_resized

        output_path = os.path.join(self.output_dir, "modified_" + image_name)

        await asyncio.to_thread(cv2.imwrite, output_path, image)

        return await self.detect_objects(output_path)
```

image_service.py

```
import os
import shutil
from fastapi import UploadFile
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy.future import select
from app.models.model import Image
from app.schemas.image_schema import ImageCreate
import posixpath
UPLOAD_DIR = "static/uploads"
os.makedirs(UPLOAD_DIR, exist_ok=True)
async def save_uploaded_file(file: UploadFile,user_id:int, db: AsyncSession) -> Image:
    file_path = posixpath.join(UPLOAD_DIR, file.filename)
    with open(file_path, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)
    image_data = ImageCreate(filename=file.filename,user_id=user_id, file_path=file_path)
    db_image = Image(**image_data.model_dump())
    db.add(db_image)
    await db.commit()
    await db.refresh(db_image)
    return db_image
async def save_uploaded_file_by_url(file_path:str,user_id:int,db:AsyncSession) -> Image:
```

```
file_name = file_path.split("/").pop()
    image_data = ImageCreate(filename=file_name,
user_id=user_id,file_path=file_path)
    db_image = Image(**image_data.model_dump())
    db.add(db_image)
    await db.commit()
    await db.refresh(db_image)
    return db_image
async def get_image(db: AsyncSession, image_id: int) -> Image | None:
```

Продовження додатку Б

```
result = await db.execute(select(Image).filter(Image.id == image_id))
    return result.scalars().first()
async def get_image_by_filename(db: AsyncSession, filename: str) -> Image | None:
    result = await db.execute(select(Image).filter(Image.filename == filename))
    return result.scalars().first()
```

user_service.py

```
from pydantic import EmailStr
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy.future import select
from passlib.context import CryptContext
from fastapi import HTTPException
from app.models.model import User
from app.schemas.user_schema import UserUpdate, ChangePassword, UserCreate
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
async def hash_password(password: str) -> str:
    return pwd_context.hash(password)
async def create_user(db: AsyncSession, user_data: UserCreate):
    hashed_password = await hash_password(user_data.password)
db_user = User(
        username=user_data.username,
email=user_data.email,
password_hash=hashed_password,
    )
    db.add(db_user)
    await db.commit()
    await db.refresh(db_user)
    return db_user
```

```
async def get_user(db: AsyncSession, user_id: int):
    result = await db.execute(select(User).filter(User.id == user_id))
    return result.scalars().first()
async def get_user_by_email(db: AsyncSession, email: EmailStr) -> User:
    result = await db.execute(select(User).filter(User.email == email))
    return result.scalars().first()
async def update_user_profile(db: AsyncSession, user: User, update_data:
UserUpdate):
    async with db.begin():
        stmt = select(User).filter(User.id == user.id)
        result = await db.execute(stmt)
        user_db = result.scalar()
        if not user_db:
            raise HTTPException(status_code=404, detail="User not found")
        for key, value in update_data.model_dump(exclude_unset=True).items():
            setattr(user_db, key, value)
        await db.commit()
      await db.refresh(user_db)
        return user_db
async def update_user_password(db: AsyncSession, user: User, password_data:
ChangePassword):
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    user.password_hash = pwd_context.hash(password_data.new_password)
    await db.commit()
    await db.refresh(user)
 return {"message": "Password updated successfully"}
async def user_exists(db: AsyncSession, email: EmailStr) -> bool:
    result = await db.execute(select(1).where(User.email == email))
    return result.scalar() is not None
```

yolo_routes.py

```
import asyncio
import os
from fastapi import APIRouter, HTTPException, UploadFile, File, Depends
```

```
from sqlalchemy.ext.asyncio import AsyncSession

from sqlalchemy import text

from app.auth.auth import get_current_user

from app.models.model import User

from app.schemas.yolo_schema import YoloBase, YoloReplace

from app.services.yolo_service import YOLOService

from app.services.image_service import save_uploaded_file, get_image, save_uploaded_file_by_url

from app.services.detection_service import create_detection

from app.db.database import get_async_db

from app.schemas.detection_schema import DetectionCreate

from app.services.log_service import log_action

from app.services.lama_service import remove_object_lama, replace_object_lama

from app.config import YOLO_MODEL_PATH

router = APIRouter()

yolo = YOLOService(YOLO_MODEL_PATH)

async def check_image_exists(image_path: str):

    if not await asyncio.to_thread(os.path.exists, image_path):

        raise HTTPException(status_code=404, detail="Зображення не знайдено")

