ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка модулю обробки зображень військової техніки за допомогою YOLOv8: інтеграція Laravel 10 REST API, Filament і Telegram Bot»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артем ПЕРЕГОН
(підпис)

Виконав: здобувач вищої освіти групи ПД-41
Артем ПЕРЕГОН

Керівник: Олег ІЛЬЇН
д.т.н., професор

Рецензент:

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Перегону Артему Дмитровичу

1. Тема кваліфікаційної роботи: «Розробка модулю обробки зображень військової техніки за допомогою YOLOv8: інтеграція Laravel 10 REST API, Filament і Telegram Bot»
керівник кваліфікаційної роботи д.т.н., професор Олег ІЛЬЇН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: опис методів обробки зображень та розпізнавання об'єктів, методів побудови архітектури систем для автоматизованого виявлення об'єктів, технічна документація інструментів розробки веб-застосунків та інтеграції з Telegram Bot, технічна документація щодо використання YOLOv8 для обробки зображень.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз методів та засобів обробки зображень в задачі ідентифікації військової техніки.
2. Проектування модулю обробки зображень в системі збору повідомлень про військову техніку.
3. Розробка модулю обробки зображень в системі збору повідомлень про військову техніку

5. Тестування модулю обробки зображень в системі збору повідомлень про військову техніку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Архітектура додатку.
5. Схема бази даних.
6. Обробка задач через черги в laravel.
7. Процес виявлення об'єктів за допомогою YOLOv8.
8. Архітектура додатку.
9. Екранні форми.
10. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів обробки зображень	07.03-14.03.2024	
3	Аналіз та вибір інструментів для розробки модулю обробки зображень в системі збору повідомлень про військову техніку	15.03-20.03.2024	
4	Проектування модулю обробки зображень	21.03-31.03.2024	
5	Розробка модулю обробки зображень	01.04-23.04.2024	
6	Тестування модулю обробки зображень	24.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Артем ПЕРЕГОН
(підпис)

Керівник
кваліфікаційної роботи _____ Олег ІЛЬЇН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 1 табл., 38 рис., 20 джерел.

Мета роботи – спрощення процесу збору та обробки зображень оператором у системі збору повідомлень про військову техніку за рахунок автоматизації основних етапів.

Об'єкт дослідження – процес збору та обробки зображень у системі збору повідомлень про військову техніку.

Предмет дослідження – програмне забезпечення для автоматизованого збору та обробки зображень у системі збору повідомлень про військову техніку.

Короткий зміст роботи: Проаналізовано сучасні алгоритми та методи обробки зображень для ідентифікації військової техніки. Проведено порівняльний аналіз існуючих рішень та визначено їх сильні і слабкі сторони. Розроблено вимоги до системи обробки зображень "Defense-Scan" та реалізовано її основні функціональні можливості, включаючи інтеграцію з Telegram Bot для збору даних та використання YOLOv8 для аналізу зображень. Програмний модуль розроблено з використанням Laravel 10 REST API та Filament для створення серверної архітектури. Проведено комплексне тестування системи, що підтвердило її функціональність, продуктивність та безпеку. В роботі використано методи структурного аналізу, системного проектування, розробки програмного забезпечення, обробки та передачі зображень. Сферою використання застосунку є моніторинг та ідентифікація військової техніки на зображеннях, отриманих через Telegram Bot. Застосунок також може бути використаний для автоматизації процесів збору, обробки та класифікації зображень у військових дослідженнях, забезпечуючи високий рівень ефективності та мінімізацію помилок, пов'язаних з людським фактором.

КЛЮЧОВІ СЛОВА: РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, LARAVEL, DEFENSE-SCAN, ВІЙСЬКОВА ТЕХНІКА

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface.

YOLOv8 - You Only Look Once, version 8.

REST - Representational State Transfer.

UI - User interface.

AI - Artificial intelligence.

Filament - Адміністративний інтерфейс для Laravel.

Laravel - PHP веб-фреймворк.

JSON - JavaScript Object Notation.

HTTP - Hypertext Transfer Protocol.

DTO - Data Transfer Object.

IDE - Integrated Development Environment.

ВСТУП

Обґрунтування вибору теми та її актуальність: У контексті зростаючої глобалізації та посилення оборонних зусиль, розробка технологій для точного та швидкого збору інформації про військову техніку стає критично важливою. Використання засобів масової інформації, зокрема соціальних мереж і месенджерів як Telegram, для оперативного збору даних набуває нового значення у воєнний час. Технології машинного навчання, такі як YOLOv8, дозволяють автоматизувати процес ідентифікації військової техніки на зображеннях, що значно підвищує швидкість та точність обробки інформації. Враховуючи важливість швидкого реагування на воєнні загрози та необхідність забезпечення національної безпеки, розробка модулю обробки зображень у системі збору повідомлень про військову техніку є актуальною та важливою задачею. Laravel та Filament забезпечують надійну базу для розробки веб-інтерфейсу, що дозволяє ефективно управляти даними та взаємодіяти з користувачами, роблячи технологію доступною для широкого кола операторів.

Об'єкт дослідження – процес збору та обробки зображень у системі збору повідомлень про військову техніку.

Предмет дослідження – програмне забезпечення для автоматизованого збору та обробки зображень у системі збору повідомлень про військову техніку.

Мета роботи – спрощення процесу збору та обробки зображень оператором у системі збору повідомлень про військову техніку за рахунок автоматизації основних етапів.

Методи дослідження – структурний аналіз та системне проектування, методи та патерни розробки програмного забезпечення, методи обробки та передачі зображень, методи тестування програмного забезпечення.

Завдання дослідження:

1. Провести аналіз сучасних аналогічних систем обробки зображень, визначивши їх сильні та слабкі сторони. На основі цього аналізу підготувати порівняльний огляд.

2. Сформулювати функціональні та нефункціональні вимоги до системи, виходячи з аналізу існуючих аналогів і специфікацій проекту.

3. Вивчити потенційні технології, інструменти та бібліотеки, які можуть бути використані для реалізації проекту, зокрема для інтеграції з Telegram Bot та обробки зображень через YOLOv8.

4. Розробити програмний модуль відповідно до раніше визначених вимог і технологічних виборів, з акцентом на інтеграцію з Telegram Bot.

5. Організувати комплексне тестування розробленої системи для перевірки її функціональності, продуктивності та безпеки.

6. Провести апробацію роботи.

Практична значущість дослідження полягає у наданні оборонним структурам та військовим дослідникам засобу для спрощення процесу ідентифікації військової техніки на зображеннях, отриманих від громадян, в тому числі шляхом використання Telegram Bot.

Завдяки автоматизації процесів збору, аналізу та класифікації зображень, система мінімізує людський фактор у помилках та покращує оперативність прийняття рішень. Це важливо для оборонних застосувань, де швидкість та точність прийняття рішень мають критичне значення. Система може використовуватися для моніторингу об'єктів військової техніки у зонах бойових дій чи на окупованих територіях.

Галузь використання – оборонна промисловість.

Робота пройшла апробацію на Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» (24.04.2024, ДУІКТ, м. Київ). За результатами участі опубліковано тези доповідей [1, 2].

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ОБРОБКИ ЗОБРАЖЕНЬ В ЗАДАЧІ ІДЕНТИФІКАЦІЇ ВІЙСЬКОВОЇ ТЕХНІКИ

1.1 Аналіз предметної галузі

У сучасному світі, де інформаційні технології еволюціонують з неймовірною швидкістю, здатність оперативно аналізувати та реагувати на військові загрози є критично важливою. Військові організації та оборонні агенції в усьому світі активно інтегрують технології штучного інтелекту та машинного навчання для підвищення ефективності обробки військових даних. Системи, які можуть автоматизувати збір та аналіз інформації, зокрема через обробку зображень, стають невід'ємною частиною військових стратегій. Проект "Defense-Scan", реалізований на основі Laravel, Filament, Telegram Bot та YOLOv8, відповідає цій потребі, забезпечуючи швидке виявлення та класифікацію військової техніки на зображеннях, отриманих з різних джерел.

Використання Telegram Bot як інтерфейсу для збору зображень дозволяє користувачам легко та швидко передавати інформацію безпосередньо з місця подій, що є особливо важливим у кризових ситуаціях. Інтеграція з YOLOv8, однією з найпотужніших технологій розпізнавання об'єктів на зображеннях, підвищує точність та швидкість аналізу даних. Laravel і Filament забезпечують надійну та гнучку платформу для розробки веб-додатків, що дозволяє ефективно управляти великими обсягами даних і взаємодіяти з користувачами.

Великі технологічні компанії, такі як Google та Facebook, активно розробляють технології збору та аналізу візуальної інформації, підтверджуючи значення та потенціал автоматизованих систем у вирішенні практичних задач. Проект "Defense-Scan" відповідає цим тенденціям, надаючи користувачам потужний інструмент для моніторингу та аналізу військових активів. Це не тільки підвищує оперативність реагування на потенційні загрози, але й дозволяє зберігати високий рівень обізнаності та готовності до дій в будь-який час доби.За допомогою

нього організації можуть надавати інформацію про військові рухи та зміни в обстановці, що дає змогу оперативно реагувати на будь-які зміни.

Також важливим аспектом є потреба у швидкому та точному аналізі візуальних даних, який є критичним у військових додатках. Розробка власного модулю обробки зображень, який використовує алгоритми штучного інтелекту та машинного навчання, як YOLOv8, вимагає глибоких знань у програмуванні та алгоритмах обробки даних. Процес розробки такого модулю може бути складним та часозатратним, включаючи створення, оптимізацію та інтеграцію з іншими системами, такими як Telegram для збору зображень і Laravel для управління даними.

Для вирішення цих задач та забезпечення ефективності процесу було розроблено систему "Defense-Scan". Ця система дозволяє оперативно збирати, аналізувати та класифікувати військову техніку з зображень, наданих користувачами через Telegram. Головна мета даного проекту полягає у тому, щоб надати військовим та аналітичним структурам інструмент для швидкого і точного аналізу візуальної інформації, що покращує рівень інформованості та реакцію на можливі загрози. Функціональність системи спрямована на задоволення базових потреб військових аналітиків, а також на можливість легкої інтеграції з існуючими військовими інформаційними системами.

Система "Defense-Scan" включає модульну архітектуру, яка дозволяє гнучко налаштовувати функціональні можливості залежно від специфічних потреб користувача. Крім того, проект розрахований на масштабування та розгортання в різних середовищах, забезпечуючи надійність та доступність сервісу для широкого кола користувачів. Розробка і підтримка такої системи вимагає високого рівня технічної експертизи, але вона забезпечує значні переваги в контексті національної безпеки та оборони.

Цільовою аудиторією системи "Defense-Scan" є військові аналітики та командири, які використовують передові технології для аналізу обстановки та планування операцій. Вони зацікавлені у швидкому та точному ідентифікуванні військових об'єктів та техніки, що дозволяє їм підтримувати високий рівень

готовності до реагування на зміни в тактичній обстановці. Саме ці користувачі шукають надійні та ефективні інструменти для вдосконалення своїх оперативних можливостей.

Крім того, система може зацікавити науковців і розробників у галузі оборонних технологій, які працюють над створенням і впровадженням інноваційних рішень для обробки і аналізу зображень. Ці фахівці прагнуть інтегрувати сучасні алгоритми машинного навчання та штучного інтелекту в системи моніторингу та розвідки, що забезпечує їм перевагу в розробці передових технологій.

У системи "Defense-Scan" присутні певні технічні обмеження, зокрема обмеження пов'язані з обробкою великого обсягу даних в реальному часі. Однак, завдяки масштабованій архітектурі та використанню ефективних алгоритмів, система забезпечує високу продуктивність та надійність при обробці зображень, що надходять з різних джерел. Потенційні обмеження на обробку даних можуть впливати на швидкість аналізу, але загальна конструкція системи дозволяє ефективно масштабувати ресурси для забезпечення стабільної роботи.

Серед аналогів цього рішення можна виділити такі системи, як Google Search by Image, JPEGsnoop та Foto Forensics, які також використовуються для аналізу зображень. Однак, особливість "Defense-Scan" полягає у спеціалізації на військовому застосуванні та інтеграції з комунікаційними інструментами, такими як Telegram, що забезпечує надзвичайну оперативність та точність у військових застосуваннях.

1.2 Google Search By Image

Google Search by Image - це потужний інструмент для зворотного пошуку зображень, який дозволяє користувачам завантажувати зображення для пошуку схожих або ідентичних зображень в інтернеті. Ця функція використовує передові алгоритми для аналізу візуального контенту, що дозволяє ідентифікувати, відслідковувати та аналізувати зображення з точки зору їх використання та

походження (рис.1.1). Цей сервіс надзвичайно корисний для виявлення авторських прав, дослідження вмісту та отримання інформації про об'єкти на фотографіях.

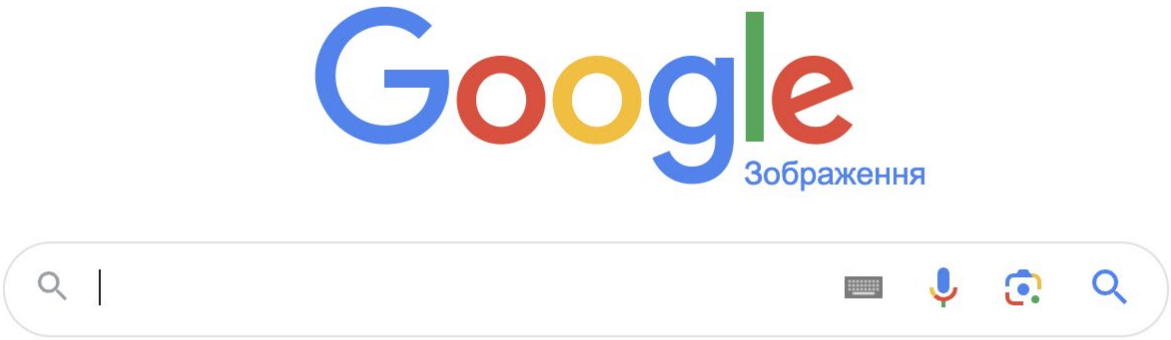


Рис. 1.1 Приклад головної сторінки Google Search By Image

Однією з ключових функцій Google Search by Image є виявлення контексту. Інструмент аналізує зображення, використовуючи передові алгоритми комп'ютерного зору, щоб надавати деталізовану інформацію про можливі локації, об'єкти або теми, пов'язані з ним. Наприклад, завантаживши фотографію відомої пам'ятки, користувач може отримати інформацію про її історію, місцезнаходження та інші пов'язані деталі (рис.1.2). Це робить сервіс надзвичайно корисним для освітніх цілей, подорожей та досліджень.

Доступ до джерел - ще одна важлива складова функціоналу Google Search by Image. Користувачі отримують прямі посилання на сторінки, де зображення було використано, що допомагає в ідентифікації авторських прав або походження зображення. Ця функція є важливою для фотографів, художників та інших творчих професіоналів, оскільки вона дозволяє відстежувати використання їхніх робіт в Інтернеті. Вона також корисна для журналістів та дослідників, які можуть перевіряти джерела інформації та підтверджувати її достовірність.

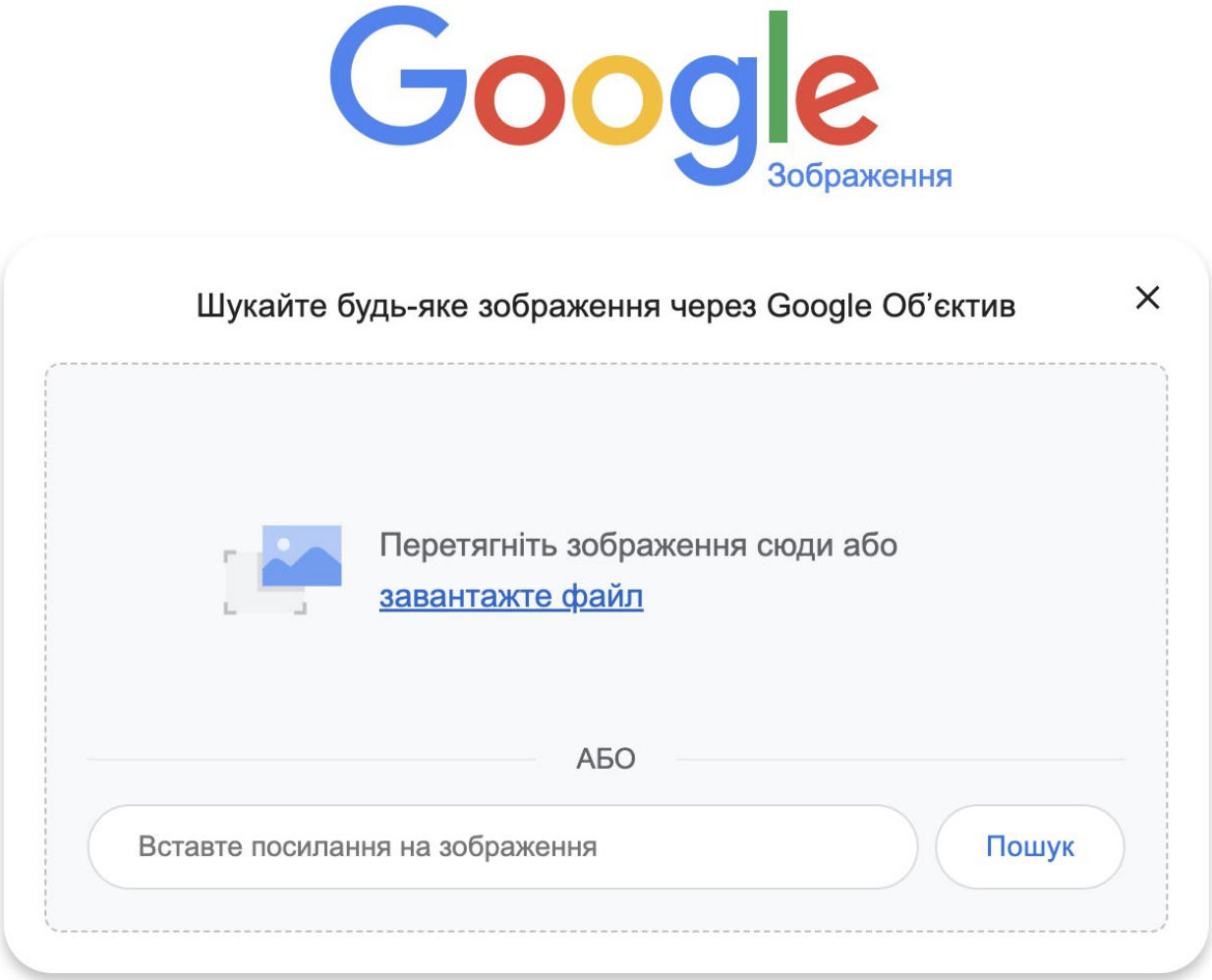


Рис. 1.2 Модальне вікно завантаження зображення для пошуку

Наприклад, якщо використовується зображення танка, інструмент може виявити його походження або інші випадки його використання на різних веб-сторінках. Результати відображаються поруч із посиланнями на веб-сайти, де зображення було опубліковано, включно з ресурсами як Вікіпедія, новинні сайти, блоги чи інші онлайн-ресурси, описаний функціонал зображено на рисунку 1.3.

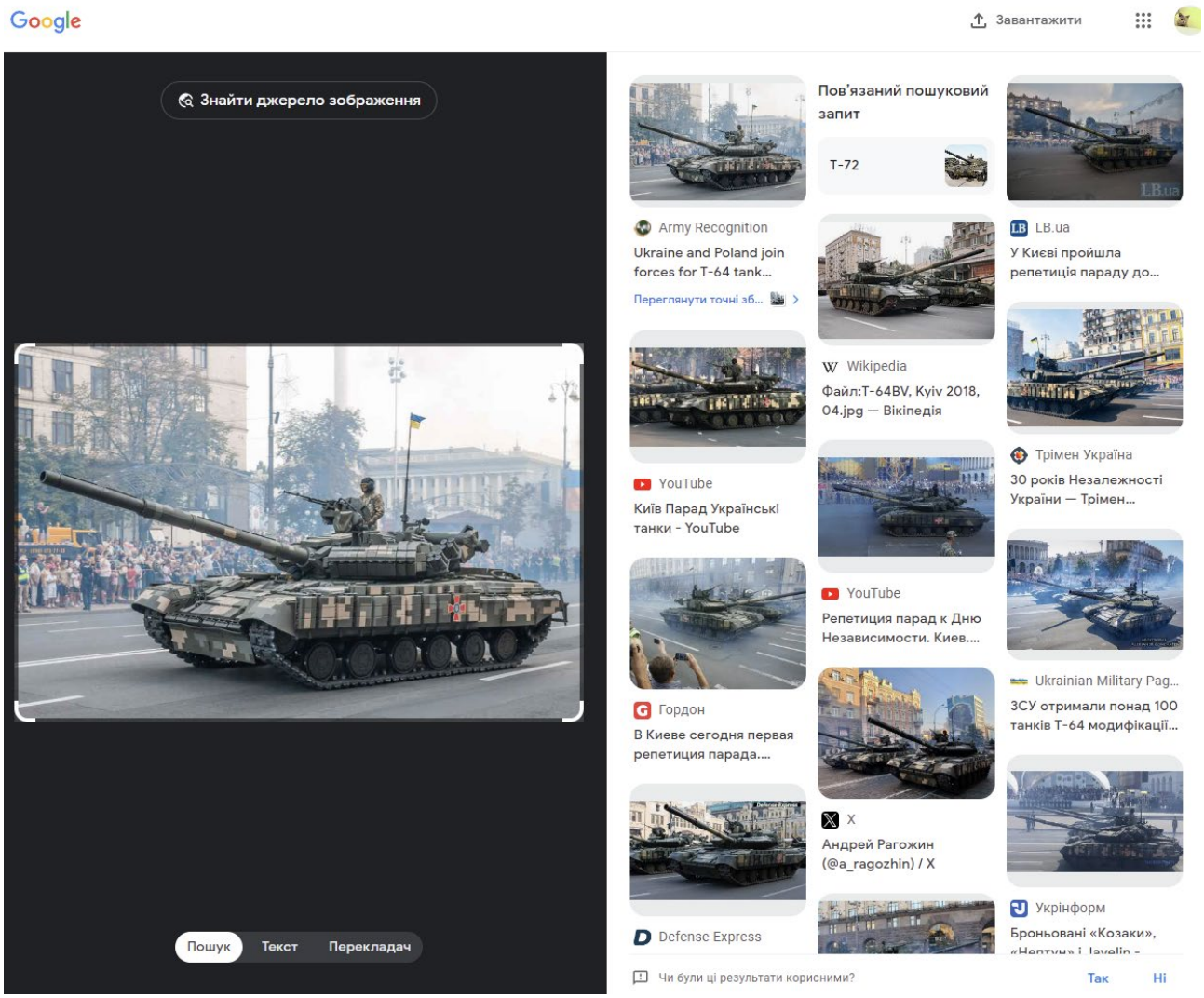


Рис. 1.3 Сторінка результатів пошуку Google Search by Image

Переваги Google Search By Image:

–Велика база даних. Google володіє однією з найбільших і найбільш обширних баз даних зображень у світі, що дозволяє користувачам отримувати різноманітні та вичерпні результати при зворотному пошуку зображень. Ця база даних є надзвичайно цінною для виявлення візуального контенту, що використовується в різних контекстах по всьому світу, надаючи користувачам широкі можливості для дослідження та верифікації.

