

POLITECHNIKA WROCŁAWSKA
Wydział Informatyki i Telekomunikacji
Kierunek: IST

ZESPOŁOWE PRZEDSIEWZIĘCIE INFORMATYCZNE

**LarvixON AI – diagnostyka toksyczności
osocza z wykorzystaniem analizy
behawioralnej larw**

Autorzy:

Mikołaj Kubś
Krzysztof Kulka
Martyna Łopianiak
Patryk Łuszczek

Opiekun pracy:

dr inż. Natalia Piórkowska

Słowa kluczowe: ML, AI, Web Application, Full Stack

WROCŁAW 2025

Spis treści

1 Wykaz symboli, oznaczeń i akronimów	4
2 Cel i zakres przedsięwzięcia	5
3 Słownik pojęć	5
4 Stan wiedzy w obszarze przedsięwzięcia	5
4.1 Analiza literatury i badania wstępne	6
5 Założenia wstępne	6
5.1 Grupa docelowa i środowisko operacyjne	7
5.2 Założenia technologiczne i sprzętowe	7
5.3 Założenia dotyczące danych wejściowych (wideo)	7
5.4 Założenia prawne i bezpieczeństwo danych	8
6 Specyfikacja wymagań na produkt programowy	9
6.1 Wymagania funkcjonalne	9
6.1.1 Epic 1: Zarządzanie użytkownikiem	9
6.1.2 Epic 2: Zarządzanie analizami	9
6.1.3 Epic 3: Informacja o systemie	10
6.2 Wymagania niefunkcjonalne	10
6.2.1 Wydajność i skalowalność	10
6.2.2 Bezpieczeństwo i ochrona danych	10
6.2.3 Użyteczność i kompatybilność	10
6.2.4 Niezawodność i obsługa błędów	11
6.3 Realizacja wymagań i zarządzanie projektem	11
7 Projekt produktu programowego	12
7.1 Backend	12
7.1.1 Architektura systemu i stos technologiczny	12
7.1.2 Struktura modułowa	14
7.1.3 Projekt Interfejsu Programistycznego (API)	15
7.1.4 Logiczna architektura warstwowa	15
7.1.5 Schemat bazy danych	16
7.2 Interfejs użytkownika	18
7.2.1 Architektura systemu i stos technologiczny	18
7.2.2 Zastosowane podejścia projektowe	20
7.2.3 Zastosowane wzorce architektoniczne	20
7.2.4 Zastosowane wzorce projektowe	21

7.2.5	Prototyp interfejsu	21
7.3	Model uczenia maszynowego do predykcji	27
7.3.1	Architektura hybrydowa i wdrożenie	27
7.3.2	Preprocessing i ekstrakcja danych wideo	27
7.3.3	Trening iteracyjny i zarządzanie modelem	29
7.3.4	Serwis predykcyjny (FastAPI) i wdrożenie modelu	31
7.4	Serwis pacjentów	33
7.4.1	API serwisu	33
7.4.2	Działanie serwisu	34
7.4.3	Deployment	35
7.4.4	Architektura	35
7.5	Symulacja	35
7.5.1	Struktura i zewnętrzne biblioteki	35
7.5.2	Namespaces	35
7.5.3	Zastosowane wzorce projektowe	36
7.6	Strona z dokumentacją	37
8	Implementacja	37
8.1	Środowisko wytwórcze i proces kontroli wersji	37
8.1.1	Strategia branchowania i Code Review	37
8.1.2	Ciągła integracja (CI/CD)	38
8.2	Backend	38
8.2.1	Struktura projektu i organizacja kodu	38
8.2.2	Konfiguracja środowiska uruchomieniowego	40
8.2.3	Asynchroniczne przetwarzanie i integracja z modelem ML	41
8.2.4	Generowanie raportów	42
8.2.5	Automatyczna dokumentacja API	43
8.2.6	Testy	43
8.3	Interfejs użytkownika	45
8.3.1	Implementacja widoków i modułów	45
8.3.2	Obsługa błędów przy komunikacji z API	57
8.3.3	Zarządzanie stanem	60
8.3.4	Automatyczna dokumentacja kodu	63
8.3.5	Testy akceptacyjne i z udziałem użytkowników	64
8.4	Implementacja modelu uczenia maszynowego (Larvixon ML)	65
8.4.1	Architektura Modelu i Warstwy	65
8.4.2	Trening, Walidacja i Wnioskowanie	66
8.5	Implementacja symulacji	67
8.5.1	Algorytm fizycznego ruchu larw	67
8.5.2	Wyliczenie wartości modyfikatorów narkotyków	68

9 Demonstracja produktu programowego	68
9.1 Przykładowy workflow wykonywania analizy	68
10 Dodatkowe materiały online	72

1 Wykaz symboli, oznaczeń i akronimów

Akronim	Definicja
AI	<i>Artificial Intelligence</i> – sztuczna inteligencja.
API	<i>Application Programming Interface</i> – interfejs programowania aplikacji, zestaw reguł komunikacji między systemami komputerowymi.
BLoC	<i>Business Logic Component</i> – wzorzec projektowy oddzielający logikę biznesową od warstwy prezentacji, popularny w ekosystemie Flutter.
CI/CD	<i>Continuous Integration / Continuous Delivery</i> – praktyki inżynierii oprogramowania polegające na częstym integrowaniu zmian w kodzie, automatycznym wdrażaniu aplikacji i automatycznych testach.
CLI	<i>Command Line Interface</i> – interfejs wiersza poleceń.
CNN	<i>Convolutional Neural Network</i> – konwolucyjna sieć neuronowa, typ sieci głębokiej specjalizujący się w przetwarzaniu obrazów.
DTO	<i>Data Transfer Object</i> – prosty obiekt służący do przesyłania danych między procesami lub warstwami aplikacji.
FHIR	<i>Fast Healthcare Interoperability Resources</i> – standard wymiany danych medycznych stworzony przez organizację HL7.
GPU	<i>Graphics Processing Unit</i> – procesor graficzny, wykorzystywany w projekcie do przyspieszania obliczeń modelu ML.
GUID/UUID	<i>Globally Unique Identifier/Universally Unique Identifier</i> – uniwersalny identyfikator zapisany jako 128-bitowa liczba.
HL7	<i>Health Level Seven</i> – zbiór międzynarodowych standardów wymiany informacji medycznych.
HTTP	<i>Hypertext Transfer Protocol</i> – protokół przesyłania dokumentów hipertekstowych, podstawa komunikacji w WWW.
JWT	<i>JSON Web Token</i> – standard bezpiecznego przesyłania informacji między stronami jako obiekt JSON, używany do autoryzacji.
LSTM	<i>Long Short-Term Memory</i> – typ rekurencyjnej sieci neuronowej (RNN) zdolny do uczenia się długoterminowych zależności w danych sekwencyjnych.
ML	<i>Machine Learning</i> – uczenie maszynowe.
ORM	<i>Object-Relational Mapping</i> – technika mapowania obiektów w kodzie programu na relacyjną bazę danych.

Akronim	Definicja
REST	<i>Representational State Transfer</i> – styl architektoniczny przeznaczony do projektowania aplikacji sieciowych, oparty na bezstanowym modelu komunikacji.

2 Cel i zakres przedsięwzięcia

Celem projektu jest opracowanie szybkiego systemu diagnostycznego opartego na sztucznej inteligencji do wykrywania toksycznych substancji w osoczu pacjentów.

Kluczowa metoda polega na analizie wzorców ruchowych larw *Galleria mellonella*, którym podano próbki osocza. System wykorzystuje algorytmy uczenia maszynowego do identyfikacji charakterystycznych, subtelnego reakcji ruchowych larw na toksyczne substancje.

Ostatecznym celem jest wdrożenie nowatorskiej metody, która wykryje toksyczne związki, przyczyniając się do poprawy skuteczności leczenia i skrócenia czasu interwencji medycznej w stanach zagrożenia życia.

3 Słownik pojęć

Poniższa tabela zawiera definicje kluczowych pojęć oraz akronimów wykorzystywanych w dokumentacji.

Termin	Definicja
Drzewo widgetów	Hierarchiczna struktura organizująca widgety w relacji rodzic-dziecko, odwzorowująca budowę interfejsu użytkownika w aplikacji.
Galleria mellonella (G. mellonella)	Barciak większy, motylka – nocny owad z rzędu motyli, rodziny omańcowatych.
Ksenobiotyk	Substancja chemiczna obecna w organizmie, która nie jest w nim naturalnie wytwarzana ani nie występuje w nim w normalnych warunkach (np. lek, toksyna).
Widget	Podstawowy element budulcowy interfejsu w technologii Flutter.

4 Stan wiedzy w obszarze przedsięwzięcia

Dotychczasowe metody wykrywania ksenobiotyków opierają się głównie na analizie chemicznej, która mimo dużej dokładności wymaga długich procedur i specjalistycznego sprzętu. Coraz większe zainteresowanie budzą więc podejście wykorzystujące modele biologiczne i algorytmy sztucznej inteligencji do pośredniego wykrywania substancji toksycznych. Larwy *Galleria mellonella* zyskują na popularności w toksykologii dzięki niskim kosztom, szybkim

reakcjom i pewnym podobieństwom ich odpowiedzi immunologicznej do reakcji ssaków. Zmiany w ich zachowaniu, zwłaszcza w aktywności ruchowej, można łatwo mierzyć przy użyciu systemów wizyjnych, co stwarza dobre warunki do automatyzacji całego procesu diagnostycznego.

4.1 Analiza literatury i badania wstępne

Kluczowym etapem inicjującym prace nad projektem było przeprowadzenie szczegółowej analizy literatury i podobnych, istniejących rozwiązań w obszarze diagnostyki toksykologicznej wspomaganej sztuczną inteligencją. Proces badawczy koncentrował się na trzech głównych obszarach:

1. **Modelowanie behawioralne larw *Galleria mellonella*:** Analizowano dotychczasowe wykorzystanie larw jako modelu biologicznego (*in vivo*) do oceny toksyczności. Szczerąną uwagę zwrócono na publikacje opisujące **zależność zmian motorycznych od stężenia ksenobiotyków**, co stanowiło empiryczną podstawę dla naszego podejścia diagnostycznego.
2. **Nowoczesne metody komputerowego przetwarzania obrazu (Computer Vision):** Przeprowadzono *research* w zakresie zaawansowanych algorytmów śledzenia ruchu oraz ekstrakcji cech behawioralnych z sekwencji wideo. Analizowano efektywność tradycyjnych algorytmów śledzenia oraz nowoczesnych metod opartych na głębokim uczeniu (np. *DeepLabCut*), mając na uwadze ich wydajność i szybkość wymaganą przez system czasu rzeczywistego.
3. **Architektury modeli klasyfikacyjnych:** Zbadano nowoczesne podejścia do modelowania danych czasowo-przestrzennych. Analiza ta potwierdziła, że **hybrydowe architektury łączące konwolucyjne sieci neuronowe (CNN) do ekstrakcji cech przestrzennych z sieciami rekurencyjnymi (LSTM)** do modelowania dynamiki czasowej są optymalne dla problemu klasyfikacji sekwencji behawioralnych.

Wyniki analizy wykazały, że choć literatura opisuje poszczególne elementy składowe projektu (wykorzystanie *G. mellonella*, śledzenie owadów, modelowanie sekwencyjne), **brak jest kompleksowego, wdrożonego rozwiązania komercyjnego lub badawczego**, które integrowałoby wszystkie te aspekty w jedną, skalową platformę do szybkiej diagnostyki klinicznej. Ta luka technologiczna stanowiła główne uzasadnienie dla podjęcia prac nad projektem LarvixON.

5 Założenia wstępne

Projekt systemu LarvixON został oparty na szeregu kluczowych założeń, wynikających zarówno z potrzeb biznesowych, specyfiki środowiska medycznego, jak i ograniczeń tech-

nologicznych związanych z przetwarzaniem obrazu oraz uczeniem maszynowym. Poniżej przedstawiono szczegółową charakterystykę przyjętych założeń.

5.1 Grupa docelowa i środowisko operacyjne

Głównymi odbiorcami systemu są pracownicy laboratoriów diagnostycznych oraz personel medyczny szpitali (toksykologzy, lekarze SOR). W związku z tym przyjęto następujące założenia dotyczące środowiska pracy:

- **Dostępność wieloplatformowa:** System musi być dostępny zarówno na stacjonarnych stacjach roboczych w laboratoriach (przeglądarka internetowa), jak i na urządzeniach mobilnych (tablety/smartfony), umożliwiając szybki podgląd wyników przy łóżku pacjenta.
- **Intuicyjność obsługi:** Interfejs użytkownika (UI) musi być maksymalnie uproszczony i czytelny, aby nie obciążać poznawczo personelu pracującego pod presją czasu.
- **Model pracy:** System działa w architekturze klient-serwer, gdzie ciężkie obliczenia (wnioskowanie modelu ML) są delegowane do chmury, aby nie obciążać urządzeń końcowych użytkownika.

5.2 Założenia technologiczne i sprzętowe

Ze względu na złożoność obliczeniową algorytmów analizy wideo oraz wymagania dotyczące skalowalności, przyjęto następujące założenia technologiczne:

1. **Jednolita baza kodu (frontend):** W celu optymalizacji procesu wytwarzego i utrzymania, aplikacja kliencka zostanie zrealizowana w technologii *Flutter*, co pozwoli na skompilowanie natywnych aplikacji na systemy Android/iOS oraz wersji Web z jednego kodu źródłowego.
2. **Akceleracja GPU (serwis ML):** Serwer odpowiedzialny za analizę wideo musi być wyposażony w karty graficzne (GPU) wspierające bibliotekę CUDA, co jest niezbędne do efektywnego działania modelu hybrydowego CNN+LSTM w rozsądny czasie.
3. **Konteneryzacja:** Całość rozwiązania serwerowego (API, bazy danych, worker ML) zostanie osadzona w kontenerach *Docker*, co zapewni powtarzalność środowiska uruchomieniowego i łatwość wdrażania (CI/CD).

5.3 Założenia dotyczące danych wejściowych (wideo)

Skuteczność modelu diagnostycznego jest ściśle skorelowana z jakością dostarczonego materiału wideo. Przyjęto następujące założenia dla modułu analizy obrazu:

- **Standaryzacja nagrania:** Materiał wideo musi przedstawiać szalkę Petriego sfilmowaną z góry ("lot ptaka"), z zachowaniem odpowiedniego oświetlenia eliminującego silne cienie, które mogłyby zostać błędnie zinterpretowane przez algorytm detekcji kształtów.
- **Format danych:** System będzie obsługiwał format wideo .mp4, z ograniczeniem rozmiaru wideo do 3 GB (praktyczne ograniczenie wynikające z ograniczonych funduszy przy studenckiej licencji Azure).
- **Model biologiczny:** System zakłada, że na nagraniu znajdują się larwy *Galleria mellonella* w odpowiednim stadium rozwojowym, a ich reakcje ruchowe są wynikiem ekspozycji na badaną substancję.

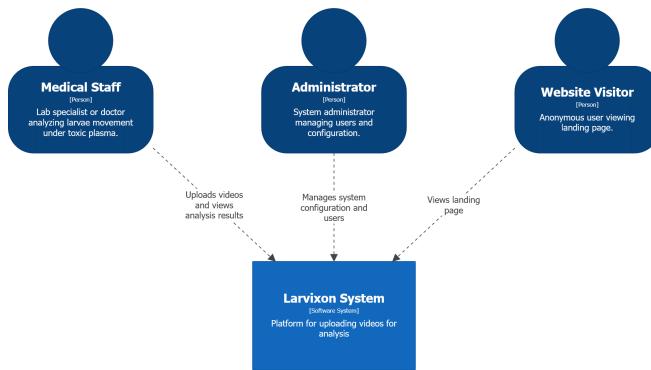
5.4 Założenia prawne i bezpieczeństwo danych

Ponieważ system przetwarza dane mogące posłużyć do identyfikacji pacjenta (w powiązaniu z wynikami badań), projekt uwzględnia rygorystyczne wymogi bezpieczeństwa:

- **Zgodność z RODO:** System musi realizować zasady *Privacy by Design*. Dane osobowe pacjentów są przechowywane w oddzielnym mikroserwisie (Patient Service).
- **Szyfrowanie:** Cała komunikacja sieciowa odbywa się kanałem szyfrowanym (HTTPS / TLS), a wrażliwe dane w bazie (np. hasła) są hashowane przy użyciu silnego algorytmu PBKDF2, domyślnego w Django.
- **Interoperacyjność:** Struktura danych pacjentów powinna być zgodna lub możliwa do mapowania na standardy medyczne takie jak HL7 FHIR, co umożliwi potencjalną integrację z systemami szpitalnymi w przyszłości.

