SYSTEM ŚLEDZĄCY PORUSZAJĄCE SIĘ OBIEKTY

Dokumentacja projektowa PZSP2

WERSJA WSTĘPNA

28.04.2023

Semestr 5.

Zespół nr 4 w składzie:
Hubert Gołębiowski, Filip Pokrop,
Łukasz Główka, Jakub Rozkosz

Mentor zespołu: dr. hab. inż. Krzysztof Cabaj, profesor uczelni

Spis treści

Spis treści

1 Wprowadzenie

1.1 Cel projektu

Celem projektu jest stworzenie aplikacji webowej, umożliwiającej wykrywanie oraz śledzenie dronów na materiałach wideo przesyłanych przez użytkowników. System ma umożliwiać wybór różnych metod śledzenia, dostosowanych do potrzeb użytkowników. Wyniki będą przedstawiane w formie wygenerowanego filmu z zaznaczonymi wykrytymi obiektami, a także w formie pliku tekstowego z adnotacjami pozwalającymi na odtworzenie pozycji wykrytego obiektu na filmie.

1.2 Wstępna wizja projektu

Proces wykrywania dronów jest podzielony na dwa etapy - detekcję obiektów oraz ich śledzenie. Detekcja zajmuje się wykrywaniem obiektów na pojedynczych klatkach filmowych, które są traktowane jak niezależne zdjęcia. Dane o wykrytych obiektach są następnie przekazywane do odpowiednich metod śledzenia, które są w stanie stwierdzić, czy wykryty obiekt jest tym samym, który został wykryty kilka klatek wcześniej. Aby zrealizować postawione nam zadanie postanowiliśmy skorzystać z odpowiednich metod uczenia maszynowego.

Do wykrywania dronów użyliśmy architektury sieci neuronowych YOLO (You Only Look Once), która jest jedną z najbardziej popularnych metod detekcji obiektów w czasie rzeczywistym (szybka i skuteczna w wykrywaniu obiektów). W naszym projekcie korzystamy z pre-trenowanych wag modelu YOLO, które zostały wyuczone na dużym zbiorze danych COCO. Następnie przeprowadzamy dalsze treningi, aby model był w stanie wyspecjalizować się w wykrywaniu dronów. Do treningu wykorzystujemy zbiór nagrań wideo z dronami udostępniony przez właścicielkę tematu oraz odpowiadające im adnotacje, które wymagają odpowiedniego przekształcenia, aby ich format był kompatybilny z YOLO.

W przypadku śledzenia, użytkownik będzie miał możliwość skorzystania z jednej z trzech metod:

- metody śledzenia SORT, DeepSORT
- metody śledzenia obiektu po cechach i wzorcach poruszania się takie jak CSRT, MEDIANFLOW, KCF
- wykrywanie dronów bazując na optycznym przepływie (optical flow) i identyfikacja wykrytych obiektów

Aby film mógł zostać poddany metodzie detekcji YOLO, musi on zostać najpierw podzielony na poszczególne klatki filmowe. Ten proces jest odwracany, gdy tworzymy nagranie wyjściowe z zaznaczonymi obiektami. Do wszelkich zadań związanych z obróbką zdjęć i filmów wykorzystaliśmy bibliotekę OpenCV.

Wytrenowany przez nas model będzie częścią aplikacji internetowej, która umożliwi użytkownikom wstawienie swoich własnych filmów do analizy lub wybranie dostępnych nagrań z dronami. Aplikacja będzie w stanie na bieżąco przetwarzać wideo, wykrywać drony i śledzić ich ruch, a wynikiem analizy będzie wideo z zaznaczoną ramką śledzącą drona, które użytkownik będzie mógł obejrzeć na stronie, lub pobrać wraz z adnotacjami zawierającymi informacje o wykrytych dronach. Użytkownik oprócz możliwości wyboru metody śledzenia, będzie miał również możliwość dostrojenia progu pewności, od którego dron

ma zostać wykryty. Dzięki temu użytkownicy będą mogli dostosować wyniki detekcji do swoich potrzeb i wymagań.

2 Metodologia wytwarzania

Praca w zespole została zorganizowana w następujących krokach:

1. Zrozumienie wymagań projektu - spotkanie z właścicielem tematu projektu oraz wywiedzenie się jakie są konkretne wymagania projektu, jego cele oraz oczekiwania odnośnie jego wyników.

2. Podział zadań - ustalono, że do wykonania są poniższe elementy:

- wykorzystanie modelu sztucznej inteligencji YOLO oraz wytrenowanie go
- przekształcenie bazy danych z danymi trenującymi - wyodrębnienie poszczególnych klatek z filmów oraz odpowiednie przeformatowanie adnotacji do formatu YOLO
- *zaimplementowanie metod śledzenia obiektów (SORT, DeepSORT, optical flow, CSRT, MEDIANFLOW, KCF)
- zintegrowanie wytrenowanego modelu YOLO z metodami śledzenia - zbudowanie programu realizującego wykrywanie i śledzenie drona na filmie wejściowym
- zaprojektowanie serwera API - dodawanie filmów, generacja miniaturek, integracja z modelami wykrywania obiektów i z bazą danych
- zbudowanie frontendu - zaprojektowanie interfejsu użytkownika i integracja z API backendowym
- zaprojektowanie modelu danych warstwy trwałości projektu
- stworzenie bazy danych wykorzystywanej w projekcie

Każdy z członków zespołu dostał przydział zadań.

3. Komunikacja w zespole: narzędziem do komunikacji w zespole jest Discord, na którym utworzyliśmy kanał do konwersacji oraz udostępniania plików. Na spotkaniach, które odbywają się co tydzień lub w ramach potrzeby częściej, przedstawiamy efekty swojej pracy oraz dodatkowo oceniamy postępy całego projektu i wyznaczamy nowe zadania do wykonania.

3 Analiza wymagań

3.1 Wymagania użytkownika i biznesowe

- wymagania biznesowe:

Zaprojektowanie aplikacji webowej, która pozwoli na wykrywanie i śledzenie dronów na podstawie nagrań wideo. Głównym problemem, który chcemy rozwiązać, jest potrzeba szybkiego i skutecznego wykrywania dronów. Nasza platforma będzie udostępniała możliwość zapoznania się z różnymi metodami

wykrywania i śledzenia dronów, ocenienia ich skuteczności oraz szybkości działania, dzięki czemu użytkownicy będą mogli zapoznać się z nowymi technologiami w tej dziedzinie.

- *wymagania użytkowe:*

Wymagane cechy użytkowe aplikacji to: łatwa obsługa, przejrzysty i nieskomplikowany interfejs użytkownika, szybkość działania, wysoka skuteczność w wykrywaniu dronów, możliwość dostosowania progu pewności oraz dostępność z dowolnego urządzenia z dostępem do Internetu.

- *wymagania systemowe:*

Nasze rozwiązanie opiera się na architekturze sieci neuronowej YOLO, co zapewnia szybkość działania i wysoką skuteczność w wykrywaniu obiektów. Aplikacja będzie działała na serwerze, który będzie przetwarzał dane wideo i generował wyniki analizy. Wymagane cechy systemowe to: skalowalność, dostępność w chmurze, optymalizacja działania na różnych urządzeniach, takich jak komputery, smartfony czy tablety.

3.2 Wymagania funkcjonalne i niefunkcjonalne

Najważniejszymi wymaganiami funkcjonalnymi są:

- możliwość wyboru i wgrania materiału w postaci video
- wybranie i możliwość uruchomienia wybranego scenariusza wykrywania dronów i śledzenia ich
- zapisywania adnotacji w określonym formacie
- zapisywanie wyników (filmów z zaznaczonymi obiektami)

Wymagania niefunkcjonalne to przede wszystkim łatwość obsługi systemu oraz szeroki zbiór metod wykrywania obiektów i metod ich śledzenia. Dobrze, aby metody działały w czasie rzeczywistym.

3.3 Przypadki użycia

Użytkownik

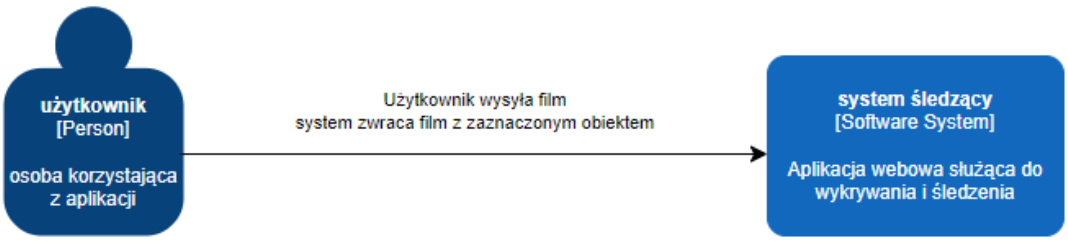
1. Tworzenie konta
    - użytkownik naciska przycisk register
    - użytkownik wpisuje adres email i hasło
    - użytkownik rejestruje konto jeżeli email jest już w bazie danych zwracany jest błąd
2. Logowanie
    - użytkownik wpisuje email oraz hasło
    - Jeżeli dane są poprawne następuje przekierowanie do panelu głównego
    - Jeżeli dane są błędne zwracany jest błąd i prośba o ponowne wprowadzenie danych
3. Wgranie filmu
    - użytkownik loguje się
    - wybiera film który chce wgrać
    - naciska przycisk wgraj
    - jeżeli plik jest w odpowiednim formacie oraz nie ma pliku o takiej samej nazwie plik jest wgrywany

4. Przetworzenie filmu
    - użytkownik loguje się
    - wgrywa film lub wybiera film wgrany wcześniej
    - wybiera rodzaj detekcji oraz trackera
    - naciska przycisk start i oczekuje na przetworzenie pliku
    - gdy plik jest gotowy wyświetlany jest on użytkownikowi
5. Pobranie filmu
    - użytkownik loguje się do systemu
    - wybiera przetworzony film lub zleca jego przetworzenie
    - gdy film jest przetworzony użytkownik może go pobrać
6. Usuwanie filmu
    - użytkownik loguje się do serwisu
    - wybiera jeden z wgranych filmów
    - naciska przycisk usuń film jest usunięty