–Точність. Завдяки розширеному використанню алгоритмів штучного інтелекту та машинного навчання, Google Search by Image забезпечує високу точність у визначенні схожості та відповідності зображень. Ці технології дозволяють не тільки знаходити зображення, що зовні схожі, але й аналізувати

візуальні елементи, що допомагає уточнювати результати пошуку за допомогою контекстуального розуміння.

−Доступність. Сервіс від Google доступний безкоштовно і може бути використаний на будь-якому пристрої з доступом до інтернету. Ця універсальність робить зворотній пошук зображень доступним для широкої аудиторії, включаючи студентів, дослідників, журналістів та професіоналів у різних областях.

Недоліки Google Search By Image:

−Конфіденційність. Завантаження зображень на сервери Google може нести ризики для конфіденційності користувачів, оскільки є потенціал для використання цих даних у комерційних або інших цілях без відома користувача. Це ставить питання про захист особистих даних і приватності користувачів.

−Обмеження авторського права. Пошук зображень може виявити контент, що охороняється авторським правом, і його неправомірне використання може призвести до юридичних наслідків. Користувачам необхідно бути особливо обережними при використанні знайдених через сервіс зображень, щоб уникнути порушення авторських прав.

−Залежність від алгоритмів. Результати зворотного пошуку зображень можуть значно варіюватися, залежно від алгоритмів, які Google використовує у даному момент. Алгоритми можуть змінюватися, що може впливати на релевантність та актуальність результатів пошуку, що іноді може привести до невідповідностей або неправильного тлумачення даних

Цей інструмент є критично важливим для дослідників, журналістів, юристів, і всіх, хто працює в сферах, де точне розуміння і законне використання візуального контенту має особливе значення. Завдяки можливостям швидкого пошуку та точного визначення контексту, Google Search by Image сприяє ефективному знаходженню, аналізу та управлінню візуальним контентом в інтернеті.

1.3 JPEGsnoop

JPEGsnoop - це спеціалізоване програмне забезпечення для аналізу метаданих файлів, що встановлюється на комп'ютери з операційною системою Windows. Цей інструмент дозволяє глибокий аналіз не лише JPEG зображень, але й інших форматів файлів, включно з AVI, DNG, PDF, і THM. Користувачі, які зацікавлені в детальному дослідженні технічних характеристик зображення, можуть використовувати JPEGsnoop для вивчення структури файлу, виявлення можливих змін та аналізу історії редагувань. Інтерфейс програми організовано таким чином, що зліва розміщене завантажене зображення, тоді як результати аналізу метаданих відображаються по центру, з вкладками для швидкого доступу до різних типів даних, таких як EXIF-дані, цвітові гістограми, та інформація про компресію зображення. Головку сторінку JPEGsnoop показано на рисунку 1.4, екран JPEGsnoop з відображенням метаданих представлено на рис.1.5.

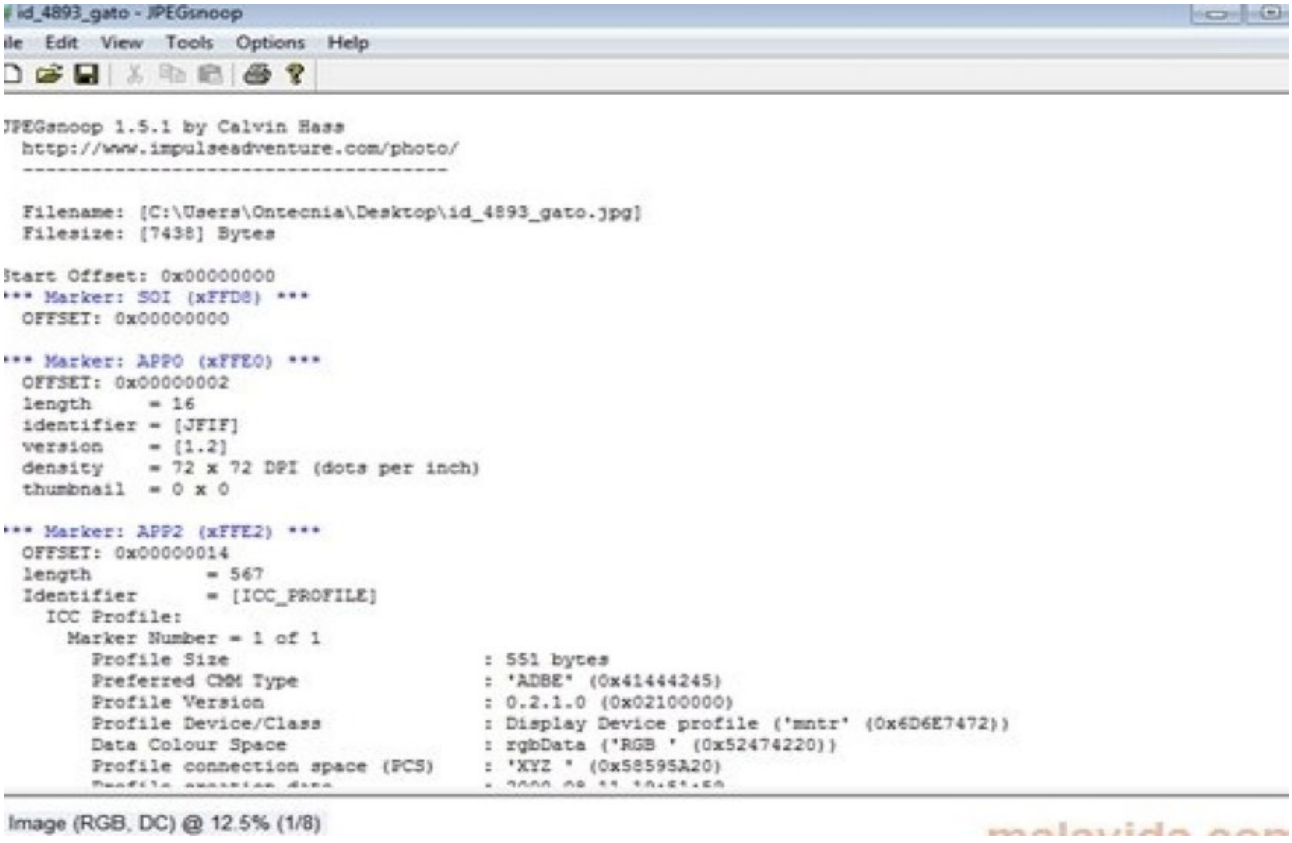


Рис. 1.4 Головна сторінка JPEGsnoop

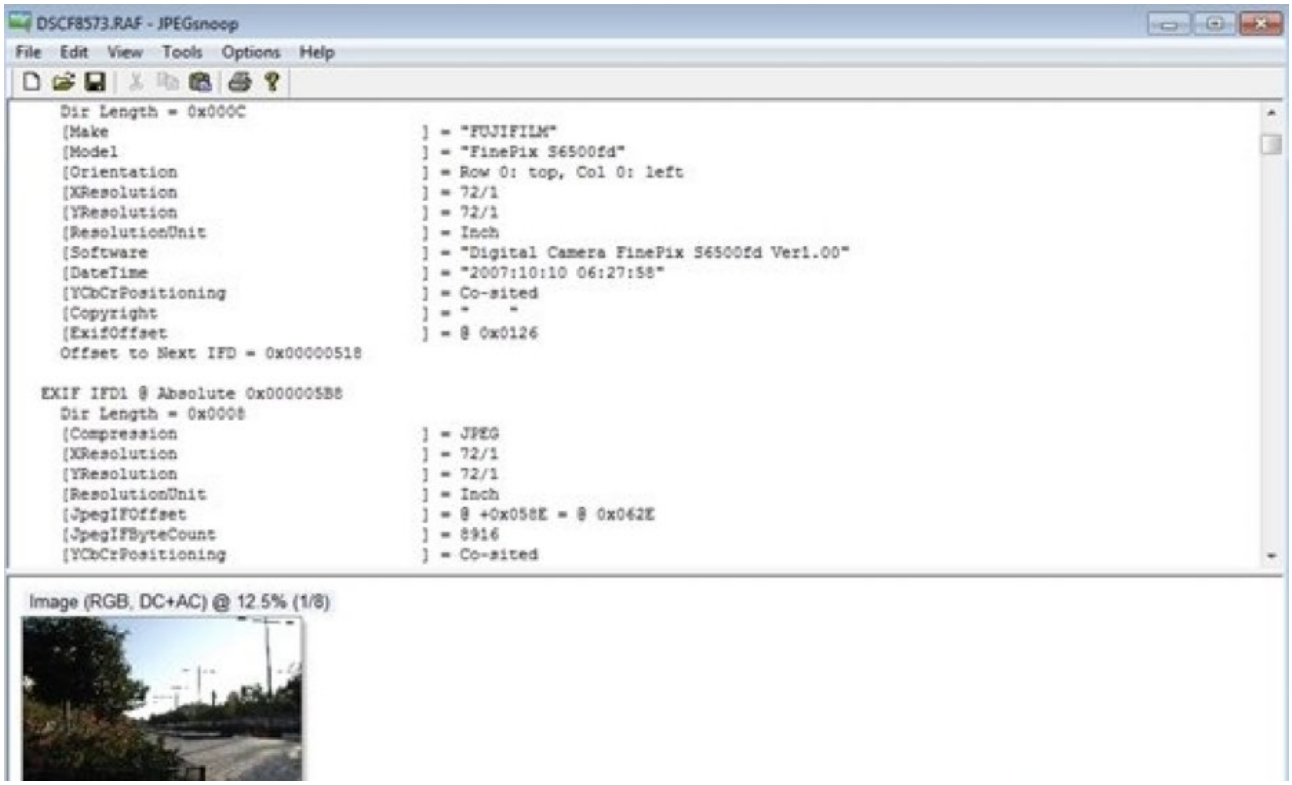


Рис. 1.5 Екран JPEGsnoop з відображенням метаданих

Переваги JPEGsnoop:

– Детальний аналіз метаданих EXIF. JPEGsnoop надає винятково детальний огляд метаданих, включаючи деталі про камеру, налаштування об'єктива, дату та час зйомки, параметри експозиції, і використані фільтри або редагування. Ця інформація може бути надзвичайно корисною для фотографів, які аналізують свої роботи, а також для юридичних експертів та інших спеціалістів, яким необхідно встановити справжність зображення.

– Виявлення редагувань. Програма автоматично аналізує, чи зазнало зображення цифрової обробки або модифікації, що дозволяє користувачам визначити, чи було зображення альтеровано після первісної зйомки. Ця функція є незамінною для роботи з доказами в правових справах або для перевірки автентичності історичних фотографій.

– Підтримка численних форматів файлів. JPEGsnoop підтримує аналіз широкого спектру форматів зображень та мультимедійних файлів, роблячи його ідеальним інструментом для професіоналів, які працюють з різноманітними медіа-файлами.

– Експорт даних. Користувачі можуть легко експортувати зібрані метадані до текстових файлів для подальшого аналізу або архівування, що дозволяє інтегрувати дані в дослідницькі звіти або презентації.

Недоліки JPEGsnoop:

– Обмежена доступність. JPEGsnoop доступний тільки для користувачів Windows, що виключає значну частину користувачів, які працюють на інших операційних системах.

– Технічні вимоги. Для ефективного використання програми користувачам потрібно мати базові знання про метадані та обробку зображень, що може становити складність для непрофесіоналів.

– Користувацький інтерфейс. Інтерфейс програми може здатися складним та застарілим, особливо для користувачів, звиклих до сучасних веб-додатків та мобільних додатків.

JPEGsnoop є потужним інструментом для детального аналізу зображень, який задовольняє потреби професіоналів, які шукають глибоке розуміння технічних аспектів фотографій. Його можливості детального аналізу та підтримка численних форматів роблять його незамінним ресурсом для фотографів, аналітиків, правоохоронних органів і дослідників. Однак обмеження за платформою та складність інтерфейсу можуть вимагати додаткових зусиль для освоєння програми, особливо для нових користувачів. Враховуючи ці фактори, JPEGsnoop залишається цінним інструментом у руках досвідчених користувачів, які можуть повністю реалізувати його потенціал.

1.4 Foto Forensics

Foto Forensics є веб-платформою, призначеною для детального аналізу цифрових зображень, яка забезпечує користувачам інструменти для виявлення змін і перевірки автентичності фотографій. Використовуючи методи аналізу помилок рівня (ELA), сервіс здатен виявляти ділянки на зображенні, які могли бути модифіковані або ретушовані. Цей веб-сайт надає зручний інтерфейс для

завантаження зображень або введення URL, дозволяючи користувачам швидко отримати доступ до різних інструментів аналізу і детальних звітів про зміни в зображеннях. Приклад головної сторінки Foto Forensics зображено на рисунку 1.6.

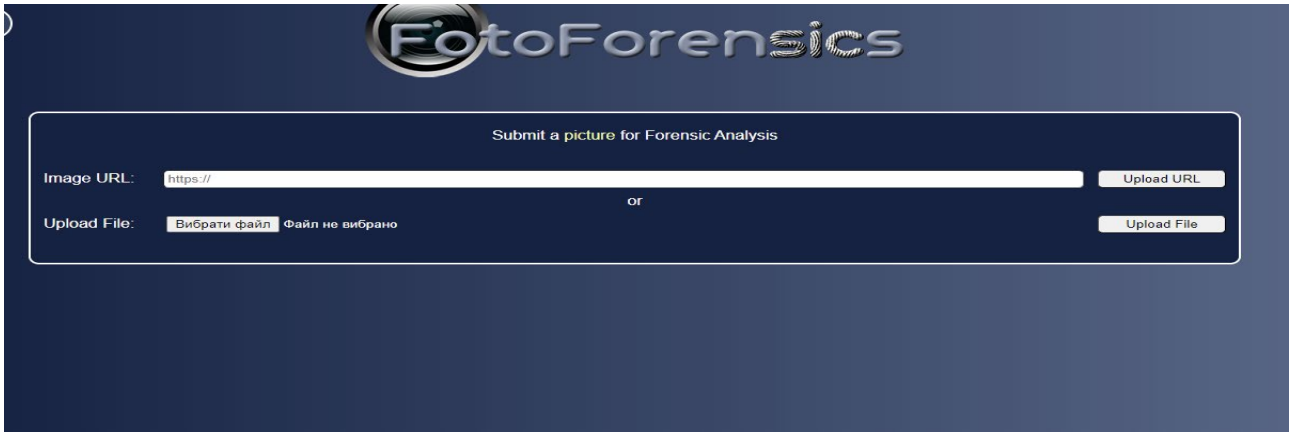


Рис. 1.6 Приклад головної сторінки Foto Forensics

Приклади сторінок аналізу зображень за допомогою сервісу Foto Forensics наведені на рисунках 1.7 – 1.9. На цих рисунках зображені різні інтерфейси та результати аналізу зображень, що надаються сервісом.

На рисунку 1.7 представлена сторінка аналізу зображення Digest Foto Forensics, яка показує загальний огляд і деталі аналізу завантаженого зображення. Сторінка аналізу зображення ELA Foto Forensics, яка надає інформацію про рівні помилок в зображенні, що дозволяє виявити можливі маніпуляції (рис.1.8). На рисунку 1.9 зображена сторінка аналізу зображення Metadata Foto Forensics, яка показує метадані зображення, такі як інформація про камеру, час створення та інші технічні деталі.

Рис. 1.7 Сторінка аналізу зображення Digest Foto Forensics

Рис. 1.8 Сторінка аналізу зображення ELA Foto Forensics

Рис. 1.9 Сторінка аналізу зображення Metadata Foto Forensics

Переваги Foto Forensics:

−Виявлення змін у зображенні. Метод ELA, який використовує Foto Forensics, дозволяє ідентифікувати області на фотографії, які могли бути змінені після її створення, що є надзвичайно корисним для фотожурналістів, дослідників і правоохоронних органів.

−Доступ до EXIF-даних. Користувачі можуть переглядати детальну інформацію про параметри зйомки та інші технічні аспекти зображень, що допомагає визначити автентичність та оригінальність фотографій.

−Широка доступність. Запущений у 2012 році як некомерційний проект, Foto Forensics пропонує свої послуги широкому колу користувачів, роблячи детальний аналіз зображень доступнішим.

−Підтримка різних форматів. Фото Forensics підтримує обробку зображень великих розмірів (до 10,000×10,000 пікселів) і файлів до 8 МБ, у тому числі формати JPEG, PDF, RAW, AVI, MOV і DNG, роблячи його універсальним інструментом для аналізу.

Недоліки Foto Forensics:

−Обмеження на малі зображення. Малі зображення (менше ніж 100×100 пікселів) не можуть бути ефективно аналізовані, що може стати проблемою при роботі з дуже малими або низькороздільними знімками.

−Технічні знання для інтерпретації. Необхідність володіння знаннями для правильної інтерпретації результатів ELA може ускладнити використання платформи для недосвідчених користувачів.

−Залежність від інтернет-з'єднання. Як веб-базована служба, Foto Forensics вимагає стабільного інтернет-з'єднання для функціонування, що може обмежити його використання в регіонах з обмеженим доступом до мережі.

−Комерційна версія для масового використання. Для користувачів, які потребують частого аналізу великої кількості зображень, може бути необхідно підписатися на комерційну версію, що вимагає додаткових витрат.

Foto Forensics став важливим ресурсом у галузі аналізу зображень, пропонуючи користувачам потужні інструменти для перевірки автентичності фотографій. Його здатність виявляти редагування та надавати детальні метадані робить його цінним для професіоналів у сферах, де точність зображень має вирішальне значення. Однак, потреба у технічних знаннях і залежність від інтернет-з'єднання можуть обмежити його універсальність для деяких користувачів.

1.5 Порівняльна таблиця

Для обґрунтування вибору програмного забезпечення для обробки зображень у системі збору повідомлень про військову техніку, було проведено порівняння кількох існуючих рішень. У таблиці 1.1 наведено порівняльну характеристику таких інструментів, як Google Search by Image, JPEGsnoop, Foto Forensics та Defense-Scan.

Таблиця 1.1

Порівняльна таблиця аналогів

Показник	Google Search by Image	JPEGsnoop	Foto Forensics	Defense-Scan
Платформа	Windows, MacOs, Web Android, iOS	Windows	Windows, MacOs, Web Android, iOS	Windows, MacOs, Web Android, iOS
Цільовий об'єкт розпізнавання	Загальне розпізнавання об'єктів	Деталізований аналіз зображень	Зміни в зображеннях	Військова техніка
Підтримка різних форматів і розмірів	JPG, PNG, GIF	JPG, TIFF, RAW	JPEG, PNG	JPG, PNG, GIF, TIFF, RAW
Конфіденційність	Ризики використання даних	Локальний аналіз	Дані не зберігаються	Високий рівень захисту даних
Інтеграція з іншими системами	Обмежена інтеграція	Обмежена інтеграція	Обмежена інтеграція	Інтеграція через REST API
Автоматизація процесів	Ручне введення даних	Вимагає ручного втручання	Мінімальна автоматизація	Повна автоматизація

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Загальні вимоги до розробки

В роботі розробляється модуль обробки зображень для системи збору повідомлень про військову техніку на платформі Laravel, використовуючи мову програмування PHP версії 8.1. Система має забезпечувати прийом, аналіз і класифікацію зображень військової техніки, отриманих через Telegram Bot, з подальшою ідентифікацією об'єктів за допомогою технології YOLOv8.

Основний функціонал поділяється на кілька підсистем: прийом зображень через інтегрований Telegram Bot, їх збереження та класифікація з використанням нейронної мережі YOLOv8 [7,8], та відображення результатів в адміністративній панелі на Filament.

Для управління даними використовується база даних PostgreSQL, яка інтегрується з Laravel через ORM Eloquent, забезпечуючи ефективне зберігання та швидкий доступ до зображень і результатів їх аналізу. Враховуючи потенційно високу навантаженість системи, для оптимізації обробки вхідних даних та кешування використовується Redis.

Проектування системи має бути здійснене з дотриманням принципів чистої архітектури та шаблонів проектування, що дозволить забезпечити гнучкість та масштабованість проекту. Всі компоненти системи повинні бути упаковані в Docker контейнери для забезпечення легкості розгортання та портабельності рішення.

Розробка повинна виконуватися з використанням інтегрованого середовища розробки JetBrains PhpStorm, що оптимізовано для роботи з PHP та фреймворком Laravel, а також забезпечує потужні інструменти для роботи з базами даних та відладки коду.

Це технічне завдання ставить за мету створення надійної та ефективної системи для моніторингу військової техніки, здатної обробляти великі об'єми даних в реальному часі з мінімальними затримками, забезпечуючи високий рівень точності та доступності аналітичних даних для кінцевих користувачів.

2.2 Моделювання вимог до програмного забезпечення

Діаграма варіантів використання (Use Case Diagram) є важливим елементом у проектуванні програмного забезпечення, оскільки вона демонструє взаємодію користувачів із системою. Вона відображає, які дії можуть виконувати користувачі, а також основні функціональні можливості системи. У контексті розробки модулю обробки зображень у системі збору повідомлень про військову техніку на основі Laravel, Filament, Telegram Bot та YOLOv8, діаграма використання допомагає візуалізувати основні сценарії використання системи.

На діаграмі варіантів використання (рис. 2.1) представлено два основних актори: Користувач та Адміністратор. Користувач може почати чат з Телеграм ботом, заповнити анкету про себе та надіслати фото. Адміністратор має більше функцій, таких як вхід в систему, вихід з системи, перегляд зображень від користувачів, перегляд користувачів системи, редагування та видалення користувачів системи. Додатково, діаграма показує включення таких сервісів як аутентифікація, розпізнавання зображень та отримання інформації про користувачів.