@router.post("/upload_file")

async def upload_file(file: UploadFile = File(...),current_user:User = Depends(get_current_user), db: AsyncSession = Depends(get_async_db)):

    image = await save_uploaded_file(file,current_user.id, db)

    return {"image_id":image.id}

@router.get("/get_image")

async def get_image_front(id: int, db: AsyncSession = Depends(get_async_db)):

 image = await get_image(db, id)

    return {"image_path":image.file_path}

@router.post("/detect/{id}")

async def detect_objects(id: int, db: AsyncSession = Depends(get_async_db)):

    image = await get_image(db,id)

    try:

        results = await yolo.detect_objects(image.file_path)

    except Exception as e:
```

```
detections = []
    for obj in results["detections"]:
        detected_class = obj.get("detected_class", "unknown")
   print(f"Detected class: {detected_class}")
        detection_data = DetectionCreate(
            image_id=image.id,
            x1=obj["x1"],
 y1=obj["y1"],
  x2=obj["x2"],
            y2=obj["y2"],
            detected_class=detected_class,
            confidence=obj.get("confidence", 0.0)
        )
        detection = await create_detection(db, detection_data)
        detections.append(detection)
    await log_action(db, action="Object detection", user_id=image.user_id,
                                details=f"Detected {len(detections)} objects in
{image.filename}")
    return {
        "image_id": image.id,
        "filename": image.filename,
        "detections": results["detections"],
        "processed_image": f"/static/uploads/{image.filename}"
    }
@router.post("/replace/")
async def replace_object_route(
data: YoloReplace,
    current_user:User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
    image = await get_image(db,data.image_id)
    if image is None:
        raise HTTPException(404, "Image record not found in DB")
    x1, y1, x2, y2 = data.x1, data.y1, data.x2, data.y2
    replacement_path = replacement_file.file_path
```

```
image_path = image.file_path
    try:
     modified_image_path = await asyncio.to_thread(
            replace_object_lama,
            image_path,
            {"x1": x1, "y1": y1, "x2": x2, "y2": y2},
            replacement_path
)
    except Exception as e:
          raise HTTPException(status_code=500, detail=f"Помилка заміни об'єкта: {str(e)}")
                                modified_image = await save_uploaded_file_by_url(modified_image_path,current_user.id,db)
    await log_action(db, action="Object replacement", user_id=None,
                     details=f"Replaced object in {image.filename} at [{x1}, {y1}, {x2}, {y2}]")
    return {
        "message": "Об'єкт замінено",
        "processed_image": modified_image.id
    }
@router.post("/remove/")
async def remove_object_route(
    data: YoloBase,
    current_user:User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
    image = await get_image(db,data.image_id)
    if image is None:
        raise HTTPException(404, "Image record not found in DB")
    x1, y1, x2, y2 = data.x1, data.y1, data.x2, data.y2
    try:
        modified_image_path = await asyncio.to_thread(
            remove_object_lama,
            image.file_path,
            {"x1": x1, "y1": y1, "x2": x2, "y2": y2}
    except Exception as e:
```

```
import traceback
        traceback.print_exc()
raise HTTPException(status_code=500, detail=f"Помилка видалення об'єкта: {e}")
                                modified_image              =              await
save_uploaded_file_by_url(modified_image_path,current_user.id,db)
    delete_query = text("""
        DELETE d
FROM detections d
WHERE d.image_id = :image_id

AND d.x1 = :x1 AND d.y1 = :y1
          AND d.x2 = :x2 AND d.y2 = :y2
    """)
    await db.execute(delete_query, {
        "image_id": image.id,
        "x1": x1, "y1": y1,
        "x2": x2, "y2": y2
    })
    await db.commit()
    await log_action(
        db,
        action="Object removal",
        user_id=None,
        details=f"Removed object in {image.filename} at [{x1}, {y1}, {x2}, {y2}]"
    )
    return {
        "message": "Об'єкт успішно видалено",
        "updated_image_id": modified_image.id
    }
```

user_routes.py