4 Definicja architektury

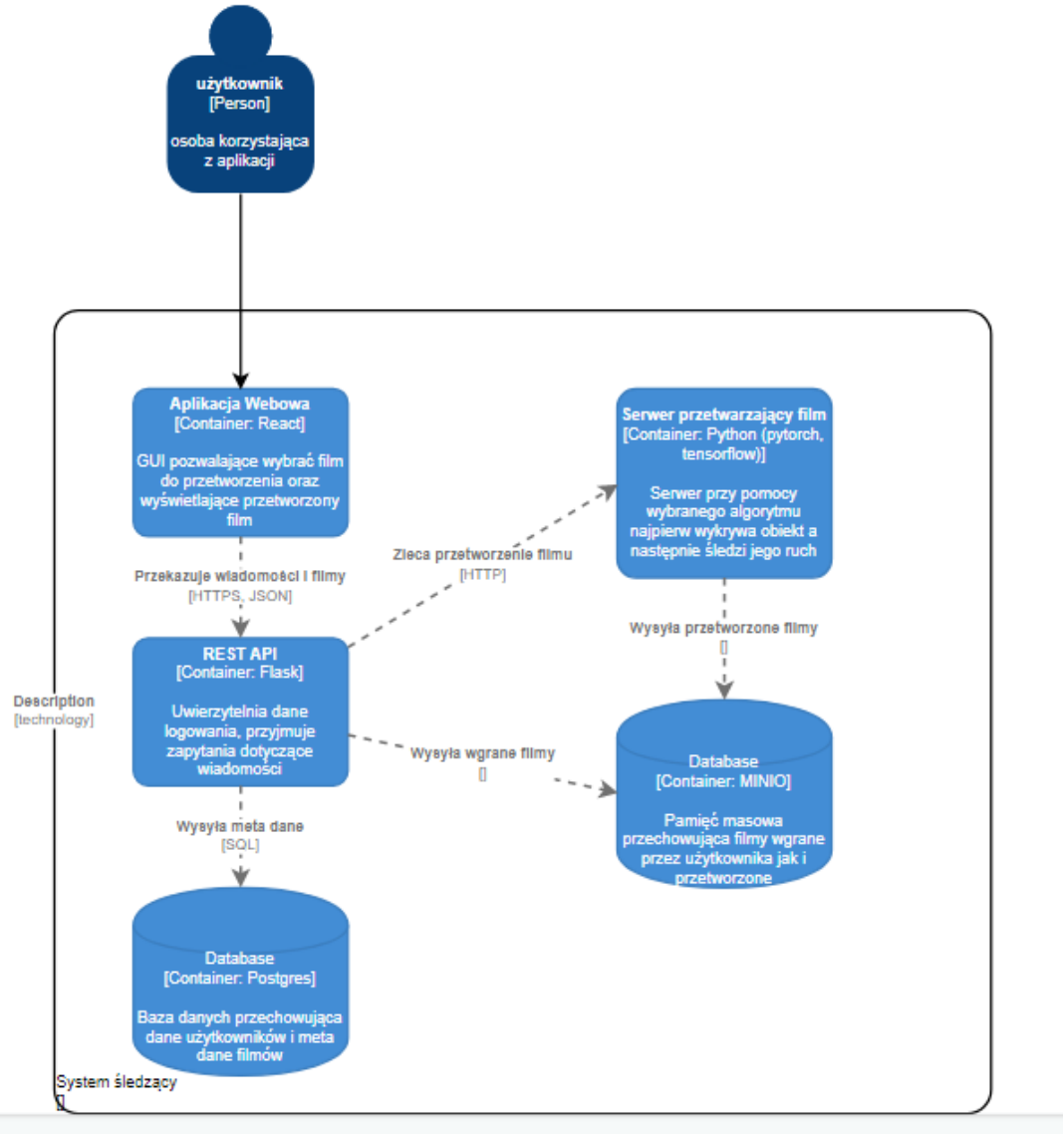
Architektura systemu została przez nas zobrazowana przy pomocy modelu C4:

1. CONTEXT



Jedynym aktorem w naszym układzie jest użytkownik. Ma on dostęp do aplikacji webowej. Poprzez tą aplikacje może wgrywać, oglądać i pobierać materiały wideo.

2. CONTAINERS



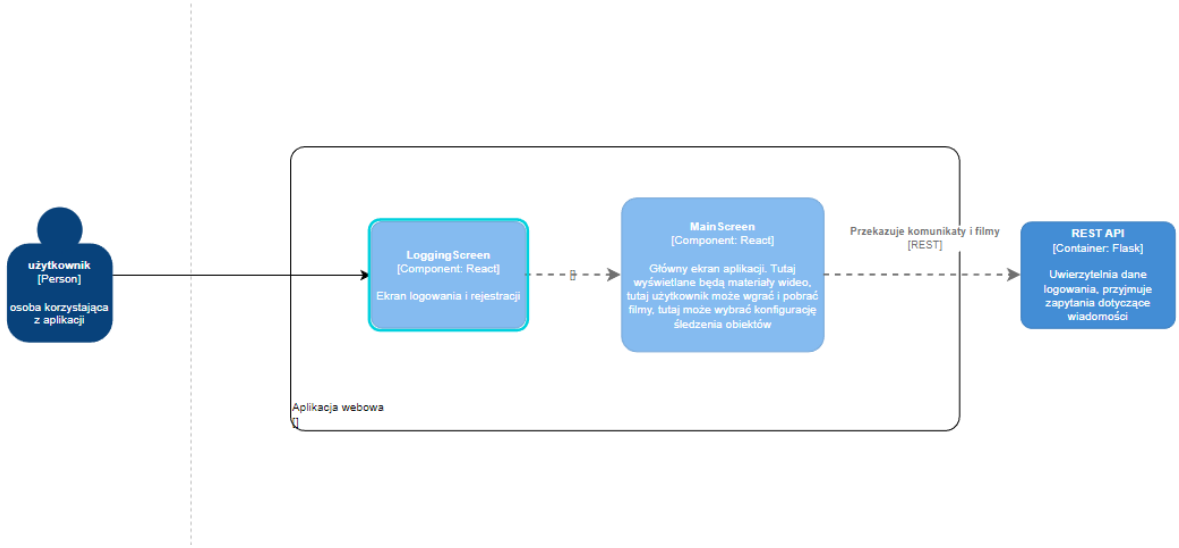
W naszym systemie możemy wyszczególnić 5 niezależnych komponentów, które docelowo znajdą się w odrębnych kontenerach dockerowych:

- Aplikacja Webowa - serwer z aplikacją internetową
- REST API - kluczowy serwer aplikacji, odpowiadający za komunikację klienta z resztą systemu. Przetwarza zapytania i odpowiednio zarządza resztą komponentów.
- Serwer przetwarzający film - serwer zajmujący się przetwarzaniem materiałów wideo. Na tym serwerze odbywają się wszystkie etapy potrzebne do wygenerowania wyjściowego filmu z zaznaczonymi dronami. Komunikuje się z klientem poprzez serwer REST API, jednak ma także bezpośredni dostęp do bazy MinIO, aby pobierać i zapisywać filmy w bazie.

- Baza filmów MinIO - baza danych do przechowywania obiektów. Wykorzystamy ją do magazynowania filmów wgranych przez użytkownika, filmów wygenerowanych przez modele śledzenia oraz wygenerowanych przez nie adnotacji.
- Baza PostgreSQL - relacyjna baza danych, którą wykorzystamy do przechowywania informacji o zarejestrowanych użytkownikach i wgranych przez nich filmach w postaci metadanych. Dzięki tym informacjom będziemy w stanie powiązać fizyczne materiały wideo w bazie MinIO z ich właścicielami.

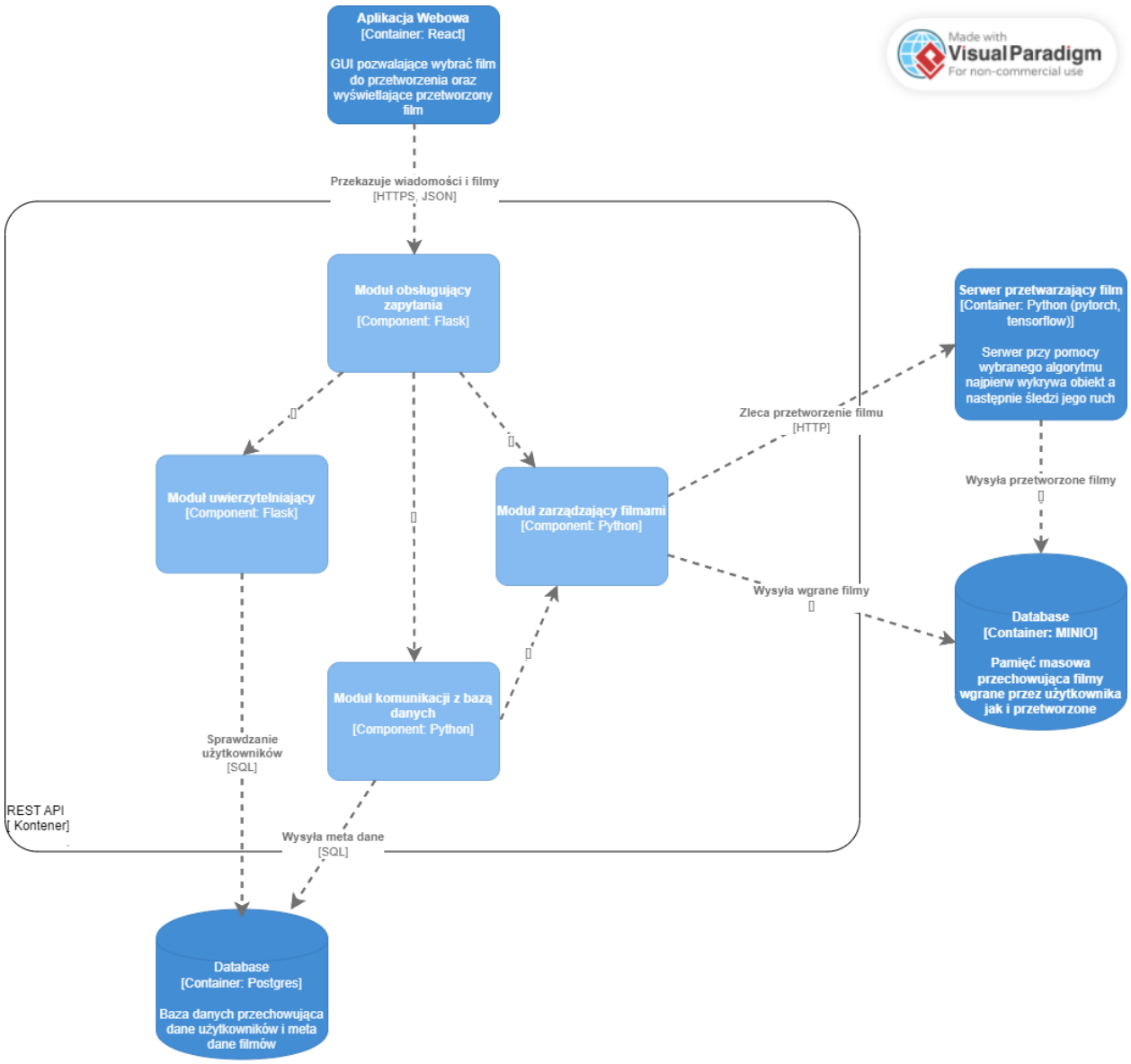
3. COMPONENTS

Aplikacja webowa:



W aplikacji webowej możemy wyodrębnić 2 komponenty: ekran logowania, oraz ekran główny, który zawiera wszystkie funkcjonalności naszej aplikacji.