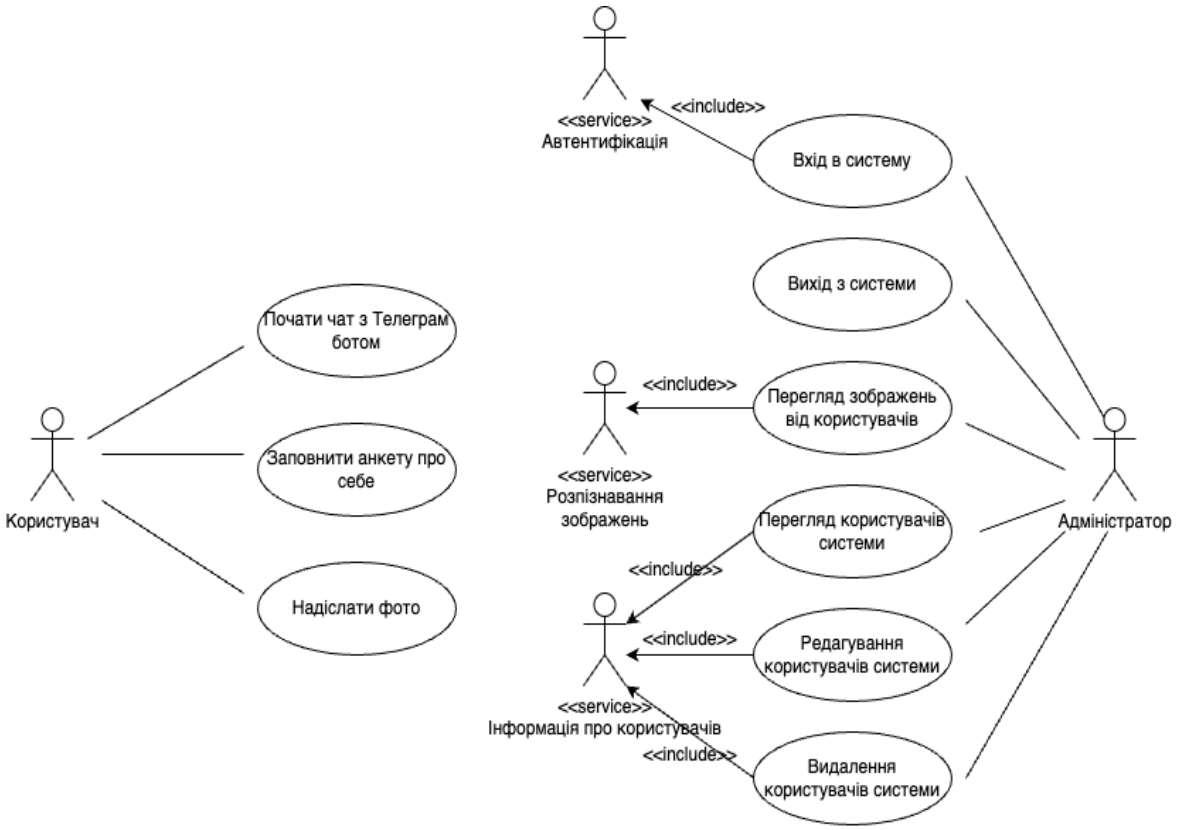


Рис. 2.1 Діаграма варіантів використання

Діаграма використання надає загальне уявлення про функціональні можливості системи та взаємодію між користувачами і системою. Це допомагає визначити основні вимоги до системи та забезпечити чітке розуміння функціоналу, що має бути реалізований. Такий підхід сприяє створенню більш ефективного та інтуїтивно зрозумілого інтерфейсу для кінцевих користувачів.

Функціональні вимоги визначають конкретний набір функцій, які система повинна підтримувати для забезпечення необхідної роботи як для користувачів, так і для адміністраторів. Нефункціональні вимоги, у свою чергу, охоплюють характеристики та обмеження системи, які мають бути враховані для забезпечення її ефективності та надійності.

Функціональні вимоги

Користувач:

- Авторизація через Telegram.
- Заповнення анкети з інформацією про себе.
- Надсилання зображень через Telegram.

Адміністратор:

- Управління користувачами.
- Пошук користувачів
- Перегляд надісланих зображень.
- Пошук зображень

Нефункціональні вимоги

- Мінімальний розмір зображення не менше 100x100 пікселів.
- Рекомендований розмір зображення становить 1920x1080 пікселів.
- Максимальний розмір зображення 3840x2160 пікселів.
- Сервери, на яких розгорнуто систему, повинні мати щонайменше 16 ГБ оперативної пам'яті та процесори не менше Intel i7 або еквівалентні AMD.
- Використання реляційної бази даних PostgreSQL
- Веб-додаток повинен бути сумісний з останніми версіями основних браузерів (Chrome, Firefox, Safari та Edge).

2.3 Моделювання архітектури системи

Моделювання архітектури системи відіграє важливу роль у процесі розробки програмного забезпечення, оскільки воно дозволяє візуально представити всі компоненти системи та їх взаємодію. Це допомагає забезпечити чітке розуміння структури та функціонування додатку, що є критичним для успішної реалізації проекту. Основною метою моделювання є створення діаграм, які ілюструють всі елементи системи та їх взаємозв'язки, полегшуючи планування, розробку та підтримку програмного забезпечення.

Архітектура системи збору повідомлень про військову техніку включає кілька ключових компонентів: Telegram Bot, Admin Client, вебсервер (на базі PHP), Redis, модуль обробки зображень, YOLOv8 та базу даних. Взаємодія між цими компонентами забезпечує ефективну обробку зображень, надісланий користувачами, ідентифікацію військової техніки та управління даними.

Telegram Bot: Користувачі надсилають зображення через Telegram Bot, який працює з API Telegram для отримання повідомлень та зображень.

Admin Client: Адміністративний клієнт дозволяє адміністраторам системи керувати користувачами та переглядати надіслані зображення. Він використовує технологію Livewire для інтерактивної взаємодії з вебсервером.

Вебсервер (PHP): Основний сервер додатку, реалізований на базі PHP, отримує запити від Telegram Bot та Admin Client, обробляє їх і взаємодіє з іншими компонентами системи. Вебсервер також створює черги для обробки зображень.

Redis: Використовується для кешування даних та управління чергами задач. Коли користувач надсилає зображення, воно потрапляє в чергу Redis для подальшої обробки.

Модуль обробки зображень: Основний компонент, що відповідає за обробку зображень та ідентифікацію військової техніки. Виконує завдання на основі запланованих чергових задач, отриманих з Redis.

YOLOv8: Використовується для ідентифікації об'єктів на зображеннях. Модель YOLOv8 обробляє зображення, надіслані модулем обробки зображень, і повертає результати.

База даних: Використовується для зберігання інформації про користувачів, зображення та результати обробки. Вебсервер здійснює запити до бази даних для вставки, оновлення та отримання необхідних даних.

Рис. 2.2 ілюструє загальну архітектуру системи збору повідомлень про військову техніку.

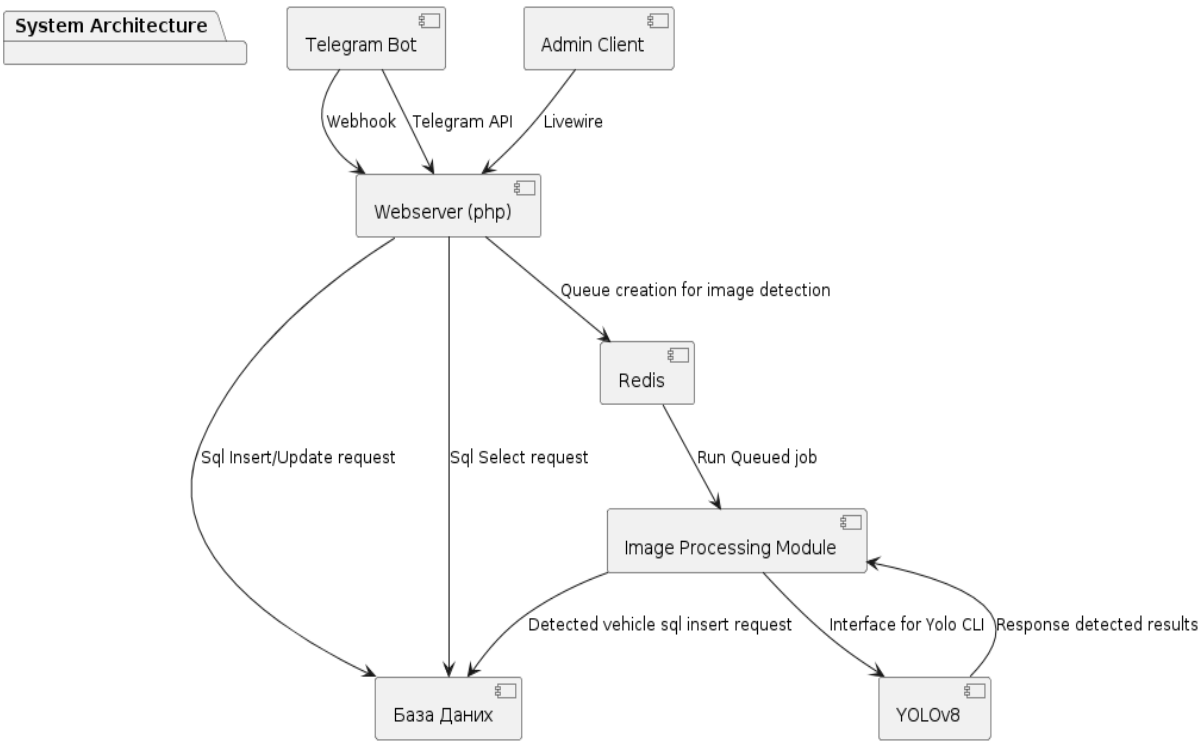


Рис. 2.2 Архітектура додатку “Defense Scan”

2.4 Діаграма класів

Діаграма класів є важливим інструментом у розробці програмного забезпечення, який дозволяє візуалізувати структуру системи, визначити основні класи, їх атрибути, методи, а також зв'язки між класами. У контексті розробки модулю обробки зображень у системі збору повідомлень про військову техніку на основі Laravel, Filament, Telegram Bot та YOLOv8, діаграма класів допомагає структурувати програмний код, забезпечуючи чітке розуміння ролей і взаємодій кожного компонента системи.

На діаграмі класів на рис. 2.3 представлені такі основні класи, як User, Provider, Request, а також інтерфейси та трейти для реалізації додаткових функцій, таких як робота з медіафайлами та запитами. Вона демонструє, як користувачі (Users) та провайдери (Providers) взаємодіють з запитами (Requests) та як система обробляє медіафайли, використовуючи морфні зв'язки.

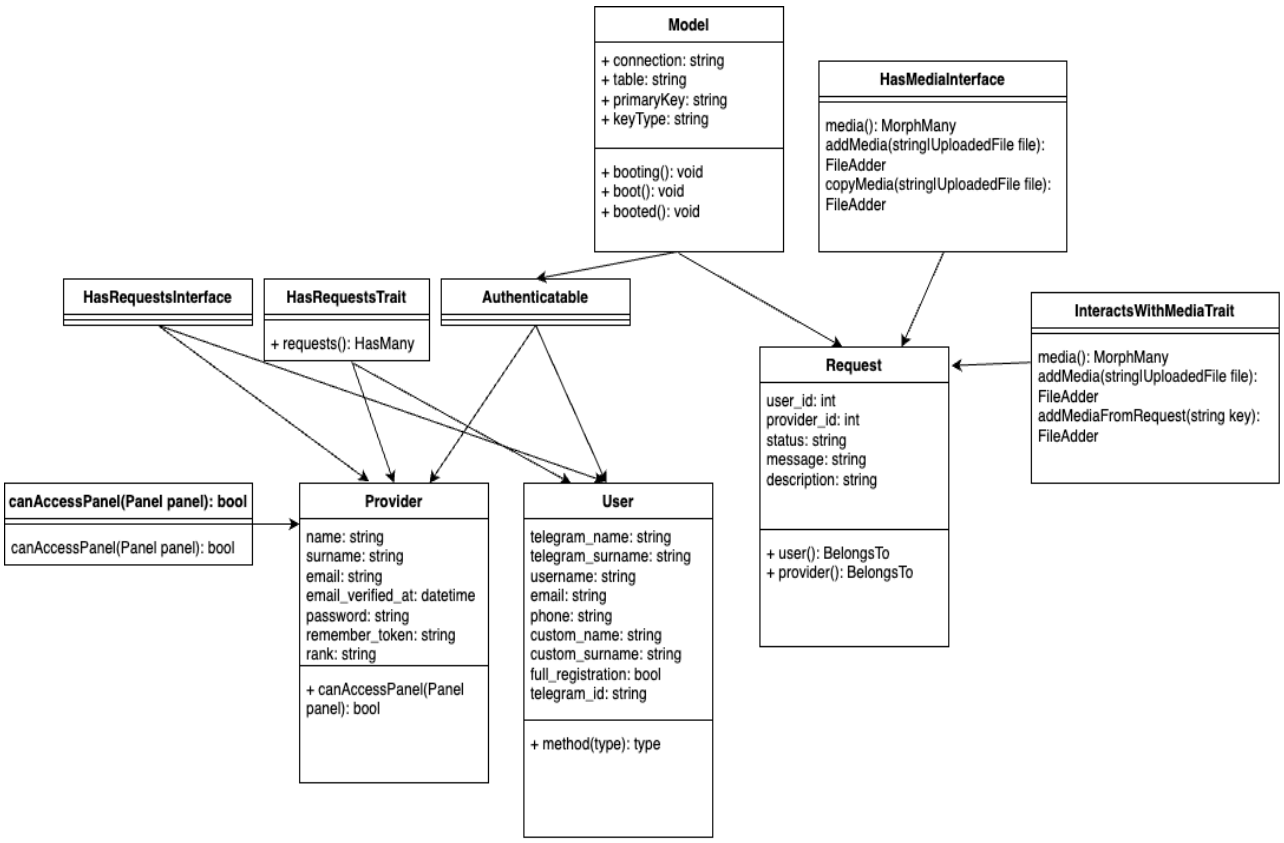


Рис. 2.3 Діаграма класів

Діаграма класів є критичним елементом в проектуванні програмного забезпечення для системи збору повідомлень про військову техніку. Вона дозволяє чітко визначити структуру програми, взаємодію між різними компонентами та сприяє створенню зрозумілого та підтримуваного коду. Це забезпечує ефективну розробку та масштабування системи у майбутньому.

2.5 Файлова архітектура

Файлова архітектура проєкту є важливим аспектом у розробці програмного забезпечення, оскільки вона визначає структуру каталогів і файлів, що використовуються в проєкті. Така структура допомагає організувати код таким чином, щоб забезпечити легкість у підтримці, масштабуванні та зрозумілості для розробників. У контексті розробки модулю обробки зображень у системі збору повідомлень про військову техніку на основі Laravel, Filament, Telegram Bot та

YOLOv8, файлова архітектура відображає, як різні компоненти системи розподілені та взаємодіють між собою.

Файлова архітектура додатку представлена на рисунку 2.4.

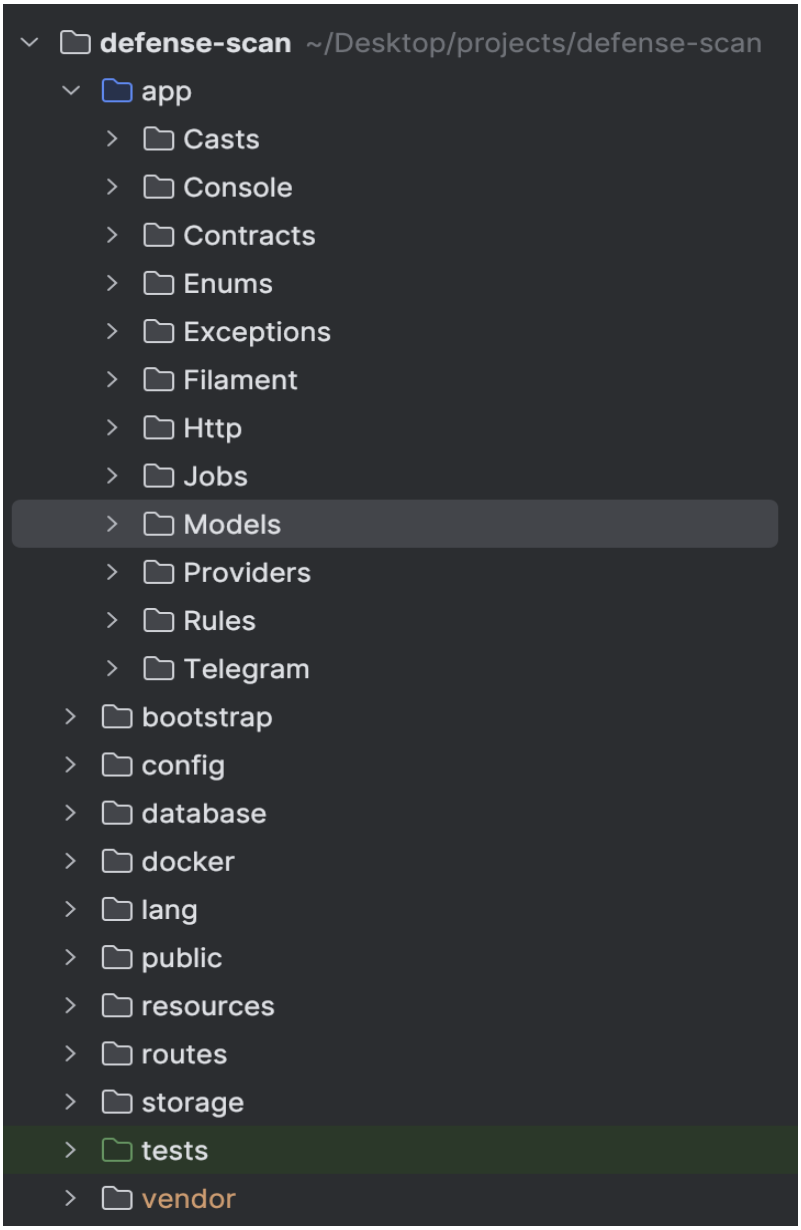


Рис. 2.4 Файлова архітектура додатку

Файлова архітектура проєкту є критичним елементом для успішної розробки програмного забезпечення. Вона забезпечує структуровану організацію коду, що полегшує його підтримку та розвиток.

3 РОЗРОБКА ТА ОПИС ФУНКЦІОНУВАННЯ МОДУЛЮ ОБРОБКИ ЗОБРАЖЕНЬ В СИСТЕМІ ЗБОРУ ПОВІДОМЛЕНЬ ПРО ВІЙСЬКОВУ ТЕХНІКУ

3.1 Обґрунтування вибору засобів розробки

Laravel – це популярний PHP фреймворк з відкритим кодом, який забезпечує розробників потужним інструментарієм для створення надійних та високонавантажених веб-додатків. Завдяки своїй модульній архітектурі та великій кількості вбудованих функцій, Laravel дозволяє швидко та ефективно розробляти серверні додатки. Використання цього фреймворку у проекті забезпечує надійну платформу для обробки вхідних зображень від користувачів через Telegram Bot на основі NutGram. Laravel також забезпечує зручну інтеграцію з базами даних, системами кешування та чергами задач, що є критичними для проектів з високими вимогами до продуктивності та надійності.[3,6]

Основні переваги Laravel:

Модульність та розширюваність. Laravel дозволяє легко розширювати функціональність додатка за допомогою пакетів. Це робить його ідеальним вибором для проектів різної складності, де потрібно швидко додавати нові можливості.

Інтеграція з базами даних. Laravel підтримує різні типи баз даних, включаючи MySQL, PostgreSQL, SQLite та SQL Server. В нашому проекті було обрано PostgreSQL, що забезпечило надійність та продуктивність зберігання даних.

Черги задач та обробка подій. Використання Redis для керування чергами задач дозволяє ефективно розподіляти навантаження між процесами, що особливо важливо для обробки великої кількості вхідних зображень.

Безпека та аутентифікація. Laravel надає вбудовані засоби для забезпечення безпеки додатків, включаючи аутентифікацію користувачів, захист від CSRF та XSS атак, шифрування даних та інше.

Екосистема. Laravel має велику екоистему для розробки, тесутвання та підтримки додатку на серверах

Екосистема. Laravel має велику екосистему для розробки, тестування та підтримки додатку на серверах. Це включає в себе інструменти для розгортання, як-от Laravel Forge та Envoyer, які дозволяють легко керувати додатками на сервері. Крім того, Laravel Horizon забезпечує зручний моніторинг черг задач, а Laravel Telescope пропонує інструменти для відлагодження та моніторингу додатка в реальному часі. Також є безліч пакетів та бібліотек, які розширюють функціональність Laravel, роблячи його одним з найгнучкіших фреймворків для PHP.

Архітектура системи, що побудована на Laravel 10, зображена на рисунку 3.1.

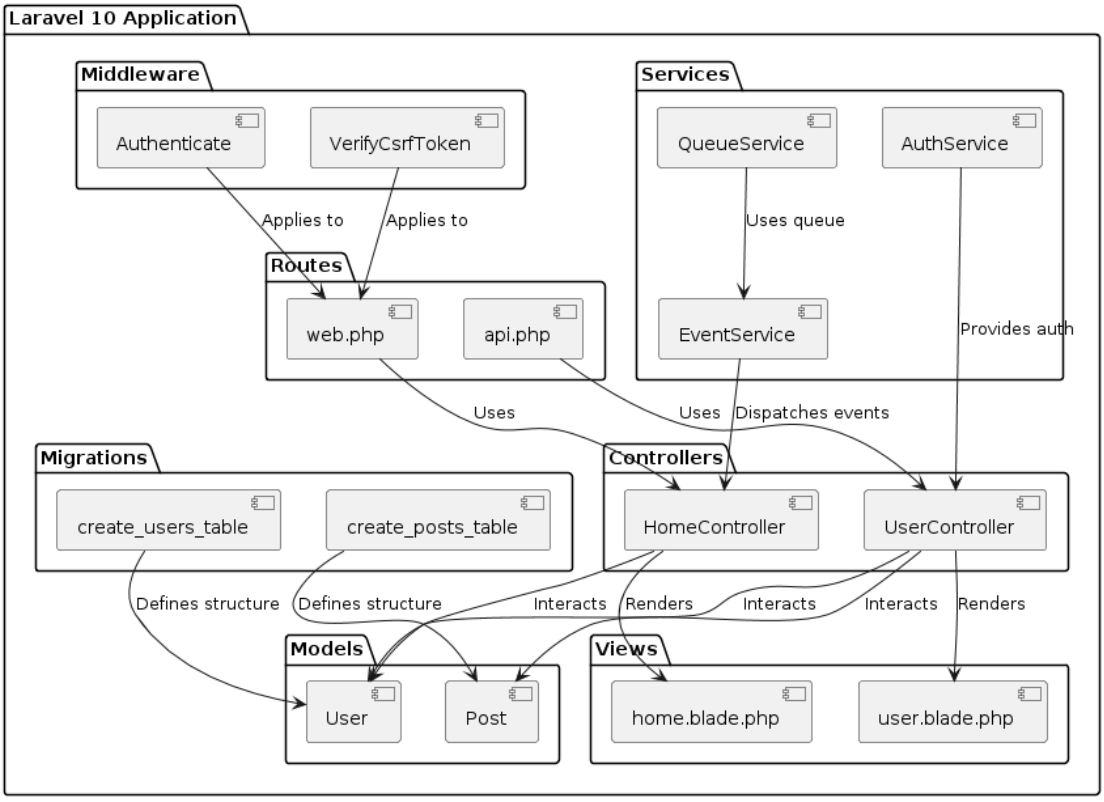


Рис. 3.1 Архітектура системи, побудована на базі фреймворку Laravel

Ця діаграма ілюструє взаємодію між різними компонентами фреймворку.

Laravel надає розробникам зручні інструменти для спрощення процесу розробки та підтримки додатків.

Artisan Command – це вбудований командний інтерфейс Laravel, який забезпечує широкий набір команд для полегшення розробки (рис 3.2). Розробники можуть використовувати Artisan для створення моделей, контролерів, міграцій баз даних та багато іншого. Наприклад, команда php artisan make:model створює нову модель, а php artisan migrate запускає всі міграції баз даних. Artisan також дозволяє створювати власні команди, що значно спрощує процес автоматизації задач.