```
from logging import getLogger
from fastapi import APIRouter, Depends, HTTPException
from fastapi import UploadFile, File
import shutil
```

```
from app.db.database import get_async_db
from app.models.model import User
from app.schemas.user_schema import UserResponse, UserUpdate, ChangePassword
from app.auth.auth import get_current_user
from app.services.user_service import update_user_profile, update_user_password
from app.services.log_service import log_info, log_error
router = APIRouter(prefix="/user_profile", tags=["Profile"])
UPLOAD_DIR = "static/avatars/"
@router.post("/me/avatar")
async def upload_avatar(
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
    try:
        filename = f"{current_user.id}_{file.filename}"
        file_path = os.path.join(UPLOAD_DIR, filename)
        with open(file_path, "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)
        current_user.avatar_url = f"/static/avatars/{filename}"
        await db.commit()
        #await log_info(f"User {current_user.email} uploaded new avatar")
        return {"avatar_url": current_user.avatar_url}
    except Exception as e:
        getLogger('root').info(e)
        #await log_error(f"Error uploading avatar for {current_user.email}: {e}")
        raise HTTPException(status_code=400, detail="Avatar upload failed")
@router.get("/get-profile", response_model=UserResponse)
async def get_my_profile(current_user: User = Depends(get_current_user)):
    #await log_info(f"User {current_user.email} accessed their profile")
    return current_user
@router.put("/profile-update", response_model=UserResponse)
async def update_my_profile(
    update_data: UserUpdate,
```

```
db: AsyncSession = Depends(get_async_db)
):
    try:
        updated_user = await update_user_profile(db, current_user, update_data)
await log_info(f"User {current_user.email} updated their profile")
return updated_user
 except Exception as e:
        await log_error(f"Error updating profile for {current_user.email}: {e}")
        raise HTTPException(status_code=400, detail="Profile update failed")
@router.put("/me/change-password")
async def change_password(
    password_data: ChangePassword,
    current_user: User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
    try:
        await update_user_password(db, current_user, password_data)
        await log_info(f"User {current_user.email} changed password")
        return {"detail": "Password updated successfully"}
    except Exception as e:
        await log_error(f"Error changing password for {current_user.email}: {e}")
       raise HTTPException(status_code=400, detail="Password change failed")
@router.delete("/delete-account")
async def delete_my_account(
    current_user: User = Depends(get_current_user),
    db: AsyncSession = Depends(get_async_db)
):
    try:
        await db.delete(current_user)
        await db.commit()
#await log_info(f"User {current_user.email} deleted their account")
        return {"detail": "Account deleted successfully"}
    except Exception as e
```

detection_routes.py

```
import logging
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.ext.asyncio import AsyncSession
from app.db.database import get_async_db
from app.schemas.detection_schema import DetectionCreate, DetectionUpdate, DetectionResponse
from sqlalchemy.future import select
from app.models.model import Detection
from app.services.detection_service import (
    create_detection,
    get_detections,
    get_detection_by_id,
    update_detection,
    delete_detection,
)
router = APIRouter()
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
@router.post("/", response_model=DetectionResponse)
async def create_new_detection(
    detection_data: DetectionCreate,
    db: AsyncSession = Depends(get_async_db)
):
    logger.info(f"Creating detection: {detection_data}")
    detection = await create_detection(db, detection_data)
logger.info(f"Detection created with ID: {detection.id}")
    return detection
@router.get("/", response_model=list[DetectionResponse])
async def get_all_detections(db: AsyncSession = Depends(get_async_db)):
    logger.info("Fetching all detections")
return await get_detections(db)
@router.get("/{detection_id}", response_model=DetectionResponse)
async def get_detection(detection_id: int, db: AsyncSession = Depends(get_async_db)):
```