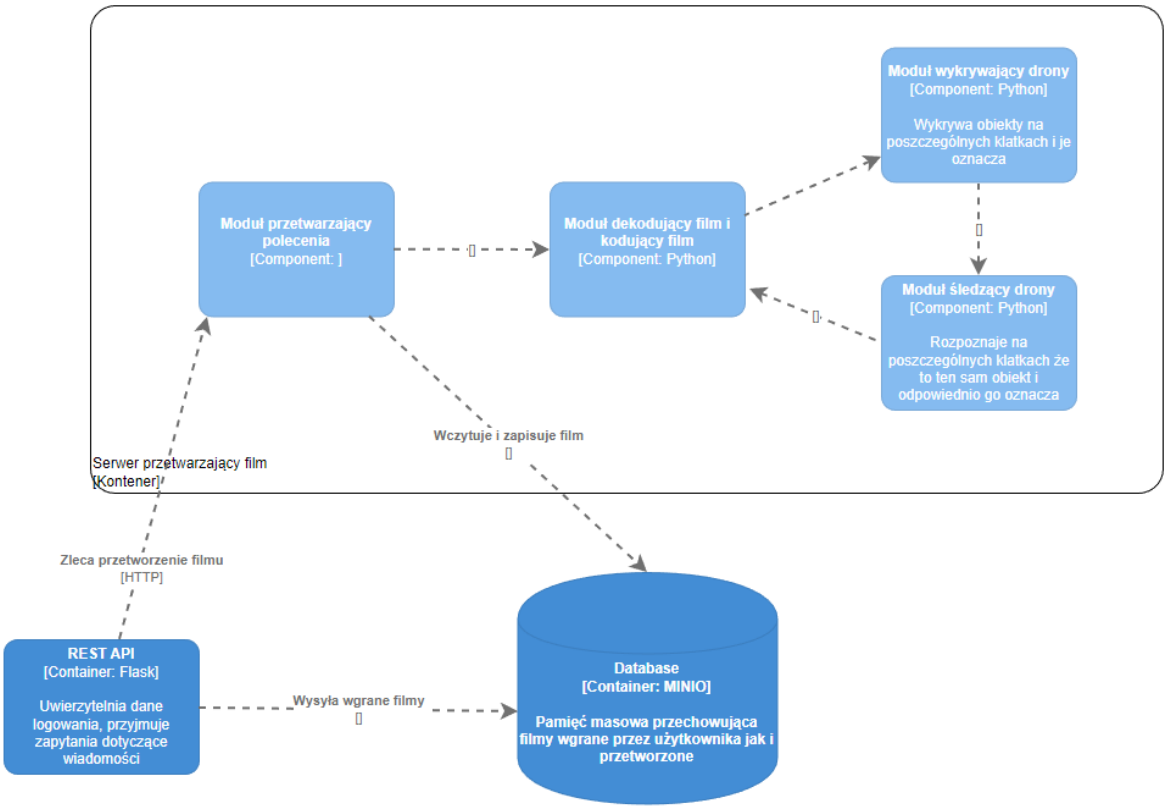
Serwer REST API:



Serwer API składa się z 4 głównych komponentów:

- Moduł obsługujący zapytania - odpowiada za komunikację z klientem i przetwarzanie jego zapytań.
- Moduł uwierzytelniający - odpowiada za uwierzytelnianie sesji.
- Moduł komunikacji z bazą danych - odpowiada za komunikację z bazą danych PostgreSQL.
- Moduł zarządzający filmami - odpowiada za zapisywanie i pobieranie filmów z bazy MinIO oraz komunikuje się z serwerem przetwarzającym filmy w celu pozyskania nagrań z wykrytymi dronami.

Serwer przetwarzający filmy:



Serwer przetwarzający dane składa się z 4 głównych komponentów:

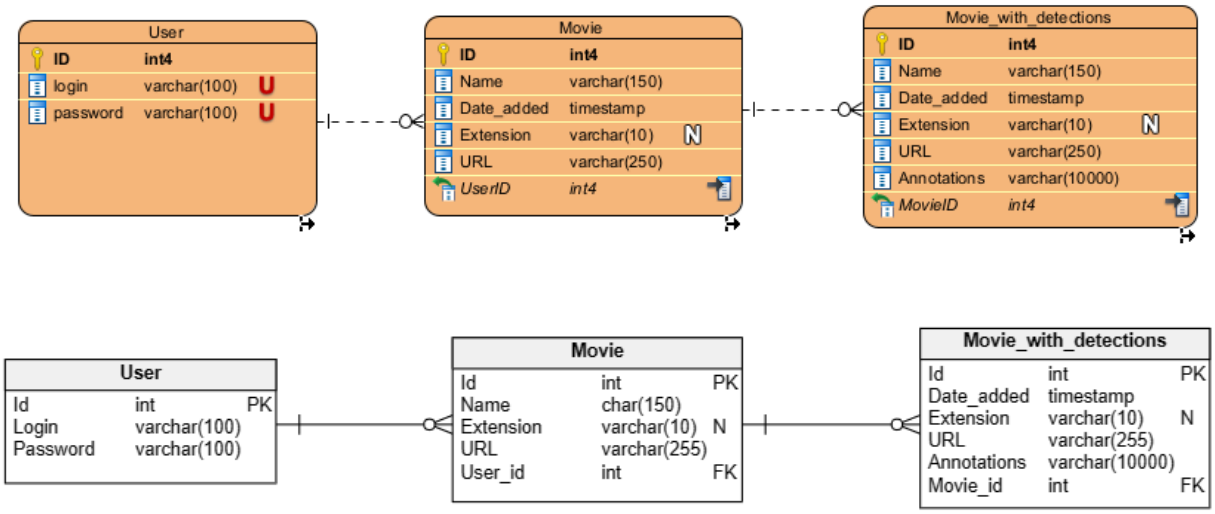
- Moduł przetwarzający polecenia - odpowiada za komunikację z Serwerem API i przetwarzanie jego poleceń. Komunikuje się również z bazą danych MinIO w celu pobierania filmów wgranych przez użytkownika i zapisywania wygenerowanych danych.
- Moduł dekodujący i kodujący filmy - odpowiada za przetwarzanie wideo, m. in. za podział materiału na klatki filmowe oraz za zakodowanie wyjściowego filmu.
- Moduł wykrywający drony - odpowiada za wykrywanie dronów na poszczególnych klatkach filmowych.
- Moduł śledzący drony - odpowiada za śledzenie dronów na podstawie klatek z wykrytymi obiektami.

Zastosowane szablony architektoniczne:

- 3 mikroserwisy: backend, frontend oraz śledzenie obiektów. Komunikacja odbywa się poprzez protokół REST.
- mikroserwisy działają w kontenerach i porozumiewają się z MinIO oraz bazą PostgreSQL
- kontenery są tworzone na serwerze Ubuntu

5 Dane trwałe

5.1 Model logiczny danych



Model danych składa się z trzech tabel: User, Movie, Movie_with_detections.

User:

- przechowuje informacje o użytkownikach, którzy założyli konto w aplikacji
- przechowuje szyfrowane hasło

Movie:

- przechowuje metadane filmów (nie poddanych procesowi detekcji), które znajdują się na koncie użytkownika
- jedną z kolumn jest adres URL do miejsca, w którym przechowywana jest zawartość filmu
- na klucz obcy User_id został nałożony indeks

Movie_with_detections:

- przechowuje metadane filmów, które są wyjściem modelu sieci neuronowej, czyli zawierają na klatkach filmu zaznaczone detekcje dronów
- również znajduje się adres URL
- jedną z kolumn są adnotacje, które są typem string o dużej maksymalnej długości, gdyż przechowują adnotacje (koordynaty drona) do wykrytych obiektów z klatek filmu
- na klucz obcy Movie_id został nałożony indeks

5.2 Przetwarzanie i przechowywanie danych

Do przechowywania danych o użytkownikach i plikach wideo zostanie wykorzystany postgreSQL.

Serwer komunikować się będzie z bazą danych przy użyciu SQLAlchemy.

Do przechowywania plików wideo będziemy korzystać z minIO.

6 Specyfikacja analityczna i projektowa

Odnośnik do repozytorium kodu:

https://gitlab-stud.elka.pw.edu.pl/lglowka1/grupasledcza

Języki programowania: Python, JavaScript

Frameworki: Flask, React

Moduły: PyTorch, OpenCV, Ultralytics

Środowisko programowania: VSC

Diagram klas modułu dotyczącego trackingu:

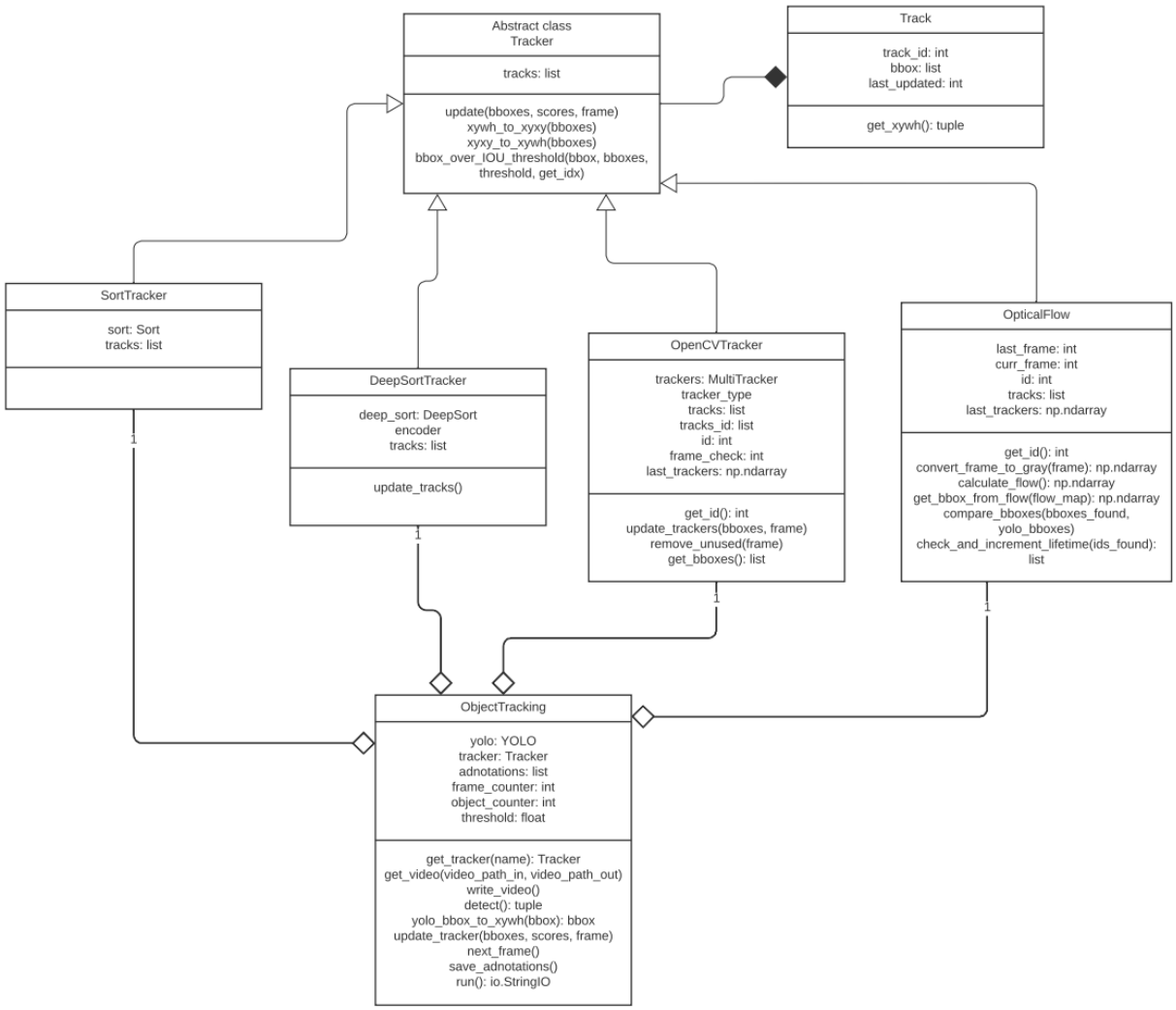


Diagram klas modułu yolo:

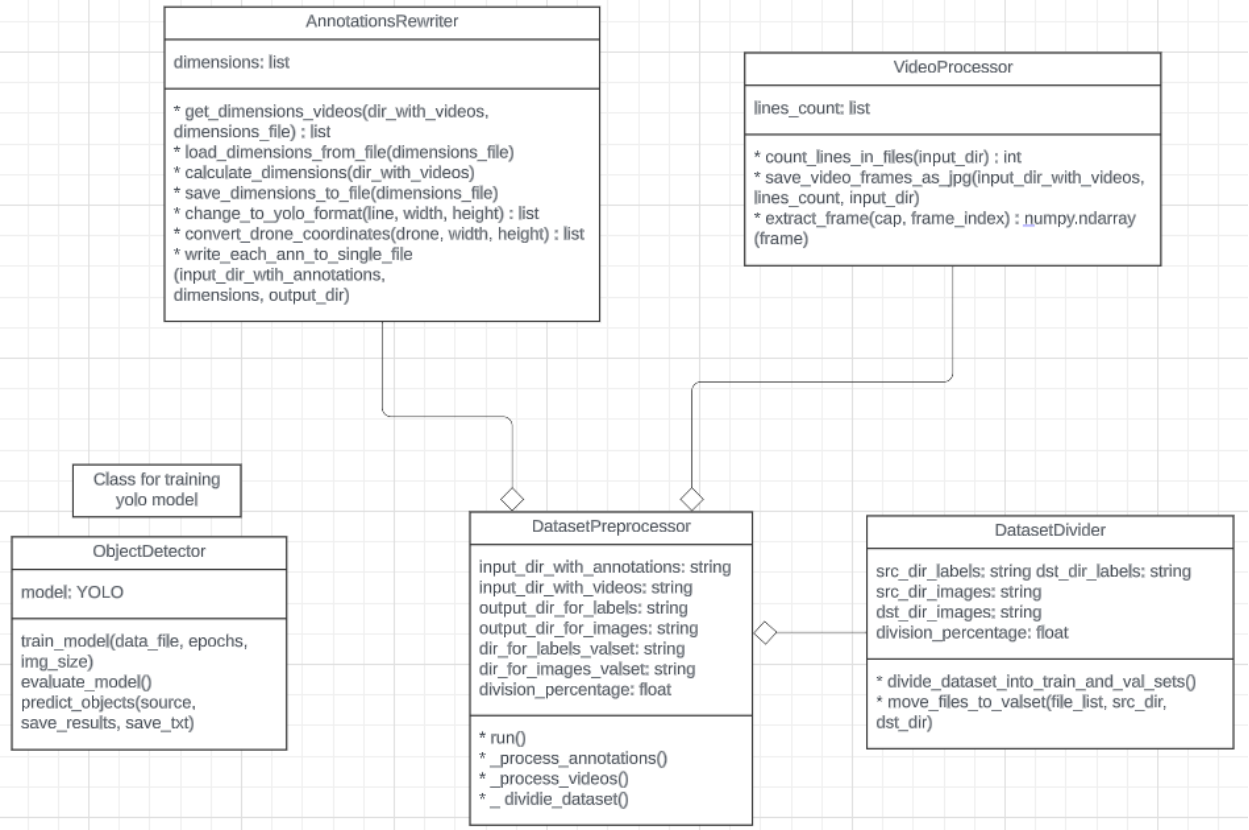
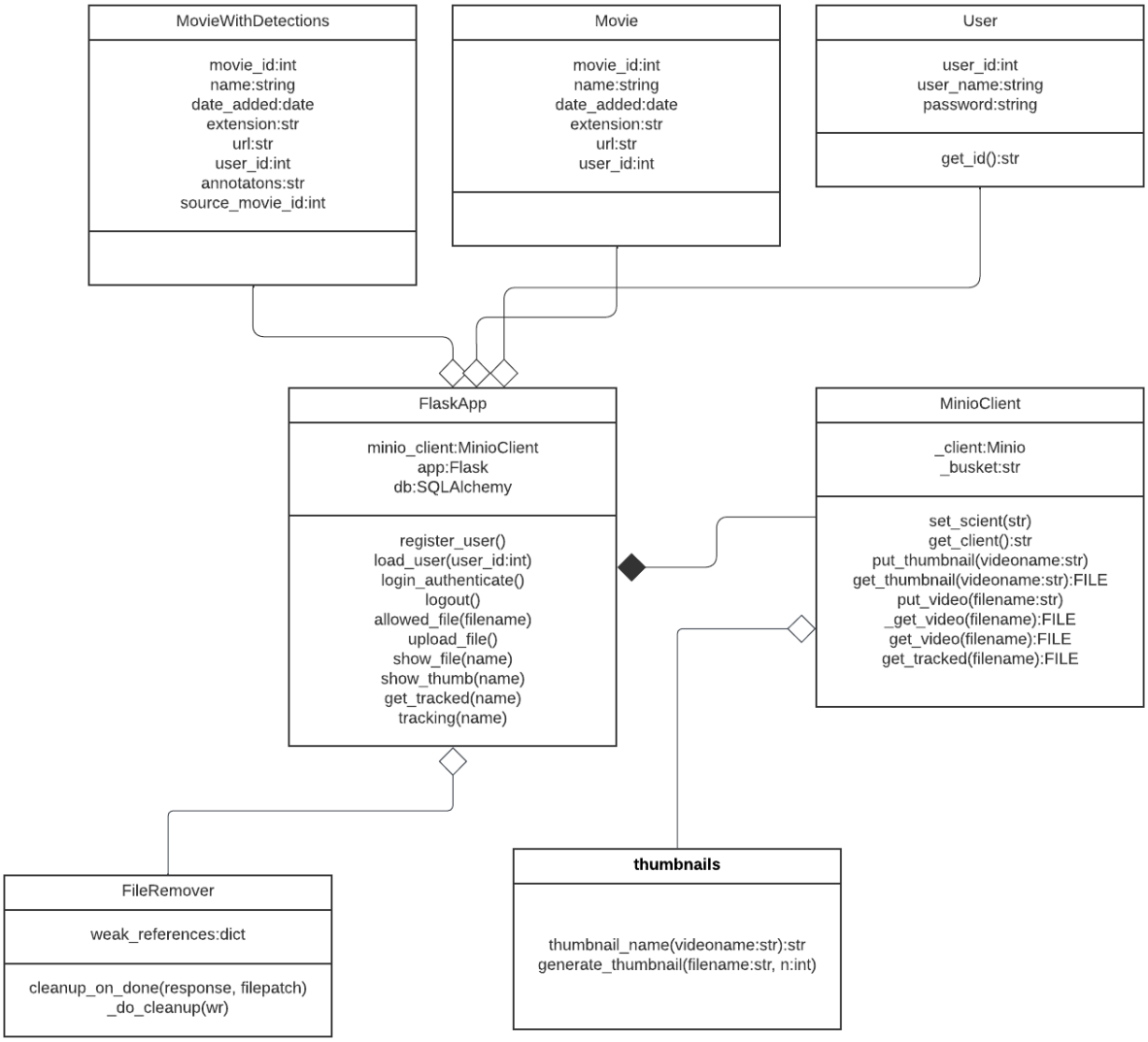


Diagram klas backendu:



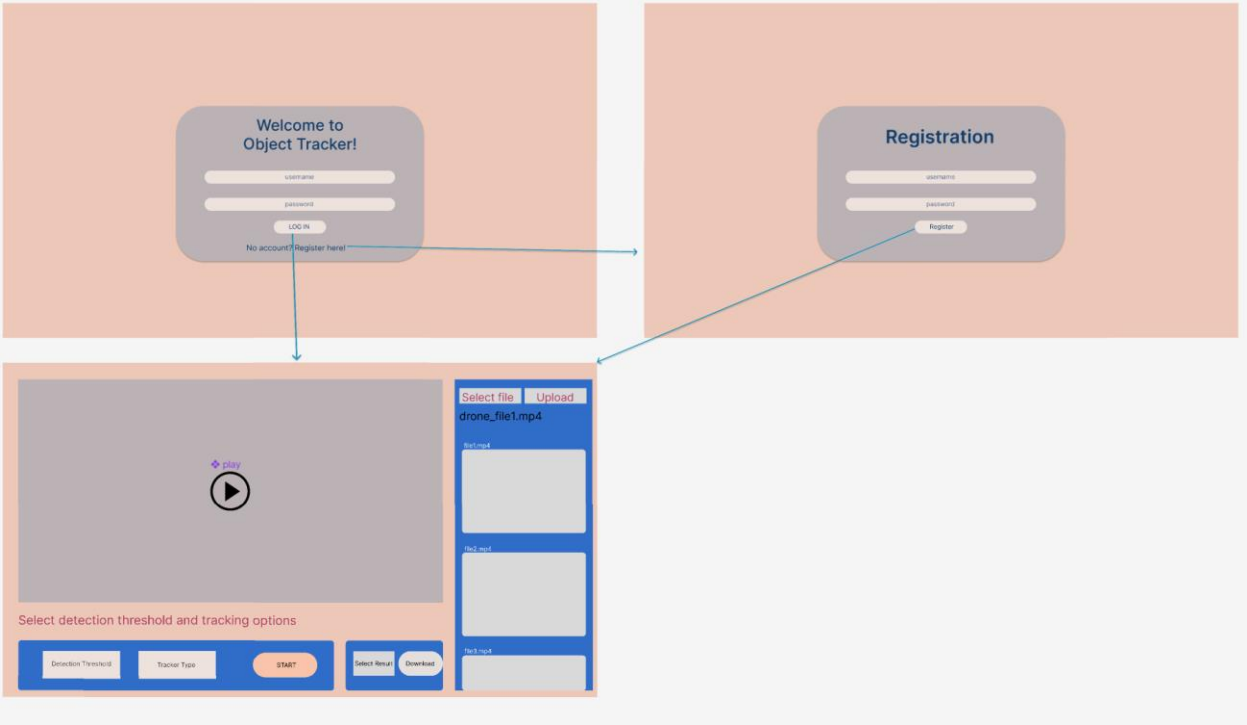
———————

Liczba plików 46

Liczba linii kodu ok. 3000

7 Projekt standardu interfejsu użytkownika

Prototyp interfejsu został zamodelowany w programie Figma. Przedstawia 3 ekrany, które występują w aplikacji. Logowanie i rejestracja posiadają podobny interfejs i przekierowują do głównego ekranu. Aby zapewnić najszybszy dostęp do plików, miniaturki są pokazywane z prawej strony. Opcje wykrywania i śledzenia są umieszczone pod załadowanym filmem, obok przycisku pobierania rezultatów.



Interfejs powstał na podstawie przeanalizowania aktorów, ich historii oraz use case-ów.

Aktorem w naszym systemie jest klient zainteresowany metodami śledzenia i wykrywania obiektów. To on korzysta z wszystkich funkcji systemu.

Historie:

- klient wybiera film z dronem i porównuje skuteczność detekcji obiektu przez zmienianie współczynnika wykrycia
- klient wybiera film z dronem i porównuje skuteczność śledzenia obiektu na podstawie różnych metod śledzenia
- klient wgrywa film i sprawdza poprawność śledzenia obiektu przez metodę optical flow
- klient wgrywa film i pobiera adnotacje w formie .txt lub .json
- klient wgrywa film ze swoim dronem i pobiera film z dronem zaznaczonym ramką

USE CASE	Śledzenie poruszających się dronów
Opis	Klient wybiera oraz wgrywa film, który przedstawia poruszającego się drona. Używa aplikacji, aby uzyskać film z zaznaczonym dronem.

Aktor	Klient
Etapy	1. Zalogowanie/Zarejestrowanie się użytkownika 2. Wybranie oraz załadowanie filmu z dronem 3. Wybranie progu wykrywania. 4. Wybranie metody śledzenia dla wolnego lotu. 5. Obejrzenie wyniku śledzenia i zweryfikowanie wyniku. 6. Pobranie filmu z zaznaczonymi ramkami oraz pliku z adnotacjami.
Wariacje etapów	Wybierana metoda śledzenia drona może się różnić. W przypadku szybko poruszającego się drona najlepsze wyniki zwrócą metody śledzenia obiektu po cechach i wzorcach poruszania się.
Warunek	Chęć założenia/posiadanie konta użytkownika.