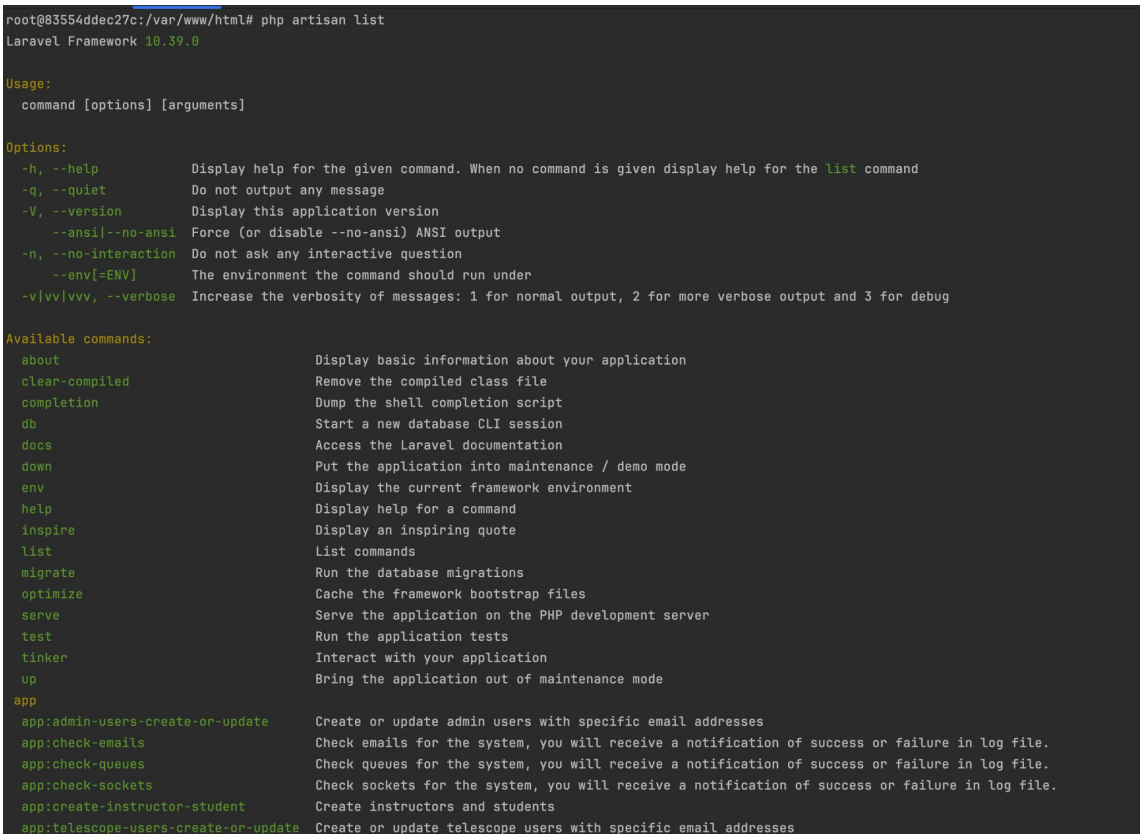


Рис. 3.2 Основні Artisan команди.

Laravel Tinker – це інтерактивна оболонка для взаємодії з додатком у режимі реального часу. Tinker дозволяє розробникам запускати команди, взаємодіяти з моделями, виконувати запити до бази даних та тестувати код безпосередньо в консолі (рис 3.3). Це інструмент для швидкого тестування та налагодження додатків.

```
root@e12abc749910:/var/www/html# php artisan tinker
Psy Shell v0.12.3 (PHP 8.2.18 — cli) by Justin Hileman
> $user = User::find(1);
[!] Aliasing 'User' to 'App\Models\User' for this Tinker session.
= App\Models\User {#7108
    id: 1,
    telegram_name: "Katrine",
    telegram_surname: "Wolff",
    username: "wmitchell",
    email: "chansen@example.com",
    phone: null,
    custom_name: "Anna",
    custom_surname: "Carter",
    #full_registration: true,
    telegram_id: "91773",
    created_at: "2024-05-13 17:54:07",
    updated_at: "2024-05-13 17:54:07",
  }

>
```

Рис. 3.3 Взаємодія з базою даних, за допомогою Tinker.

Pest – це сучасний фреймворк для тестування додатків на Laravel. Pest забезпечує простий та виразний синтаксис для написання тестів, що робить процес тестування більш зрозумілим та зручним (рис 3.4). Pest підтримує всі основні функції, такі як юніт-тести, тестування HTTP-запитів, а також інтеграцію з іншими інструментами Laravel для забезпечення повного циклу тестування додатків.[15]

```
root@e12abc749910:/var/www/html# php artisan test

  PASS  Tests\Unit\ExampleTest
  ✓ that true is true                                                    0.08s

  PASS  Tests\Feature\ExampleTest
  ✓ the application returns a successful response                        1.36s

  Tests:    2 passed (2 assertions)
  Duration: 2.55s

root@e12abc749910:/var/www/html#
```

Рис. 3.4 Запуск тестів з використанням Pest.

Laravel Sail – це легкий інтерфейс для роботи з Docker, який забезпечує просте налаштування та запуск середовища розробки для додатків Laravel. Sail дозволяє розробникам швидко розгортати додатки у контейнерах, що забезпечує консистентність середовища розробки на різних машинах. На рисунку 3.5 показано, як використовувати Laravel Sail для налаштування середовища розробки.[11]

Рис. 3.5 Запуск Docker контейнерів, за допомогою Laravel Sail

Redis – це система керування базами даних в оперативній пам’яті з відкритим кодом, яка використовується для забезпечення швидкого доступу до даних. Вона підтримує різні структури даних, такі як рядки, списки, множини та хеші, що дозволяє ефективно кешувати дані та керувати чергами задач. Використання Redis у проекті дозволяє оптимізувати обробку зображень, забезпечуючи швидкий доступ до необхідних даних та розподіл навантаження між процесами. [13]

PostgreSQL – це потужна система управління реляційними базами даних з відкритим кодом. Вона забезпечує високу надійність, масштабованість та розширюваність, підтримуючи різні типи даних і розширені функції, такі як транзакції та індекси. У проекті PostgreSQL використовується для зберігання та керування великими обсягами даних, забезпечуючи надійну основу для роботи додатка. [12]

Git – це інструмент керування версіями, який дозволяє розробникам відстежувати зміни у коді та керувати проектами. Git підтримує спільну роботу над проектами, дозволяючи об'єднувати зміни з різних гілок коду та вирішувати конфлікти. Крім того, Git надає можливість створювати відгалуження (branches), що забезпечує збереження основного коду під час експериментів та розробок. [20]

GitHub – це онлайн-платформа, яка побудована на основі Git і полегшує співпрацю між розробниками. Вона дозволяє зберігати проекти в хмарі, керувати доступом до них, відстежувати зміни та обговорювати проблеми. GitHub надає інструменти для злиття коду, створення запитів на злиття (pull requests) та управління версіями. Це одна з найпопулярніших платформ для спільної роботи над кодом, яку використовують як окремі розробники, так і великі компанії.

Telegram – це безкоштовний месенджер з відкритим кодом, який дозволяє користувачам надсилати текстові повідомлення, файли та мультимедійні дані. У проекті Telegram використовується як інтерфейс для взаємодії з користувачами, що дозволяє їм надсилати зображення для обробки. Інтеграція з Telegram Bot API дозволяє автоматизувати процес отримання зображень та відправки результатів аналізу. [18, 19]

Postman – це популярний інструмент для тестування API, який дозволяє розробникам створювати, тестувати та документувати API-запити. У проекті Postman використовується для перевірки коректності роботи REST API, розробленого на базі Laravel. Це забезпечує зручність і швидкість тестування API, що сприяє покращенню якості коду.

Docker – це платформа для автоматизації розгортання додатків у контейнерах. Вона дозволяє ізолювати додатки та їх залежності у контейнерах, що забезпечує консистентність середовища розробки, тестування та виробництва. Використання Docker у проекті спрощує процес розгортання додатка та забезпечує його стабільну роботу на різних платформах. Рис. 3.6 - 3.7 показують Docker контейнери додатку та їх налаштування. [14]

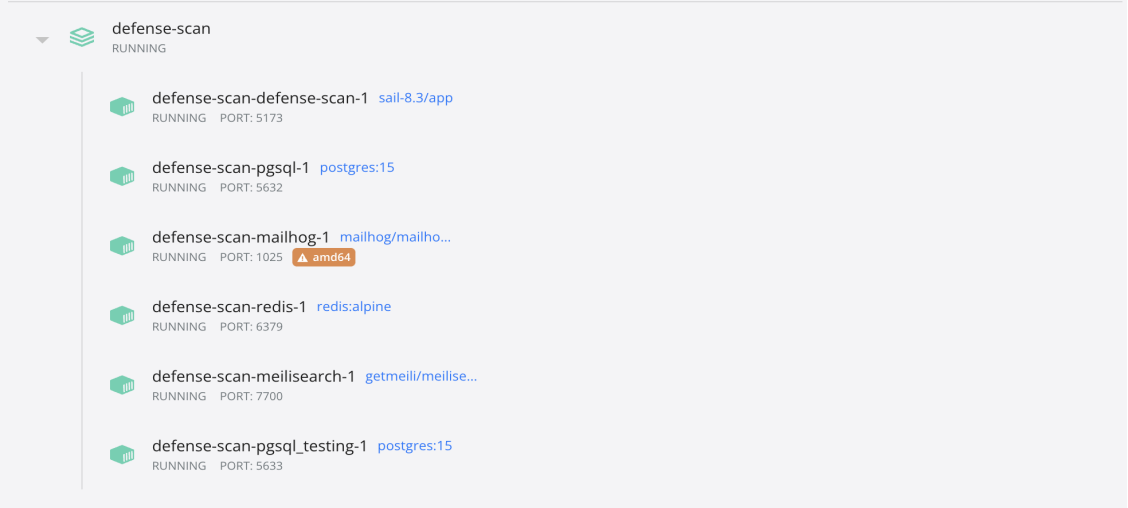


Рис. 3.6 Docker контейнери додатку.

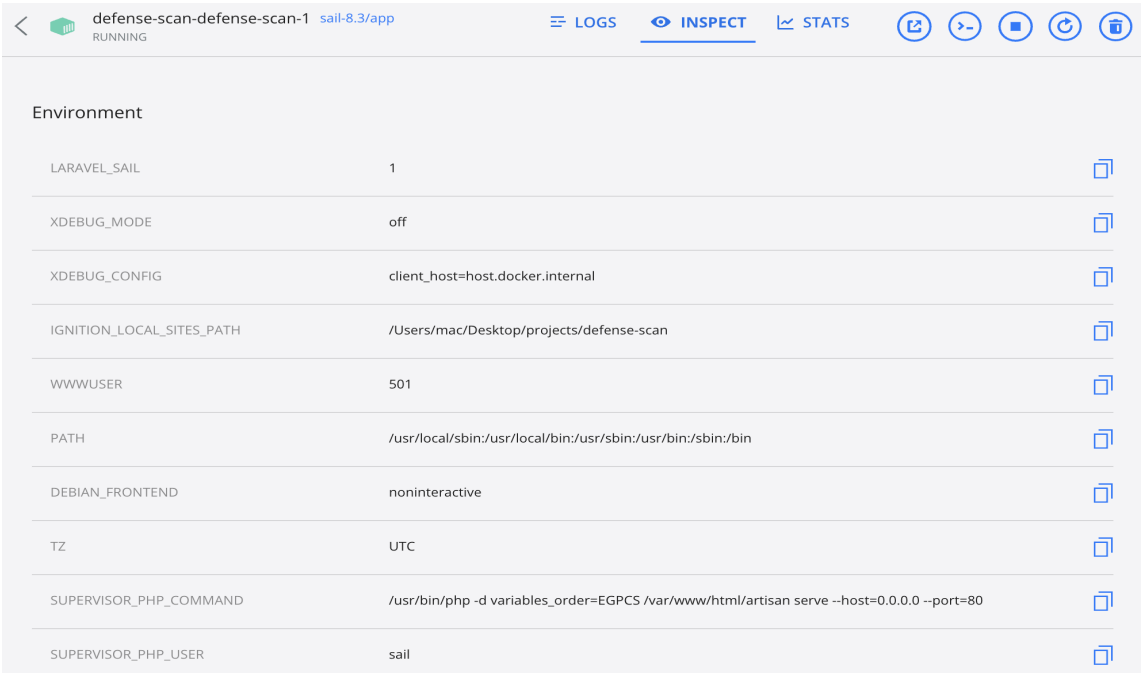


Рис. 3.7 Панель інспектування контейнера defense-scan-defense-scan-1, який працює за допомогою інструмента Laravel Sail.

PHPStorm – це інтегроване середовище розробки (IDE), створене компанією JetBrains, спеціалізоване на розробці PHP-додатків. PHPStorm підтримує широкий спектр технологій веб-розробки, включаючи HTML, CSS, JavaScript, TypeScript, та інші. Це середовище розробки пропонує потужні інструменти для налагодження, тестування та рефакторингу коду, що значно підвищує продуктивність

розробників. PHPStorm також інтегрується з системами контролю версій, такими як Git, що робить його ідеальним вибором для командної розробки. [16]

YOLOv8, розроблена Ultralytics, є новітньою версією популярної моделі для виявлення об'єктів і сегментації зображень. Побудована на основі передових досягнень у галузі глибокого навчання та комп'ютерного зору, YOLOv8 забезпечує високу продуктивність як за швидкістю, так і за точністю. Її адаптивний дизайн дозволяє використовувати модель на різних апаратних платформах, від граничних пристроїв до хмарних API. YOLOv8 є останньою версією цієї моделі, що включає нові можливості та покращення. Вона підтримує широкий спектр завдань комп'ютерного зору, таких як виявлення об'єктів, сегментація, оцінка поз, відстеження та класифікація. [7, 8]

3.2 Застосування Webhook Pattern

Webhook Pattern є одним із способів асинхронної взаємодії між клієнтом і сервером, де сервер автоматично надсилає запити на клієнтський сервер при виникненні певних подій. У контексті мого дипломного проекту, який присвячений розробці модуля обробки зображень у системі збору повідомлень про військову техніку на основі Laravel, Filament, Telegram Bot та YOLOv8, цей патерн використовується для взаємодії з Telegram API. Замість того, щоб постійно опитувати сервер Telegram на наявність нових повідомлень (polling), мій сервер приймає HTTP POST запити від Telegram, коли користувач надсилає нове повідомлення. Це забезпечує ефективну і реальну взаємодію в реальному часі.

Використання Webhook Pattern значно підвищує продуктивність системи, оскільки зменшує навантаження на мережу та серверні ресурси, уникаючи непотрібних запитів. Це дозволяє зосередити ресурси на обробці зображень і аналізі даних, що надходять, забезпечуючи швидку та ефективну відповідь на повідомлення користувачів.

Діаграма роботи патерну Webhook представлена на рис. 3.8.

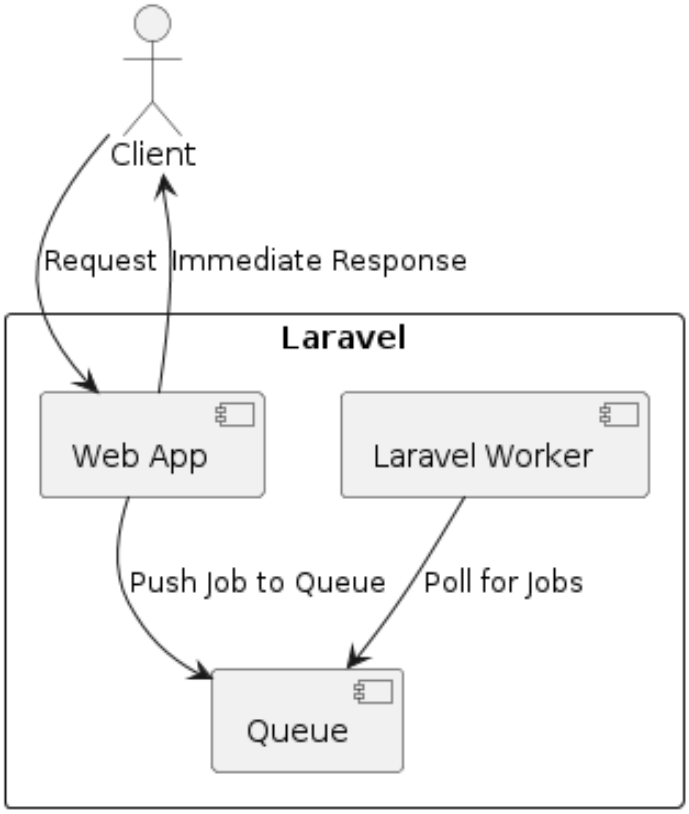


Рис. 3.8 Діаграма роботи патерну Webhook

Webhook Pattern використовується для взаємодії з Telegram API у системі збору повідомлень про військову техніку. Це дозволяє нашому серверу отримувати повідомлення в режимі реального часу, автоматично обробляючи HTTP POST запити від Telegram, коли користувачі надсилають нові повідомлення.

3.3 Розробка бази даних

У якості бази даних вибір пав на PostgreSQL завдяки її надійності, розширеним функціям та гнучкості в роботі з різними форматами даних. [4, 12] Використання цієї системи управління базами даних дозволило створити стійку платформу, здатну легко адаптуватися до зростаючого обсягу даних без

компромісу між продуктивністю та масштабованістю. Таблиці бази даних представлені на рис. 3.9.

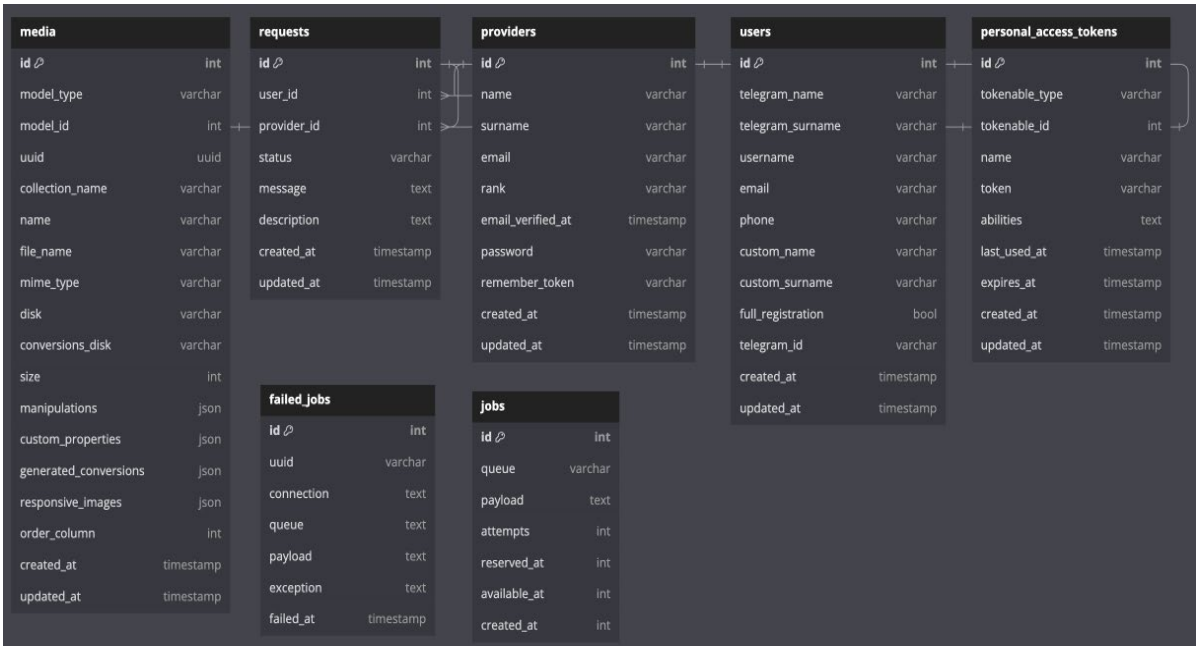


Рис. 3.9 Таблиці бази даних додатку

В Laravel у якості механізму для роботи з базою даних слугує Eloquent ORM (Object-Relational Mapping). Це потужний інструмент, що дозволяє розробникам взаємодіяти з базою даних через об'єктно-орієнтований синтаксис. Використовуючи Eloquent, можна легко виконувати CRUD (Create, Read, Update, Delete) операції, визначати зв'язки між моделями та застосовувати різні методи запитів та фільтрації даних.

Основні особливості Eloquent ORM

- Моделі. Кожна таблиця бази даних має відповідну модель в Laravel. Моделі містять методи для виконання запитів до бази даних.
- Міграції. Laravel використовує міграції для створення та модифікації структури бази даних.
- Зв'язки. Eloquent дозволяє легко визначати зв'язки між моделями, такі як "один до одного", "один до багатьох" та "багато до багатьох".

3.3.1 Модель Provider

Модель Provider представляє адміністраторів у системі. Вона використовує трейт HasApiTokens, HasFactory, Notifiable, InteractsWithMedia та HasRequests. Поля fillable включають основні атрибути постачальника, такі як ім'я, прізвище, електронна пошта, перевірка електронної пошти, пароль, токен та ранг. Поля, які приховуються, включають пароль та токен. Поля casts визначають типи атрибутів. Модель Provider та міграції для створення таблиці можна переглянути на рисунках 3.10 – 3.11. [20]

```
Artem Peregon
class Provider extends Authenticatable implements FilamentUser, HasMedia, HasRequestInterface
{
    use HasApiTokens, HasFactory, Notifiable, InteractsWithMedia, HasRequests;


    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    no usages
    protected $fillable = [
        'name',
        'surname',
        'email',
        'email_verified_at',
        'password',
        'remember_token',
        'rank'
    ];


    /**
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];
```

Рис. 3.10 Модель Provider

```php
/**
 * Run the migrations.
 */
Artem Peregon
public function up(): void
{
    Schema::create( table: 'providers', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'name')->index();
        $table->string( column: 'surname')->index();
        $table->string( column: 'email')->nullable()->unique();
        $table->string( column: 'rank')->nullable()->index();
        $table->timestamp( column: 'email_verified_at')->nullable();
        $table->string( column: 'password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Рис. 3.11 Міграція для створення таблиці providers

3.3.2 Модель User

Модель User представляє користувачів системи. Вона використовує трейт HasApiTokens, HasFactory, Notifiable, InteractsWithMedia та HasRequests. Поля fillable включають основні атрибути користувача, такі як ім'я та прізвище в Telegram, ім'я користувача, електронну пошту, телефон, власні ім'я та прізвище, повну реєстрацію та ідентифікатор Telegram. Поля, які приховуються, включають повну реєстрацію. Поля casts визначають типи атрибутів. Модель User та міграції для створення таблиці можна переглянути на рисунках 3.12 – 3.13.

```php
Artem Peregon
class User extends Authenticatable implements HasMedia, HasRequestInterface
{
    use HasApiTokens, HasFactory, Notifiable, InteractsWithMedia, HasRequests;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    no usages
    protected $fillable = [
        'telegram_name',
        'telegram_surname',
        'username',
        'email',
        'phone',
        'custom_name',
        'custom_surname',
        'full_registration',
        'telegram_id'
    ];
```

Рис. 3.12 Модель User

```php
/**
 * Run the migrations.
 */
Artem Peregon
public function up(): void
{
    Schema::create( table: 'users', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'telegram_name')->nullable()->index();
        $table->string( column: 'telegram_surname')->nullable()->index();
        $table->string( column: 'username')->index();
        $table->string( column: 'email')->nullable()->index()->unique();
        $table->string( column: 'phone')->nullable()->index();
        $table->string( column: 'custom_name')->nullable();
        $table->string( column: 'custom_surname')->nullable();
        $table->boolean( column: 'full_registration')->default( value: false);
        $table->string( column: 'telegram_id')->unique();
        $table->timestamps();
    });
}
```

Рис. 3.13 Міграція для створення таблиці users

3.3.3 Модель Request

Модель Request представляє запити від користувачів. Вона використовує трейт HasFactory та InteractsWithMedia. Поля fillable включають ідентифікатори користувачів та постачальників, статус, повідомлення та опис запиту. Поля casts визначають типи атрибутів. Модель Request та міграції для створення таблиці можна переглянути на рисунках 3.14 – 3.15.