6 Specyfikacja wymagań na produkt programowy

6.1 Wymagania funkcjonalne



Rysunek 1: Diagram kontekstu systemowego (warstwa C1 modelu architektury C4)

6.1.1 Epic 1: Zarządzanie użytkownikiem

- US-1.1** Jako nowy użytkownik chcę móc zarejestrować się w systemie, aby uzyskać dostęp do platformy.
- US-1.2** Jako zarejestrowany użytkownik chcę móc zalogować się do systemu, aby uzyskać dostęp do moich analiz.
- US-1.3** Jako zalogowany użytkownik chcę móc edytować swoje dane, aby aktualizować informacje o sobie.

6.1.2 Epic 2: Zarządzanie analizami

- US-2.1** Jako zalogowany użytkownik chcę móc przesłać nagranie wideo larw, aby system mógł je przeanalizować.
- US-2.2** Jako zalogowany użytkownik chcę widzieć listę wszystkich moich analiz, aby śledzić postęp i przeglądać wyniki.
- US-2.3** Jako zalogowany użytkownik chcę zobaczyć szczegółły konkretnej analizy, aby przeanalizować wyniki i podjąć decyzje medyczne.
- US-2.4** Jako zalogowany użytkownik chcę móc usunąć analizę, aby pozbyć się niepotrzebnych danych.

- US-2.5** Jako zalogowany użytkownik chcę zlecić wykonanie analizy ponownie, jeśli mam wątpliwości lub analiza się nie powiodła.
- US-2.6** Jako zalogowany użytkownik chcę móc wyeksportować wyniki analizy w formacie dokumentu PDF, aby móc wydrukować wyniki i podzielić się nimi z członkami personelu.

6.1.3 Epic 3: Informacja o systemie

- US-3.1** Jako użytkownik chcę wysłać wiadomość do zespołu, aby zadać pytanie dotyczące systemu.

6.2 Wymagania niefunkcjonalne

6.2.1 Wydajność i skalowalność

- NFR-1.1** System powinien umożliwiać asynchroniczne przetwarzanie plików wideo, aby proces analizy ML nie blokował interfejsu użytkownika ani głównego wątku serwera.
- NFR-1.2** Czas odpowiedzi API dla standardowych operacji (pobranie listy, logowanie) nie powinien przekraczać 500 ms w warunkach stabilnego łącza internetowego.
- NFR-1.3** System powinien obsługiwać pliki wideo o wielkości do 3 GB.

6.2.2 Bezpieczeństwo i ochrona danych

Ze względu na przetwarzanie danych osobowych pacjentów, system musi spełniać podwyższone standardy bezpieczeństwa.

- NFR-2.1** Komunikacja między klientem a serwerem musi być szyfrowana przy użyciu protokołu HTTPS.
- NFR-2.2** Hasła użytkowników muszą być przechowywane w bazie danych w formie zaszyfrowanej, przy użyciu bezpiecznych algorytmów.
- NFR-2.3** Dostęp do danych pacjentów oraz wyników analiz musi być chroniony mechanizmem uwierzytelniania opartym na tokenach JWT.
- NFR-2.4** System musi wymuszać politykę haseł (minimum 8 znaków), aby zminimalizować ryzyko nieautoryzowanego dostępu.

6.2.3 Użyteczność i kompatybilność

- NFR-3.1** Aplikacja kliencka musi być możliwa do skompilowania i uruchomienia jako natywna aplikacja instalowalna na systemach operacyjnych: Android, Windows oraz Linux, a także jako aplikacja przeglądarkowa.

- NFR-3.2** Interfejs użytkownika musi być responsywny i adaptować się nie tylko do rozdzielczości ekranu, ale także do metody wprowadzania danych (obsługa dotyku na urządzeniach mobilnych / obsługa myszy i klawiatury na systemach desktopowych).
- NFR-3.3** Aplikacja musi wspierać internacjonalizację i umożliwiać dynamiczną zmianę języka interfejsu (polski/angielski) bez konieczności przeładowania strony.
- NFR-3.4** Architektura aplikacji klienckiej rozwiązania musi pozwalać na utrzymanie wspólnej bazy kodu dla wszystkich wspieranych platform, z wyjątkiem specyficznych modułów natywnych.
- NFR-3.5** System powinien informować użytkownika o statusie długotrwałych operacji za pomocą wskaźników postępu lub ekranów szkieletowych.

6.2.4 Niezawodność i obsługa błędów

- NFR-4.1** W przypadku niedostępności serwera lub braku połączenia internetowego, aplikacja powinna wyświetlić czytelny komunikat błędu i umożliwić ponowienie próby.
- NFR-4.2** System powinien walidować przesyłane pliki wideo pod kątem formatu przed rozpoczęciem procesu przetwarzania, aby uniknąć błędów modelu ML.
- NFR-4.3** Model powinien zwracać błąd dla nagrani, na których nie znajdują się larwy.

6.3 Realizacja wymagań i zarządzanie projektem

W celu efektywnej organizacji pracy zespołu oraz monitorowania postępów w implementacji zdefiniowanych wymagań funkcjonalnych, wykorzystano platformę GitHub Projects. Przyjęto metodę zwinnego zarządzania projektem (Agile) z wykorzystaniem tablicy Kanban. Do jasnego monitorowania prac i komunikacji wykorzystano serwer Discord. Do kanałów *pull-requests* i *issues* dodano webhooki GitHub, pozwalające na automatyczne wysłanie wiadomości w przypadku rozpoczęcia/zakończenia issue lub pull requesta w projekcie.

Proces zarządzania wymaganiami przebiegał następująco:

- Definicja zadań:** Każde zdefiniowane wcześniej *User Story* zostało przełożone na konkretne zgłoszenia (*Issues*) w repozytorium GitHub.
- Szacowanie i priorytetyzacja:** Zadania trafiały do *Backlogu* produktu, gdzie nadawano im priorytety oraz przypisywano do odpowiednich repozytoriów (backend, frontend, ML, symulacja, ...).
- Cykł życia zadania:** Zadania przechodziły przez kolejne statusy na tablicy Kanban: *Ready*, *In Progress*, *In Review* oraz *Done*.

Taka organizacja pracy pozwoliła na zachowanie *traceability*, ponieważ każda funkcja w finalnym systemie ma swoje odzwierciedlenie w zrealizowanym zadaniu w systemie zarządzania projektem.

Kluczowym elementem procesu wytwórczego była również współpraca z interesariuszem – Uniwersytetem Medycznym we Wrocławiu. Zespół dwukrotnie konsultował postępy prac oraz ewoluujące wymagania, co pozwoliło na bieżące dostosowywanie aplikacji do rzeczywistych potrzeb personelu medycznego i weryfikację założeń projektowych.

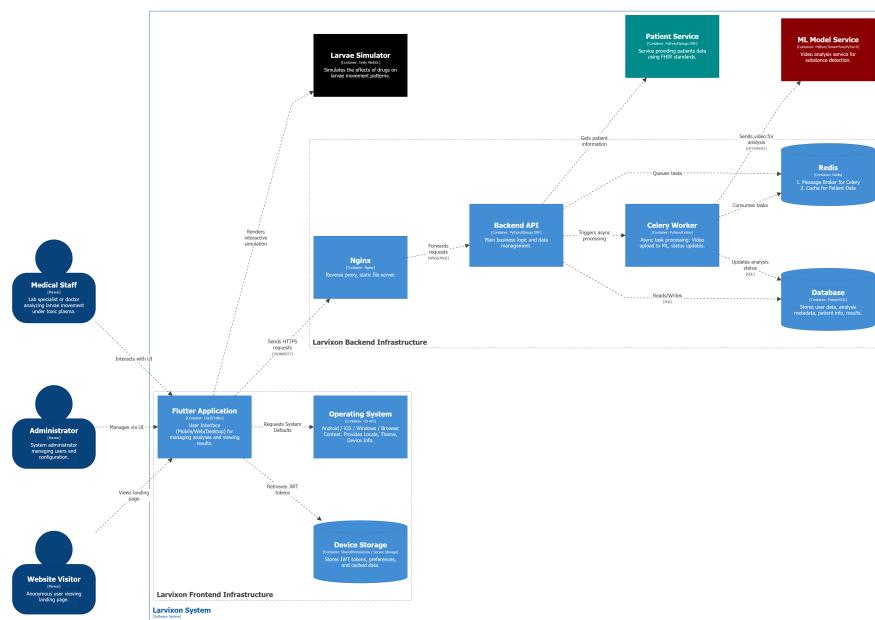
7 Projekt produktu programowego

7.1 Backend

Część serwerowa systemu LarvixON (larvixon-backend) to niejako połaczanie między interfejsem użytkownika a modelem uczenia maszynowego. Została zrealizowana w oparciu o język Python, ze względu na prostą składnię i bogactwo bibliotek. Dodatkowo w Pythonie dostępny jest framework Django, który umożliwia szybkie i bezpieczne wdrażanie nowych funkcjonalności do backendu. Ponadto wykorzystano też Django REST Framework do wystawienia standardyzowanego API (RESTful API), przez które część frontendowa komunikuje się z backendem.

7.1.1 Architektura systemu i stos technologiczny

Architektura backendu została zaprojektowana zgodnie z wzorcem modułowego monolitu. Głównym celem było odseparowanie logiki biznesowej od długotrwałych procesów obliczeniowych.



Rysunek 2: Diagram kontenerów (warstwa C2 modelu architektury C4)

Jak przedstawiono na Rysunku 2, system składa się z następujących elementów infrastruktury:

- **Django App (Backend API):** Główna aplikacja obsługująca żądania HTTP, autoryzację i logikę biznesową.
- **Celery Worker:** Procesy tła odpowiedzialne za asynchroniczne przetwarzanie wideo i komunikację z modelem ML, co zapobiega blokowaniu interfejsu użytkownika.
- **Redis:** Broker wiadomości kolejkowych oraz pamięć podręczna.
- **PostgreSQL:** Relacyjna baza danych przechowująca trwałe dane systemu.
- **Nginx:** Serwer odwrotnego proxy (*Reverse Proxy*) obsługujący ruch sieciowy i pliki statyczne.

Decyzja o doborze powyższego stosu technologicznego była podejmowana planowanym wdrożeniem systemu w środowisku chmurowym **Microsoft Azure**. Wykorzystano usługi *Azure App Service* w modelu *Code-based* do hostowania aplikacji Django.

W celu zapewnienia wysokiej skalowalności oraz optymalizacji kosztów, zamiast przechowywania danych w lokalnym systemie plików instancji, zdecydowano się na wykorzystanie dedykowanych *Managed Services*:

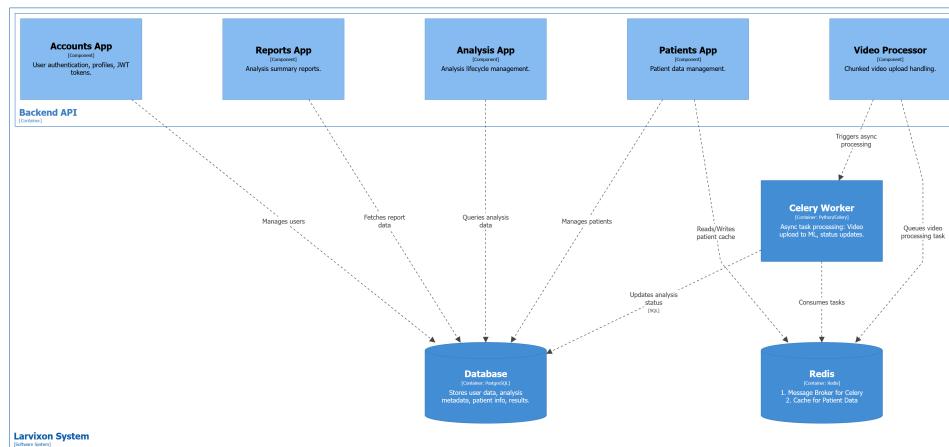
- **Azure Database for PostgreSQL** – jako relacyjna baza danych przechowująca metadane użytkowników i analiz.

- **Azure Cache for Redis** – jako szybka pamięć podręczna oraz broker wiadomości dla zadań asynchronicznych.
- **Azure Blob Storage** – jako skalowalny magazyn obiektowy dla plików wideo.

Integracja warstwy aplikacji z Azure Blob Storage została zrealizowana przy użyciu biblioteki `django-storages`. Dzięki temu pliki przesyłane przez użytkowników nie obciążają systemu plików serwera, lecz trafiają bezpośrednio do chmury, co umożliwia przetwarzanie dowolnej liczby nagrani bez ryzyka przepełnienia dysku lokalnego.

7.1.2 Struktura modułowa

W celu zachowania zasady *Separation of Concerns* (rozdziału odpowiedzialności), kod aplikacji został podzielony na niezależne domeny funkcjonalne.



Rysunek 3: Diagram komponentów backendu (warstwa C3 modelu architektury C4)

Wyróżniono następujące moduły (Rys. 3):

- **accounts** – Zarządzanie tożsamością: rejestracja, logowanie, zmienianie danych osobowych, obsługa tokenów JWT.
- **analysis** – Zarządzanie cyklem życia analizy, przechowywanie wyników i udostępnianie ich poprzez API.
- **patients** – Komunikacja z serwisem pacjentów.
- **reports** – Moduł generujący sformatowane raporty PDF na podstawie wyników analizy.
- **videoprocessor** – Obsługa przesyłania plików wideo do analizy.

7.1.3 Projekt Interfejsu Programistycznego (API)

Komunikacja między warstwą prezentacji a serwerem odbywa się w oparciu o architekturę REST. Wybór tego stylu architektonicznego zapewnia bezstanowość komunikacji, co klu- czowo wpływa na skalowalność systemu oraz łatwość integracji z różnymi platformami klienckimi.

W celu zapewnienia spójności komunikacji między backendem a frontendem, przyjęto standard OpenAPI (*Swagger*). Pozwolił on na sformalizowany opis struktury endpointów, wymaganych typów danych oraz kodów błędów.

Zaprojektowano następujące konwencje wykorzystania metod HTTP:

- **GET** – pobieranie zasobów (np. listy analiz, danych użytkownika), operacja bezpieczna i idempotentna.
- **POST** – tworzenie nowych zasobów (np. rejestracja użytkownika, start analizy).
- **PUT/PATCH** – aktualizacja istniejących danych (np. edycja zdjęcia profilowego czy stanu analizy).
- **DELETE** – usuwanie zasobów.

W Tabeli 3 przedstawiono projekt kluczowych endpointów systemu, realizujących główne wymagania funkcjonalne.

Tabela 3: Projekt kluczowych endpointów API systemu LarvixON

Metoda	Endpoint	Opis funkcjonalny
GET	/api/patients/	Pobranie listy pacjentów (most między wybieraniem pacjenta do analizy na frontendzie a osobnym serwisem pacjentów).
POST	/api/video/upload/	Przesłanie pliku video.
GET	/api/analysis/	Pobranie listy analiz.
GET	/api/analysis/{id}/	Pobranie wyników konkretnej analizy.
GET	/api/reports/{id}/pdf/	Wygenerowanie raportu PDF.

7.1.4 Logiczna architektura warstwowa

Wewnątrz modułów backendu zastosowano architekturę warstwową (*Layered Architecture*), co zapewniło separację odpowiedzialności oraz wysoką testowalność. Kod podzielono na następujące warstwy logiczne:

1. **Warstwa prezentacji (widoki i serializery):** Oparta na generycznych klasach *Django REST Framework*. Jej rola została ograniczona do walidacji danych wejściowych, autoryzacji użytkownika oraz delegowania zadań do warstwy serwisowej. Nie zawiera ona logiki biznesowej.