8 Specyfikacja testów

Rodzaje testów:

W ramach testów aplikacji przewidziane były następujące rodzaje testów:

Testy jednostkowe: Testy skupiające się na testowaniu poszczególnych komponentów aplikacji, takich jak funkcje, klasy czy moduły. Do testowania modułu zajmującego się detekcją i śledzeniem dronów wykorzystywane zostało narzędzia *unittest*.

Testy integracyjne: przeprowadzono z wykorzystaniem metody bottom-up. Początkowo testowano moduł przetwarzający flm przetwarzając kilka filmów. Następnie zintegrowano go z backendem testując po kolei wszystkie zapytania REST API. następnie backend zintegrowano z bazą danych. Na koniec cały system zintegrowano z frontendem. Potwierdzenie integracji systemu uzyskano poprzez ręczne przejście przez wszystkie scenariusze użycia z rozdziału 3.3.

Testy systemowe: W celu jak najlepszego odwzorowania środowiska produkcyjnego aplikację uruchomiono na maszynie z systemem Linux-Ubuntu 22.04 LTS. Podczas testów wykonano scenariusze testowe z rozdziału 3.3. Aplikacja spełniała swoje podstawowe funkcjonalności zawarte w założeniach. Dokładniejsze testy sprawdzające sytuacje wyjątkowe nie zostały przeprowadzone z powodu krótkiego czasu na wdrożenie rozwiązania.

Opis sposobu realizacji poszczególnych rodzajów testów

Testy jednostkowe:

Testy jednostkowe w aplikacji są realizowane przy użyciu frameworka unittest, który umożliwia strukturalne i automatyczne testowanie poszczególnych komponentów aplikacji. Dla każdej klasy, funkcji lub modułu w aplikacji tworzony jest oddzielny plik testowy, który zawiera zestaw testów jednostkowych odpowiadających poszczególnym funkcjonalnościom.

Struktura testów jednostkowych w aplikacji wygląda następująco:

1. Tworzenie oddzielnego pliku testowego dla każdej klasy, funkcji lub modułu. Na przykład, jeśli mamy klasę *AnnotationsRewriter* w pliku *annotations_rewriting.py*, to tworzymy plik testowy o nazwie *test_annotations_rewriting.py*.
2. Importowanie modułu unittest oraz innych potrzebnych modułów lub klas do testowanego pliku testowego.
3. Tworzenie klasy testowej dziedziczącej po klasie *unittest.TestCase*. Ta klasa będzie zawierać metody testowe.
4. W metodzie setUp inicjalizujemy niezbędne zasoby, takie jak pliki tymczasowe czy instancje klasy, które będą wykorzystywane w testach.
5. W metodzie tearDown usuwamy utworzone zasoby, aby przywrócić początkowy stan środowiska (o ile konieczne).
6. Dla każdej funkcji, klasy lub modułu tworzymy oddzielne metody testowe, które będą testować różne scenariusze i przypadki użycia. W metodach testowych wykorzystujemy asercje (np. *assertEqual, assertTrue, assertFalse* z modułu *unittest* czy też *np.testing.assert_array_equal* z modułu *numpy*) do porównywania oczekiwanych wyników z rzeczywistymi wynikami.
7. Jeśli testowana funkcjonalność wymaga mockowania zależności, takich jak funkcje zewnętrzne lub obiekty, można skorzystać z *MagicMock* z modułu *unittest.mock*. *MagicMock* pozwala na tworzenie atrap obiektów i kontrolowanie ich zachowania w celu przetestowania konkretnych scenariuszy. Na przykład, w przypadku testu jednostkowego klasy *DatasetDivider* (Detection\Yolo\tests\test_dataset_division.py), możemy zastosować *MagicMock* dla funkcji *random.sample*, aby kontrolować jej zwracane wartości i przetestować konkretny scenariusz.
8. Uruchomienie testów przy użyciu metody unittest.main(), która automatycznie znajduje i wykonuje wszystkie testy w pliku testowym.

Przykład struktury plików testowych:

- yolo/
    - annotations_rewriting.py
    - tests/
        - test_annotations_rewriting.py

Miary jakości testów

Wszystkie przeprowadzone testy zostały zakończone sukcesem.

Pokrycie kodu - W projekcie został wykorzystany moduł *coverage* (instalacja poprzez wykonanie komendy *pip install coverage*), który umożliwia analizę pokrycia kodu. Dzięki temu mogliśmy pozyskać informację na temat liczby obszarów kodu, które zostały wykonane podczas uruchamiania testów.

Aby zbadać pokrycie kodu, w folderze, w którym znajdują się pliki źródłowe oraz folder tests/ z plikami testowymi, wywołaliśmy taką komendę:
python3 -m coverage run -m unittest discover tests/

Po zakończeniu uruchomienia testów z *coverage*, mogliśmy wygenerować raport pokrycia kodu za pomocą polecenia:
python3 -m coverage html

Aby móc wygenerować taki raport, w folderze, w którym wykonywana jest powyższa komenda musza znajdować się pliki *config-3.py* oraz *config.py* (w naszym przypadku są to po prostu puste pliki).

Polecenie to tworzy specjalny folder, w którym znajduje się między innymi plik index.html, w którym zapisany jest raport z pokrycia kodu. Znajdują się tam również pliki html odnoszące się do poszczególnych testowanych plików, w których można zobaczyć, które linijki kodu zostały pominięte w testach. Na poniższych obrazach można zaobserwować takie raporty z wykonanych testów jednostkowych:

Coverage report: 95%

coverage.py v7.2.7, created at 2023-06-02 20:48 +0200

Module	statements	missing	excluded	coverage
annotations_rewriting.py	80	12	0	85%
config-3.py	0	0	0	100%
config.py	0	0	0	100%
dataset_division.py	25	0	0	100%
tests\test_annotations_rewriting.py	67	1	0	99%
tests\test_dataset_division.py	53	1	0	98%
tests\test_videos_to_frames.py	48	1	0	98%
videos_to_frames.py	36	1	0	97%
Total	**309**	**16**	**0**	**95%**

coverage.py v7.2.7, created at 2023-06-02 20:48 +0200

Detection\Trackers\deep_sort_tracker.py	45	1	0	98%
Detection\Trackers\opencv_trackers.py	57	3	0	95%
Detection\Trackers\optical_flow.py	77	17	0	78%
Detection\Trackers\tracker.py	42	1	0	98%

Detection\object_tracking.py	79	20	0	75%
Detection\Yolo\annotations_rewriting.py	80	12	0	85%
Detection\Yolo\dataset_division.py	25	0	0	100%
Detection\Yolo\videos_to_frames.py	36	1	0	97%

9 Wirtualizacja/konteneryzacja

Docelowym miejscem uruchamiania systemu jest serwer bigubu. Nasz system będzie się składał z pięciu kontenerów. Wszystkie kontenery będą korzystać z sieci 172.21.121.0/24. Adresy poszczególnych kontenerów będą ustawiane statycznie co znacznie uprości konfigurację poszczególnych elementów systemu.

Kontenery budujące system:

- db - Kontener z bazą danych przechowującej informacje o użytkownikach i metadanych filmu. Kontener oparty na oficjalnym obrazie dockerowym postgresa. Skonfigurowane zostały nazwa użytkownika, hasło, oraz plik generujący tabele.
- minio - Kontener służący jako pamięć masowa przechowującą filmy. Kontener oparty na oficjalnym obrazie dockerowym minio. Skonfigurowane zostały nazwa użytkownika, hasło, wolumin pamięci
- front - Kontener na którym uruchamiany jest frontend stworzony w react. Oparty o własny obraz dockerowy stworzony na podstawie oficjalnego kontenera node-alpine. Podczas budowania instalowane są biblioteki reacta.
- back - Kontener na którym uruchamiany jest backend stworzony we flasku. Oparty o własny obraz dockerowy stworzony na podstawie oficjalnego obrazu pythona3. Podczas tworzenia obrazu instalowanie są niezbędne pakiety takie jak:
    - Flask
    - OpenCV
    - Minio
    - SQLAlchemy
    - psycopg2
- detection - Kontener na którym uruchamiany jest serwer przetwarzający filmy wideo. Kontener oparty o własny obraz stworzony na podstawie oficjalnego obrazu pythona3. Podczas tworzenia obrazu instalowane są niezbędne pakiety takie jak:
    - opencv
    - tensorflow
    - ultralytics
    - scikit-learn
    - Flask
    - minio

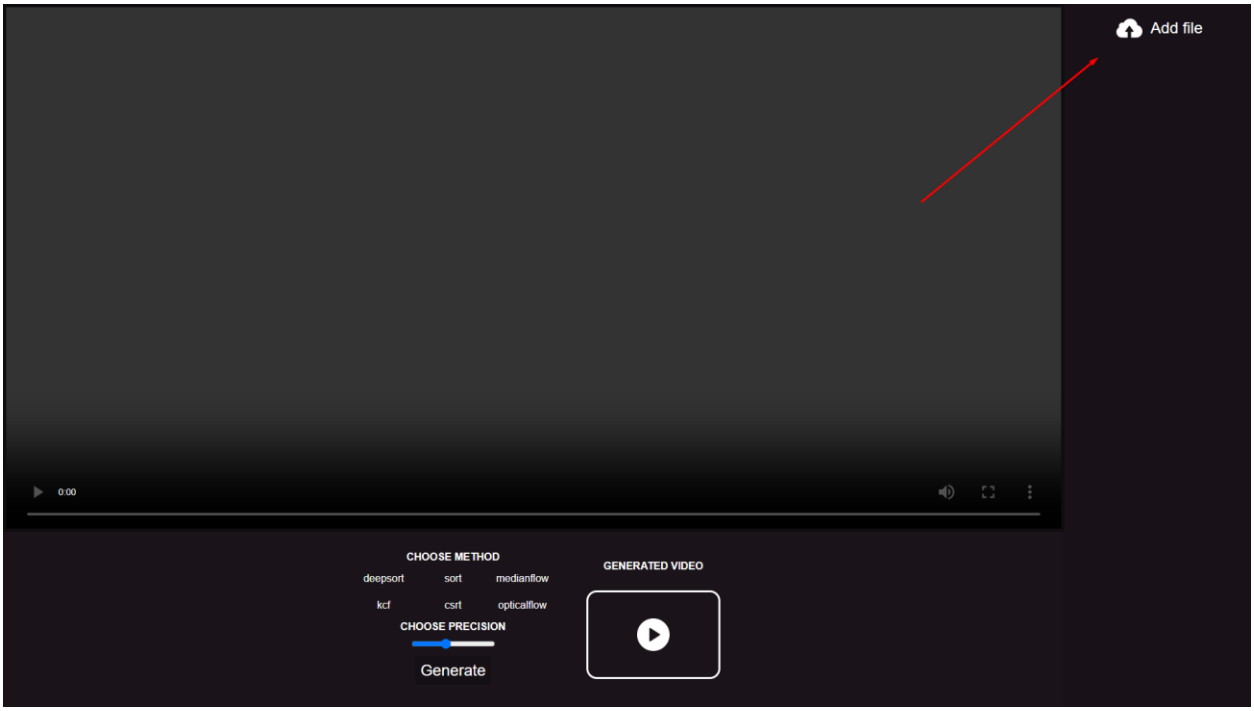
Kontener	adres ip
db	172.21.121.4
minio	172.21.121.6
front	172.21.121.3
back	172.21.121.2
detection	172.21.121.5