```
Artem Peregon
class Request extends Model implements HasMedia
{
    use HasFactory, InteractsWithMedia, InteractsWithMedia;


    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    no usages
    protected $fillable = [
        'user_id',
        'provider_id',
        'status',
        'message',
        'description',
    ];
```

Рис. 3.14 Модель Request

```
/**
 * Run the migrations.
 */
Artem Peregon
public function up(): void
{
    Schema::create( table: 'requests', function (Blueprint $table) {
        $table->id();
        $table->foreignIdFor( model: User::class)->unsigned()->nullable()->constrained()->cascadeOnDelete();
        $table->foreignIdFor( model: Provider::class)->unsigned()->nullable()->constrained()->cascadeOnDelete();
        $table->string( column: 'status')->nullable()->index();
        $table->text( column: 'message')->nullable();
        $table->text( column: 'description')->nullable();
        $table->timestamps();
    });
}
```

Рис. 3.15 Міграція для створення таблиці requests

3.4 Навчання моделі YOLOv8

У цьому розділі буде описано процес навчання моделі YOLOv8 для розпізнавання об'єктів на зображеннях військової техніки. Навчання моделі є одним з ключових етапів створення системи автоматичного збору повідомлень про військову техніку. Для цього необхідно правильно підготувати дані, провести навчання моделі, валідацію та оцінку її продуктивності. Нижче розглянуто всі необхідні кроки для успішного навчання моделі YOLOv8, включаючи збір і анотацію даних, розділення вибірки, процес навчання, валідацію та оцінку моделі. Також буде представлено результати навчання та їх аналіз. [9]

3.4.1 Збір та анотування даних

Для успішного навчання моделі YOLOv8 необхідно зібрати великий набір зображень, які містять об'єкти, що потребують розпізнавання. Ці зображення повинні бути різноманітними за умовами зйомки, ракурсами та освітленням, щоб забезпечити модель різнобічною інформацією. У нашому випадку це можуть бути зображення військової техніки, зняті з різних кутів і в різних умовах. [5]

Після збору зображень необхідно створити анотації для кожного об'єкта на зображеннях. Анотації містять інформацію про координати рамок навколо об'єктів і їхні класи. Це можна зробити за допомогою спеціальних інструментів, таких як LabelImg для формату XML (Pascal VOC) або LabelMe для формату TXT (COCO). Правильні анотації є критично важливими, оскільки від них залежить точність розпізнавання об'єктів моделлю.

3.4.2 Розділення даних

Після створення анотацій дані потрібно розділити на три вибірки: тренувальну, валідаційну та тестову. Зазвичай, тренувальна вибірка становить 70-80% від загальної кількості даних, валідаційна - 10-20%, а тестова - 10%.

Тренувальна вибірка використовується для навчання моделі, валідаційна - для моніторингу її продуктивності під час навчання, а тестова - для остаточної оцінки моделі після навчання. Розділення даних дозволяє уникнути перенавчання та забезпечує об'єктивну оцінку якості моделі. [10]

3.4.3 Процес навчання моделі

Модель YOLOv8 навчається методом зворотного поширення помилки (backpropagation) з використанням градієнтного спуску (gradient descent). Під час навчання модель проходить через кожне зображення в тренувальній вибірці, прогнозує об'єкти та порівнює ці прогнози з реальними анотаціями. Відхилення між прогнозами і реальними даними визначається за допомогою функції втрат (loss function).

Градієнтний спуск використовується для оновлення вагів моделі, щоб зменшити втрати. Після кожної ітерації ваги коригуються в напрямку, протилежному градієнту функції втрат, що дозволяє моделі поступово покращувати свої прогнози. Цей процес повторюється протягом багатьох епох, поки модель не досягне бажаного рівня точності. Старт та кінець навчання можна переглянути на рисунках 3.16 – 3.17.

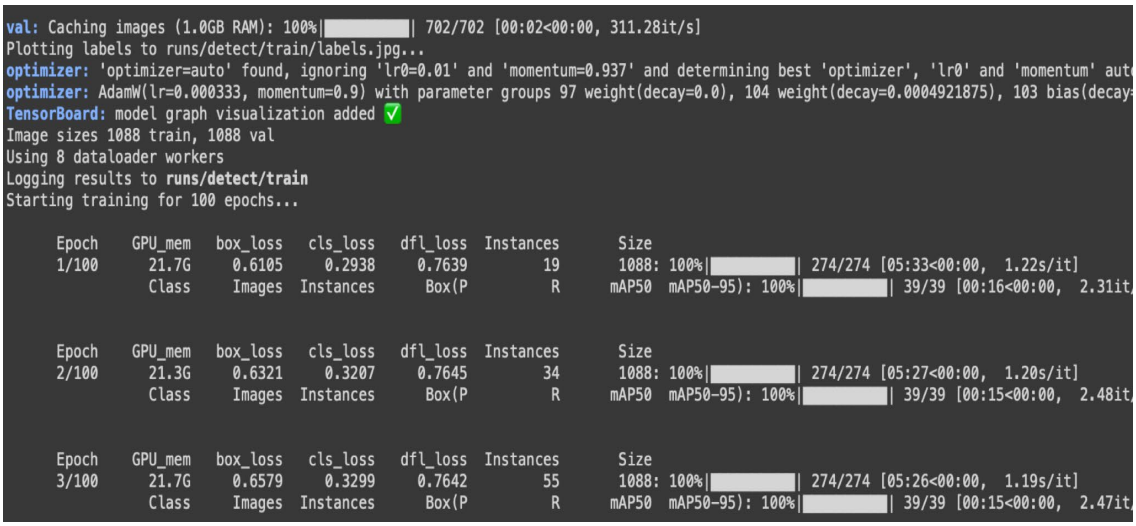


Рис. 3.16 Початок навчання моделі з параметрами.

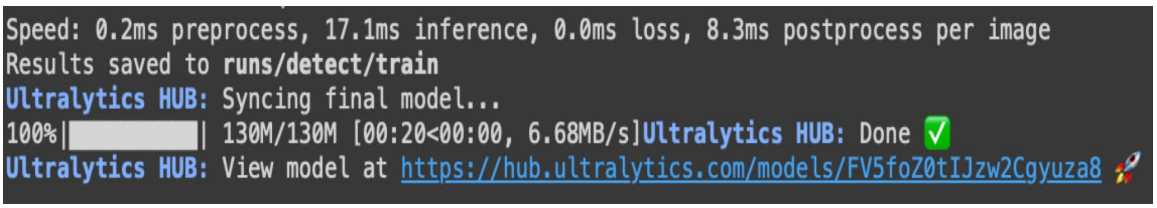


Рис. 3.17 Завершення процесу навчання моделі YOLOv8.

Епоха (epoch) - це один повний цикл навчання моделі на всій тренувальній вибірці. Протягом однієї епохи модель обробляє всі зображення в тренувальному наборі один раз. Епоха складається з декількох ітерацій, де кожна ітерація обробляє частину тренувальної вибірки, відому як batch.

Кількість епох визначає, скільки разів модель перегляне весь тренувальний набір. Наприклад, якщо встановлено 100 епох, то модель перегляне кожне зображення в тренувальному наборі 100 разів. Зазвичай, збільшення кількості епох покращує точність моделі до певного моменту, після чого може початися перенавчання.

3.4.4 Валідація моделі

Валідація моделі здійснюється після кожної епохи для моніторингу її продуктивності на валідаційній вибірці. Це допомагає визначити, чи модель не перенавчається на тренувальних даних і як добре вона здатна узагальнювати нові дані. Під час валідації модель обробляє валідаційну вибірку, і її прогнози порівнюються з реальними анотаціями, що дозволяє оцінити точність і стабільність моделі.

Основні характеристики валідації:

- Box Loss (Втрати рамок) - показує різницю між передбаченими і реальними рамками об'єктів. На графіку видно, що втрати для тренувальної вибірки (синя лінія) поступово зменшуються, тоді як втрати для валідаційної вибірки (помаранчева лінія) залишаються стабільними.

● Class Loss (Втрати класів) - відображає точність визначення класів об'єктів у кожному детектуванні. Графік показує зменшення втрат для тренувальної вибірки і стабільні втрати для валідаційної вибірки.

● Object Loss (Втрати об'єктів) - відображає втрати, пов'язані з виявленням об'єктів. Графік на рис. 3.18 демонструє стабільні втрати як для тренувальної, так і для валідаційної вибірок протягом процесу навчання.

Рис. 3.18 Графіки втрат моделі YOLOv8 протягом навчання

3.5 Схема процесу розпізнавання зображень в YOLOv8

Схема процесу розпізнавання зображень в YOLOv8 демонструє, як система обробляє зображення для виявлення об'єктів. Ця схема показує взаємодію між різними компонентами, такими як завантажувач зображень, препроцесор, модель YOLOv8, процесор результатів та візуалізатор. На рисунку 3.19 представлена схема розпізнавання зображень YOLOv8, яка демонструє, як всі ці компоненти взаємодіють між собою для ефективного розпізнавання об'єктів.

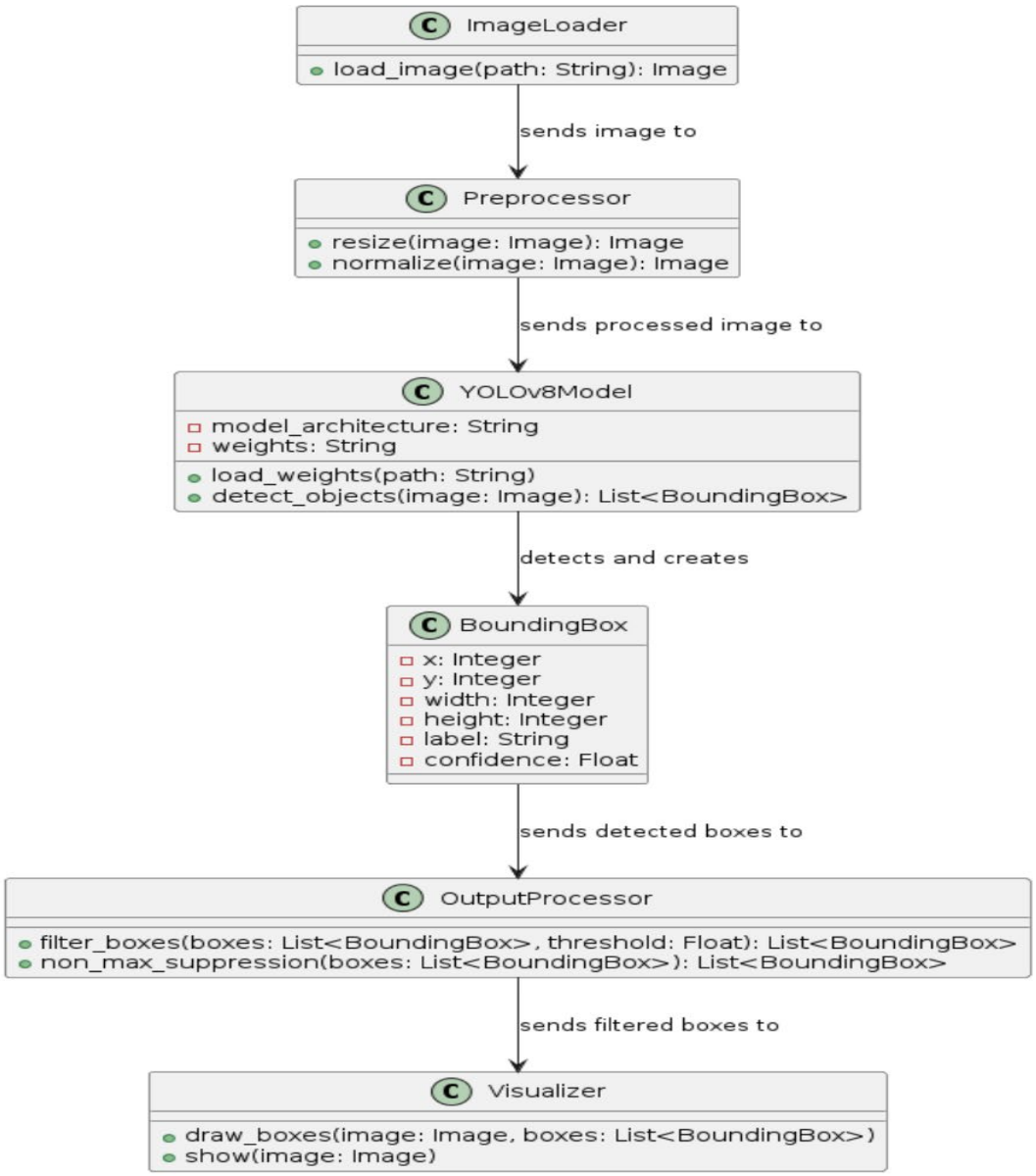


Рис. 3.19 Схема процесу розпізнавання зображень в YOLOv8

3.6 Опис панелі адміністратора

Ключовим елементом проєкту є панель адміністратора, створена для управління та моніторингу даних платформи. Вона надає функції для перегляду та керування користувачами, обробки запитів та інших сутностей системи. Панель включає можливості додавання, редагування та видалення користувачів, обробки користувацьких запитів, а також відстеження різних аспектів роботи платформи.

На рис. 3.20 представлено список користувачів з можливістю перегляду, редагування та видалення кожного з них. Кожен користувач має атрибути, такі як ім'я в Telegram, прізвище, email та ім'я користувача.

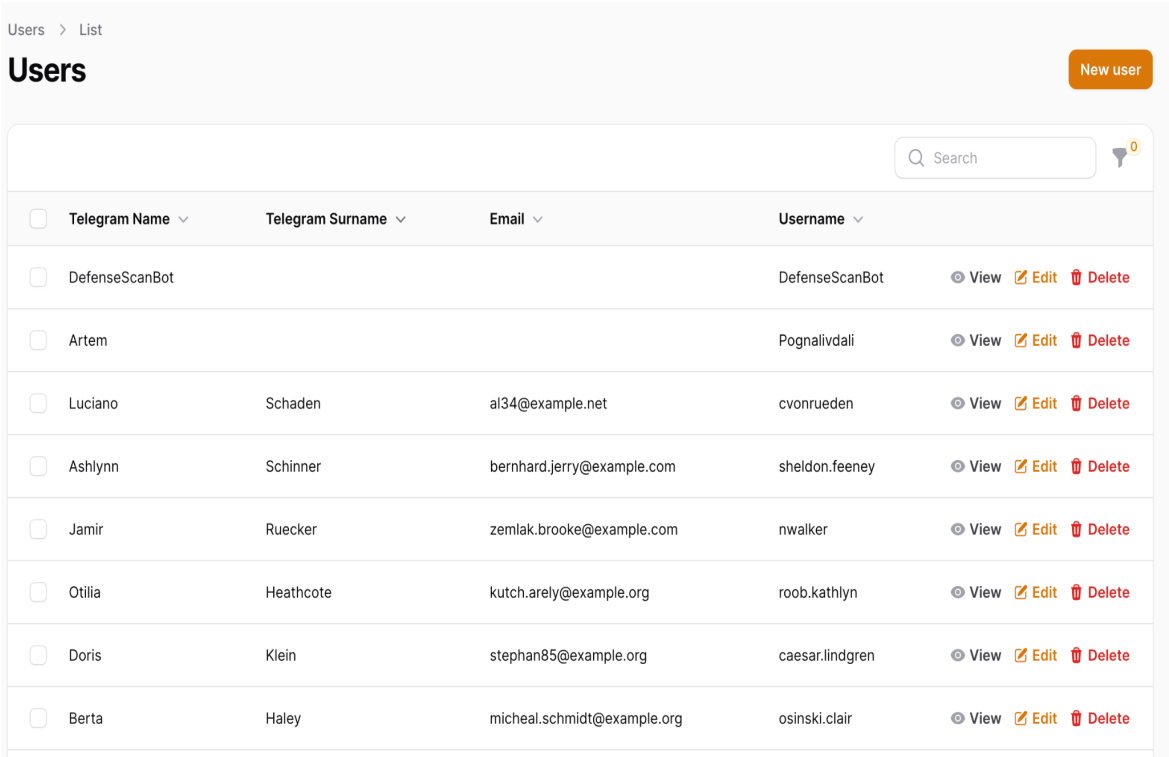


Рис. 3.20 Список користувачів.

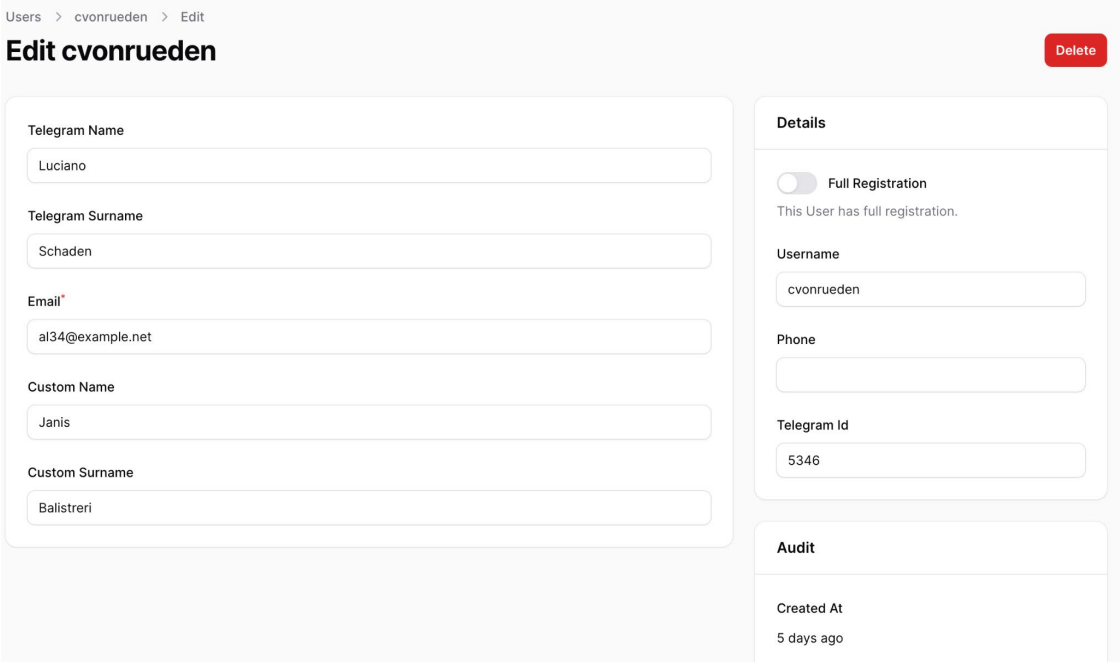


Рис. 3.21 Детальна сторінка користувача

На рис. 3.22 показано список звернень від користувачів. Адміністратор може переглядати деталі кожного звернення, змінювати його статус, а також редагувати або видаляти звернення.

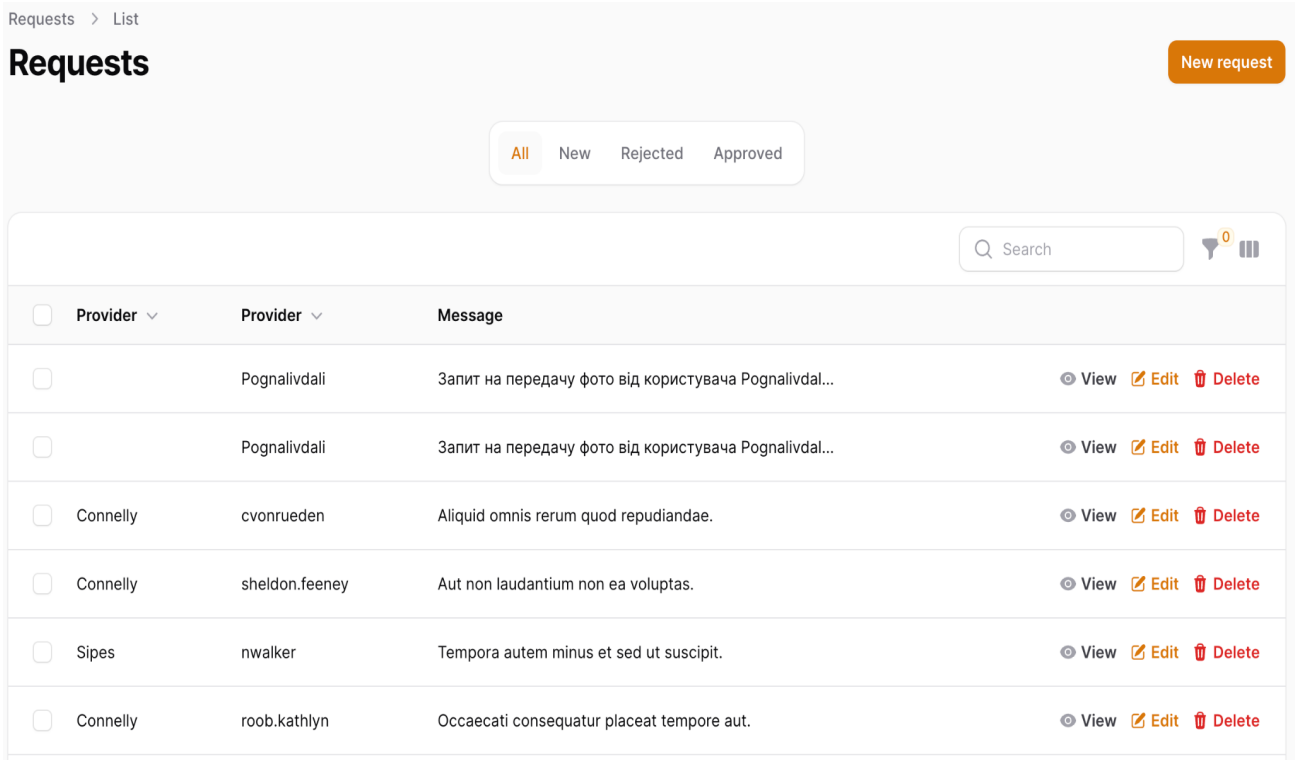


Рис. 3.22 Список звернень від користувачів

На рис. 3.23 детально показано процес обробки звернення користувача. Адміністратор може переглянути оригінальне зображення, з яким було зроблено звернення, а також модифіковане зображення з результатами обробки. Також можна змінити статус звернення, вибрати провайдера, який оброблятиме запит, та переглянути інформацію про користувача, що створив звернення.

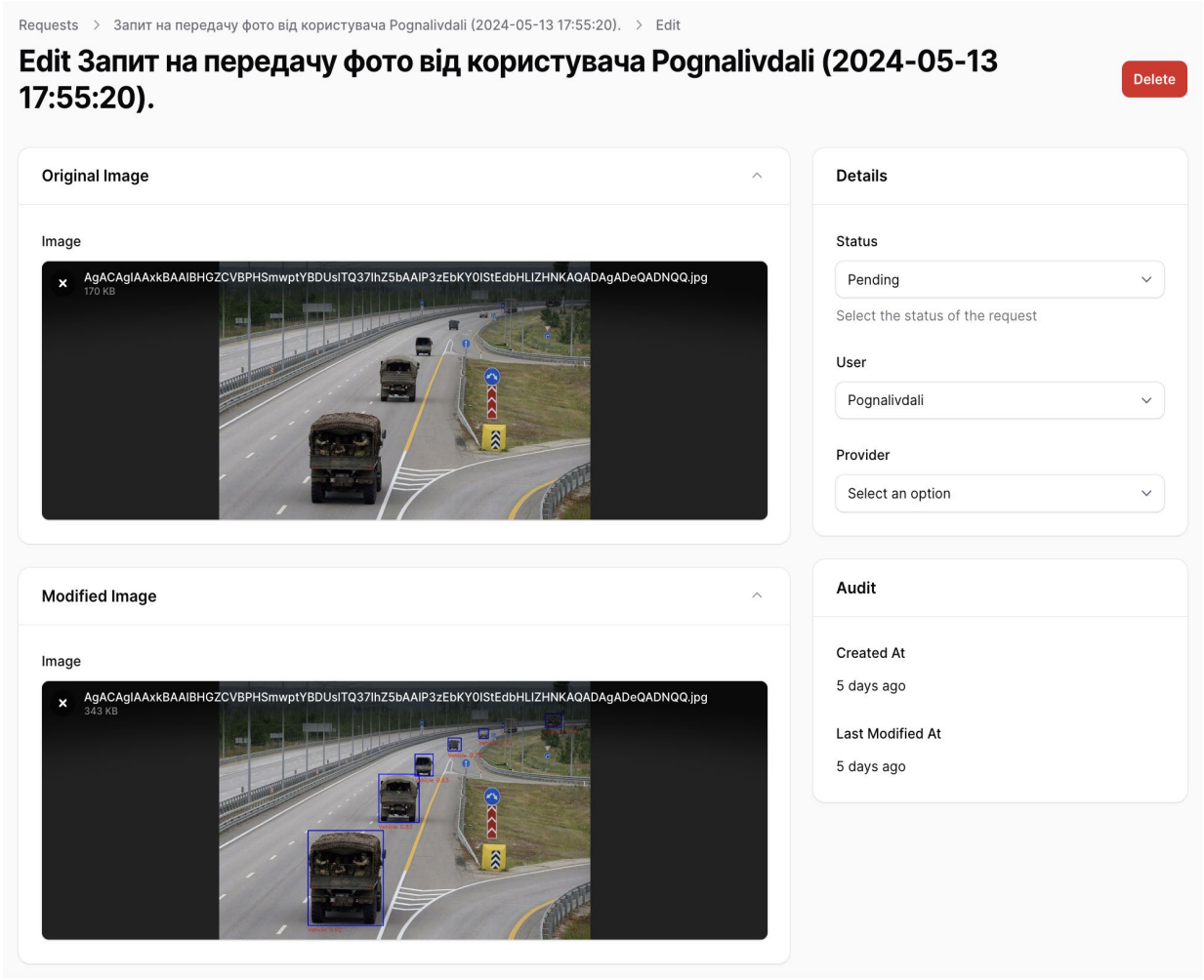


Рис. 3.23 Сторінка редагування звернення

4 ТЕСТУВАННЯ МОДУЛЮ ОБРОБКИ ЗОБРАЖЕНЬ В СИСТЕМІ ЗБОРУ ПОВІДОМЛЕНЬ ПРО ВІЙСЬКОВУ ТЕХНІКУ

4.1 Unit тести

Unit тестування є важливою складовою процесу розробки програмного забезпечення. Це метод тестування, що дозволяє перевірити коректність окремих модулів або функцій програми незалежно від інших частин системи. Метою unit тестування є виявлення та усунення помилок на ранніх етапах розробки, що сприяє підвищенню якості кінцевого продукту. Unit тести допомагають розробникам впевнитись, що кожен окремий компонент системи працює згідно зі своїми специфікаціями, що зменшує ризик виникнення проблем у майбутньому.

Застосування unit тестування в розробці модулю обробки зображень є надзвичайно важливим, оскільки дозволяє виявити та виправити помилки, які можуть виникати під час аналізу та обробки даних зображень. Завдяки цьому забезпечується стабільність та надійність роботи модулю, що особливо важливо в контексті військових застосувань, де точність та надійність є критичними факторами.

Pest - це сучасний та елегантний фреймворк для тестування PHP, який фокусується на простоті. Pest був ретельно розроблений для того, щоб повернути радість від тестування в PHP, забезпечуючи простий і зрозумілий синтаксис для написання тестів. Його використання дозволяє швидко створювати зрозумілі та ефективні тести, що сприяє кращій якості коду та більш надійному програмному забезпеченню. [15]

Однією з ключових переваг використання фреймворку Laravel для unit тестування є його інтеграція з Pest, що дозволяє створювати моки для об'єктів та їх методів, забезпечуючи можливість тестування взаємодії між компонентами без необхідності їх реальної реалізації. [17]

Перевірка коректності обробки медіа та логування результатів роботи модулю обробки зображень. Тест використовує Mockery для створення мока

об'єктів Media, Request, FileAdder, та Process, що дозволяє імітувати їх поведінку та перевірити роботу окремих компонентів системи в ізоляції. Це особливо корисно для тестування взаємодії між різними частинами системи та забезпечення того, що вони правильно обмінюються даними. Під час тесту перевіряється, чи правильно обробляється медіа-файл, створюються необхідні директорії та чи ведеться логування успішного виконання процесу обробки зображень (рис. 4.1).