2. **Warstwa logiki biznesowej (serwisy):** Jest to kluczowy element systemu, orkiestrujący operacje między bazą danych a zewnętrznymi API. Zastosowano tu **Wzorzec Strategii** z wykorzystaniem klas abstrakcyjnych (ABC). Zdefiniowanie kontraktów (np. `BasePatientService`) umożliwiło stworzenie wymiennych implementacji:

- **Produkcyjnej** – komunikującej się z rzeczywistym systemem FHIR.
- **Mockowej** – zwracającej spreferowane dane, wykorzystywanej przy lokalnym developmencie, bez konieczności uruchamiania wszystkich serwisów. Testy nie testują mockowych wersji serwisów, tylko produkcyjne, dzięki mockowaniu odpowiedzi HTTP do zewnętrznych serwisów.

Warstwa ta realizuje również mechanizm *caching* (Redis), redukując liczbę zapytań do zewnętrznych API.

3. **Warstwa obsługi błędów:** Zamiast generycznych wyjątków, zaimplementowano hierarchię **wyjątków domenowych** (np. `PatientServiceUnavailableError` czy `AnalysisProcessingError`). Pozwala to warstwie prezentacji na precyzyjne mapowanie błędów logicznych na odpowiednie kody statusu HTTP (np. 503 zamiast ogólnego 500) oraz generowanie czytelnych komunikatów dla klienta.
4. **Warstwa danych:** Standardowa warstwa modeli Django ORM, mapująca obiekty na tabele relacyjnej bazy danych PostgreSQL.

Taki podział umożliwił niezależne testowanie logiki biznesowej (Unit Tests) bez konieczności angażowania warstwy HTTP czy zewnętrznych zależności.

7.1.5 Schemat bazy danych

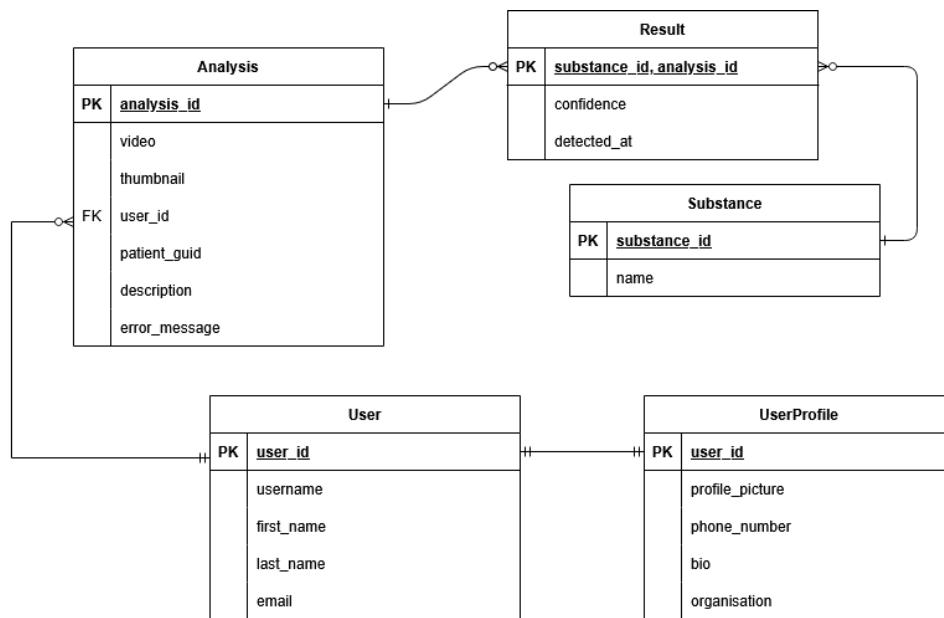
Warstwa danych systemu LarvixON została zaimplementowana w oparciu o relacyjną bazę danych PostgreSQL. Projekt bazy danych został zrealizowany dwuetapowo: najpierw opracowano model konceptualny odzwierciedlający powiązania biznesowe, a następnie przełożono go na model fizyczny zarządzany przez system migracji Django ORM.

7.1.5.1 Model konceptualny Model konceptualny przedstawia główne encje systemu oraz zachodzące między nimi relacje. Centralnym punktem systemu jest powiązanie użytkownika z przeprowadzanymi analizami oraz wynikami detekcji substancji.

Zidentyfikowano następujące encje i relacje:

- **Użytkownik – Profil:** Relacja jeden-do-jednego (1:1). Wydzielenie profilu pozwala na separację danych autoryzacyjnych (e-mail, nazwa użytkownika) od danych informacyjnych (zdjęcie, bio, organizacja).
- **Użytkownik – Analiza:** Relacja jeden-do-wielu (1:N). Jeden użytkownik może zlecić wiele analiz, ale każda analiza ma jednego właściciela.

- **Analiza – Wynik:** Relacja jeden-do-wielu (1:N). Pojedyncza analiza wideo może skutkować wykryciem wielu substancji z różnym prawdopodobieństwem.
- **Substancja – Wynik:** Relacja jeden-do-wielu (1:N). Ta sama substancja może pojawić się w wynikach wielu różnych analiz.



Rysunek 4: Model konceptualny bazy danych (diagram ERD)

7.1.5.2 Model fizyczny i implementacja Implementacja fizyczna została zrealizowana przy użyciu podejścia *Code-First*. Modele zdefiniowane w języku Python (klasy dziedziczące po `models.Model`) zostały przetłumaczone na schemat SQL.

Poniżej omówiono kluczowe tabele w podziale na moduły funkcjonalne.

1. Moduł tożsamości (accounts)

Tabela `accounts_user` rozszerza standardowy model użytkownika Django, wykorzystując adres e-mail jako unikalny identyfikator logowania. Tabela `accounts_userprofile` przechowuje dodatkowe atrybuty, takie jak ścieżka do zdjęcia profilowego czy numer telefonu.

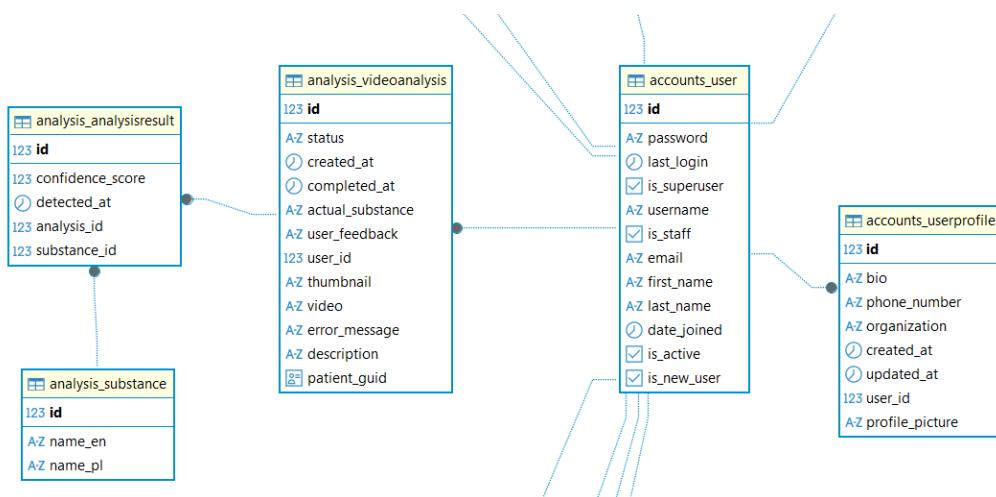
2. Moduł analizy (analysis)

Jest to kluczowa część schematu bazy danych. Główna tabela `videoanalysis` przechowuje metadane procesu przetwarzania. Szczegółową strukturę tej tabeli przedstawiono poniżej na Rysunku 5

Table "public.analysis_videoanalysis"				
Column	Type	Collation	Nullable	Default
id	bigint		not null	generated by default as identity
status	character varying(20)		not null	
created_at	timestamp with time zone		not null	
completed_at	timestamp with time zone			
actual_substance	character varying(100)			
user_feedback	text		not null	
user_id	bigint		not null	
thumbnail	character varying(100)			
video	character varying(100)			
error_message	text			
description	text		not null	
patient_guid	uuid			

Rysunek 5: Struktura fizyczna tabeli analizy

Wyniki odebrane od modelu ML są przechowywane w tabeli analysisresult, która łączy konkretną analizę z substancją i przechowuje pewność z jaką model określił, że dana substancja znajdowała się w osoczu pacjenta. Zastosowano tu klucz unikalny złożony (UNIQUE TOGETHER) na parach (analysis_id, substance_id), co zapobiega dublowaniu wyników tej samej substancji w ramach jednej analizy.



Rysunek 6: Model fizyczny bazy danych wygenerowany z DBeaver

7.2 Interfejs użytkownika

Projekt larvixon-frontend odpowiada za prezentację funkcjonalności systemu oraz dostarczenie użytkownikowi intuicyjnego i czytelnego interfejsu.

7.2.1 Architektura systemu i stos technologiczny

Aplikacja została zbudowana w architekturze warstwowej, inspirowanej zasadami Clean Architecture [8], co zapewnia separację odpowiedzialności oraz wysoką testowalność. Poniżej przedstawiono odpowiedzialności poszczególnych warstw:

Warstwa danych Zarządza źródłami danych, obsługuje komunikację z API oraz mapuje DTO na encje domenowe.

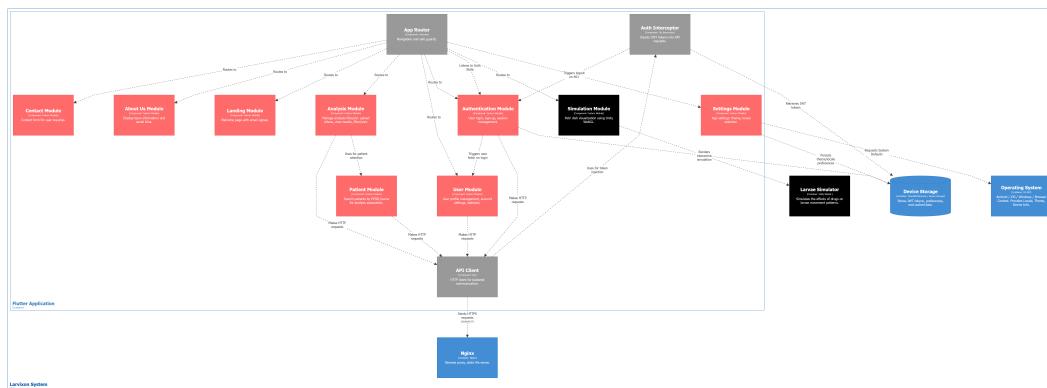
Warstwa domeny Definiuje encje biznesowe, określa interfejsy repozytoriów i zawiera logikę biznesową aplikacji.

Warstwa prezentacji Zawiera interfejs użytkownika, odpowiada za wyświetlanie danych i interakcję z użytkownikiem oraz wykorzystuje wzorzec BLoC do separacji logiki prezentacji od widoków i zarządzania stanem.

7.2.1.1 Kluczowe technologie i biblioteki Podstawą technologiczną projektu jest Flutter w wersji 3.35.1. Poniżej zostały wymienione kluczowe biblioteki usprawniające implementację aplikacji.

- **flutter_bloc** – zarządzanie stanem w warstwie prezentacji.
- **dio** – klient HTTP obsługujący komunikację z API.
- **go_router** – deklaratywny routing i nawigacja w aplikacji.
- **fpdart** – biblioteka wprowadzająca paradymat programowania funkcyjnego, wykorzystywana do bezpiecznej obsługi błędów oraz operacji na typach opcjonalnych.
- **flutter_unity_widget** – biblioteka umożliwiająca integrację symulacji w aplikacji.

7.2.1.2 Struktura modułów Projekt został podzielony *per feature*, co oznacza, że każda funkcjonalność aplikacji znajduje się w osobnym folderze. Każdy moduł może zawierać widoki UI (warstwa prezentacji), repozytoria i definicje encji (warstwa domeny) oraz źródła danych i obiekty transferu danych (warstwa danych).



Rysunek 7: Diagram komponentów frontendu (warstwa C3 modelu architektury C4)

Poniżej przedstawiono moduły wraz z opisem ich przeznaczenia:

core Infrastruktura aplikacji. Zawiera m.in.: klienta API z rozszerzeniami obsługującymi błędy oraz sesję użytkownika, router aplikacji, definicje motywów oraz stałe.

I10n Warstwa internacjonalizacji. Zawiera pliki z tłumaczeniami oraz wygenerowane klasy. Wspiera tłumaczenia tekstów w widokach.

about_us Moduł prezentacyjny informacji o zespole.

analysis Kluczowy moduł zarządzania cyklem analizy.

authentication Moduł sesji użytkownika: logowanie/rejestracja.

common Biblioteka współdzielonych fragmentów logiki.

contact Moduł kontaktu z zespołem.

landing Prosty moduł prezentacyjny strony powitalnej.

patient Moduł wspierający wyszukiwanie pacjentów.

settings Preferencje aplikacji m.in. motyw oraz język.

simulation Integracja z Unity WebGL do symulacji zachowań larw.

user Zarządzanie kontem i danymi użytkownika.

7.2.2 Zastosowane podejścia projektowe

- **Domain-Driven Design (DDD)** – podejście projektowe koncentrujące się na modelu domen i logice biznesowej. W projekcie wykorzystano elementy taktyczne DDD, takie jak encje czy repozytoria.
- **SOLID** – zestaw zasad poprawiających czytelność, modularność i testowalność kodu.
- **Clean Code** – nacisk na prostotę, czytelność i utrzymywalność kodu.
- **DRY** – eliminacja powielania logiki w aplikacji.
- **KISS** – ograniczenie zbędnej złożoności implementacji.

7.2.3 Zastosowane wzorce architektoniczne

- **Clean Architecture** – podział na warstwy w celu separacji odpowiedzialności i ułatwienia testowania.
- **BLoC Architecture** – separacja logiki biznesowej od UI, przewidywalne zarządzanie stanem.
- **Event-Driven Architecture** – komunikacja oparta na wydarzeniach i zmianach stanu.

- **Dependency Injection** – centralne zarządzanie zależnościami.
- **Repository Pattern** – abstrakcja nad źródłami danych, umożliwia łatwą wymianę implementacji.

7.2.4 Zastosowane wzorce projektowe

7.2.4.1 Kreacyjne

- **Singleton** – wykorzystany w serwisie zarządzającym tokenami JWT.
- **Factory** – konstrukcja obiektów błędów zwracanych przez API.

7.2.4.2 Strukturalne

- **Adapter / Mapper** – konwersja DTO na encje domenowe.
- **Facade** – ujednolicenie dostępu do funkcji różniących się między platformami.
- **Proxy** – proxy autoryzacji dodaje token JWT do żądań i kontroluje błędy.
- **Decorator** – dodawanie reużywalnej funkcjonalności, na przykład onHoverExtension.

7.2.4.3 Behawioralne

- **Command** – enkapsulacja żądań jako obiektów przy zarządzaniu stanem.
- **Template** – stosowany na przykład w validatorach (adres e-mail, minimalna długość, legalne znaki).