```
logger.info(f"Fetching detection with ID: {detection_id}")
   detection = await get_detection_by_id(db, detection_id)
    if not detection:
        logger.warning(f"Detection with ID {detection_id} not found")
        raise HTTPException(status_code=404, detail="Detection not found")
    return detection
@router.put("/{detection_id}", response_model=DetectionResponse)
async def update_existing_detection(
    detection_id: int,
    detection_data: DetectionUpdate,
    db: AsyncSession = Depends(get_async_db)
):
    logger.info(f"Updating detection {detection_id} with data: {detection_data}")
    detection = await update_detection(db, detection_id, detection_data)
    if not detection:
        logger.warning(f"Detection with ID {detection_id} not found for update")
        raise HTTPException(status_code=404, detail="Detection not found")
    logger.info(f"Detection {detection_id} updated successfully")
    return detection
@router.delete("/{detection_id}")
async def delete_existing_detection(detection_id: int, db: AsyncSession =
Depends(get_async_db)):
    logger.info(f"Deleting detection with ID: {detection_id}")
    detection = await delete_detection(db, detection_id)
    if not detection:
       logger.warning(f"Detection with ID {detection_id} not found for deletion")
        raise HTTPException(status_code=404, detail="Detection not found")
 logger.info(f"Detection {detection_id} deleted successfully")
    return {"message": "Detection deleted successfully"}
```

auth.py

```
from datetime import datetime, timedelta, timezone
from typing import Optional
from jose import JWTError, jwt
from passlib.context import CryptContext
```

```
from fastapi import Depends, HTTPException, status

from fastapi.security import OAuth2PasswordBearer

from sqlalchemy.ext.asyncio import AsyncSession

from sqlalchemy.future import select

from app.config import settings

from app.db.database import get_async_db

from app.models.model import User, Admin

from app.auth.schema import TokenData

SECRET_KEY = settings.SECRET_KEY

ALGORITHM = "HS256"

ACCESS_TOKEN_EXPIRE_MINUTES = 120

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def verify_password(plain_password: str, hashed_password: str) -> bool:

    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:

    return pwd_context.hash(password)

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")

def create_access_token(data: dict, expires_delta: Optional[timedelta] = None) -
> str:

    to_encode = data.copy()

     expire = datetime.now(timezone.utc) + (expires_delta if expires_delta else
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))

    to_encode.update({"exp": expire})

    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

async def get_current_user(token: str = Depends(oauth2_scheme), db: AsyncSession
= Depends(get_async_db)):

    credentials_exception = HTTPException(

        status_code=status.HTTP_401_UNAUTHORIZED,

        detail="Could not validate credentials",

        headers={"WWW-Authenticate": "Bearer"},

    )

    try:

        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])

username: str = payload.get("sub")

        if username is None:
```

```
token_data = TokenData(username=username)
    except JWTError:
        raise credentials_exception
         result = await db.execute(select(User).filter(User.username == token_data.username))
    user = result.scalars().first()
if user is None:
result = await db.execute(select(Admin).filter(Admin.username == token_data.username))
        user = result.scalars().first()
    if user is None:
        raise credentials_exception
    if isinstance(user, User) and user.is_banned:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Your account has been banned"
        )
    return user
async def get_current_admin(token: str = Depends(oauth2_scheme), db: AsyncSession = Depends(get_async_db)):
    payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
    role: str = payload.get("role")
    if role != "admin":
        raise HTTPException(status_code=403, detail="Admin access required")
        result = await db.execute(select(Admin).filter(Admin.username == payload.get("sub")))
    admin = result.scalars().first()
    if not admin:
        raise HTTPException(status_code=401, detail="Invalid token")
    return admin
async def authenticate_user(db: AsyncSession, email: str, password: str):
  result = await db.execute(select(User).filter(User.email == email))
    user = result.scalars().first()
    role = "user"
    if user is None:
```

```
user = result.scalars().first()
        role = "admin"
    if user and verify_password(password, user.password_hash):
   return {"user": user, "role": role}
    return None
```

mail.py