```php
it( description: 'handles media and logs successfully', function () {
    Queue::fake();
    Storage::fake( disk: 'media');

    $request = Request::factory()->create();

    $media = Mockery::mock( ...args: Media::class)->makePartial();
    $media->shouldReceive( ...methodNames: 'getPath')->andReturn(storage_path( path: 'app/public/telegram_chats_photos/fake.jpg'));
    $media->shouldReceive( ...methodNames: 'getFileName')->andReturn('fake.jpg');

    $mediaCollection = new MediaCollection([$media]);

    $requestMock = Mockery::mock( ...args: Request::class)->makePartial();
    $requestMock->shouldReceive( ...methodNames: 'getMedia')->with(RequestMediaTypeEnum::ORIGINAL->value)->andReturn($mediaCollection);

    $fileAdderMock = Mockery::mock( ...args: FileAdder::class)->makePartial();
    $fileAdderMock->shouldReceive( ...methodNames: 'toMediaCollection')->andReturn($media);
    $requestMock->shouldReceive( ...methodNames: 'addMedia')->andReturn($fileAdderMock);

    $process = Mockery::mock( ...args: Process::class)->makePartial();
    $process->shouldReceive( ...methodNames: 'isSuccessful')->andReturn(true);
    $process->shouldReceive( ...methodNames: 'run')->andReturnSelf();
    $process->shouldReceive( ...methodNames: 'getOutput')->andReturn('Detection successful');

    $this->instance( abstract: Process::class, $process);

    Storage::makeDirectory( path: 'public/detection/' . $request->id);

    Log::shouldReceive('info')->once()->with( ...args: 'Detection successful', Mockery::type( expected: 'array'));

    $job = new DetectionJob($requestMock);

    $job->handle();

    $requestMock->shouldHaveReceived( method: 'addMedia')
        ->with(Mockery::type( expected: 'string'))->once();

    $fileAdderMock->shouldHaveReceived( method: 'toMediaCollection')
        ->with(RequestMediaTypeEnum::MODIFIED->value, DetectionJob::DISK)->once();
});
```

Рис. 4.1 Тест обробки зображення

Перевірка обробки випадку, коли медіа-файл не існує. Тест створює фейковий медіа-файл з нульовим розміром, після чого перевіряється, чи правильно модуль обробки зображень обробляє цю ситуацію. Тест використовує виключення FileDoesNotExist, щоб переконатися, що система правильно реагує на відсутність

файлу. Це дозволяє переконатися, що модуль обробки зображень здатен обробляти помилки та виключення належним чином, забезпечуючи стабільність та надійність роботи системи (рис. 4.2).

```php
it( description: 'fails when file does not exist', function () {
    Queue::fake();
    Storage::fake( disk: 'media');

    $request = Request::factory()->create();

    $media = Media::make([
        'id' => 1,
        'model_type' => Request::class,
        'model_id' => $request->id,
        'collection_name' => RequestMediaTypeEnum::ORIGINAL->value,
        'name' => 'non_existent',
        'file_name' => 'non_existent.jpg',
        'mime_type' => 'image/jpeg',
        'disk' => 'media',
        'size' => 0,
        'manipulations' => [],
        'custom_properties' => [],
        'responsive_images' => [],
        'generated_conversions' => [],
        'order_column' => 1,
    ]);

    $request->media()->save($media);

    Storage::makeDirectory( path: 'public/detection/' . $request->id);

    $job = new DetectionJob($request);