7.2.4.4 Dodatkowe techniki

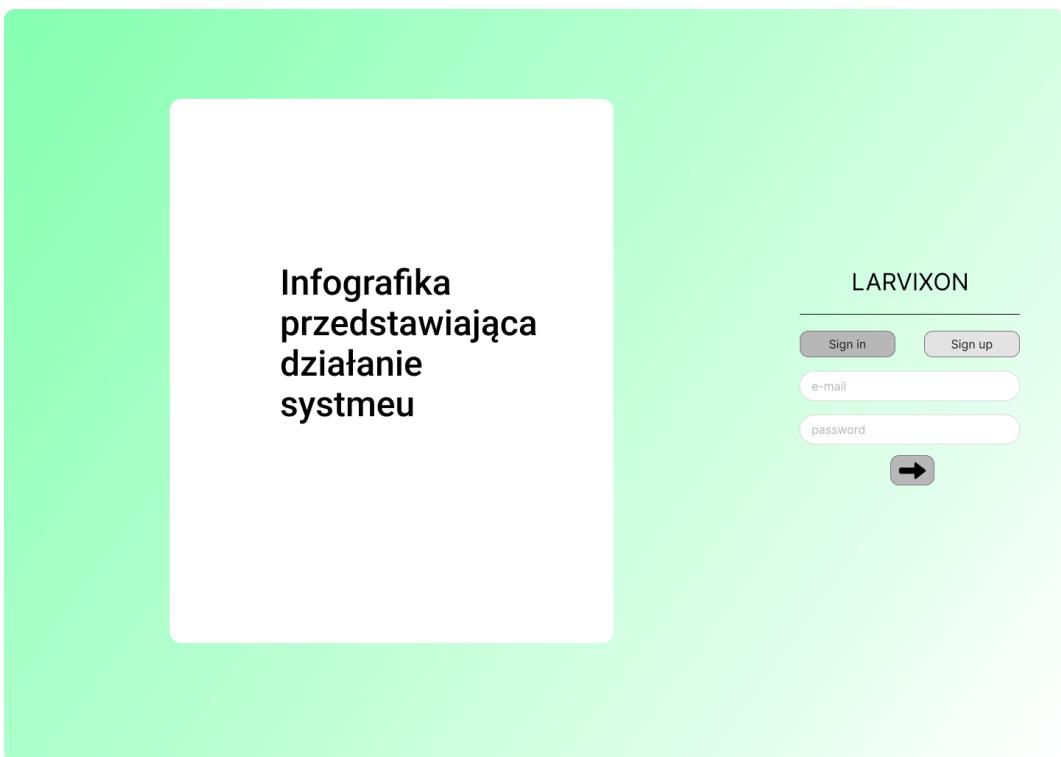
- Programowanie funkcyjne – wymuszenie obsługi błędów bez wyjątków.
- Responsive design – adaptacyjne layouty dla różnych urządzeń.

7.2.5 Prototyp interfejsu

Prototyp interfejsu został opracowany w narzędziu *Figma*.

7.2.5.1 Ekran powitalny

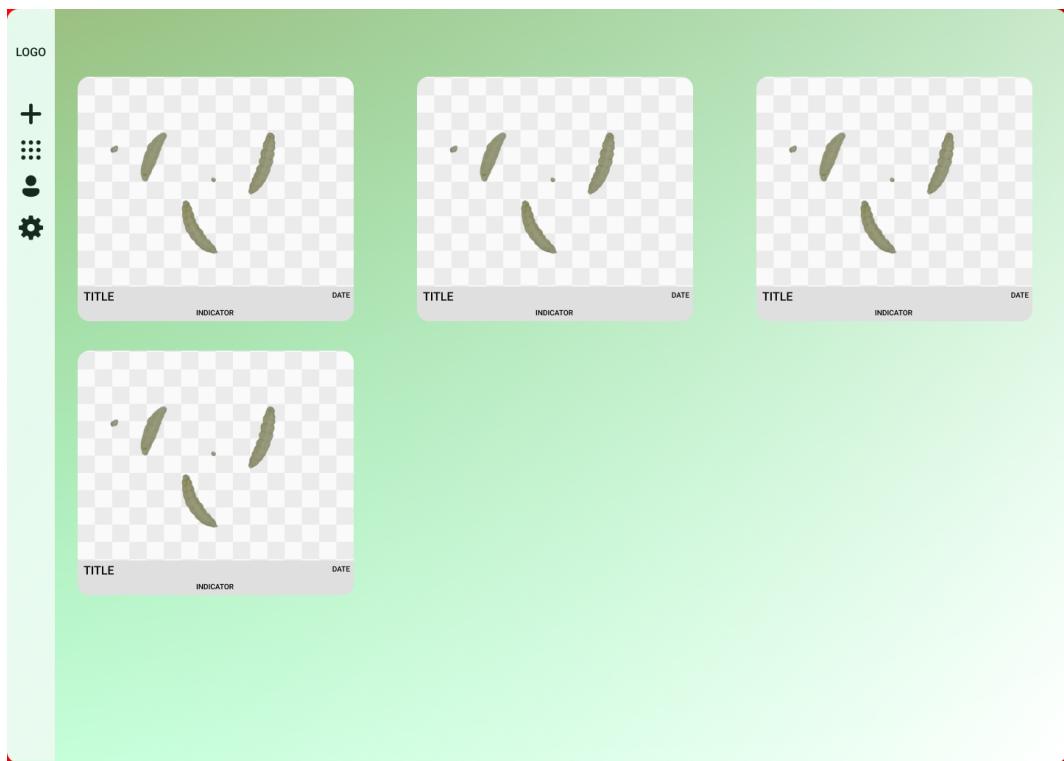
Widok powitalny umożliwiający użytkownikowi uwierzytelnienie w systemie. Zawiera formularz logowania oraz krótką i przejrzystą infografikę obrazującą działanie systemu.



Rysunek 8: Ekran powitalny wraz z formularzem logowania.

7.2.5.2 Lista analiz

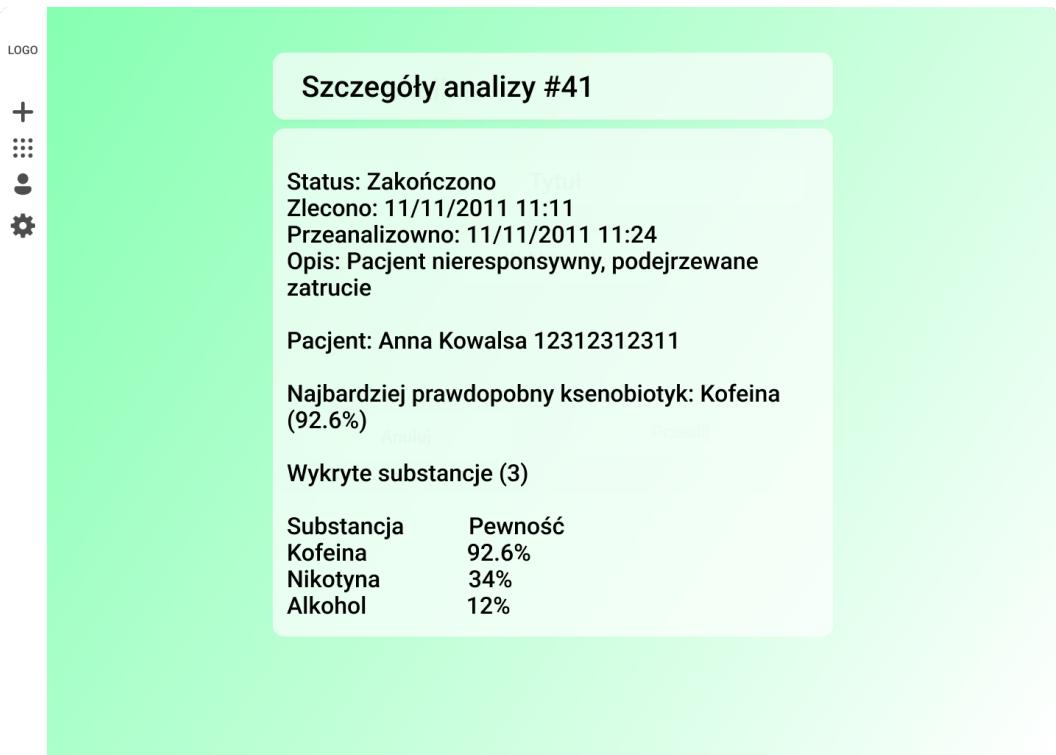
Widok prezentuje siatkę wszystkich wykonanych analiz. Umożliwia filtrowanie analiz ze względu na ich status oraz datę wykonania, sortowanie według daty czy statusu oraz przechodzenie do widoku szczegółowego analizy.



Rysunek 9: Widok główny wykonanych analiz.

7.2.5.3 Szczegóły analizy

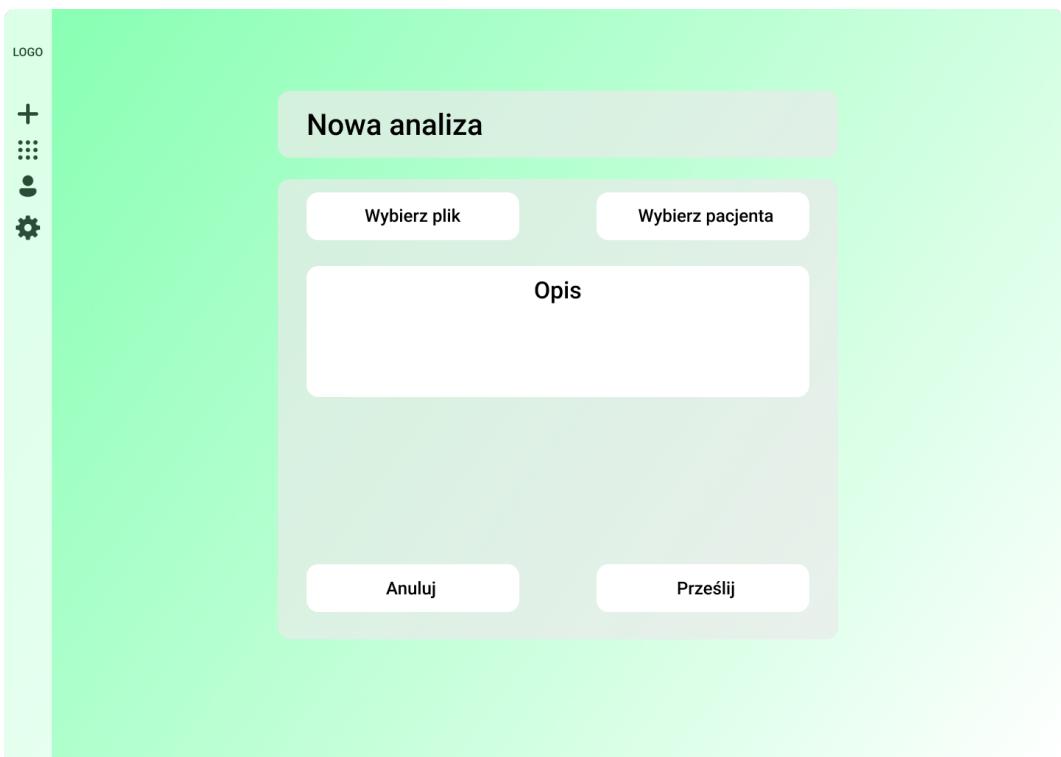
Widok prezentujący dane dotyczące wybranej analizy, takie jak status, data zlecenia oraz data wykonania, opis, powiązanego pacjenta oraz listę wyników.



Rysunek 10: Widok szczegółowy analizy.

7.2.5.4 Zlecenie analiz

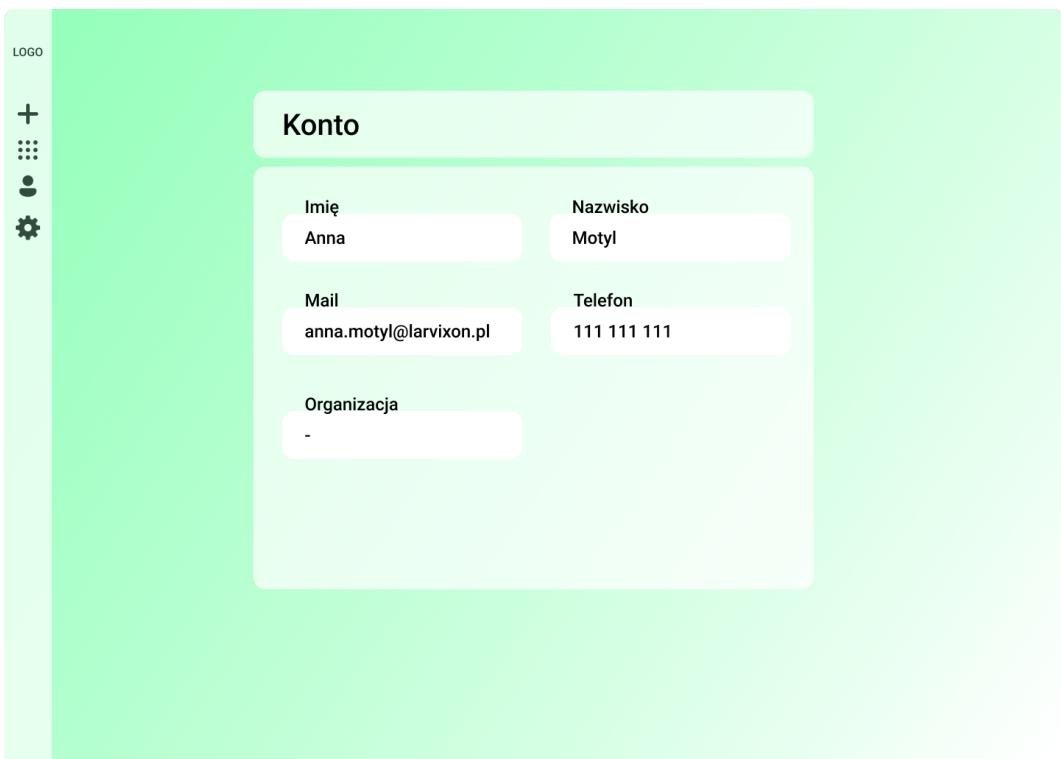
Widok umożliwiający zlecenie nowej analizy. Przedstawia formularz z polem umożliwiającym wybór nagrania, wyborem pacjenta oraz wprowadzeniem krótkiego opisu.



Rysunek 11: Widok zlecania analiz.

7.2.5.5 Konto użytkownika

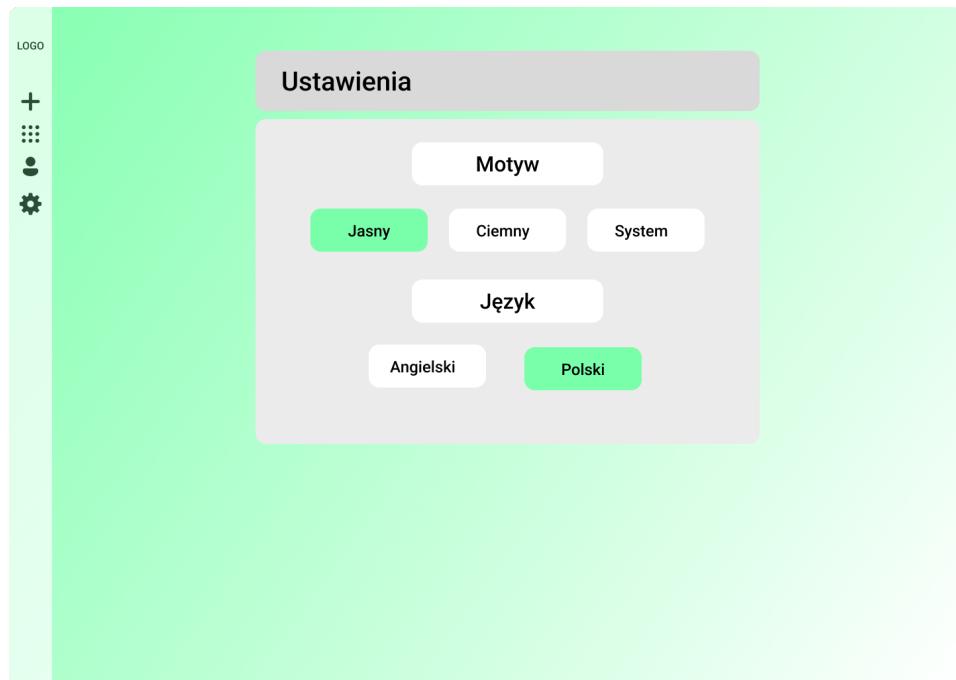
Widok umożliwiający zarządzanie danymi profilu użytkownika, takimi jak imię, nazwisko, numer telefonu czy organizację.



Rysunek 12: Widok konta użytkownika

7.2.5.6 Ustawienia

Widok umożliwiający zarządzanie preferencjami dotyczącymi motywu czy języka aplikacji.



Rysunek 13: Widok ustawień.

7.3 Model uczenia maszynowego do predykcji

Model uczenia maszynowego jest sercem systemu LarvixON AI i odpowiada za automatyczną klasyfikację substancji aktywnych na podstawie behawioralnej odpowiedzi larw *Galleria mellonella*.

7.3.1 Architektura hybrydowa i wdrożenie

Zastosowano zaawansowaną, hybrydową architekturę głębokiego uczenia zaprojektowaną do analizy danych czasowo-przestrzennych (w naszym projekcie - sekwencji klatek w czasie). Model został zaimplementowany przy użyciu frameworku **PyTorch**.