```
from datetime import datetime, timedelta, timezone
from json import dumps, loads
from pathlib import Path
from fastapi import HTTPException
from fastapi_mail import ConnectionConfig, FastMail, MessageSchema, MessageType
from pydantic import EmailStr
from sqlalchemy.ext.asyncio import AsyncSession
from jose import JWTError, jwt
from app.config import settings
from app.auth.schema import SignUpArgs
from app.services.user_service import get_user_by_email
mail_config =
ConnectionConfig(MAIL_USERNAME='minecraftytor@gmail.com',MAIL_PASSWORD='drqd
lowa voez fzuf',MAIL_FROM='minecraftytor@gmail.com',
                              MAIL_PORT='465',MAIL_SERVER='smtp.gmail.com',
                               MAIL_STARTTLS=False,MAIL_SSL_TLS=True,USE_CREDENT
IALS=True,
                                 TEMPLATE_FOLDER=Path(__file__).parent / 'email-
templates' / 'build')
fm = FastMail(mail_config)
SECRET_KEY = settings.SECRET_KEY
ALGORITHM = "HS256"
async def send_confirmation_email(token:str,signup_args:SignUpArgs):
      subject = f'Image editor service - email confirmation for user
{signup_args.username}'
    link = f'http://localhost:5173/confirm-email/{token}
message =
MessageSchema(subject=subject,recipients=[signup_args.email],template_body={'pro
ject_name':'Image editor
service','username':signup_args.username,'valid_hours':14,'link':link},subtype=M
essageType.html)
```

```
await fm.send_message(message,template_name='signup_confirmation.html')

async def send_reset_password_email(db:AsyncSession,email:EmailStr,token:str):

    user = await get_user_by_email(db,email)

if not user:

       raise HTTPException(status_code=404)

    subject = f'Image editor service - password recovery for user {user.username}'

    link = f'http://localhost:5173/reset-password/{token}'

message                                                                    =
MessageSchema(subject=subject,recipients=[email],template_body={'project_name':'
Image                                                                  editor
service','username':user.username,'valid_hours':14,'link':link},subtype=MessageT
ype.html)

    await fm.send_message(message,template_name='reset_password.html')

def generate_password_reset_token(email: EmailStr) -> str:

  expires = datetime.now(timezone.utc) + timedelta(hours=14)

encoded_jwt   =   jwt.encode({"exp":   expires,   "sub":   email},   SECRET_KEY,
algorithm=ALGORITHM)

  return encoded_jwt

def validate_password_reset_token(token: str) -> EmailStr | None:

  try:

    decoded_token = jwt.decode(token, SECRET_KEY, algorithms=ALGORITHM)

    return str(decoded_token["sub"])

  except JWTError:

    raise HTTPException(status_code=400, detail="Invalid token")

def generate_signup_confirmation_token(signup_args: SignUpArgs) -> str:

    expires = datetime.now(timezone.utc) + timedelta(hours=14)

    encoded_jwt = jwt.encode(

        {"exp": expires, "sub": dumps(signup_args.model_dump())},

        SECRET_KEY, algorithm=ALGORITHM

    )

    return encoded_jwt

def validate_signup_confirmation_token(token: str) -> SignUpArgs:

  try:

    decoded_token = jwt.decode(token, SECRET_KEY, algorithms=ALGORITHM)

    return SignUpArgs(**loads(decoded_token["sub"]))

  except JWTError as e:
```

main.py

```
from fastapi import FastAPI
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.middleware.cors import CORSMiddleware
from starlette.middleware.sessions import SessionMiddleware
from app.routes.yolo_routes import router as yolo_router
from app.routes.admin_routes import router as admin_router
from app.routes.detection_routes import router as detection_router
from app.routes.log_routes import router as log_router
from app.routes.user_routes import router as user_router
from app.auth.routes import router as auth_router
import os
app = FastAPI()
if not os.path.exists("static/uploads"):
  os.makedirs("static/uploads")
app.mount("/static", StaticFiles(directory="static"), name="static")
origins = [
    "http://localhost",
    "http://localhost:5173",
    "http://127.0.0.1:5173",
    "http://127.0.0.1:8000"
]
app.add_middleware(
  CORSMiddleware,
  allow_origins=origins,
  allow_credentials=True,
  allow_methods=["*"],
  allow_headers=["*"],
)
app.add_middleware(SessionMiddleware,
secret_key='config.fastapi.middleware_secret',                same_site="none",
https_only=True)
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: ***Розробка web-додатку редагування зображень на основі декомпозиції об'єктів засобами нейронної мережі Yolo8***

Обсяг пояснювальної записки 59 аркушів:

6 таблиць;

29 рисунків;

2 додатки.

Дата завершення роботи: *12 червня 2025р*.

Підпис студента- _____*Паламаренко Н.Р.*