    $job->handle();
})->throws( exception: FileDoesNotExist::class);
```

Рис. 4.2 Тест перевірки обробки випадку

ВИСНОВКИ

В результаті виконання дипломної роботи розроблено модуль обробки зображень у системі збору повідомлень про військову техніку, який базується на використанні Laravel, Filament, Telegram Bot та YOLOv8. Актуальність цієї роботи полягає в автоматизації процесу ідентифікації військової техніки, що є надзвичайно важливим для ефективного аналізу та прийняття рішень у військових операціях.

1. Проведено детальний аналіз предметної області та існуючих рішень у сфері обробки зображень та розпізнавання об'єктів. Це дозволило визначити основні переваги та недоліки додатку

2. Визначено функціональні та нефункціональні вимоги до системи. До функціональних вимог віднесено авторизацію через Telegram, заповнення анкети з інформацією про себе, надсилання зображень, управління користувачами, пошук користувачів та перегляд надісланих зображень. Нефункціональні вимоги включають підтримку різних форматів зображень, вимоги до серверів, використання реляційної бази даних PostgreSQL, та сумісність з основними веб-браузерами.

3. Виконано проектування програмного забезпечення, включаючи визначення загальних вимог, розробку діаграм варіантів використання та архітектури системи. Було створено діаграми класів для структурування програмного коду. Архітектура включає ключові компоненти, такі як Telegram Bot, вебсервер на базі PHP, Redis, модуль обробки зображень, YOLOv8 та базу даних PostgreSQL

4. Вибрано технічні засоби для реалізації системи, включаючи Laravel для побудови бекенду, Filament для створення адміністративної панелі, PostgreSQL для зберігання даних та Redis для кешування і черг задач. Telegram Bot використано для забезпечення зручного каналу взаємодії з користувачами, а YOLOv8 для високоточного розпізнавання об'єктів на зображеннях.

5. Розроблено модуль відповідно до встановлених вимог, з використанням таких технологій, як Laravel для побудови бекенду, Filament для створення адміністративної панелі, PostgreSQL для зберігання даних та Redis для кешування і черг задач.

6. Проведено тестування модуля для забезпечення його надійності. Виконано unit тести за допомогою PEST PHP, що дозволило перевірити функціональність та стабільність роботи компонентів системи.

Робота пройшла апробацію на наукових конференціях, за результатами апробації опубліковано тези доповідей:

1. Перегон А.Д., Ільїн О.Ю. Обробка зображень для автоматизованого виявлення військової техніки у системі збору повідомлень. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 121-122.

2. Перегон А.Д., Ільїн О.Ю. Дизайн архітектури модулю обробки зображень для автоматизованого виявлення військової техніки у системі збору повідомлень. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 122-124.

ПЕРЕЛІК ПОСИЛАНЬ

1. Перегон А.Д., Ільїн О.Ю. Обробка зображень для автоматизованого виявлення військової техніки у системі збору повідомлень. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 121-122.
2. Перегон А.Д., Ільїн О.Ю. Дизайн архітектури модулю обробки зображень для автоматизованого виявлення військової техніки у системі збору повідомлень. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 122-124.
3. Мэтт Стаффер., Laravel. Повне керівництво. 2-е видання
4. Allen G. Taylor., Database Development For Dummies 1st Edition
5. Автомобільна техніка ворога [Електронний ресурс] – Режим доступу до ресурсу: https://sprotyvg7.com.ua/lesson/avtomobilna-texnika.
6. Laravel [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/.
7. Ultralytics [Електронний ресурс] – Режим доступу до ресурсу: https://docs.ultralytics.com/.
8. Довідник по Ultralytics YOLO [Електронний ресурс] – Режим доступу до ресурсу: https://docs.ultralytics.com/ru/guides/.
9. Військова бібліотека: автомобілі і бронетехніка – Режим доступу до ресурсу: https://www.ukrmilitary.com/p/library-vehicle.html.
10. Roboflow. Швидкий старт [Електронний ресурс] – Режим доступу до ресурсу: https://blog.roboflow.com/getting-started-with-roboflow/.

11. Laravel Sail [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/docs/11.x/sail.
12. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/.
13. Laravel Redis [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/docs/11.x/redis.
14. Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/.
15. PestPHP [Електронний ресурс] – Режим доступу до ресурсу: https://pestphp.com/.
16. PHPStorm [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/help/phpstorm/using-code-editor.html.
17. Laravel Testing [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/docs/11.x/testing.
18. Laravel Nutgram [Електронний ресурс] – Режим доступу до ресурсу: https://nutgram.dev/docs/configuration/laravel/.
19. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api.
20. Git [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/what-is-github-and-how-to-work/.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка модулю обробки зображень військової техніки за допомогою YOLOv8: інтеграція Laravel 10 REST API, Filament і Telegram Bot

Виконав студент 4 курсу
Групи ПД-41
Перегон Артем Дмитрович
Керівник роботи
Д.т.н., професор, професор кафедри ІПЗ Ільїн Олег Юрійович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – спрощення процесу збору та обробки зображень оператором у системі збору повідомлень про військову техніку за рахунок автоматизації основних етапів.
- **Об’єкт дослідження** – процес збору та обробки зображень у системі збору повідомлень про військову техніку.
- **Предмет дослідження** – програмне забезпечення для автоматизованого збору та обробки зображень у системі збору повідомлень про військову техніку.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз сучасних алгоритмів та програмних засобів обробки зображень, які можуть бути використані для ідентифікації військової техніки, визначити їх переваги та недоліки.
2. Визначити функціональні та нефункціональні вимоги до модулю обробки зображень у системі збору повідомлень про військову техніку.
3. Дослідити та визначити засоби розробки, технології, бібліотеки, що можуть бути використані для реалізації завдань збору та обробки зображень з метою ідентифікації військової техніки.
4. Спроєктувати та програмно реалізувати модуль обробки зображень, як частину системі збору повідомлень про військову техніку.
5. Провести тестування модулю обробки зображень .

3

АНАЛІЗ АНАЛОГІВ

Показник	Google Search by Image	JPEGsnoop	Foto Forensics	Defense-Scan
Платформа	Windows, MacOs, Web Android, iOS	Windows	Windows, MacOs, Web Android, iOS	Windows, MacOs, Web Android, iOS
Цільовий об'єкт розпізнавання	Загальне розпізнавання об'єктів	Деталізований аналіз зображень	Зміни в зображеннях	Військова техніка
Підтримка різних форматів і розмірів	JPG, PNG, GIF	JPG, TIFF, RAW	JPEG, PNG	JPG, PNG, GIF, TIFF, RAW
Конфіденційність	Ризики використання даних	Локальний аналіз	Дані не зберігаються	Високий рівень захисту даних
Інтеграція з іншими системами	Обмежена інтеграція	Обмежена інтеграція	Обмежена інтеграція	інтеграція через REST API
Автоматизація процесів	Ручне введення даних	Вимагає ручного втручання	Мінімальна автоматизація	Повна автоматизація

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:
Користувач:

- Авторизація через Telegram
- Надсилання зображень через Telegram

Адміністратор:

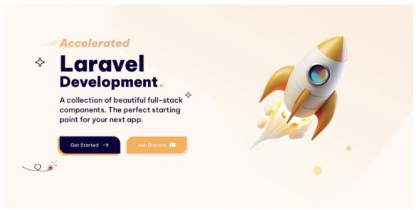
- Управління користувачами
- Перегляд надісланих зображень

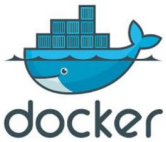
Нефункціональні вимоги:

- Мінімальний розмір зображення - Система повинна обробляти зображення розміром не менше 100x100 пікселів.
- Рекомендований розмір зображення - Оптимальний розмір зображення для обробки — 1920x1080 пікселів.
- Максимальний розмір зображення - Система повинна підтримувати обробку зображень до 3840x2160 пікселів.
- Сервери, на яких розгорнуто систему, повинні мати щонайменше 16 ГБ оперативної пам'яті та процесори не менше Intel i7 або еквівалентні AMD.
- Використання реляційної бази даних PostgreSQL

5

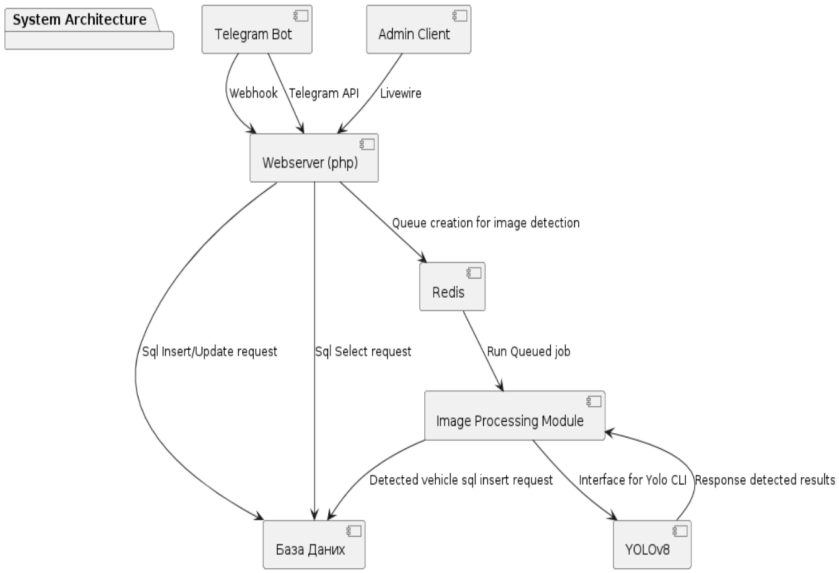
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ





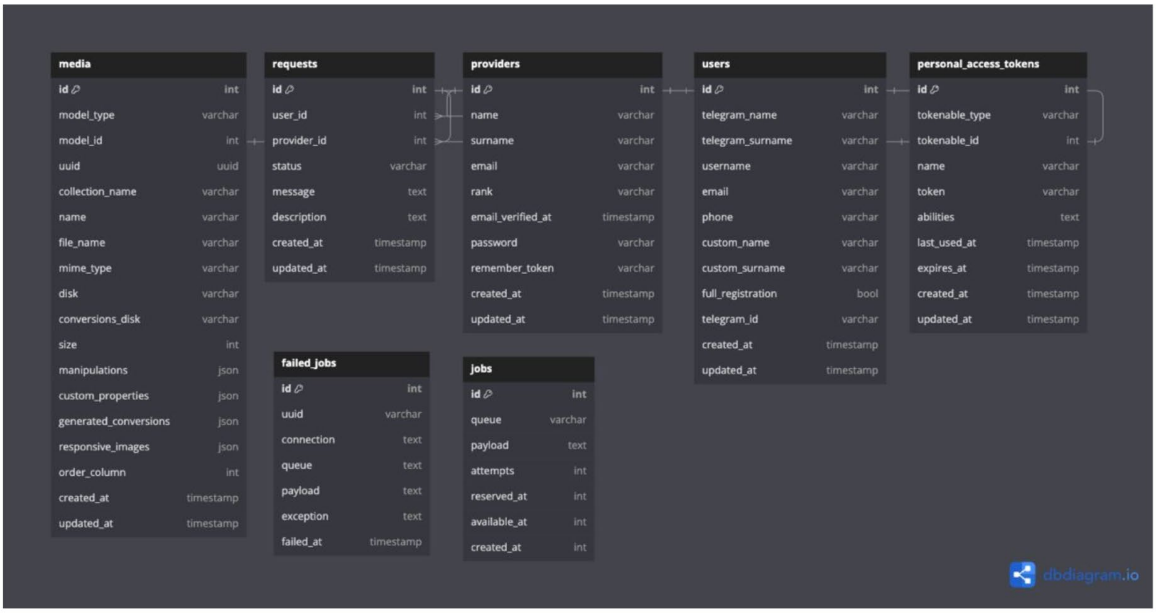
6

АРХІТЕКТУРА ДОДАТКУ



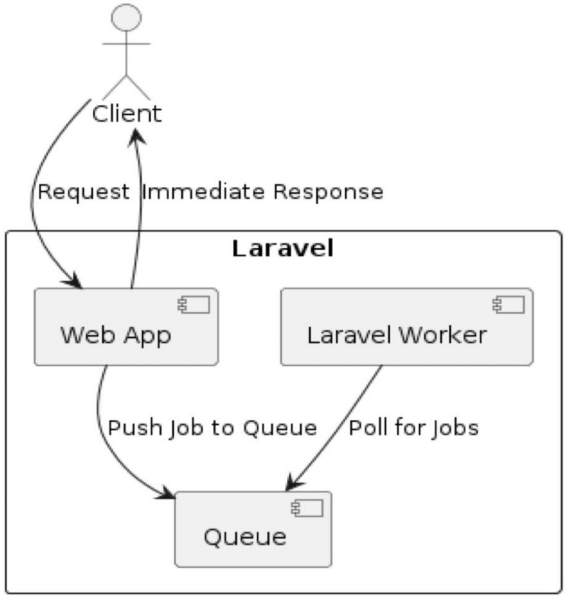
7

СХЕМА БАЗИ ДАНИХ



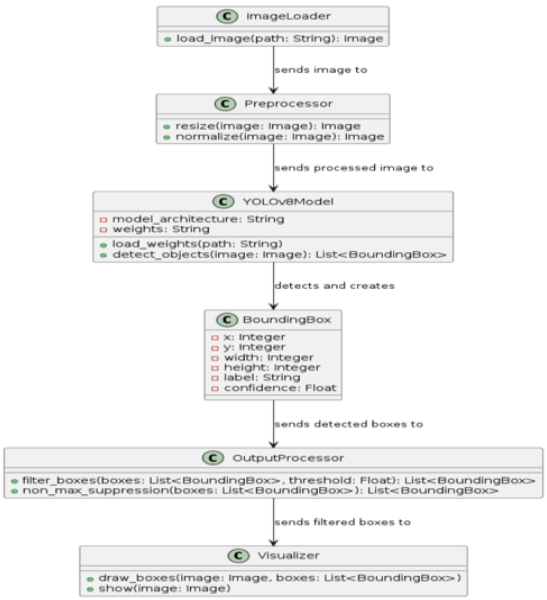
8

ОБРОБКА ЗАДАЧ ЧЕРЕЗ ЧЕРГИ В LARAVEL 10



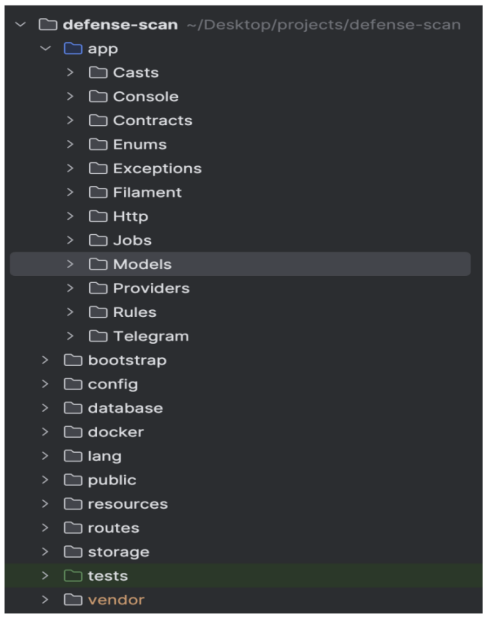
9

СХЕМА ПРОЦЕСУ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ В YOLOV8



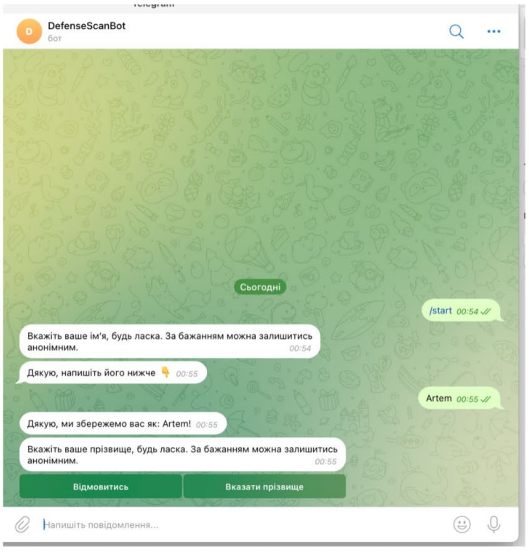
10

ФАЙЛОВА АРХІТЕКТУРА ДОДАТКУ

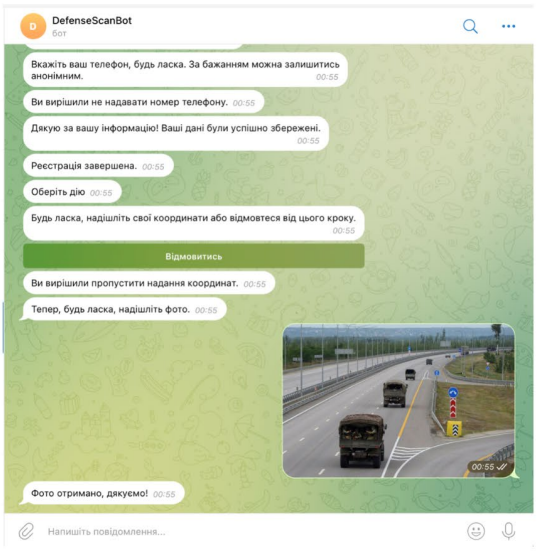


11

ЕКРАННІ ФОРМИ



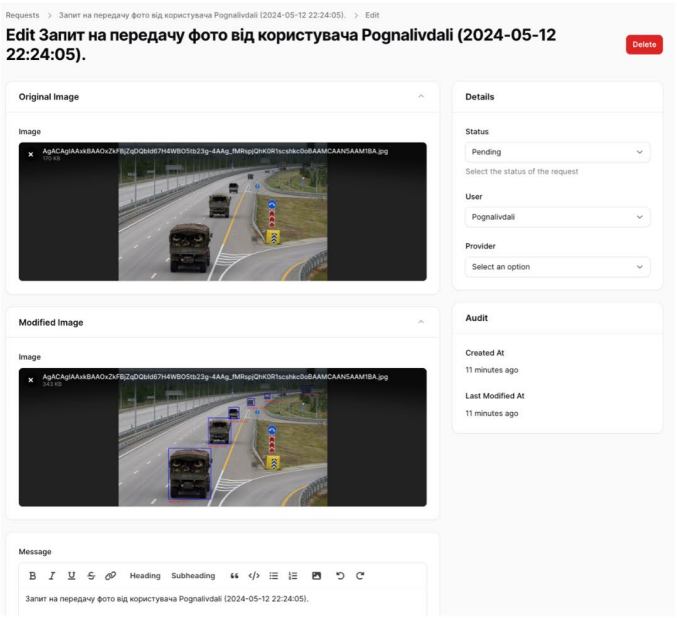
Реєстрація користувача



Відправлення даних через Telegram Bot

12

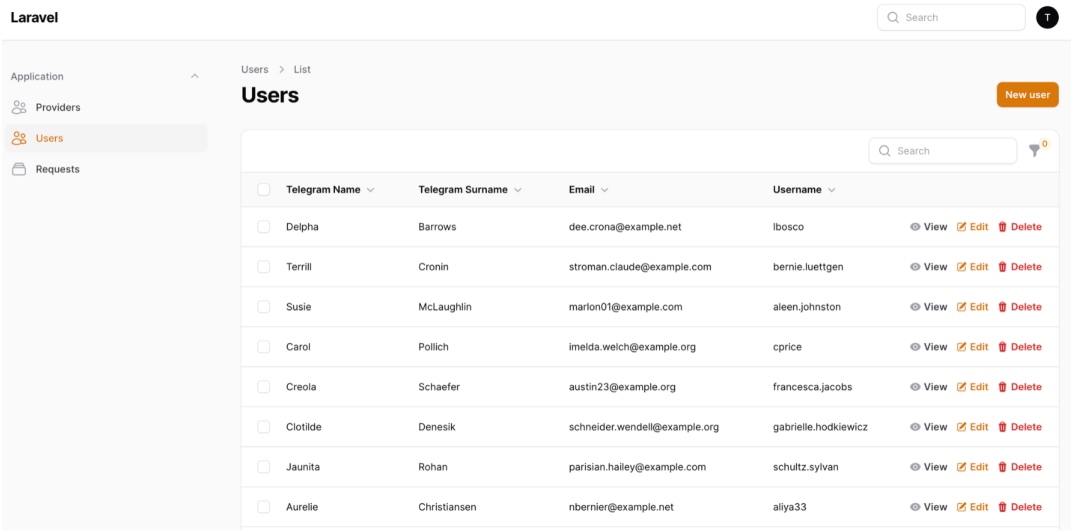
ЕКРАННІ ФОРМИ



Обробка та аналіз запиту користувача

13

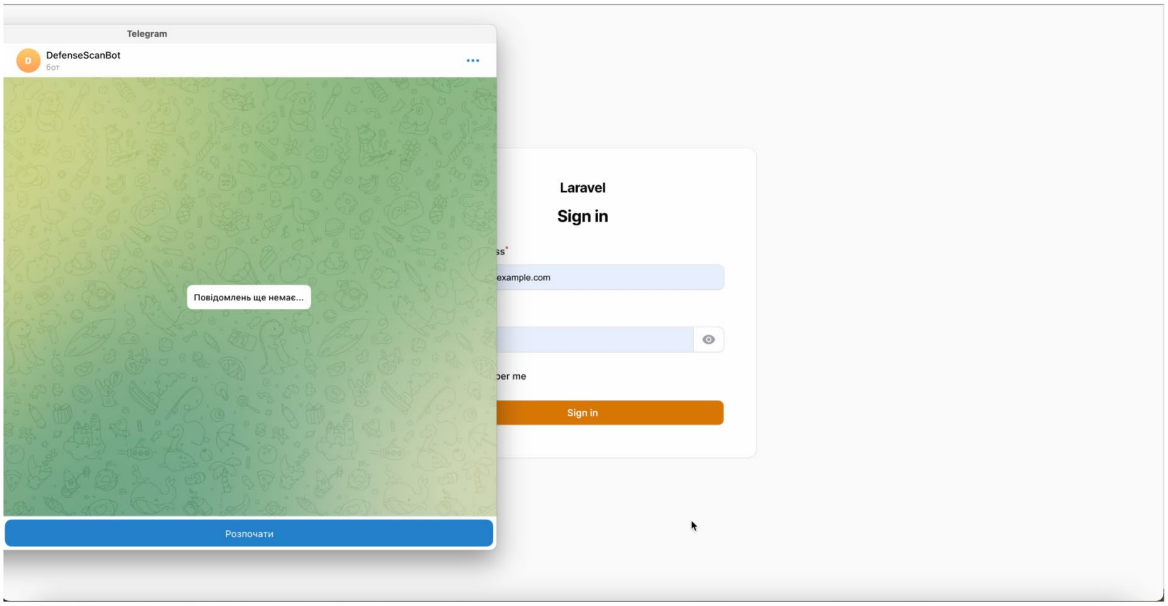
ЕКРАННІ ФОРМИ



Керування користувачами в адміністративній панелі Filament 3

14

ВІДЕО РОБОТИ ДОДАТКУ



15

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Перегон А.Д., Ільїн О.Ю. «Обробка зображень для автоматизованого виявлення військової техніки у системі збору повідомлень». Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – С. 121-122.
2. Перегон А.Д., Ільїн О.Ю. «Дизайн архітектури модулю обробки зображень для автоматизованого виявлення військової техніки у системі збору повідомлень». Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – С. 122-124.

16

ВИСНОВКИ

1. Проведено детальний аналіз предметної області та існуючих рішень у сфері обробки зображень та розпізнавання об'єктів. Це дозволило визначити основні переваги та недоліки наявних підходів і обрати оптимальні технології для реалізації проекту. Зокрема, основними недоліками існуючих рішень є недостатня точність розпізнавання, обмежена підтримка форматів зображень, та низький рівень автоматизації процесів.
2. Виконано проектування програмного забезпечення, включаючи визначення функціональних та нефункціональних вимог, розробку діаграми варіантів використання та архітектури системи. Створено діаграми класів для структурування програмного коду.
3. Вибрано сучасні технології та інструменти для реалізації проекту, зокрема Laravel 10 для серверної частини, PostgreSQL для бази даних, YOLOv8 для аналізу зображень, і Telegram Bot для інтерфейсу користувача.
4. Розроблено модуль відповідно до встановлених вимог, з використанням таких технологій, як Laravel для побудови бекенду, Filament для створення адміністративної панелі, PostgreSQL для зберігання даних та Redis для кешування і черг задач.
5. Проведено тестування модуля для забезпечення його надійності. Виконано unit тести за допомогою PEST PHP, що дозволило перевірити функціональність та стабільність роботи компонентів системи.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
class Provider extends Authenticatable implements
FilamentUser, HasMedia, HasRequestInterface
{
    use HasApiTokens, HasFactory, Notifiable,
InteractsWithMedia, HasRequests;


    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'surname',
        'email',
        'email_verified_at',
        'password',
        'remember_token',
        'rank'
    ];


    /**
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];


    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'name' => 'string',
        'surname' => 'string',
        'email' => 'string',
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
        'remember_token' => 'string',
        'rank' => 'string'
    ];


    /**
     * @param Panel $panel
     * @return bool
     */
    public function canAccessPanel(Panel $panel): bool
    {
        return true;
    }
}
```

```
class Request extends Model implements HasMedia
{
    use HasFactory, InteractsWithMedia,
InteractsWithMedia;


    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'user_id',
        'provider_id',
        'status',
        'message',
        'description',
    ];


    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'user_id' => 'integer',
        'provider_id' => 'integer',
        'status' => 'string',
        'message' => 'string',
        'description' => 'string',
    ];


    /**
     * @return BelongsTo
     */
    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }


    /**
     * @return BelongsTo
     */
    public function provider(): BelongsTo
    {
        return $this->belongsTo(Provider::class);
    }
}

class User extends Authenticatable implements
HasMedia, HasRequestInterface
{
```

```
    use HasApiTokens, HasFactory, Notifiable,
InteractsWithMedia, HasRequests;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'telegram_name',
        'telegram_surname',
        'username',
        'email',
        'phone',
        'custom_name',
        'custom_surname',
        'full_registration',
        'telegram_id'
    ];


    /**
     * @var array<int, string>
     */
    protected $hidden = [
        'full_registration',
    ];


    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'telegram_name' => 'string',
        'telegram_surname' => 'string',
        'username' => 'string',
        'email' => 'string',
        'phone' => Phone::class,
        'custom_name' => 'string',
        'custom_surname' => 'string',
        'full_registration' => 'boolean',
        'telegram_id' => 'string'
    ];
}

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table)
{
            $table->id();
            $table->string('telegram_name')->nullable()-
>index();
            $table->string('telegram_surname')->nullable()-
>index();
            $table->string('username')->index();
            $table->string('email')->nullable()->index()-
>unique();
            $table->string('phone')->nullable()->index();
            $table->string('custom_name')->nullable();
            $table->string('custom_surname')->nullable();
            $table->boolean('full_registration')->default(false);
            $table->string('telegram_id')->unique();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('users');
    }
};

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('failed_jobs', function (Blueprint
$table) {
            $table->id();
            $table->string('uuid')->unique();
            $table->text('connection');
            $table->text('queue');
            $table->longText('payload');
            $table->longText('exception');
            $table->timestamp('failed_at')->useCurrent();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('failed_jobs');
    }
};

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('jobs', function (Blueprint $table) {
```

```
            $table->bigIncrements('id');
            $table->string('queue')->index();
            $table->longText('payload');
            $table->unsignedTinyInteger('attempts');
            $table->unsignedInteger('reserved_at')->nullable();
            $table->unsignedInteger('available_at');
            $table->unsignedInteger('created_at');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('jobs');
    }
};

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('media', function (Blueprint $table) {
            $table->id();

            $table->morphs('model');
            $table->uuid('uuid')->nullable()->unique();
            $table->string('collection_name');
            $table->string('name');
            $table->string('file_name');
            $table->string('mime_type')->nullable();
            $table->string('disk');
            $table->string('conversions_disk')->nullable();
            $table->unsignedBigInteger('size');
            $table->json('manipulations');
            $table->json('custom_properties');
            $table->json('generated_conversions');
            $table->json('responsive_images');
            $table->unsignedInteger('order_column')->nullable()->index();

            $table->nullableTimestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('media');
    }
};

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('providers', function (Blueprint $table) {
            $table->id();
            $table->string('name')->index();
            $table->string('surname')->index();
            $table->string('email')->nullable()->unique();
            $table->string('rank')->nullable()->index();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('providers');
    }
};

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('requests', function (Blueprint $table) {
            $table->id();
            $table->foreignIdFor(User::class)->unsigned()->nullable()->constrained()->cascadeOnDelete();
            $table->foreignIdFor(Provider::class)->unsigned()->nullable()->constrained()->cascadeOnDelete();
            $table->string('status')->nullable()->index();
            $table->text('message')->nullable();
            $table->text('description')->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('requests');
```

```
    }
};

// Set locale for each user
$bot->middleware(function (Nutgram $bot, $next) {
    $message = $bot->message();
    if ($message && $message->from) {
        // Знайти користувача за telegram_id
        $user = User::query()->where('telegram_id',
$message->from->id)->first();

        // Перевірка, чи існує користувач та чи він має
full_registration == false
        if (!$user || !$user->full_registration) {
            User::query()->updateOrCreate(
                ['telegram_id' => $message->from->id],
                [
                    'username' => $message->from-
>username,
                    'telegram_name' => $message->from-
>first_name,
                    'telegram_surname' => $message->from-
>last_name,
                ]
            );
        }
    }

    $next($bot);
});

$bot->onCommand('start', HelloConversation::class);

$bot->onCallbackQueryData('send_photo',
SendPhotoConversation::class);

class HelloConversation extends Conversation
{

    /**
     * Start the conversation
     *
     * @param Nutgram $bot
     * @return void
     * @throws InvalidArgumentException
     */
    public function start(Nutgram $bot): void
    {
        // Витягуємо користувача за telegram_id
        $user = User::query()->where('telegram_id', $bot-
>message()->from->id)->first();

        // Перевірка, чи користувач має full_registration
як true
        if ($user && $user->full_registration) {
            // Якщо анкета вже пройдена, відображаємо
головне меню
            $bot->sendMessage('Оберіть дію',
reply_markup: MainMenu::build());
        } else {
            // Якщо анкета не пройдена, запускаємо процес
реєстрації
            $this->firstStep($bot);
        }
    }


    /**
     * First step of the conversation
     *
     * @param Nutgram $bot
     * @return void
     * @throws InvalidArgumentException
     */
    public function firstStep(Nutgram $bot): void
    {
        $bot->sendMessage(
            text: 'Вкажіть ваше ім’я, будь ласка. За бажанням
можна залишитись анонімним.',
            reply_markup: InlineKeyboardMarkup::make()
                ->addRow(
                    InlineKeyboardButton::make('Відмовитись',
callback_data: 'skip_name'),
                    InlineKeyboardButton::make('Вказати ім’я',
callback_data: 'set_name'),
                )
        );

        $this->next('secondStep');
    }


    /**
     * Second step of the conversation
     *
     * @param Nutgram $bot
     * @return void
     * @throws InvalidArgumentException
     */
    public function secondStep(Nutgram $bot): void
    {
        KeyboardHelper::removeKeyboard($bot);

        $data = $bot->callbackQuery()->data ?? null;
        if ($data === 'skip_name') {
            $this->thirdStep($bot);
        } elseif ($data === 'set_name') {
            $bot->sendMessage('Дякую, напишіть його нижче
');

            $this->next('thirdStep');
        } else {
            $this->firstStep($bot);
        }
    }


    /**
```

```
    * Third step of the conversation
    *
    * @param Nutgram $bot
    * @return void
    * @throws InvalidArgumentException
    */
   public function thirdStep(Nutgram $bot): void
   {
      if (!$bot->isCallbackQuery() && $bot->message()->text) {
         $name = $bot->message()->text;
         $bot->sendMessage('Дякую, ми збережемо вас як: ' . $name . '!');

         // Витягуємо користувача за telegram_id
         User::query()->where('telegram_id', $bot->message()->from->id)->update(['custom_name' => $name]);
      } elseif ($bot->isCallbackQuery() && $bot->callbackQuery()->data === 'skip_name') {
         $bot->sendMessage('Ви вирішили залишитись анонімним, ми збережемо ваше ім’я із Telegram');
      } else {
         $this->secondStep($bot);
      }

      $this->requestSurname($bot);
   }


   /**
    * Request surname from the user
    *
    * @param Nutgram $bot
    * @return void
    * @throws InvalidArgumentException
    */
   private function requestSurname(Nutgram $bot): void
   {
      $bot->sendMessage(
         text: 'Вкажіть ваше прізвище, будь ласка. За бажанням можна залишитись анонімним.',
         reply_markup: InlineKeyboardMarkup::make()
            ->addRow(

InlineKeyboardButton::make('Відмовитись', callback_data: 'skip_surname'),
               InlineKeyboardButton::make('Вказати прізвище', callback_data: 'set_surname'),
            )
      );

      $this->next('fourthStep');
   }


   /**
    * Fourth step of the conversation
    *
    * @param Nutgram $bot
    * @return void
    * @throws InvalidArgumentException
    */
   public function fourthStep(Nutgram $bot): void
   {
      KeyboardHelper::removeKeyboard($bot);

      $data = $bot->callbackQuery()->data ?? null;
      if ($data === 'skip_surname') {
         $this->fifthStep($bot);
      } elseif ($data === 'set_surname') {
         $bot->sendMessage('Дякую, напишіть його нижче');

         $this->next('fifthStep');
      } else {
         $this->thirdStep($bot);
      }
   }


   /**
    * Fifth step of the conversation
    *
    * @param Nutgram $bot
    * @return void
    * @throws InvalidArgumentException
    */
   public function fifthStep(Nutgram $bot): void
   {
      KeyboardHelper::removeKeyboard($bot);

      if (!$bot->isCallbackQuery() && $bot->message()->text) {
         $surname = $bot->message()->text;
         $bot->sendMessage('Дякую, ми збережемо вас як: ' . $surname . '!');

         // Витягуємо користувача за telegram_id
         User::query()->where('telegram_id', $bot->message()->from->id)->update(['custom_surname' => $surname]);
      } elseif ($bot->isCallbackQuery() && $bot->callbackQuery()->data === 'skip_surname') {
         $bot->sendMessage('Ви вирішили залишитись анонімним, ми збережемо ваше прізвище із Telegram');
      } else {
         $this->requestSurname($bot);
      }

      $this->requestPhone($bot);
   }


   /**
    * Request phone number from the user
```

```
    *
    * @param Nutgram $bot
    * @return void
    * @throws InvalidArgumentException
    */
   private function requestPhone(Nutgram $bot): void
   {
      $bot->sendMessage(
         text: 'Вкажіть ваш телефон, будь ласка. За
бажанням можна залишитись анонімним.',
         reply_markup: InlineKeyboardMarkup::make()
            ->addRow(

InlineKeyboardButton::make('Відмовитись',
callback_data: 'skip_phone'),
               InlineKeyboardButton::make('Вказати
телефон', callback_data: 'set_phone'),
            )
      );

      $this->next('sixthStep');
   }


   /**
    * Sixth step of the conversation
    *
    * @param Nutgram $bot
    * @return void
    * @throws InvalidArgumentException
    */
   public function sixthStep(Nutgram $bot): void
   {
      KeyboardHelper::removeKeyboard($bot);

      $data = $bot->callbackQuery()->data ?? null;
      if ($data === 'skip_phone') {
         $bot->sendMessage('Ви вирішили не надавати
номер телефону.');

         // You may want to call some method here to
finish or loop the conversation.
         $this->completeConversation($bot);
      } elseif ($data === 'set_phone') {
         $bot->sendMessage(
            text: 'Напишіть ваш номер телефону нижче
, або натисніть на кнопку нижче, щоб надати його
з контактів.',
            chat_id: $bot->chat()->id,
            reply_markup:
ReplyKeyboardMarkup::make(true)->addRow(
               new KeyboardButton('Поділитись
номером ', request_contact: true)
            )
         );

         // Continue to next step or finish conversation
depending on your application's flow.
         $this->next('completeConversation');
      } else {
         $this->fifthStep($bot);
      }
   }


   /**
    * Complete the conversation
    *
    * @param Nutgram $bot
    * @return void
    * @throws InvalidArgumentException
    */
   public function completeConversation(Nutgram $bot):
void
   {
      KeyboardHelper::removeKeyboard($bot);

      $contact = $bot->message()->contact;
      $fromId = $bot->message()->from->id;

      if ($contact && $contact->user_id === $fromId &&
PhoneValidator::validate($contact->phone_number)) {
         User::query()->where('telegram_id', $fromId)-
>update([
            'phone' => $contact->phone_number,
            'full_registration' => true,
         ]);
         $bot->sendMessage('Ваш номер успішно
збережено.');
      }

      User::query()->where('telegram_id', $fromId)-
>update(['full_registration' => true]);
      $bot->sendMessage('Дякую за вашу інформацію!
Ваші дані були успішно збережені.');

      $bot->sendMessage('Реєстрація завершена.',
reply_markup: ReplyKeyboardRemove::make(true));

      // Відправити головне меню
      $bot->sendMessage('Оберіть дію', reply_markup:
MainMenu::build());

      $this->end();
   }
}

class SendPhotoConversation extends Conversation
{
   const DISK = 'media';


   /**
    * Start the send photo conversation
    *
    * @param Nutgram $bot
    * @return void
    * @throws Throwable
```

```
    */
    public function start(Nutgram $bot): void
    {
        KeyboardHelper::removeKeyboard($bot);

        // Витягуємо користувача за telegram_id
        $user = User::query()->where('telegram_id', $bot->message()->chat->id)->first();

        $now = Carbon::now()->toDateTimeString();

        // Створення запису про запит передачі фото
        $request = Request::query()->create([
            'user_id' => $user->id,
            'status' => RequestStatusTypeEnum::PENDING->value,
            'message' => "Запит на передачу фото від користувача {$user->username} ($now).",
            'description' => "Користувач {$user->telegram_name} ({$user->phone}) надіслав запит на передачу фото."
        ]);

        // Зберігання request_id в кеші за ключем, що відноситься до telegram_id користувача
        Cache::put('request_' . $bot->message()->chat->id, $request->id, 3600);

        Log::info('Request created: ', ['request_id' => $request->id, 'request' => 'request_' . $bot->message()->chat->id]);

        $bot->sendMessage(
            text: 'Будь ласка, надішліть свої координати або відмовтеся від цього кроку.',
            reply_markup: InlineKeyboardMarkup::make()
                ->addRow(
                    InlineKeyboardButton::make('Відмовитись', callback_data: 'skip_coordinates')
                )
        );

        $this->next('requestPhoto');
    }


    /**
     * Request the user to send a photo