- **Architektura:** CNN + LSTM (*Convolutional Neural Network* połączona z *Long Short-Term Memory*).
- **Cel:** Klasyfikacja wzorców ruchowych larw w celu predykcji obecności substancji w oparciu o ich czasowo-przestrzenne cechy.
- **Wdrożenie i Serwowanie:** Moduł *ML* jest w pełni skonteneryzowany (**Docker**) i udostępnia wnioskowanie (*inference*) za pomocą wydajnego API opartego na **FastAPI**, co gwarantuje szybką integrację z *Larvixon Backend*.

7.3.2 Preprocessing i ekstrakcja danych wideo

Preprocessing w projekcie Larvixon jest złożonym procesem, który ma na celu przekształcenie surowych plików wideo przechowywanych w chmurze (S3) w ustandaryzowane sekwencje klatek gotowych do podania modelowi *CNN + LSTM* w celu treningu lub wnioskowania.

7.3.2.1 1. Pozyskiwanie danych i środowisko

Proces rozpoczyna się od pobrania surowych plików wideo:

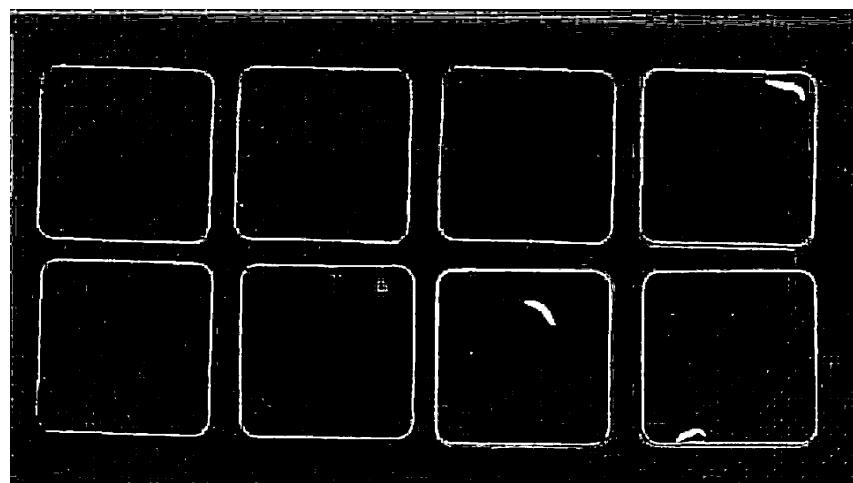
- **Pobieranie z S3:** Pliki wideo (z rozszerzeniem .mov) są pobierane z serwisu S3 (z wykorzystaniem boto3) i tymczasowo przechowywane w lokalnym katalogu (tempfile.TemporaryDirectory).
- **Kontrola Danych:** Funkcja list_s3_videos filtryuje dane, ignorując filmy, których nazwa zawiera "EtOH" (w celach weryfikacyjnych) oraz inne niepożądane pliki.

7.3.2.2 2. Segmentacja obszarów zainteresowania (ROI)

Kluczowym elementem jest funkcja extract_8_dishes_to_frame_folders, która dzieli wideo na osiem niezależnych kanałów behawioralnych:

- **Detekcja krawędzi:** Wykorzystana jest funkcja find_shapes_first_frame (z modułu find_edges_adaptive) w celu automatycznej detekcji położenia szalek w pierwszej klatce.

- **Wsparcie awaryjne (Fallback):** Jeśli detekcja nie powiedzie się lub wykrytych zostało mniej niż 8 obszarów, system przechodzi na metodę `slice_evenly` (podział ramki na 4 kolumny i 2 rzędy), z dodatkowym **buforem tolerancji** (np. 10% szerokości i wysokości, `tolerance=0.1`) w celu uniknięcia obcinania krawędzi.
- **Normalizacja etykiet:** Klasa substancji (*Class*) jest przypisywana na podstawie konfiguracji (`dish_to_class`) i opcjonalnie modyfikowana (*np.* przypisanie do klasy "Ethanol" na podstawie nazwy pliku).



Rysunek 14: Obraz kontrastowy pierwszej klatki wideo



Rysunek 15: Przykładowe wycięcie szalek Petriego za pomocą preprocessingu.

7.3.2.3 3. Synchronizacja i ekstrakcja klatek Sekwencyjna ekstrakcja klatek wymaga precyzyjnego wyboru momentu rozpoczęcia i próbkowania:

- **Próbkowanie czasowe (Sampling):** Funkcja `sample_frame_indices` wybiera określoną liczbę klatek (`num_frames`) w równych odstępach (`step`) z okresu rozpoczętego po danym przesunięciu.
- **Zapis klatek:** Na podstawie wybranych indeksów, klatki są wycinane (zgodnie z obliczonym *ROI*) i zapisywane do docelowej struktury katalogów.

7.3.2.4 4. Transformacje tensorów (PyTorch) Przed podaniem do modelu *CNN + LSTM*, klatki poddawane są standardowym transformacjom **PyTorch** (`torchvision.transforms`):

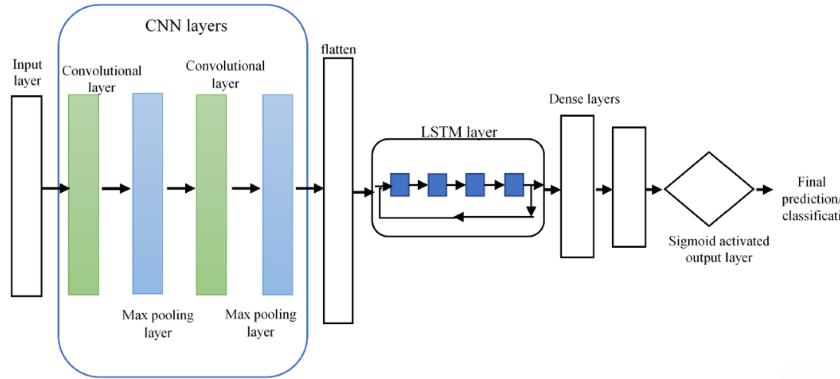
- **Skalowanie (Resize):** Klatki są skalowane do ujednoliconego wymiaru 112×112 pikseli.
- **Konwersja (ToTensor):** Konwersja obrazu do tensora.
- **Normalizacja:** Zastosowanie uśrednionej normalizacji na zbiorze danych *ImageNet* ($[0.485, 0.456, 0.406], [0.229, 0.224, 0.225]$) w celu przyspieszenia procesu uczenia.

7.3.2.5 Analiza cech i klasyfikacja Przygotowane sekwencje są przetwarzane przez warstwy hybrydowe:

- **Ekstrakcja przestrzenna (CNN):** Każda pojedyncza klatka jest przetwarzana przez warstwę **CNN** (np. *ResNet18*) w celu wydobycia cech statycznych (kształtu, położenia).
- **Modelowanie czasowe (LSTM):** Wektory cech, wygenerowane przez *CNN*, są podawane do sieci **LSTM**, która modeluje dynamikę ruchu i zależności w czasie, komplikując wzorzec behawioralny.
- **Predykcja:** Ostateczny wektor stanu jest klasyfikowany przez warstwy w pełni połączone (*Fully Connected Layers*) z użyciem funkcji *softmax*, generując rozkład prawdopodobieństwa obecności danej substancji.

7.3.3 Trening iteracyjny i zarządzanie modelem

Trening modelu jest realizowany w pętli iteracyjnej, gdzie każdy plik wideo jest przetwarzany jako odrębna sesja treningowa, w celu ciągłego doskonalenia i dostosowywania wag modelu do nowych danych.



Rysunek 16: Struktura modelu hybrydowego.

7.3.3.1 Inicjalizacja i ładowanie stanu Przed rozpoczęciem pętli treningowej następuje konfiguracja środowiska i modelu:

- **Inicjalizacja Modelu:** Tworzona jest instancja modelu `CNNLSTM` z określona liczbą klas (`num_classes`) i przenoszona na docelowe urządzenie (`device`, np. `cuda` lub `cpu`).
- **Optymalizator:** Wykorzystano optymalizator **Adam** (`torch.optim.Adam`) ze zdefiniowaną stopą uczenia (`LEARNING_RATE`).
- **Checkpointing:** Jeśli istnieje zapisany punkt kontrolny (`CHECKPOINT_PATH`), wagi modelu (`model_state`) oraz stan optymalizatora (`opt_state`) są ładowane. Umożliwia to wznowienie treningu i uniknięcie utraty postępów.

7.3.3.2 Pętla treningowa na wideo (train_one_video) Kluczowa logika treningowa, realizowana w funkcji `train_one_video`, przebiega następująco:

1. **Zestaw danych:** Tworzona jest instancja `FrameDataset`, która wczytuje sekwencje klatek (wygenerowane w preprocessingu) wraz z odpowiadającymi im etykietami.
2. **DataLoader:** Dane są ładowane w partiach (`batch_size`) za pomocą `DataLoader` z aktywowanym mieszaniem danych (`shuffle=True`).
3. **Funkcja straty:** Zastosowano **Cross-Entropy Loss** (`nn.CrossEntropyLoss`), standardową funkcję straty dla problemów klasyfikacji wieloklasowej.
4. **Proces uczenia (Epochs):** Model jest trenowany przez określoną liczbę epok (`epochs`) na sekwencjach pochodzących z pojedynczego wideo:
 - Obliczenie logitów: `logits = model(frames)`.
 - Obliczenie straty: `loss = criterion(logits, labels)`.

- Propagacja wsteczna: `loss.backward()` i aktualizacja wag `optimizer.step()`.
5. **Metryki:** Monitorowana jest bieżąca strata (running) oraz dokładność (acc), obliczana przez sumowanie poprawnych predykcji (`pred = logits.argmax(1)`).

7.3.3.3 Zapis stanu i finalizacja

Po zakończeniu treningu na danym wideo oraz po całkowitym przetworzeniu wszystkich zasobów:

- **Zapis checkpointu:** Po zakończeniu treningu na każdym wideo, model i stan optymalizatora są zapisywane jako punkt kontrolny (`config.CHECKPOINT_PATH`).
- **Finalny zapis:** Po przetworzeniu wszystkich wideo, ostateczne wagi modelu są zapisywane do pliku (`config.SAVE_PATH`), finalizując proces uczenia.
- **Optymalizacja GPU:** W razie użycia karty graficznej, pamięć cache GPU jest opróżniana (`torch.cuda.empty_cache()`), co zwiększa stabilność systemu - konieczne zwłaszcza na słabych jednostkach.

```

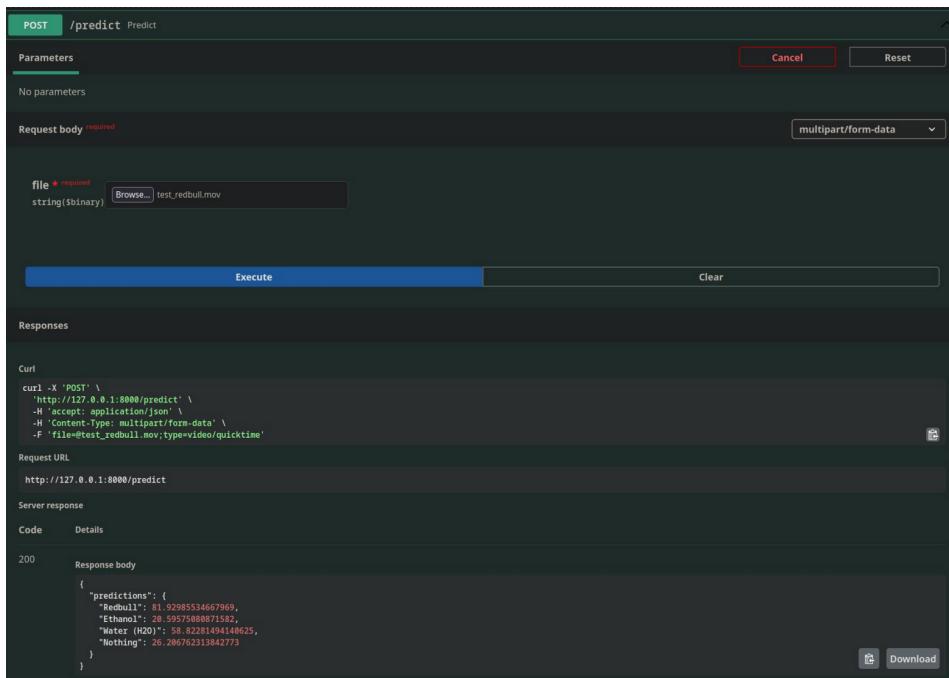
lrvixon@lrvixon-OptiPlex-5090:~/Desktop/lrvixon$ python scripts/train_real_data.py
2023-11-17 18:26:57,444 INFO  Executing training on real data from S3 with config:
2023-11-17 18:26:57,444 INFO  Device: cuda
2023-11-17 18:26:57,444 INFO  Model: CNNLSTM
2023-11-17 18:26:57,444 INFO  Loss Function: CrossEntropyLoss
2023-11-17 18:26:57,444 INFO  Learning Rate: 0.001
2023-11-17 18:26:57,444 INFO  Batch Size: 2
2023-11-17 18:26:57,444 INFO  Padding: 15
2023-11-17 18:26:57,444 INFO  Epochs per Video: 3
2023-11-17 18:26:57,444 INFO  S3 Bucket: s3nina-adam-junka-174436756
2023-11-17 18:26:57,444 INFO  Class Names: ['Bedroll', 'Egħmlu', 'Water (H2O)', 'Nothing']
2023-11-17 18:26:58,474 INFO  Downloading s3://s3nina-adam-junka-174436756/L_R_2025_BB_22_10.mov
2023-11-17 18:26:58,474 INFO  Video /tmp/tmpqqls9ln/L_R_2025_BB_22_10.mov has 33849 frames.
2023-11-17 18:27:39,241 INFO  Generated even roi_boxes with padding: [(0, 0, 528, 594), (432, 0, 576, 594), (486, 52, 524, 524), (540, 52, 486, 524), (594, 52, 540, 524), (648, 52, 594, 524), (702, 0, 576, 594), (756, 0, 528, 594), (810, 0, 576, 594), (864, 0, 528, 594), (918, 0, 576, 594), (972, 0, 528, 594), (1026, 0, 576, 594), (1080, 0, 528, 594), (1134, 0, 576, 594), (1188, 0, 528, 594), (1242, 0, 576, 594), (1296, 0, 528, 594), (1350, 0, 576, 594), (1404, 0, 528, 594), (1458, 0, 576, 594), (1512, 0, 528, 594), (1566, 0, 576, 594), (1620, 0, 528, 594), (1674, 0, 576, 594), (1728, 0, 528, 594), (1782, 0, 576, 594), (1836, 0, 528, 594), (1890, 0, 576, 594), (1944, 0, 528, 594), (1998, 0, 576, 594), (2052, 0, 528, 594), (2106, 0, 576, 594), (2160, 0, 528, 594), (2214, 0, 576, 594), (2268, 0, 528, 594), (2322, 0, 576, 594), (2376, 0, 528, 594), (2430, 0, 576, 594), (2484, 0, 528, 594), (2538, 0, 576, 594), (2592, 0, 528, 594), (2646, 0, 576, 594), (2600, 0, 528, 594), (2654, 0, 576, 594), (2708, 0, 528, 594), (2762, 0, 576, 594), (2816, 0, 528, 594), (2870, 0, 576, 594), (2924, 0, 528, 594), (2978, 0, 576, 594), (3032, 0, 528, 594), (3086, 0, 576, 594), (3140, 0, 528, 594), (3194, 0, 576, 594), (3248, 0, 528, 594), (3302, 0, 576, 594), (3356, 0, 528, 594), (3410, 0, 576, 594), (3464, 0, 528, 594), (3518, 0, 576, 594), (3572, 0, 528, 594), (3626, 0, 576, 594), (3680, 0, 528, 594), (3734, 0, 576, 594), (3788, 0, 528, 594), (3842, 0, 576, 594), (3896, 0, 528, 594), (3950, 0, 576, 594), (4004, 0, 528, 594), (4058, 0, 576, 594), (4112, 0, 528, 594), (4166, 0, 576, 594), (4220, 0, 528, 594), (4274, 0, 576, 594), (4328, 0, 528, 594), (4382, 0, 576, 594), (4436, 0, 528, 594), (4490, 0, 576, 594), (4544, 0, 528, 594), (4598, 0, 576, 594), (4652, 0, 528, 594), (4706, 0, 576, 594), (4760, 0, 528, 594), (4814, 0, 576, 594), (4868, 0, 528, 594), (4922, 0, 576, 594), (4976, 0, 528, 594), (5030, 0, 576, 594), (5084, 0, 528, 594), (5138, 0, 576, 594), (5192, 0, 528, 594), (5246, 0, 576, 594), (5200, 0, 528, 594), (5254, 0, 576, 594), (5308, 0, 528, 594), (5362, 0, 576, 594), (5416, 0, 528, 594), (5470, 0, 576, 594), (5524, 0, 528, 594), (5578, 0, 576, 594), (5632, 0, 528, 594), (5686, 0, 576, 594), (5740, 0, 528, 594), (5794, 0, 576, 594), (5848, 0, 528, 594), (5802, 0, 576, 594), (5856, 0, 528, 594), (5910, 0, 576, 594), (5964, 0, 528, 594), (6018, 0, 576, 594), (6072, 0, 528, 594), (6126, 0, 576, 594), (6180, 0, 528, 594), (6234, 0, 576, 594), (6288, 0, 528, 594), (6342, 0, 576, 594), (6396, 0, 528, 594), (6450, 0, 576, 594), (6504, 0, 528, 594), (6558, 0, 576, 594), (6612, 0, 528, 594), (6666, 0, 576, 594), (6720, 0, 528, 594), (6774, 0, 576, 594), (6828, 0, 528, 594), (6882, 0, 576, 594), (6936, 0, 528, 594), (6990, 0, 576, 594), (7044, 0, 528, 594), (7098, 0, 576, 594), (7152, 0, 528, 594), (7206, 0, 576, 594), (7260, 0, 528, 594), (7314, 0, 576, 594), (7368, 0, 528, 594), (7422, 0, 576, 594), (7476, 0, 528, 594), (7530, 0, 576, 594), (7584, 0, 528, 594), (7638, 0, 576, 594), (7692, 0, 528, 594), (7746, 0, 576, 594), (7700, 0, 528, 594), (7754, 0, 576, 594), (7808, 0, 528, 594), (7862, 0, 576, 594), (7916, 0, 528, 594), (7970, 0, 576, 594), (8024, 0, 528, 594), (8078, 0, 576, 594), (8132, 0, 528, 594), (8186, 0, 576, 594), (8240, 0, 528, 594), (8294, 0, 576, 594), (8348, 0, 528, 594), (8402, 0, 576, 594), (8456, 0, 528, 594), (8510, 0, 576, 594), (8564, 0, 528, 594), (8618, 0, 576, 594), (8672, 0, 528, 594), (8726, 0, 576, 594), (8780, 0, 528, 594), (8834, 0, 576, 594), (8888, 0, 528, 594), (8942, 0, 576, 594), (8996, 0, 528, 594), (9050, 0, 576, 594), (9104, 0, 528, 594), (9158, 0, 576, 594), (9212, 0, 528, 594), (9266, 0, 576, 594), (9320, 0, 528, 594), (9374, 0, 576, 594), (9428, 0, 528, 594), (9482, 0, 576, 594), (9536, 0, 528, 594), (9590, 0, 576, 594), (9644, 0, 528, 594), (9698, 0, 576, 594), (9752, 0, 528, 594), (9806, 0, 576, 594), (9860, 0, 528, 594), (9914, 0, 576, 594), (9968, 0, 528, 594), (10022, 0, 576, 594), (10076, 0, 528, 594), (10130, 0, 576, 594), (10184, 0, 528, 594), (10238, 0, 576, 594), (10292, 0, 528, 594), (10346, 0, 576, 594), (10300, 0, 528, 594), (10354, 0, 576, 594), (10408, 0, 528, 594), (10462, 0, 576, 594), (10516, 0, 528, 594), (10570, 0, 576, 594), (10624, 0, 528, 594), (10678, 0, 576, 594), (10732, 0, 528, 594), (10786, 0, 576, 594), (10840, 0, 528, 594), (10894, 0, 576, 594), (10948, 0, 528, 594), (10902, 0, 576, 594), (10956, 0, 528, 594), (10910, 0, 576, 594), (10964, 0, 528, 594), (10918, 0, 576, 594), (10972, 0, 528, 594), (10926, 0, 576, 594), (10980, 0, 528, 594), (10934, 0, 576, 594), (10988, 0, 528, 594), (10942, 0, 576, 594), (10996, 0, 528, 594), (10950, 0, 576, 594), (10904, 0, 528, 594), (10958, 0, 576, 594), (10912, 0, 528, 594), (10966, 0, 576, 594), (10920, 0, 528, 594), (10974, 0, 576, 594), (10938, 0, 528, 594), (10992, 0, 576, 594), (10956, 0, 528, 594), (10970, 0, 576, 594), (10934, 0, 528, 594), (10988, 0, 576, 594), (10952, 0, 528, 594), (10966, 0, 576, 594), (10920, 0, 528, 594), (10974, 0, 576, 594), (10948, 0, 528, 594), (10962, 0, 576, 594), (10926, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10932, 0, 528, 594), (10986, 0, 576, 594), (10960, 0, 528, 594), (10974, 0, 576, 594), (10942, 0, 528, 594), (10996, 0, 576, 594), (10964, 0, 528, 594), (10978, 0, 576, 594), (10946, 0, 528, 594), (10960, 0, 576, 594), (10924, 0, 528, 594), (10988, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0, 576, 594), (10932, 0, 528, 594), (10996, 0, 576, 594), (10966, 0, 528, 594), (10980, 0, 576, 594), (10954, 0, 528, 594), (10968, 0, 576, 594), (10936, 0, 528, 594), (10990, 0, 576, 594), (10962, 0, 528, 594), (10976, 0, 576, 594), (10944, 0, 528, 594), (10958, 0, 576, 594), (10922, 0, 528, 594), (10986, 0, 576, 594), (10968, 0, 528, 594), (10982, 0, 576, 594), (10950, 0, 528, 594), (10964, 0,
```