10 Bezpieczeństwo

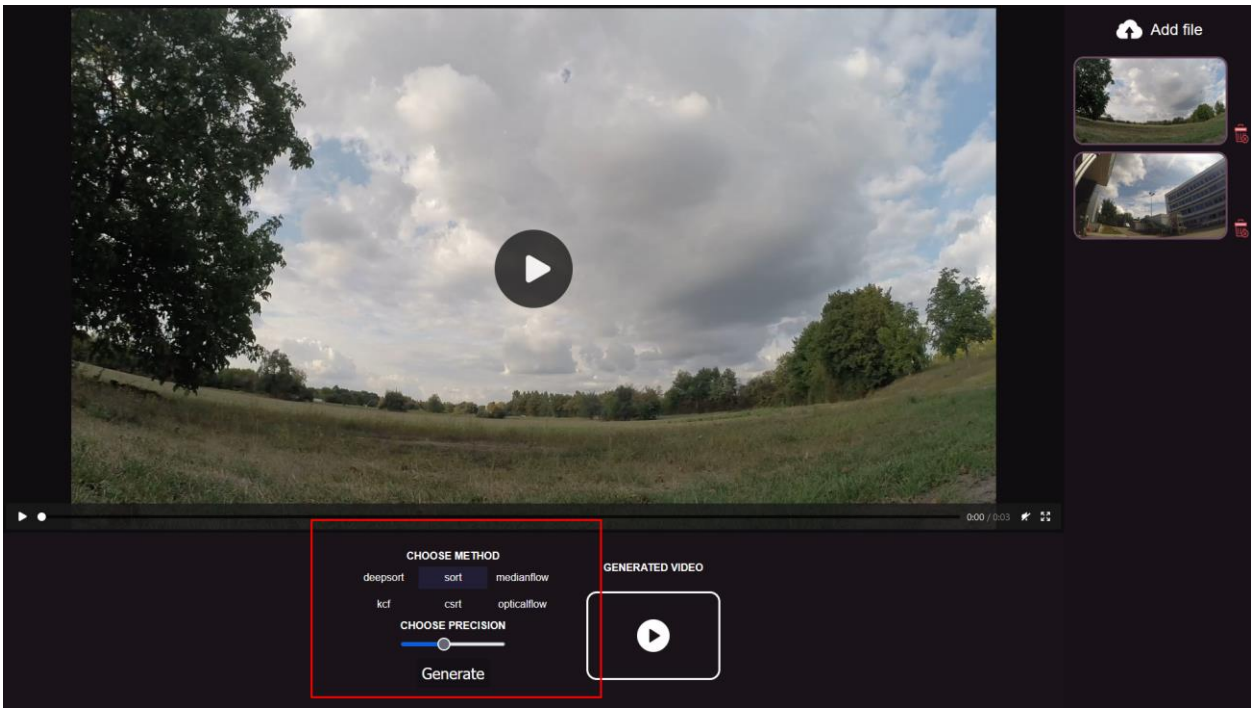
W naszej aplikacji niewiele jest kwestii, które wymagałyby silnych zabezpieczeń. Aby zapewnić podstawowy poziom bezpieczeństwa, dane uwierzytelniania są hashowane po stronie serwera, przed zapisaniem w bazie danych za pomocą biblioteki flask_bcrypt.

11 Podręcznik użytkownika

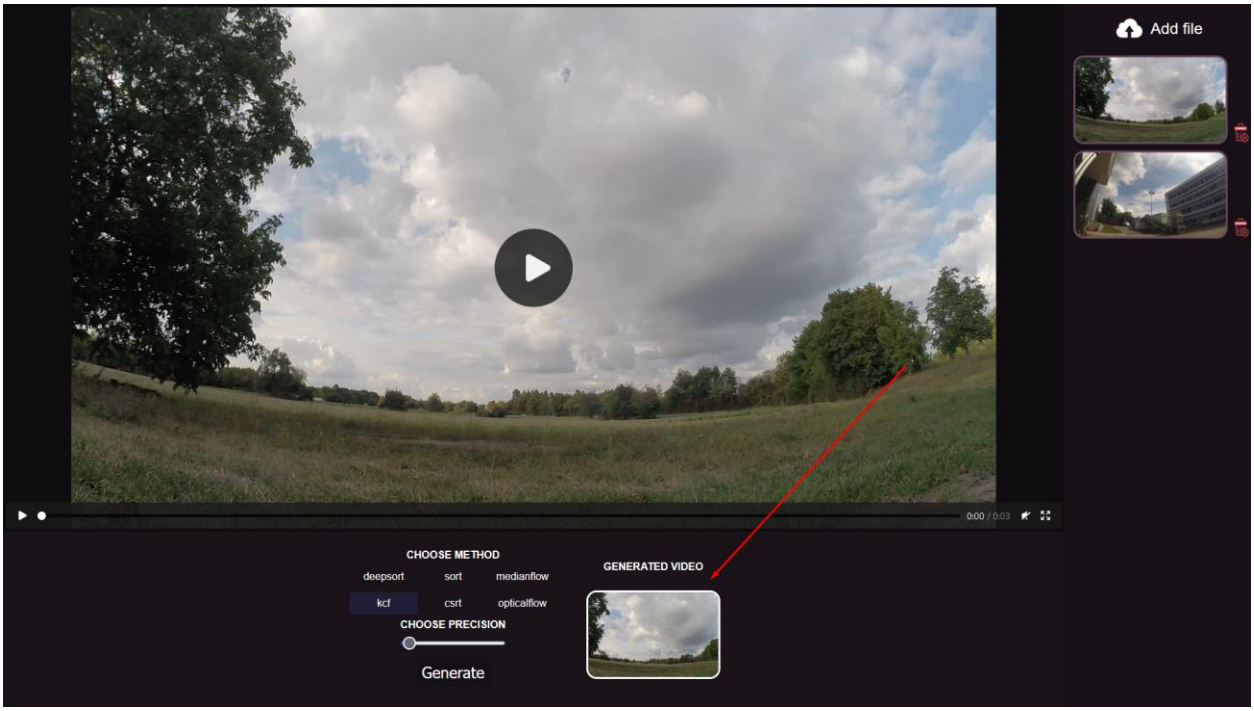
Po wejściu na stronę, zostaniemy poproszeni o zalogowanie się do serwisu. Jeśli nie posiadamy konta, możemy je założyć za pomocą przycisku **Register**. Po zalogowaniu ujrzymy ekran główny:



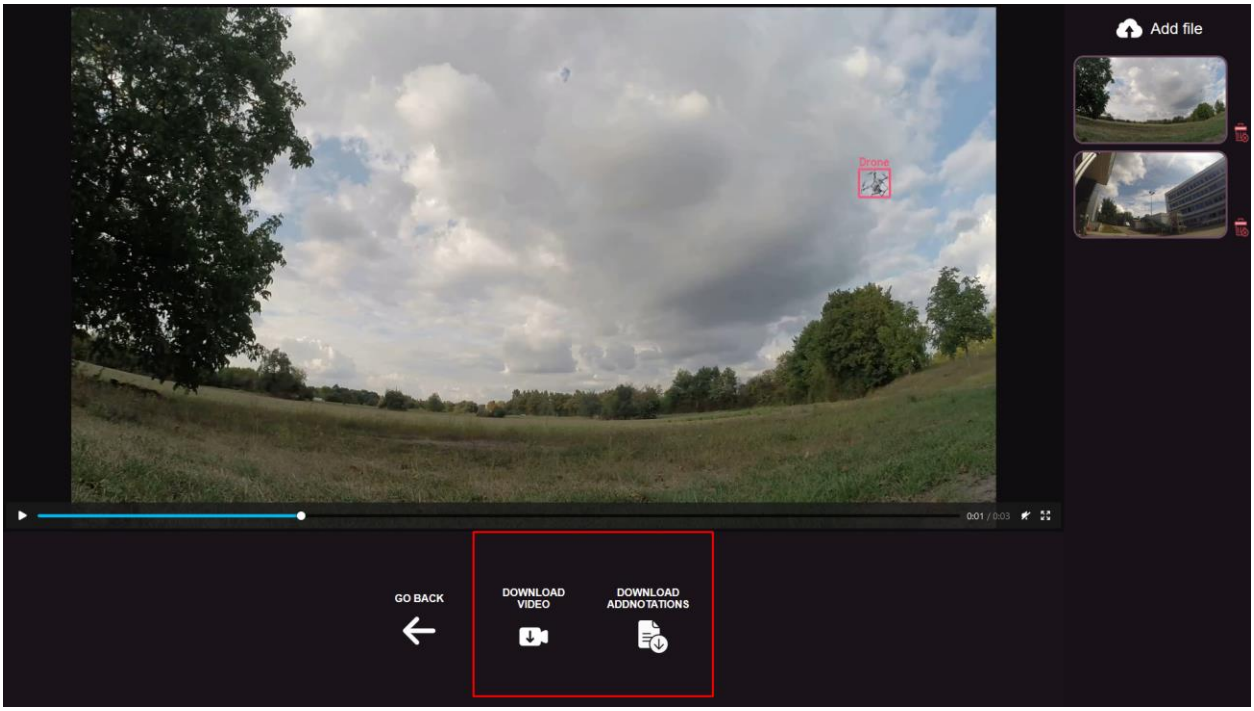
Za pomocą zaznaczonego przycisku **Add file** możemy dodać nasz pierwszy film (lub kolejny). Po dodaniu filmu możemy go obejrzeć. Jeśli chcemy wygenerować film z zaznaczonymi dronami, musimy skorzystać z panelu na dole ekranu.



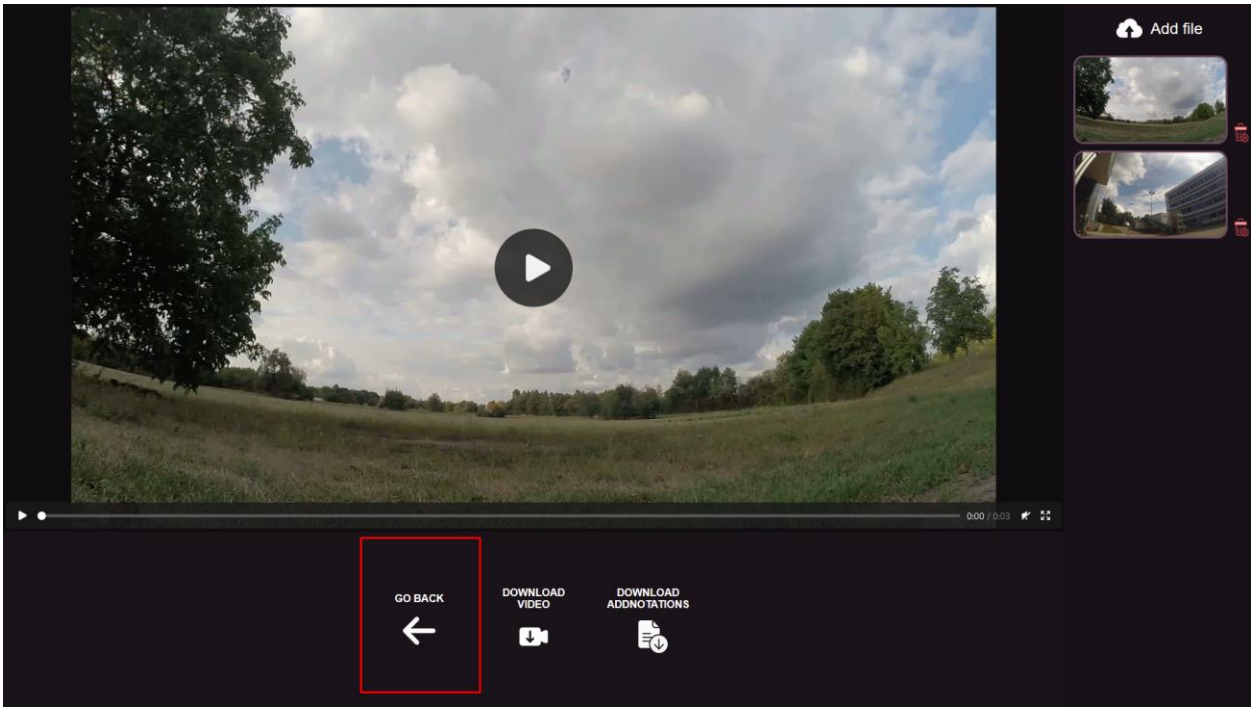
Wybieramy tam docelowy tracker oraz precyzję, a następnie klikamy przycisk **Generate.** Po wygenerowaniu filmu pojawi się obok jego miniaturka.