     *
     * @param Nutgram $bot
     * @return void
     * @throws InvalidArgumentException
     */
    public function requestPhoto(Nutgram $bot): void
    {
        if ($bot->isCallbackQuery() && $bot->callbackQuery()->data === 'skip_coordinates') {
            $bot->sendMessage('Ви вирішили пропустити надання координат.');
        } elseif ($bot->message()->location) {
            $location = $bot->message()->location;
            $bot->sendMessage("Координати отримано: Широта {$location->latitude}, Довгота {$location->longitude}.");
        }

        $bot->sendMessage('Тепер, будь ласка, надішліть фото.');

        $this->next('handlePhoto');
    }


    /**
     * Handle the photo sent by the user
     *
     * @param Nutgram $bot
     * @return void
     * @throws InvalidArgumentException
     * @throws Throwable
     */
    public function handlePhoto(Nutgram $bot): void
    {
        if ($bot->message()->photo) {
            try {
                $requestId = Cache::get('request_' . $bot->message()->chat->id);

                Log::info('Photo request: ', [
                    'request_id' => $requestId,
                    'photos' => $bot->message()->photo
                ]);

                /** @var Request $request */
                $request = Request::query()->find($requestId);
                $photo = last($bot->message()->photo);
                // Get the photo file via Telegram's getFile method and store it locally
                $path = $bot->getFile($photo->file_id)->save(storage_path("app/public/telegram_chats_photos/{$photo->file_id}.jpg"));
                $fullPath = storage_path("app/public/telegram_chats_photos/{$photo->file_id}.jpg");

                // Add the photo to the media library
                $request->addMedia($fullPath)
                    ->toMediaCollection(RequestMediaTypeEnum::ORIGINAL->value, self::DISK);

                // Optionally, delete the photo after adding it to the media library
                Storage::delete($path);
```

```
            $bot->sendMessage('Фото отримано, дякуємо!');

            DetectionJob::dispatch($request);

            Cache::forget('request_' . $bot->message()->chat->id);

            // Inform user and restart the conversation
            $bot->sendMessage('Якщо ви хочете надіслати ще одне фото, почніть новий запит.');

            $this->start($bot);
        } catch (Throwable $e) {
            Log::error($e->getMessage());
            $bot->sendMessage('Сталася помилка. Будь ласка, спробуйте знову.');

            $this->start($bot); // Restart the conversation even in case of error
        }
    } else {
        $bot->sendMessage('Будь ласка, надішліть фото для продовження.');
        $this->next('handlePhoto');
    }
  }
}

enum RequestMediaTypeEnum: string
{
  use Values, Names, InvokableCases;

  case ORIGINAL = 'original';
  case MODIFIED = 'modified';
}

enum RequestStatusTypeEnum: string
{
  use Values, Names, InvokableCases;

  case PENDING = 'pending';
  case ACCEPTED = 'accepted';
  case REJECTED = 'rejected';
}

class DetectionJob implements ShouldQueue
{
  use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

  const DISK = 'media';


  /**
   * Create a new job instance.
   */
  public function __construct(public Request $request)
  {
    //
  }


  /**
   * Execute the job.
   *
   * @return void
   * @throws FileDoesNotExist
   * @throws FileIsTooBig
   */
  public function handle(): void
  {
    $originalPhotos = $this->request->getMedia(RequestMediaTypeEnum::ORIGINAL->value);
    $scriptPath = base_path('detect_objects.py');
    $modelPath = base_path('best.pt');

    foreach ($originalPhotos as $photo) {
      $originalPath = $photo->getPath();
      $modifiedPath = storage_path('app/public/detection/' . $this->request->id . '/' . $photo->file_name);

      // Run Python script for object detection
      $process = new Process([
        'python3',
        $scriptPath,
        $modelPath,
        $originalPath,
        $modifiedPath,
        (string)$this->request->id
      ]);

      $process->run();

      // Executes after the command finishes
      if (!$process->isSuccessful()) {
        Log::error('Detection failed', ['output' => $process->getOutput()]);
        throw new ProcessFailedException($process);
      }

      Log::info('Detection successful', ['output' => $process->getOutput()]);

      // Add the modified photo to the media library
      if (file_exists($modifiedPath)) {
        $this->request->addMedia($modifiedPath)
          ->toMediaCollection(RequestMediaTypeEnum::MODIFIED->value, self::DISK);
      }
    }
  }
}

import sys
```

```
import os
import cv2
import numpy as np
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from ultralytics import YOLO  # Make sure YOLOv8 is
correctly installed.

def ensure_dir(path):
   """Ensure the directory exists, and if not, create
it."""
   directory = os.path.dirname(path)
   if not os.path.exists(directory):
      os.makedirs(directory)

def detect_objects(model_path, original_photo_path,
output_full_path, request_id):
   try:
      model = YOLO(model_path)  # Load the YOLOv8
model
      image = cv2.imread(original_photo_path)  # Read
the image

      results = model.predict(source=[image])  # Detect
objects

      ensure_dir(output_full_path)

      for result in results:
         for box in result.boxes:
            left, top, right, bottom =
np.array(box.xyxy.cpu(), dtype=int).squeeze()
            width = right - left
            height = bottom - top
            center = (left + int((right-left)/2), top +
int((bottom-top)/2))
            label = result.names[int(box.cls)]
            confidence = float(box.conf.cpu())

            cv2.rectangle(image, (left, top), (right,
bottom), (255, 0, 0), 2)
            cv2.putText(image, f"{label} {confidence:.2f}",
(left, bottom+20), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0, 0, 255), 1, cv2.LINE_AA)

      # Save the image with bounding boxes and labels
      cv2.imwrite(output_full_path, image)
      print(f"Processed image saved at
{output_full_path}")

   except Exception as e:
      print(f"An error occurred during image writing:
{str(e)}")


if __name__ == "__main__":
   if len(sys.argv) != 5:
      print("Usage: python detect_objects.py
<model_path> <original_photo_path> <output_full_path>
<request_id>")
      sys.exit(1)

   model_path, original_photo_path, output_full_path,
request_id = sys.argv[1:5]
   detect_objects(model_path, original_photo_path,
output_full_path, request_id)
```

```
class ProviderResource extends Resource
{
   protected static ?string $model = Provider::class;
   protected static ?string $slug = 'app/providers';

   protected static ?string $recordTitleAttribute =
'surname';
   protected static ?string $navigationIcon = 'heroicon-o-
users';
   protected static ?int $navigationSort = 1;


   /**
    * @return string|null
    */
   public static function getNavigationGroup(): ?string
   {
      return trans('filament-
model.navigation_group.left_group.app.name');
   }


   /**
    * @return string
    */
   public static function getLabel(): string
   {
      return strtolower(trans_choice('filament-
model.navigation_group.providers.name', 1));
   }


   /**
    * @return string|null
    */
   public static function getPluralLabel(): ?string
   {
      return trans_choice('filament-
model.navigation_group.providers.name', 2);
   }


   /**
    * @return Builder
    */
   public static function getEloquentQuery(): Builder
   {
      return parent::getEloquentQuery()-
>withoutGlobalScopes();
```

```
    }


    /**
     * @param Form $form
     * @return Form
     */
    public static function form(Form $form): Form
    {
        return $form
            ->schema([
                Forms\Components\Hidden::make('id')-
>disabled(),

                Forms\Components\Group::make()
                    ->schema([
//
Forms\Components\Section::make(trans('Image'))
//                        ->label('Image')
//                        ->translateLabel()
//                        ->schema([
//
SpatieMediaLibraryFileUpload::make('media')
//                            ->label('Image')
//                            ->translateLabel()
//                            -
>collection(ProviderMediaTypeEnum::AVATAR->value)
//                            ->maxFiles(1)
//                        ])
//                        ->collapsible(),

                    Forms\Components\Section::make()
                        ->schema([

Forms\Components\TextInput::make('name')
                                ->label('Name')
                                ->translateLabel()
                                ->nullable()
                                ->live(onBlur: true),


Forms\Components\TextInput::make('surname')
                                ->label('Surname')
                                ->translateLabel()
                                ->nullable()
                                ->live(onBlur: true),
                        ]),
                ])
                ->columnSpan(['lg' => 2]),

                Forms\Components\Group::make()
                    ->schema([

Forms\Components\Section::make(trans('Details'))
                        ->schema([

Forms\Components\Select::make('rank')
                                ->label(trans('Rank'))
                                ->translateLabel()
                                ->options([
                                    RankTypeEnum::soldier->value =>
trans('Soldier'),
                                    RankTypeEnum::sergeant->value =>
trans('Sergeant'),
                                    RankTypeEnum::lieutenant->value
=> trans('Lieutenant'),
                                ])
                                ->nullable()
                                ->helperText(trans('Select the rank of
the provider.')),


Forms\Components\TextInput::make('email')
                                ->label('Email')
                                ->translateLabel()
                                ->nullable()
                                ->unique(Provider::class, 'email',
ignoreRecord: true)
                                ->live(onBlur: true),
                        ]),


Forms\Components\Section::make(trans('Audit'))
                            ->schema([

Forms\Components\Placeholder::make('created_at')
                                ->label('Created At')
                                ->translateLabel()
                                ->content(fn(Provider $record):
?string => $record->created_at?->diffForHumans()),


Forms\Components\Placeholder::make('updated_at')
                                ->label('Last Modified At')
                                ->translateLabel()
                                ->content(fn(Provider $record):
?string => $record->updated_at?->diffForHumans()),
                            ])
                            ->columnSpan(['lg' => 1])
                            ->hidden(fn(?Provider $record) => $record
=== null),
                    ])
                    ->columnSpan(['lg' => 1]),
            ])
            ->columns(3);
    }


    /**
     * @param Table $table
     * @return Table
     */
    public static function table(Table $table): Table
    {
        return $table
            ->columns([
                TextColumn::make('name')
                    ->label('Name')
```

```
                ->translateLabel()
                ->searchable()
                ->sortable(),

            TextColumn::make('surname')
                ->label('Surname')
                ->translateLabel()
                ->searchable()
                ->sortable(),

            TextColumn::make('email')
                ->label('Email')
                ->translateLabel()
                ->searchable()
                ->sortable(),
        ])
        ->filters([
            //
        ])
        ->actions([
            Tables\Actions\ViewAction::make(),

            Tables\Actions\EditAction::make(),

            Tables\Actions\DeleteAction::make(),
        ])
        ->bulkActions([
            Tables\Actions\BulkActionGroup::make([
                Tables\Actions\DeleteBulkAction::make(),
            ]),
        ])
        ->defaultSort('created_at', 'desc');
    }


    /**
     * @return string[]
     */
    public static function getRelations(): array
    {
        return [
            RequestsRelationManager::class
        ];
    }

    public static function getPages(): array
    {
        return [
            'index' => Pages\ListProviders::route('/'),
            'create' =>
Pages\CreateProvider::route('/create'),
            'edit' =>
Pages\EditProvider::route('/{record}/edit'),
        ];
    }
}
class RequestResource extends Resource
{
    protected static ?string $model = Request::class;

    protected static ?string $slug = 'app/requests';

    protected static ?string $recordTitleAttribute =
'message';
    protected static ?string $navigationIcon = 'heroicon-o-
rectangle-stack';
    protected static ?int $navigationSort = 3;


    /**
     * @return string|null
     */
    public static function getNavigationGroup(): ?string
    {
        return trans('filament-
model.navigation_group.left_group.app.name');
    }


    /**
     * @return string
     */
    public static function getLabel(): string
    {
        return strtolower(trans_choice('filament-
model.navigation_group.requests.name', 1));
    }


    /**
     * @return string|null
     */
    public static function getPluralLabel(): ?string
    {
        return trans_choice('filament-
model.navigation_group.requests.name', 2);
    }


    /**
     * @return Builder
     */
    public static function getEloquentQuery(): Builder
    {
        return parent::getEloquentQuery()-
>withoutGlobalScopes();
    }


    /**
     * @param Form $form
     * @return Form
     */
    public static function form(Form $form): Form
    {
        return $form
            ->schema([
```

```
            Forms\Components\Hidden::make('id')->disabled(),

            Forms\Components\Group::make()
                ->schema([

Forms\Components\Section::make(trans('Original Image'))
                    ->label('Original Image')
                    ->translateLabel()
                    ->schema([

SpatieMediaLibraryFileUpload::make(RequestMediaTypeEnum::ORIGINAL->value)
                        ->label('Image')
                        ->translateLabel()
                        ->collection(RequestMediaTypeEnum::ORIGINAL->value)
                        ->maxFiles(1)
                    ])
                    ->collapsible(),


Forms\Components\Section::make(trans('Modified Image'))
                    ->label('Modified Image')
                    ->translateLabel()
                    ->schema([

SpatieMediaLibraryFileUpload::make(RequestMediaTypeEnum::MODIFIED->value)
                        ->label('Image')
                        ->translateLabel()
                        ->collection(RequestMediaTypeEnum::MODIFIED->value)
                        ->maxFiles(1)
                    ])
                    ->collapsible(),

                Forms\Components\Section::make()
                    ->schema([

Forms\Components\RichEditor::make('message')
                        ->label('Message')
                        ->translateLabel()
                        ->columnSpan('full')
                        ->live(onBlur: true),


Forms\Components\RichEditor::make('description')
                        ->label('Description')
                        ->translateLabel()
                        ->columnSpan('full')
                        ->live(onBlur: true)
                    ]),
            ])
            ->columnSpan(['lg' => 2]),

            Forms\Components\Group::make()
                ->schema([

Forms\Components\Section::make(trans('Details'))
                    ->schema([

Forms\Components\Select::make('status')
                        ->label('Status')
                        ->translateLabel()
                        ->options([

RequestStatusTypeEnum::PENDING->value => __('Pending'),

RequestStatusTypeEnum::REJECTED->value => __('Rejected'),

RequestStatusTypeEnum::ACCEPTED->value => __('Accepted'),
                        ])
                        ->nullable()
                        ->helperText(trans('Select the status of the request')),


Forms\Components\Select::make('user_id')
                        ->label('User')
                        ->translateLabel()
                        ->relationship('user', 'username')
                        ->preload()
                        ->options(User::query()->pluck('username', 'id')->toArray())
                        ->nullable(),


Forms\Components\Select::make('provider_id')
                        ->label('Provider')
                        ->translateLabel()
                        ->relationship('provider', 'surname')
                        ->preload()
                        ->options(Provider::query()->pluck('surname', 'id')->toArray())
                        ->nullable()
                    ]),


Forms\Components\Section::make(trans('Audit'))
                    ->schema([

Forms\Components\Placeholder::make('created_at')
                        ->label('Created At')
                        ->translateLabel()
                        ->content(fn(Request $record): ?string => $record->created_at?->diffForHumans()),


Forms\Components\Placeholder::make('updated_at')
                        ->label('Last Modified At')
```

```
                    ->translateLabel()
                    ->content(fn(Request $record):
?string => $record->updated_at?->diffForHumans()),
                ])
                ->columnSpan(['lg' => 1])
                ->hidden(fn(?Request $record) =>
$record === null),
            ])
            ->columnSpan(['lg' => 1]),
        ])
        ->columns(3);
  }


  /**
   * @param Table $table
   * @return Table
   * @throws Exception
   */
  public static function table(Table $table): Table
  {
     return $table
        ->columns([
           TextColumn::make('provider.surname')
              ->label('Provider')
              ->translateLabel()
              ->searchable()
              ->sortable()
              ->toggleable(),

           TextColumn::make('user.username')
              ->label('Provider')
              ->translateLabel()
              ->searchable()
              ->sortable()
              ->toggleable(),


Tables\Columns\TextColumn::make('message')
            ->label('Message')
            ->translateLabel()
            ->limit(50),
        ])
        ->filters([
           Tables\Filters\SelectFilter::make('status')
              ->label('Status')
              ->options([
                 RequestStatusTypeEnum::PENDING-
>value => __('Pending'),
                 RequestStatusTypeEnum::REJECTED-
>value => __('Rejected'),
                 RequestStatusTypeEnum::ACCEPTED-
>value => __('Accepted'),
              ])
        ])
        ->actions([
           Tables\Actions\ViewAction::make(),

           Tables\Actions\EditAction::make(),
           Tables\Actions\DeleteAction::make(),
        ])
        ->bulkActions([
           Tables\Actions\BulkActionGroup::make([
              Tables\Actions\DeleteBulkAction::make(),
           ]),
        ])
        ->defaultSort('created_at', 'desc');
  }

  public static function getRelations(): array
  {
     return [
        //
     ];
  }

  public static function getPages(): array
  {
     return [
        'index' => Pages\ListRequests::route('/'),
        'create' => Pages\CreateRequest::route('/create'),
        'edit' => Pages\EditRequest::route('/{record}/edit'),
     ];
  }
}

class UserResource extends Resource
{
  protected static ?string $model = User::class;

  protected static ?string $slug = 'app/users';

  protected static ?string $recordTitleAttribute =
'username';
  protected static ?string $navigationIcon = 'heroicon-o-
users';
  protected static ?int $navigationSort = 2;


  /**
   * @return string|null
   */
  public static function getNavigationGroup(): ?string
  {
     return trans('filament-
model.navigation_group.left_group.app.name');
  }


  /**
   * @return string
   */
  public static function getLabel(): string
  {
     return strtolower(trans_choice('filament-
model.navigation_group.users.name', 1));
  }
```

```
    /**
     * @return string|null
     */
    public static function getPluralLabel(): ?string
    {
        return trans_choice('filament-
model.navigation_group.users.name', 2);
    }


    /**
     * @return Builder
     */
    public static function getEloquentQuery(): Builder
    {
        return parent::getEloquentQuery()-
>withoutGlobalScopes();
    }


    /**
     * @param Form $form
     * @return Form
     */
    public static function form(Form $form): Form
    {
        return $form
            ->schema([
                Forms\Components\Hidden::make('id')-
>disabled(),

                Forms\Components\Group::make()
                    ->schema([
//
Forms\Components\Section::make(trans('Image'))
//                        ->label('Image')
//                        ->translateLabel()
//                        ->schema([
//
SpatieMediaLibraryFileUpload::make('media')
//                            ->label('Image')
//                            ->translateLabel()
//                            -
>collection(UserMediaTypeEnum::AVATAR->value)
//                            ->maxFiles(1)
//                        ])
//                        ->collapsible(),

                    Forms\Components\Section::make()
                        ->schema([

Forms\Components\TextInput::make('telegram_name
')
                            ->label('Telegram Name')
                            ->translateLabel()
                            ->nullable()
                            ->live(onBlur: true),

Forms\Components\TextInput::make('telegram_surname'
)
                            ->label('Telegram Surname')
                            ->translateLabel()
                            ->nullable()
                            ->live(onBlur: true),


Forms\Components\TextInput::make('email')
                            ->label('Email')
                            ->translateLabel()
                            ->required()
                            ->unique(User::class, 'email',
ignoreRecord: true)
                            ->live(onBlur: true),


Forms\Components\TextInput::make('custom_name')
                            ->label('Custom Name')
                            ->translateLabel()
                            ->nullable()
                            ->live(onBlur: true),


Forms\Components\TextInput::make('custom_surname')
                            ->label('Custom Surname')
                            ->translateLabel()
                            ->nullable()
                            ->live(onBlur: true),
                        ]),
                ])
                ->columnSpan(['lg' => 2]),

            Forms\Components\Group::make()
                ->schema([

Forms\Components\Section::make(trans('Details'))
                        ->schema([

Forms\Components\Toggle::make('full_registration')
                            ->label('Full Registration')
                            ->translateLabel()
                            ->helperText(trans('This User has full
registration.'))
                            ->default(false),


Forms\Components\TextInput::make('username')
                            ->label('Username')
                            ->translateLabel()
                            ->nullable()
                            ->live(onBlur: true),


Forms\Components\TextInput::make('phone')
                            ->label('Phone')
                            ->translateLabel()
```

```
                        ->nullable()
                        ->live(onBlur: true),


Forms\Components\TextInput::make('telegram_id')
                        ->label('Telegram Id')
                        ->translateLabel()
                        ->nullable()
                        ->live(onBlur: true),
                    ]),


Forms\Components\Section::make(trans('Audit'))
                    ->schema([

Forms\Components\Placeholder::make('created_at')
                        ->label('Created At')
                        ->translateLabel()
                        ->content(fn(User $record):
?string => $record->created_at?->diffForHumans()),


Forms\Components\Placeholder::make('updated_at')
                        ->label('Last Modified At')
                        ->translateLabel()
                        ->content(fn(User $record):
?string => $record->updated_at?->diffForHumans()),
                    ])
                    ->columnSpan(['lg' => 1])
                    ->hidden(fn(?User $record) => $record
=== null),
            ])
            ->columnSpan(['lg' => 1]),
        ])
        ->columns(3);
  }


  /**
   * @param Table $table
   * @return Table
   * @throws Exception
   */
  public static function table(Table $table): Table
  {
     return $table
        ->columns([
           TextColumn::make('telegram_name')
              ->label('Telegram Name')
              ->translateLabel()
              ->searchable()
              ->sortable(),

           TextColumn::make('telegram_surname')
              ->label('Telegram Surname')
              ->translateLabel()
              ->searchable()
              ->sortable(),

           TextColumn::make('email')
              ->label('Email')
              ->translateLabel()
              ->searchable()
              ->sortable(),

           TextColumn::make('username')
              ->label('Username')
              ->translateLabel()
              ->searchable()
              ->sortable(),
        ])
        ->filters([
           Tables\Filters\Filter::make('full_registration')
              ->query(fn($query) => $query-
>where('full_registration', true))
              ->label('Passed Full Registration')
              ->translateLabel(),


Tables\Filters\Filter::make('is_not_full_registration')
              ->query(fn($query) => $query-
>where('full_registration', false))
              ->label('Did Not Pass Full Registration')
              ->translateLabel(),
        ])
        ->actions([
           Tables\Actions\ViewAction::make(),

           Tables\Actions\EditAction::make(),

           Tables\Actions\DeleteAction::make(),
        ])
        ->bulkActions([
           Tables\Actions\BulkActionGroup::make([
              Tables\Actions\DeleteBulkAction::make(),
           ]),
        ])
        ->defaultSort('created_at', 'desc');
  }


  /**
   * @return string[]
   */
  public static function getRelations(): array
  {
     return [
        RequestsRelationManager::class,
     ];
  }

  /**
   * @return array|PageRegistration[]
   */
  public static function getPages(): array
  {
     return [
        'index' => Pages\ListUsers::route('/'),
        'create' => Pages\CreateUser::route('/create'),
        'edit' => Pages\EditUser::route('/{record}/edit'),
     ];
  }
}
```