7.3.4.2 Endpoint: /predict Endpoint ten przyjmuje plik wideo i zwraca predykcje prawdopodobieństwa dla każdej klasy.

1. **Odbiór wideo:** Endpoint przyjmuje wideo jako plik typu `UploadFile`, który jest tymczasowo zapisywany na dysku.
2. **Segmentacja wideo (Pre-Inference):** Wykonywana jest detekcja obszarów zainteresowania (*ROI*) za pomocą `find_shapes_first_frame`.
3. **Ekstrakcja i Preprocessing Klatek:** Wideo jest przetwarzane na sekwencję `NUM_FRAMES` klatek. Klatki są poddawane transformacjom (`transform`), łączone w tensor wejściowy, przenoszone na `DEVICE` i, w razie potrzeby, uzupełniane zerami.
4. **Wnioskowanie:** Predykcja jest wykonywana w kontekście `torch.no_grad()`. Wynik jest poddawany funkcji aktywacji `F.sigmoid`, a następnie przekształcany w prawdopodobieństwa procentowe.
5. **Odpowiedź:** Zwracany jest słownik `JSON` zawierający wyniki predykcji: `{"predictions": {"Klasa": Procent%}}`.

7.3.4.3 Endpoint: /train Endpoint umożliwia ponowne trenowanie modelu na nowych, pojedynczych plikach wideo, co wspiera adaptacyjną walidację w terenie.

1. **Argumenty:** Przyjmuje plik wideo, etykietę klasy (`class_name`) oraz liczbę epok (`epochs`).
2. **Kontynuacja treningu:** Ładowane są wagi modelu i stan optymalizatora z ostatniego checkpointu (`CHECKPOINT_PATH`), zapewniając **ciągłość treningu**.
3. **Trening On-Demand:** Wideo jest tymczasowo zapisywane, a następnie wywoływana jest funkcja `train_one_video`, która przeprowadza krótką sesję treningową na nowych danych z wykorzystaniem dostarczonej etykiety klasy.
4. **Zapis i czyszczenie:** Po treningu, nowe wagi modelu są zapisywane do `SAVE_PATH`.



Rysunek 18: Widok Swagger endpointu predict dla serwisu modelu, wraz z przykładowym outputem

7.4 Serwis pacjentów

Serwis pacjentów to prosty serwer FastAPI, służący jako symulacja zewnętrznego serwisu, na przykład szpitalnego, oferującego dane pacjentów zgodnie ze standardem HL7 FHIR R4.

7.4.1 API serwisu

Serwis składa się z 3 endpointów, widocznych na rysunku 19.

Larvixon Patients Service 1.0.0 OAS 3.1

/openapi.json

A FHIR-compliant patient service API.
This service provides endpoints for searching and retrieving patient data using the HL7 FHIR (Fast Healthcare Interoperability Resources) standard.

Features

- Search patients by PESEL, first name, or last name
- FHIR R4 compliant Patient resources
- Returns FHIR Bundle for search results

Limitations

Always returns a maximum of 100 patients per search request.

MIT

patients

- GET** /api/patients Search patients
- POST** /api/patients/patients-by-guids Get patients by GUIDs
- GET** /api/patients/{guid} Get patient by GUID

Rysunek 19: Dostępne endpointy

Pozwalają one wyszukać pacjentów po peselu/imieniu i nazwisku, pacjenta po jego unikalnym guidzie (uuid), a także grupę pacjentów po ich guidach. Endpointy są przydatne do znalezienia pacjenta do przypisania do analizy, zoptymalizowanym zapytaniem zwracającym wielu pacjentów do wielu analiz oraz zwróceniem pacjenta do konkretnej analizy.

Endpointy zwracają *bundle* pacjentów. Fragment takiej odpowiedzi widać na rysunku 20

```
{
  "resourceType": "Bundle",
  "type": "searchset",
  "total": 100,
  "entry": [
    {
      "fullUrl": "urn:uuid:7e9acaal-6e9b-45be-be13-ed95a4763a78",
      "resource": {
        "resourceType": "Patient",
        "id": "7e9acaal-6e9b-45be-be13-ed95a4763a78",
        "identifier": [
          {
            "use": "official",
            "system": "http://hl7.org/fhir/sid/pesel",
            "value": "29011116892"
          }
        ],
        "name": [
          {
            "use": "official",
            "family": "Tytko",
            "given": ["Ida"]
          }
        ],
        "telecom": [
          {
            "system": "phone",
            "value": "+48 539 659 368",
            "use": "mobile"
          },
          {
            "system": "email",
            "value": "urszula04@example.org",
            "use": "home"
          }
        ],
        "gender": "female",
        "birthDate": "1929-01-11",
        "address": [
          {
            "use": "home",
            "line": ["plac Krasickiego 17/58"],
            "city": "Kluczbork",
            "postalCode": "89-850",
            "country": "PL"
          }
        ]
      },
      {
        "fullUrl": "urn:uuid:d84a8f96-53ce-4a7d-8bbe-a4c1e567dec4",
        "resource": {
          "resourceType": "Patient"
        }
      }
    }
  ]
}
```

Rysunek 20: Fragment odpowiedzi serwisu pacjentów

7.4.2 Działanie serwisu

Do zarządzania prostą bazą danych SQLite wykorzystano 3 skrypty-komendy Python:

1. clear.py - usunięcie wszystkich pacjentów z bazy danych
2. print_x_first_patients - logowanie danych pacjentów do konsoli do testów
3. seed.py - wykorzystanie paczki Faker do wygenerowania danych pacjentów. Do finalnej wersji wygenerowano 100k pacjentów, każdy z unikalnym PESELem.

Autentykacja odbywa się za pomocą sprawdzenia wartości tokenu API przesyłanego jako header *x-api-token*. Poprawne tokeny API znajdują się w zmiennej środowiskowej *API_TOKENS*.

7.4.3 Deployment

Do *deploymentu* serwisu na Azure wykorzystano Terraform - wykorzystującą technikę Infrastructure as Code do znacznego ułatwienia, zwłaszcza długoterminowo, procesu wdrażania. Serwis został wdrożony jako *kontener docker*. Obraz serwisu budowany jest lokalnie (ze względu na studencką licencję Azure), a następnie jest *pushowany* do repozytorium Azure.

7.4.4 Architektura

Serwis składa się z warstw:

1. **API (Routing)** - endpointy i autentykacja
2. **serwis (logika biznesowa)** - wyszukiwanie pacjentów, konwersja danych z bazy danych na format FHIR
3. **Baza danych (SQLAlchemy ORM)** - model *Patient* z polami: PESEL, imię, nazwisko, data urodzenia, kontakt, adres

7.5 Symulacja

Projekt *larvixon-simulation* zajmuje się symulacją larw pod wpływem narkotyków. Dzięki temu można było wygenerować dane treningowe do walidacji architektury modelu maszynowego, gdy jeszcze nie było danych.

7.5.1 Struktura i zewnętrzne biblioteki

Projekt stosuje standardową strukturę projektów w silniku gier Unity. Skrypty znajdują się w katalogu Scripts, sprite'y w Sprites, sceny w Scenes, prefaby UI oraz larw w Prefabs. Użyto pluginów *flutter_unity_widget* (po stronie Unity w katalogu *FlutterUnityIntegration*) do integracji z Flutterem i *FastScriptReload* do sprytnego omijania powolnej komplikacji kodu C#. Dodatkowo instancje *ScriptableObjects* znajdują się w katalogach Channels i Drugs.

7.5.2 Namespaces

Wybrane, najważniejsze namespaces oraz krótki opis ich funkcjonalności:

- **Context** - przypisanie konkretnych skryptów jako wstrzykowalne moduły za pomocą Zenject, zajmującego się Dependency Injection
- **Drugs** - abstrakcyjna klasa DrugEffect z różnymi parametrami wpływającymi na ruch, po której dziedziczą konkretne narkotyki, na przykład EthanolEffect

- **Events** - zawiera różne kanały wydarzeń, do rozdzielenia wywoływania wydarzeń a obsłużeniem ich
- **Flutter** - tłumaczy wydarzenia z Fluttera na wywołanie wydarzeń w kanałach wydarzeń
- **Larvae** - logika fizycznego ruchu i renderowania. Wewnątrz jest **Larvae.States** - maszyna stanów larwy wraz ze stanami (na przykład MovingState, DeadState, Laying-DownState...) oraz stany wywołane przez narkotyki (CurlledLayingDownState)
- **Main** - zawiera dwa główne managery symulacji: GameManager do obsługi wydarzeń jak pauza, wyjście i LarvaSimulation do generowania (spawn) larw i *wstrzykiwanie* im narkotyków. Zawiera również przydatny Debugger i InputHandlers
- **Recording** - zarządzanie sesjami nagrywania, nagrywanie klatek i konwersja do filmu. Zawiera również **Recording.Config** do konfiguracji sesji nagrywania (czas symulacji, rozdzielcość, narkotyki i wiele innych). Symulacja działa niezależnie od modułu Recording
- **UI** - nasłuchiwa na wydarzenia z kanałów komunikacji, a także wywołuje niektóre wydarzenia

7.5.3 Zastosowane wzorce projektowe

- **State** – Stan larwy jest modulowany przez klasy stanów oraz maszynę stanów, która kontroluje przejścia.
- **Template Method** – Klasa DrugEffect definiuje szablon działania narkotyku, a konkretne efekty nadpisują wybrane kroki.
- **Observer** – System eventów (ScriptableObjects) umożliwia komunikację między komponentami bez bezpośrednich zależności.
- **Dependency Injection** – Zenject wstrzykuje zależności, upraszczając konfigurację i testowanie.
- **Strategy** – Każdy narkotyk to inna strategia modyfikacji ruchu i zachowania larwy.
- **Component** – Logika larwy jest podzielona na wiele MonoBehaviourów, które współpracują kompozycyjnie.
- **Facade** – GameManager udostępnia prosty interfejs do pauzy, wznowienia i restartu gry.
- **Adapter** – MainReceiver tłumaczy komunikaty z Fluttera na eventy Unity.

- **Null Object** – `MovementModifier.Normal` eliminuje potrzebę sprawdzania wartości `null`.
- **Decorator** – `DrugEffectEditor` rozszerza inspektor o dodatkowe informacje.