Po jej kliknięciu pojawi nam się podgląd wygenerowanego filmu. Z tego widoku możemy pobrać wygenerowany film, lub plik z adnotacjami.



Jeżeli chcemy wrócić do podglądu oryginalnego filmu (np. aby wygenerować film z użyciem innego trackera lub z inną precyzją), możemy skorzystać z przycisku **GO BACK.**

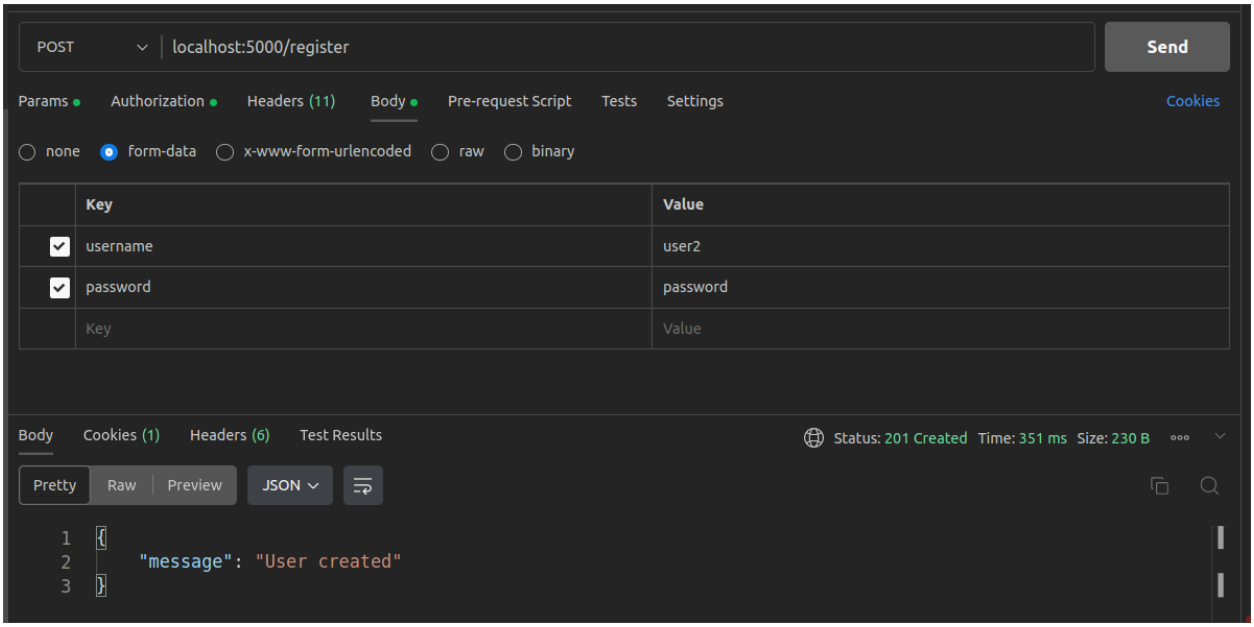


Aby usunąć film z bazy, możemy skorzystać z przycisku kosza na śmieci obok miniaturki.

11.1 REST API

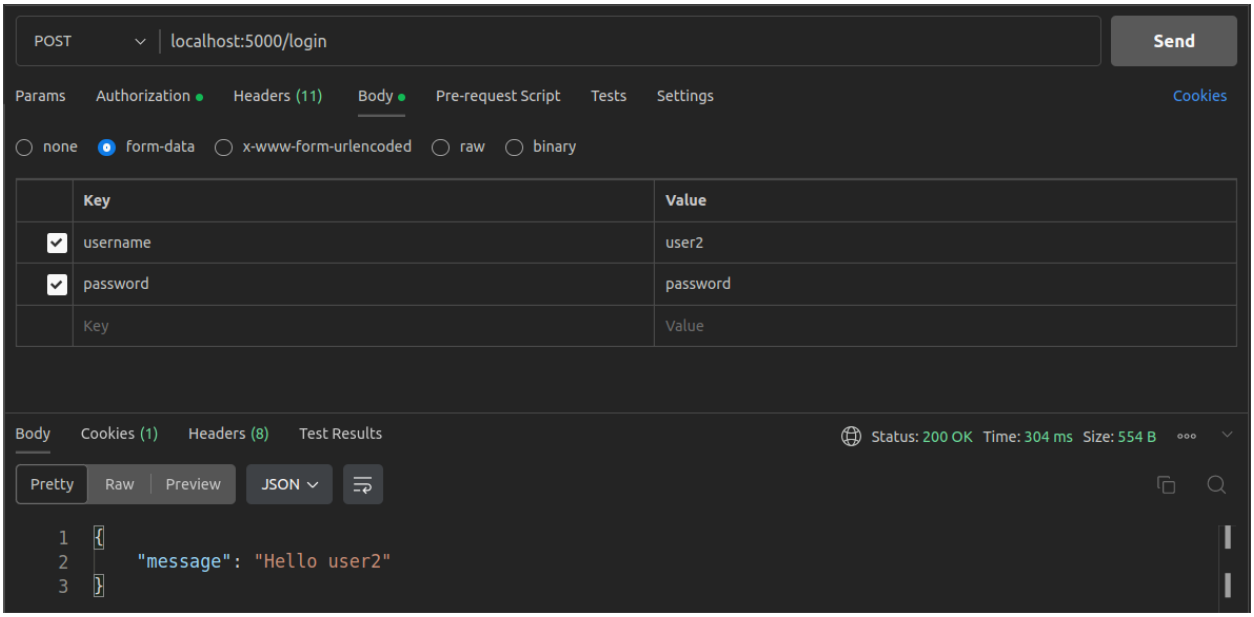
- **POST** `/register` umożliwia zarejestrowanie użytkownika wymaga przesłania loginu `'username'` i hasła `'password'` w body

  Przykład:



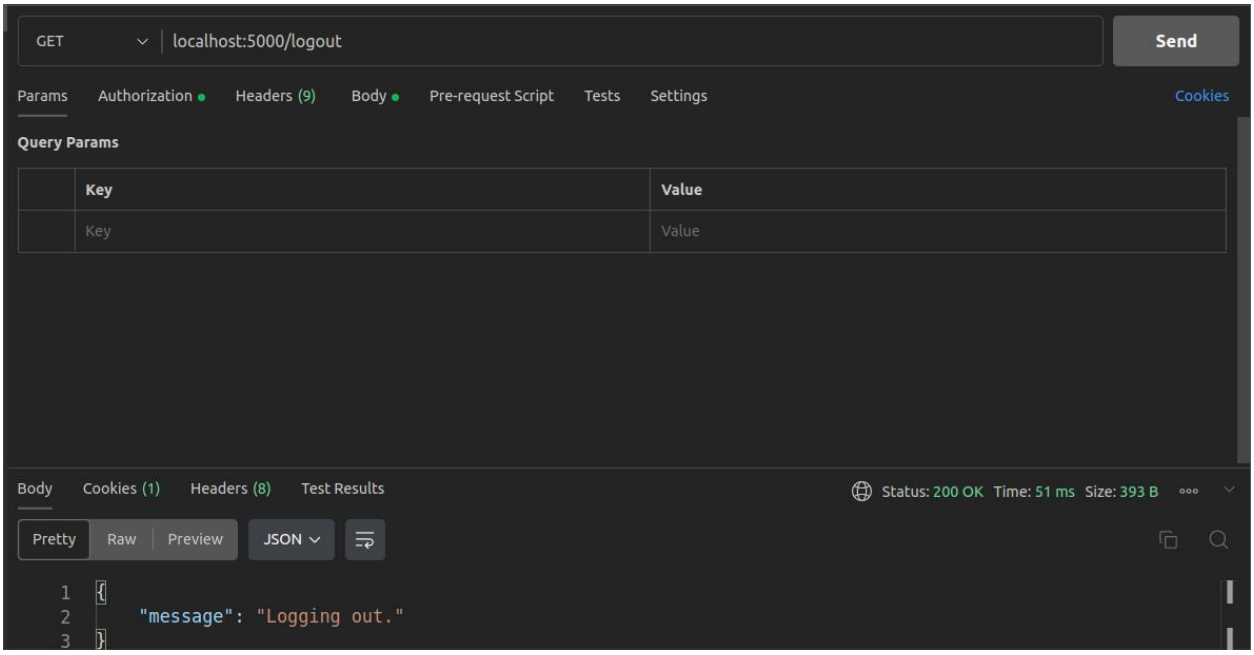
- **POST** `/login` umożliwia zarejestrowanie użytkownika wymaga przesłania loginu `'username'` i hasła `'password'` w body

Przykład:



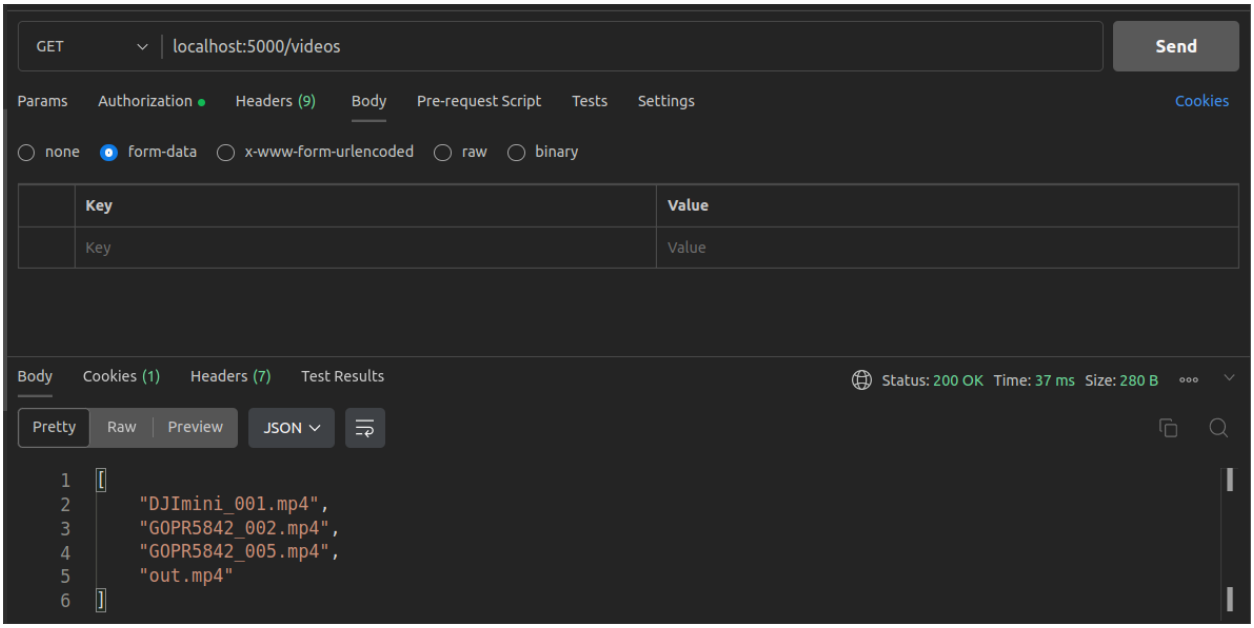
- **GET** `logout/` wylogowuje użytkownika