7.6 Strona z dokumentacją

Repozytorium *larvixon-documentation* to prosta, statyczna strona zbudowana na frameworku *Astro* z integracją *Starlight* do generowania stron dokumentacji. Strona jest hostowana na GitHub Pages. Zawiera pełną dokumentację, raport, plakat, prezentacje na seminarium oraz fiszkę. Strona jest dostępna w języku polskim oraz angielskim, aczkolwiek dokumentacja wymagana przez uczelnię nie została przetłumaczona na język angielski.

8 Implementacja

8.1 Środowisko twórcze i proces kontroli wersji

Kod źródłowy systemu został zorganizowany w architekturze *multi-repo*, gdzie poszczególne komponenty systemu (backend, frontend, model ML, symulacja, serwis pacjentów, dokumentacja) znajdują się w oddzielnych repozytoriach na platformie GitHub. Odnośniki do wszystkich repozytoriów w organizacji LarvixON-ZPI:

- larvixon-backend
- larvixon-frontend
- larvixon-model
- larvixon-patients-service
- larvixon-simulation
- larvixon-documentation

8.1.1 Strategia branchowania i Code Review

W projekcie zastosowano strategię *Feature Branch Workflow*. Główny branch `main` zawierała stabilną wersję produkcyjną kodu. Prace nad nowymi funkcjonalnościami odbywały się na dedykowanych branchach tworzonych od głównego `main`.

W celu zapewnienia wyższej jakości kodu, wprowadzono proces *Code Review*. Merge'owanie zmian z głównym branchem było możliwe tylko poprzez mechanizm *Pull Request* (PR), który wymagał:

- Akceptacji głównego *maintainer'a* repozytorium.
- Pozytywnego przejścia automatycznych testów w potoku CI.

- Rozwiązania konfliktów w kodzie.

Całkiem przydatnym wsparciem recenzji kodu okazał się GitHub Copilot. Co prawda takie narzędzia znajdują głównie literówki, proste błędy logiczne albo pozostawione przypadkiem fragmenty kodu, ale narzędzie na pewno wspomogło proces code review jako swoisty *preprocessing* pull requesta.

8.1.2 Ciągła integracja (CI/CD)

Zaimplementowano zautomatyzowane potoki wdrażania (ang. *Pipelines*) z wykorzystaniem narzędzia **GitHub Actions**. Wdrożono następujące automatyzacje:

- **Backend (Django):** Zdefiniowano workflow `django.yml`, który przy każdym wyknięciu zmian (*push*) lub otwarciu PR uruchamia środowisko Python, instaluje zależności z pliku `requirements.txt` oraz wykonuje testy jednostkowe. Dodatkowo, deployment na Azure jest zautomatyzowany w osobnym pipeline.
- **Frontend (Flutter):** Zastosowano workflow `flutter_ci.yml` oraz `release-build.yml`, które odpowiadają za analizę statyczną kodu, uruchomienie testów oraz budowanie artefaktów aplikacji (w tym wersji Web oraz Android).
- **Symulacja (Unity):** Automatyczne budowanie projektu gotowego do generacji danych na platformy Linux, macOS, Windows i WebGL przez wykorzystanie GitHub Releases.
- **Strona z dokumentacją:** Automatyczny deployment na GitHub pages.
- **Serwis pacjentów:** Automatyczny deployment serwisu na Azure dzięki wykorzystaniu Terraforma i odpowiednich sekretów.

Dzięki temu podejściu błędy regresji były wykrywane na wczesnym etapie, co znacząco przyspieszyło proces integracji poszczególnych modułów systemu.

8.2 Backend

Implementacja warstwy serwerowej została zrealizowana w języku Python z wykorzystaniem frameworka Django. Kod źródłowy zorganizowano w repozytorium systemu kontroli wersji Git, a architektura plików odzwierciedla podział na logiczne aplikacje (tzw. *Django Apps*).

8.2.1 Struktura projektu i organizacja kodu

Projekt backendu (repozytorium `larvixon-backend`) został podzielony na niezależne moduły zgodnie z zasadą *Separation of Concerns*. Każdy moduł posiada ujednoliconą strukturę katalogów, co zwiększa czytelność kodu dla nowych programistów.

Struktura pojedynczego modułu prezentuje się w następujący sposób:

```

larvixon-backend/
|-- przykładowy-moduł/
|   |-- management/      # Polecenia administracyjne
|   |-- migrations/     # Migracje
|   |-- services/        # Logika biznesowa
|   |-- tests/           # Testy jednostkowe i integracyjne (pytest)
|   |-- views/            # Widoki API

```

Struktura całościowa projektu:

```

larvixon-backend/
|-- accounts/          # Obsługa użytkowników
|-- analysis/          # Zarządzanie wynikami analiz
|-- videoprocessor/    # Logika przetwarzania wideo
|   |-- tasks.py        # Zadania asynchroniczne (Celery)
|-- reports/           # Generowanie dokumentów
|-- patients/          # Kartoteka pacjentów
|-- larvixon_site/     # Główna konfiguracja projektu
|   |-- settings.py    # Ustawienia, zmienne środowiskowe
|   |-- celery.py       # Konfiguracja instancji Celery
|-- tests/              # Testy jednostkowe i integracyjne (pytest)
|-- manage.py           # Narzędzie CLI Django
|-- Dockerfile          # Definicja obrazu kontenera

```

Każdy z powyższych modułów realizuje konkretne zadania biznesowe. W Tabeli 4 zestawiono kluczowe klasy i funkcje zaimplementowane w poszczególnych aplikacjach.

Tabela 4: Zestawienie kluczowych elementów implementacyjnych backendu

Moduł	Kluczowe elementy implementacji
accounts	Serwis AuthenticationService obsługujący logowanie, rejestrację i zarządzanie użytkownikami.
videoprocessor	Klasa VideoUploadService walidująca pliki wideo oraz zadanie process_video_task w tasks.py, które deleguje analizę do modelu ML.
patients	Klasa PatientService komunikująca się z serwisem pacjentów (dwie wersje - MockPatientService i ApiPatientService).
reports	Serwis ReportService wykorzystujący bibliotekę ReportLab do generowania dynamicznych plików PDF.
analysis	Zarządzanie analizami, filtrowanie, serializery mapujące relacyjne dane z bazy (Pacjent → Analiza → Wyniki) na format JSON.

8.2.2 Konfiguracja środowiska uruchomieniowego

Aby zapewnić spójność między środowiskiem deweloperskim a produkcyjnym, wykorzystano platformę **Docker**. Aplikacja backendowa została zdefiniowana w pliku Dockerfile, który bazuje na obrazie systemu Linux z zainstalowanym interpreterem Python 3.10. Orkiestracja usług niezbędnych do działania systemu (serwera aplikacji *Django*, bazy danych *PostgreSQL*, brokera wiadomości *Redis* oraz procesu roboczego *Celery Worker*) odbywa się za pomocą narzędzia **Docker Compose**. Poniżej przedstawiono fragment konfiguracji i zależności między komponentami (Rys. 21).

```
📄 docker-compose.yml
1   services:
2     db:
3       image: postgres:16-alpine
4       container_name: postgres
5       restart: unless-stopped
6     >   environment: ...
10    >  ports: ...
12    >  volumes: ...
14    >  healthcheck: ...
23
24     redis:
25       image: redis:alpine
26       container_name: redis
27       restart: unless-stopped
28     >  ports: ...
30     >  healthcheck: ...
35
36     backend:
37       build: .
38       container_name: backend
39       restart: unless-stopped
40     >  ports: ...
42     >  environment: ...
54     depends_on:
55       db:
56         | condition: service_healthy
57       redis:
58         | condition: service_healthy
59
60     worker:
61       build: .
62       container_name: worker
63       restart: unless-stopped
64       command: celery -A larvixon_site worker -l info
65     >  environment: ...
77     depends_on:
78       db:
79         | condition: service_healthy
80       redis:
81         | condition: service_healthy
```

Rysunek 21: Konfiguracja docker compose

8.2.3 Asynchroniczne przetwarzanie i integracja z modelem ML

Kluczowym wyzwaniem implementacyjnym było zapewnienie płynności działania interfejsu użytkownika podczas czasochłonnej analizy wideo. Zastosowano wzorzec *Task Queue* obsługiwany przez bibliotekę **Celery** oraz brokera **Redis**.

Proces przetwarzania, zaimplementowany w module `videoprocessor`, przebiega w trzech etapach:

1. **Upload:** Użytkownik przesyła plik wideo poprzez API. Serwis `VideoUploadService` zapisuje plik na dysku i tworzy rekord w bazie danych ze statusem PENDING.
2. **Kolejkowanie:** Natychmiast po zapisie uruchamiane jest zadanie asynchroniczne `process_video_task`, co pozwala na natychmiastowe zwrócenie odpowiedzi HTTP 200 do klienta.
3. **Przetwarzanie:** Worker Celery pobiera zadanie z kolejki i uruchamia logikę analizy.

Poniższy kod przedstawia implementację zadania w pliku `videoprocessor/tasks.py` (Rys. 22). Wykorzystano dekorator `@shared_task`, który rejestruje funkcję w systemie kolejkowym.

```

videoprocessor > tasks.py > ...
16     @shared_task
17     def process_video_task(analysis_id: int) -> None:
18         logger.info(f"Celery task started for analysis ID {analysis_id}")
19         analysis = None
20
21     try:
22         VideoProcessingService.process_video(analysis_id)
23
24     except VideoAnalysisNotFoundError as e:
25         logger.error(f"Analysis not found: {e}")
26         return
27
28     except (MLPredictionError, VideoFileAccessError, VideoProcessingError) as e:
29         logger.error(f"Video processing error for analysis {analysis_id}: {e}")
30     try:
31         analysis: VideoAnalysis = VideoAnalysis.objects.get(id=analysis_id)
32         analysis.status = VideoAnalysis.Status.FAILED
33         analysis.error_message = f"Processing failed: {str(e)}"
34         analysis.save()
35         logger.info(f"Updated analysis {analysis_id} status to FAILED")
36     except VideoAnalysis.DoesNotExist:
37         logger.error(f"Could not update status - analysis {analysis_id} not found")
38     except Exception as update_error:
39         logger.exception(
40             f"Error updating analysis {analysis_id} status: {update_error}"
41         )
42
43     except Exception as e:
44         logger.exception(f"Unexpected error processing analysis {analysis_id}: {e}")
45     try:
46         analysis: VideoAnalysis = VideoAnalysis.objects.get(id=analysis_id)
47         analysis.status = VideoAnalysis.Status.FAILED
48         analysis.error_message = f"Unexpected error: {str(e)}"
49         analysis.save()
50         logger.info(f"Updated analysis {analysis_id} status to FAILED")
51     except Exception as update_error:
52         logger.exception(
53             f"Error updating analysis {analysis_id} after unexpected error: {update_error}"
54         )
55
56     logger.info(f"Celery task completed for analysis ID {analysis_id}")

```

Rysunek 22: Asynchroniczne wysyłanie wideo do modelu i wyłapanie błędów

8.2.4 Generowanie raportów

System umożliwia generowanie podsumowań badań w formacie PDF. Funkcjonalność ta została wydzielona do modułu `reports`, realizując wzorzec *Service Layer*.

W pliku `reports/services/reports.py` zaimplementowano klasę generatora, która wykorzystuje bibliotekę **ReportLab**. Pozwala ona na programistyczne "rysowanie" dokumentu, co zapewnia precyzyjną kontrolę nad układem elementów (nagłówki, tabele z wynikami, stopki) oraz wysoką jakość wydruku dzięki zastosowaniu grafiki wektorowej. Wygenerowany plik nie jest zapisywany na stałe na dysku, lecz zwracany bezpośrednio jako strumień bajtów (*FileResponse*), co optymalizuje zużycie pamięci masowej.

8.2.5 Automatyczna dokumentacja API

W celu usprawnienia współpracy z zespołem frontendowym oraz ułatwienia testowania endpointów, wdrożono bibliotekę `drf-spectacular`. Generuje ona automatycznie schemat API zgodny ze standardem **OpenAPI 3.0** na podstawie kodu serializerów i widoków Django. Dzięki temu dostępna jest interaktywna dokumentacja (Rys. 23), pozwalająca na wykonywanie próbnych zapytań bezpośrednio z przeglądarki i weryfikację struktury zwracanych danych.

The screenshot shows the Swagger UI interface for the Larvixon Backend API. At the top, it displays the title "Larvixon Backend API 1.0.0 OAS 3.0" and a brief description: "API for larval behavior analysis system - handles user accounts, authentication, and video analysis tracking". A "Authorize" button is located in the top right corner. The main area is organized into sections: "Authentication", "accounts", "analysis", "patients", "reports", "token", and "videoprocessor". Under each section, there is a list of API endpoints with their corresponding HTTP methods and URLs. Most endpoints are marked with a lock icon, indicating they require authentication. The "analysis" section contains several methods for managing analyses, including GET, POST, PUT, PATCH, and DELETE requests. The "reports" section includes a GET request for PDF reports. The "videoprocessor" section includes a POST request for file uploads.

Rysunek 23: Interfejs Swagger UI wygenerowany automatycznie dla backendu LarvixON

8.2.6 Testy

Testy są integralną częścią nowoczesnych projektów informatycznych. W *larvixon-backend* opracowano testy jednostkowe (testujące metody serwisów i modeli) i integracyjne (testujące endpointy). Wykorzystano framework Django TestCase oraz Django REST Framework APITestCase, odpowiednio do testów jednostkowych i integracyjnych.

8.2.6.1 Strategia testowania

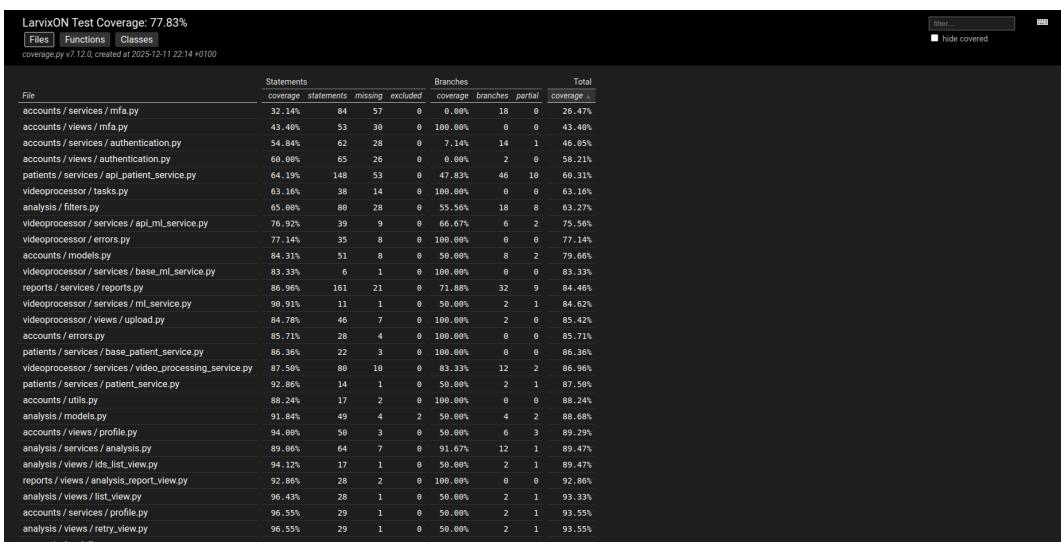
Projekt implementuje warstwową strategię testowania:

- Testy integracyjne API** - testują pełne przepływy żądań HTTP od odbioru parametrów do zwrócenia odpowiedzi. Obejmują validację autentykacji, serializacji danych i kodów statusu HTTP.
- Testy warstwy biznesowej** - testują serwisy w izolacji od warstwy HTTP, fokusując się na logice biznesowej i obsłudze błędów domenowych.
- Testy modeli** - weryfikują poprawność modeli danych, validacji na poziomie bazy danych oraz relacji między encjami.
- Testy asynchroniczne** - testują zadania Celery i przetwarzanie wideo, w tym mockowanie zewnętrznych usług ML.

8.2.6.2 Mockowanie zależności zewnętrznych Projekt wykorzystuje `unittest.mock` do izolowania testu od zewnętrznych API (usługa pacjentów, usługa ML). Pozwala to na:

- Testowanie bez dostępu do sieci
- Testowanie scenariuszy błędów (niedostępność serwisu)
- Szybkie uruchamianie testów
- Niezawodność testów (brak flakiness)

Do obliczenia pokrycia testami kodu wykorzystano paczkę Python Coverage.py. Dodano plik `.coveragerc`, ignorujący analizę pokrycia testami mockowych serwisów, komend do bazy danych czy migracji. Pokrycie testów zostało wyliczone na 77.83% (rysunek 24), co jest zadowalającym wynikiem, pokrywającym całą główną logikę biznesową.