  Przykład:



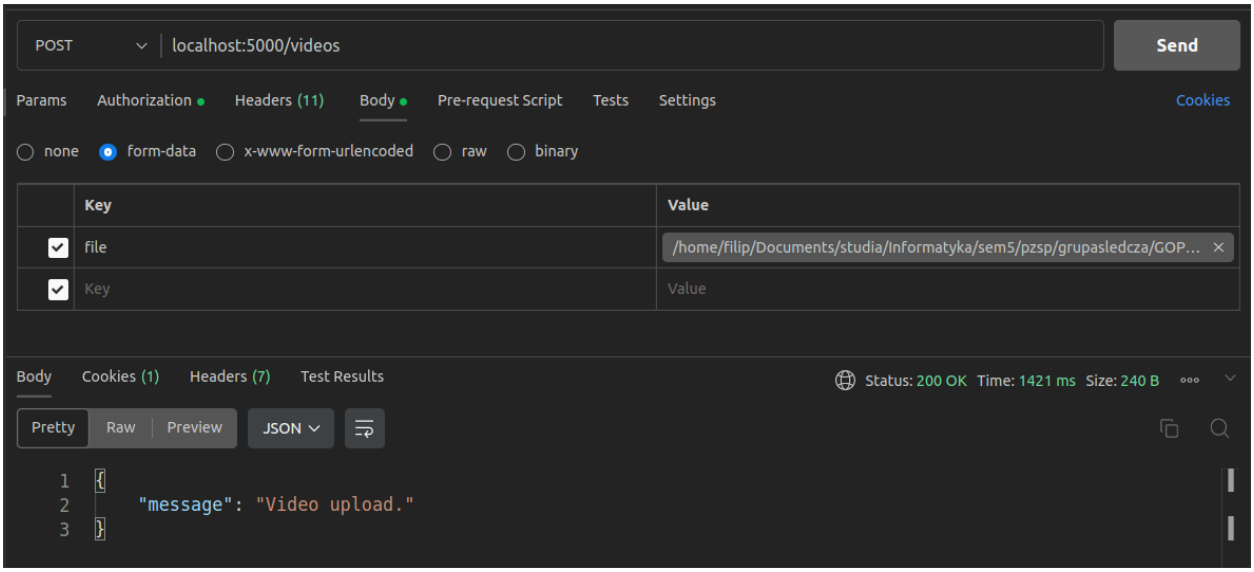
- **GET** `videos/` użytkownik musi być zalogowany. zwraca listę wgranych przez użytkownika filmów wideo w formacie json

  Przykład:



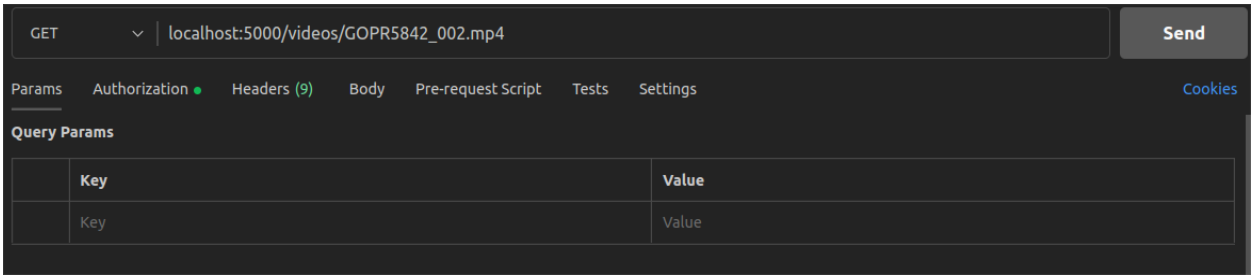
- **POST** `videos/` użytkownik musi być zalogowany. Ciało HTTP zawiera plik wideo oznaczony kluczem “file”

  Przykład:



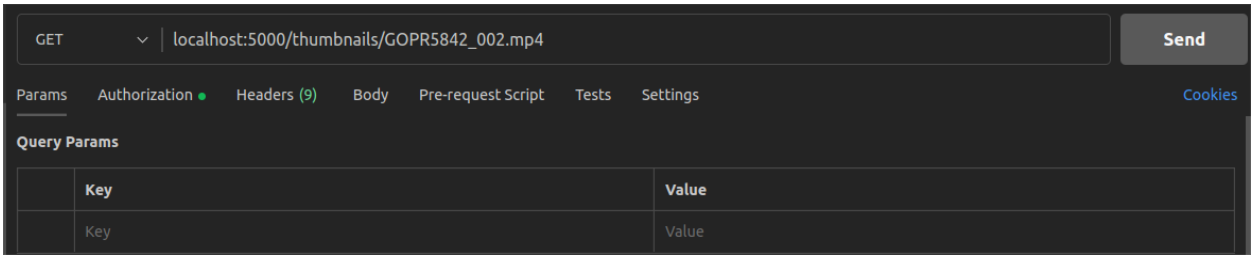
- **GET** `videos/<name>` użytkownik musi być zalogowany. zwraca film podany przez użytkownika, dodatkowo używając parametru as_attachment możemy pobrać film jako załącznik.

  Przykład:



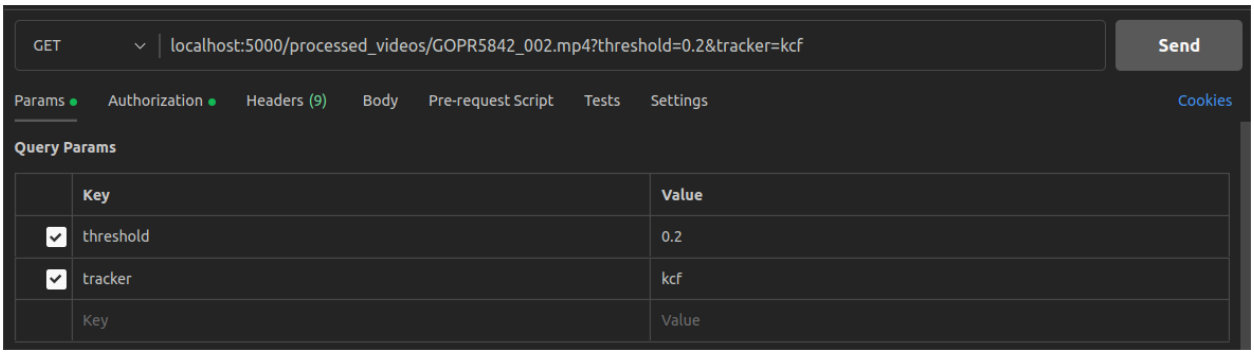
- **GET** `thumbnails/<name>` użytkownik musi być zalogowany. zwraca film podany przez użytkownika,

  Przykład:



- **GET** `processed_videos/<name>` użytkownik musi być zalogowany. Przetwarza film przesłany przez użytkownika. Użytkownik musi wprowadzić następujące parametry
  - threshold
  - tracker

  Przykład:



12 Podręcznik administratora

System budowany jest przy użyciu dockera oraz docker-compose'a. Komenda ***docker-compose up*** wywołana z poziomu repozytorium pozwala na zbudowanie 5 kontenerów, używając zawartych w folderach zależności.

W folderze */var/lib/postgresql/data* przechowywana jest zawartość bazy danych postgreSQL. Zapewnia ona persystencję danych w razie nieoczekiwanego przerwania działania systemu. Zawartość MINIO jest przechowywana w /data.

Strona uruchomionego programu dostępna jest pod adresem http://localhost:3000/.

System nie wymaga konfiguracji po pobraniu repozytorium można go od razu uruchomić.

Aktualizowanie oprogramowania odbywa się na zasadzie pobrania najnowszej wersji repozytorium. Następnie należy w katalogu głównym repozytorium wpisać następujące komendy:

```
docker compose stop

docker compose rm

docker compose build

docker compose up
```

13 Podsumowanie

Słabe strony:

- Jakość detekcji nie jest najlepsza - model nie wykrywa bardzo małych obiektów - w momencie, gdy dron mocno oddala się od kamery (jego rozmiar na klatce filmu zdecydowanie się zmniejsza), model nie potrafi dokonać detekcji. Prawdopodobne przyczyny takiego zjawiska:
  - Nie najlepszy zbiór danych treningowych
  - Sieć mogłaby być lepiej przystosowana do wykrywania małych obiektów
- Czas przetwarzania filmów jest dość długi, co prowadzi do dłuższego oczekiwania użytkownika na rezultaty. Ten wydłużony czas odpowiedzi znacząco wpływa na komfort korzystania z aplikacji, utrudniając płynne użytkowanie aplikacji.

Mocne strony:

- Oferowanie dostępu do przetestowania najnowszych i najbardziej efektywnych metod sztucznej inteligencji służących wykrywaniu i śledzeniu obiektów na obrazie. Umożliwia to użytkownikom zapoznanie się z różnorodnymi zaawansowanymi technikami, które reprezentują czołowe osiągnięcia w dziedzinie.
- Czytelny interfejs, łatwość obsługi aplikacji.
- Sprawna i regularna komunikacja w zespole.
- Zastosowanie dedykowanej bazy danych do przechowywania zdjęć w sposób obiektowy (MinIO).
- Bezpieczne logowanie - haszowane dane.

Możliwe kierunki rozwoju:

- Praca nad bardziej efektywnym treningiem modelu wykrywania dronów. Potencjalnym kierunkiem rozwoju mógłby być treningu na innym zbiorze danych lub przyjęcie innych parametrów treningowych.
- Rozszerzenie architektury sieci neuronowej YOLOv8 np. dodając moduł MDC do operacji zmniejszenia próbkowania, który zachowuje informacje o małych obiektach podczas ekstrakcji cech, ulepszyć metodę fuzji cech czy dodać moduł DC.
- Na maszynie, na której uruchamiana jest aplikacja nie udało się nam uzyskać dostępu do karty graficznej (lub też możliwe, że maszyna nie ma dedykowanej karty graficznej). Gdybyśmy mogli przenieść obliczenia sieci neuronowej na kartę graficzną (np. z wykorzystaniem narzędzia CUDA), prawdopodobnie wyniki czasowe pracy modelu byłyby nieco lepsze.
- Aby jeszcze bardziej rozbudować funkcjonalność aplikacji, można by wprowadzić możliwość przetwarzania filmów w czasie rzeczywistym. Taka funkcja znalazłaby szerokie zastosowanie w różnych systemach, które wymagają ciągłej analizy ruchu w powietrzu oraz wykrywania pojawiających się obiektów.

14 Bibliografia

https://github.com/wosdetc/challenge

https://github.com/nwojke/deep_sort

https://github.com/abewley/sort

https://github.com/ultralytics/ultralytics

https://coverage.readthedocs.io/en/7.2.7/

https://pl.legacy.reactjs.org/tutorial/tutorial.html

https://docs.opencv.org/4.x/

https://flask.palletsprojects.com/en/2.3.x/