Rysunek 24: Wygenerowany raport z pokrycia testami backendu

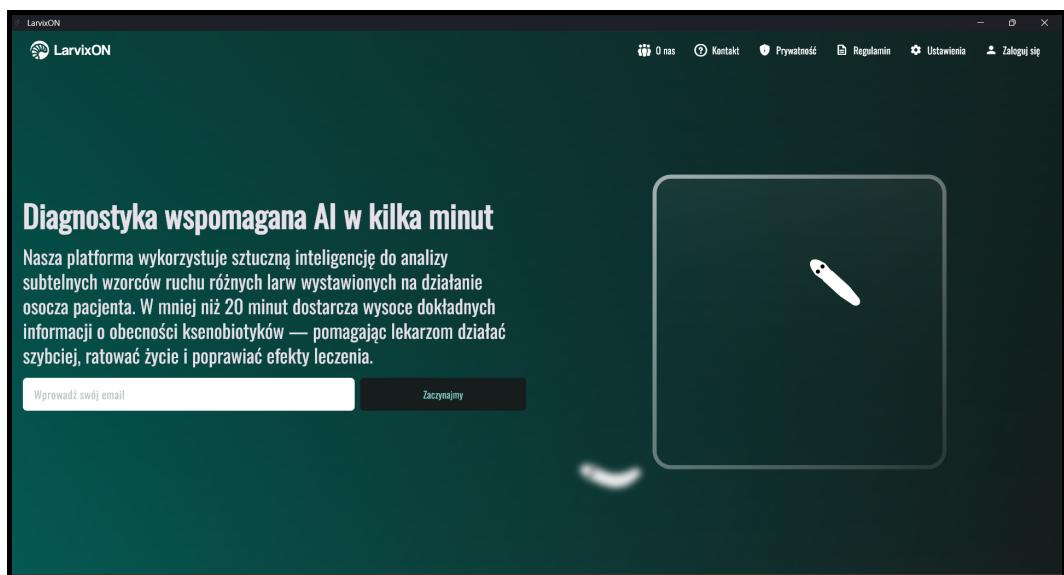
8.3 Interfejs użytkownika

8.3.1 Implementacja widoków i modułów

W ramach optymalizacji interfejsu względem prototypu, pasek nawigacyjny przeniesiono do górnej części ekranu. Zabieg ten pozwolił na wygospodarowanie miejsca dla większej liczby elementów nawigacyjnych. Dodatkowo odświeżono warstwę wizualną aplikacji, modyfikując motyw kolorystyczny oraz stylizację kafelków. Poniżej przedstawiono szczegółowe implementacyjne poszczególnych modułów.

8.3.1.1 Moduł *landing*

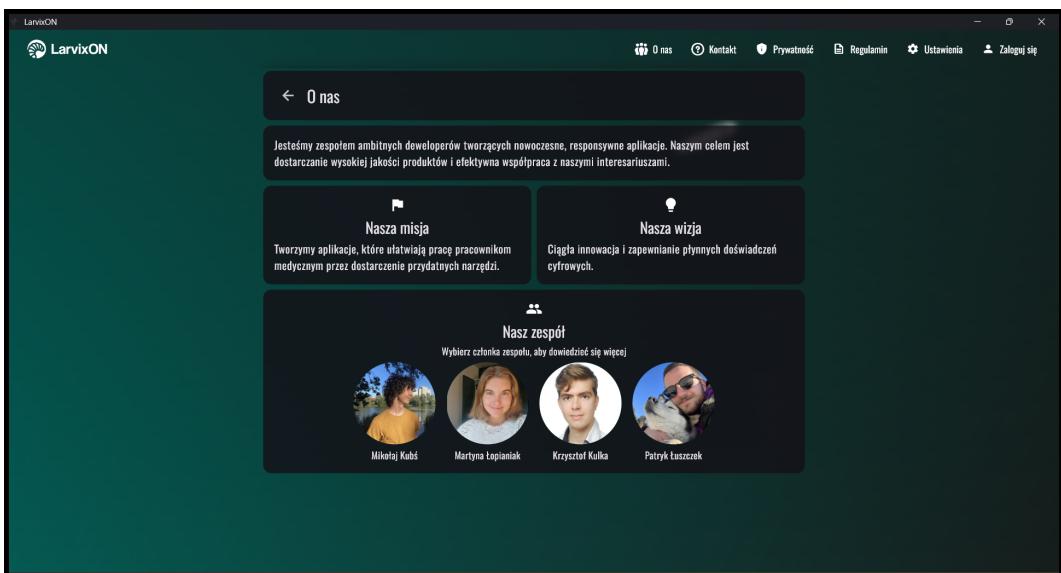
Moduł *landing* pełni rolę wizytówki aplikacji, wprowadzając użytkownika w tematykę diagnostyki wspomaganej sztuczną inteligencją. Implementacja widoku została przedstawiona na rysunku 25. W ramach usprawnień wizualnych względem prototypu (rys. 8), postanowiono rozdzielić formularz logowania od widoku powitalnego. Dzięki temu zabiegowi strona główna zyskała na przejrzystości, a miejsce poświęcono na prezentację wartości systemu oraz zachęcenie do rejestracji.



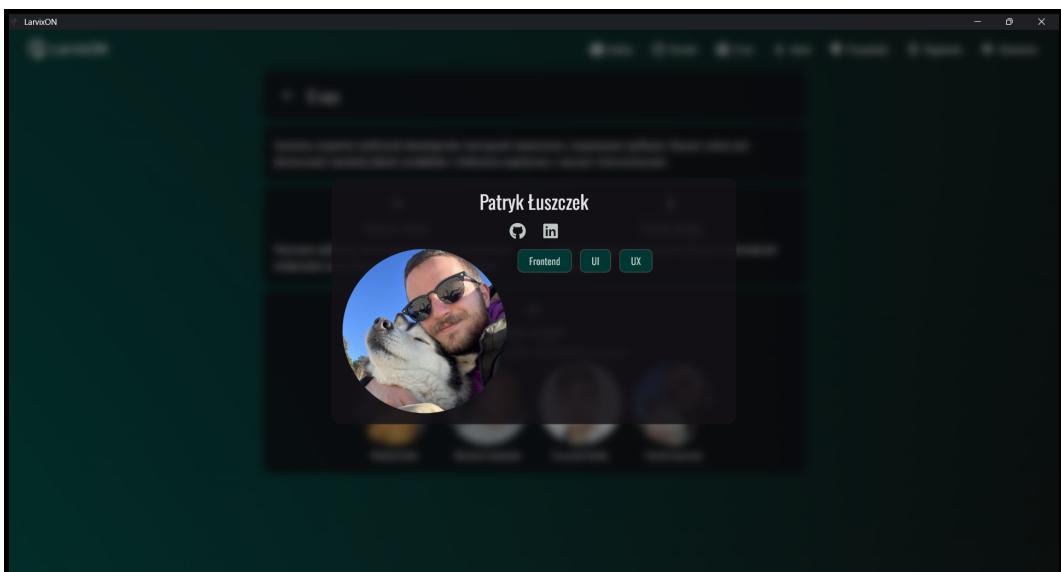
Rysunek 25: Widok powitalny

8.3.1.2 Moduł „O nas”

Moduł ten pełni funkcję informacyjną, prezentując użytkownikowi skład zespołu projektowego. Widok główny (rys. 26) zawiera krótki opis grupy oraz awatary jej poszczególnych członków. Po wybraniu jednej z osób, aplikacja wyświetla widok szczegółowy (rys. 27). Znajdują się tam rozszerzone informacje, takie jak pełniona rola w projekcie oraz aktywne odnośniki do profili zawodowych w serwisach LinkedIn i GitHub.



Rysunek 26: Widok „O nas”

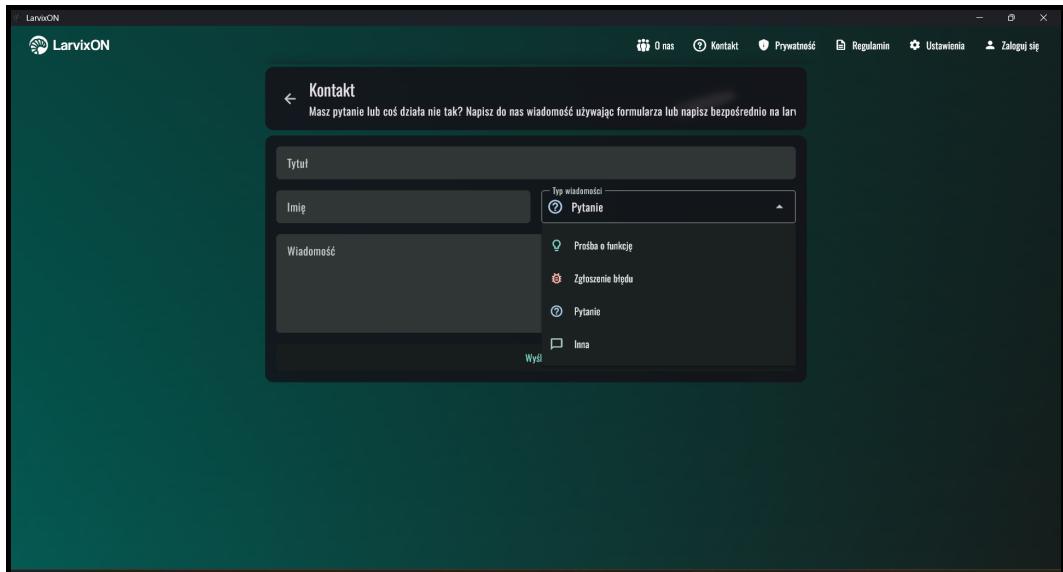


Rysunek 27: Szczegóły członka zespołu

8.3.1.3 Moduł kontaktu

Moduł kontaktu stanowi kanał komunikacji między użytkownikiem a zespołem deweloperskim (wymaganie **US-3.1**). Interfejs formularza (rys. 28) został zaprojektowany w sposób umożliwiający precyzyjne określenie celu wiadomości. Użytkownik, poza standardowymi polami tekstowymi (tytuł, imię, treść), ma do dyspozycji interaktywną listę rozwijaną „Typ wiadomości”. Pozwala ona na wstępную kategoryzację zgłoszenia, np. jako pytanie, zgłoszenie błędu czy prośbę o nową funkcję, co usprawnia proces przetwarzania nadsyłanych informacji przez administratorów systemu. Dodatkowo w celu dokładniejszego zidentyfikowania pro-

blemu, aplikacja generuje w wiadomości informacje dotyczące platformy (np. web, Android, Windows) oraz wersji aplikacji na której użytkownik napotkał błędy (np. v1.0.5)



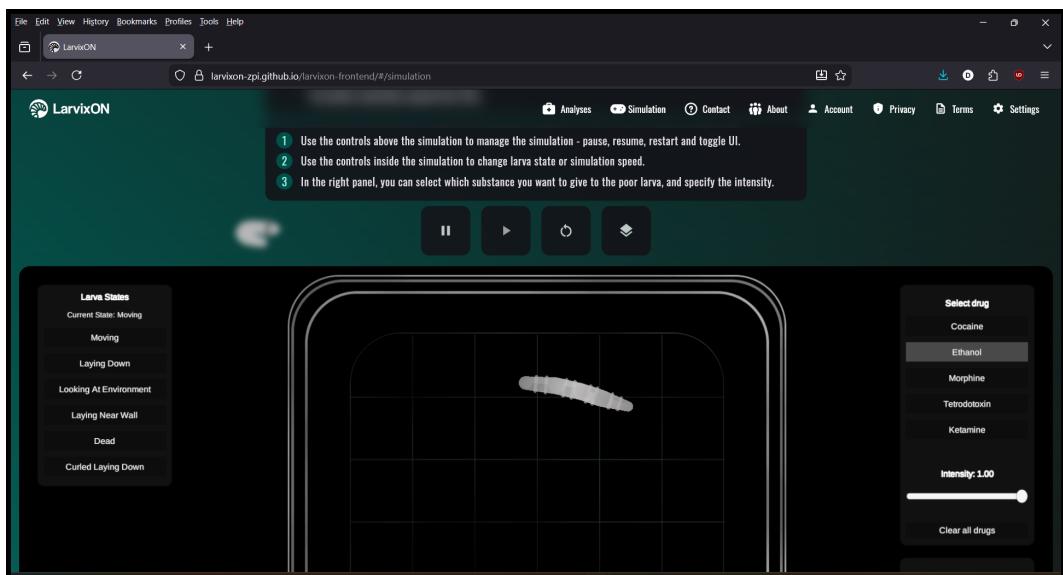
Rysunek 28: Formularz kontaktu

8.3.1.4 Moduł symulacji

W ramach rozszerzenia funkcjonalności aplikacji zaimplementowano interaktywną symulację (rys. 29), umożliwiającą obserwację reakcji larw na ekspozycję na wybrane ksenobiotyki.

Ze względów technicznych oraz użytkowych, symulacja funkcjonuje wyłącznie w wersji przeglądarkowej. Do integracji silnika Unity z aplikacją Flutter wykorzystano wtyczkę *Flutter Unity Widget*¹. Rozwiążanie to nie posiada obecnie wsparcia dla aplikacji desktopowych. Ponadto, ograniczona powierzchnia robocza ekranów mobilnych uniemożliwiały komfortową obsługę symulacji, co zadecydowało o jej wyłączeniu na tych platformach.

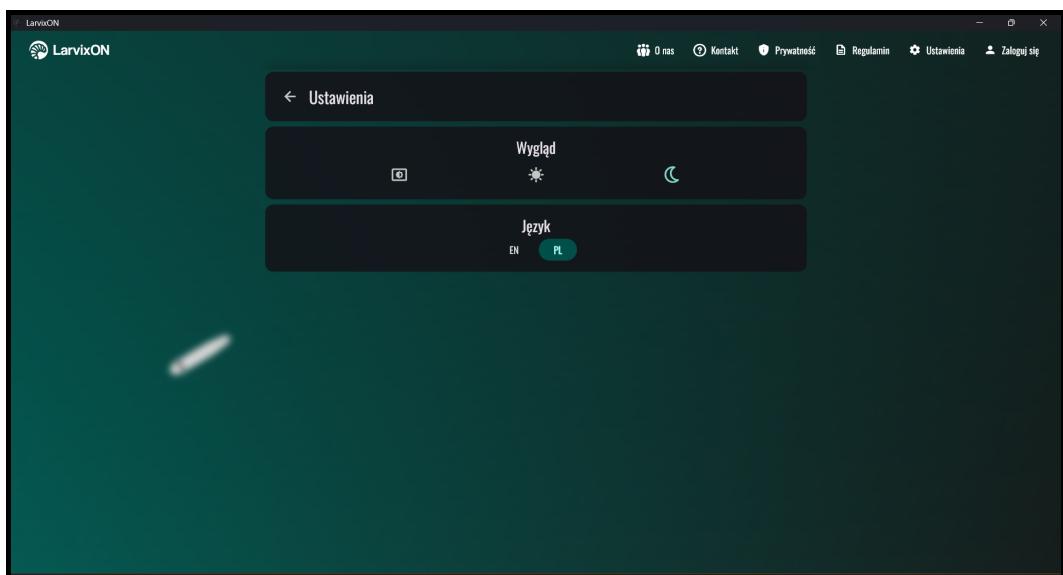
¹Dokumentacja wtyczki dostępna pod adresem: https://pub.dev/packages/flutter_unity_widget



Rysunek 29: Interfejs symulacji Unity zintegrowany z aplikacją webową

8.3.1.5 Moduł ustawień

Moduł ustawień jest odpowiedzialny za zarządzanie preferencjami użytkownika. Widok główny (rys. 30) jest tożsamy z prototypem (rys. 13). Pozwala on użytkownikowi na zmianę preferencji dotyczących motywów aplikacji (jasny lub ciemny) oraz wybór języka interfejsu spośród dostępnych opcji (??).



Rysunek 30: Ustawienia

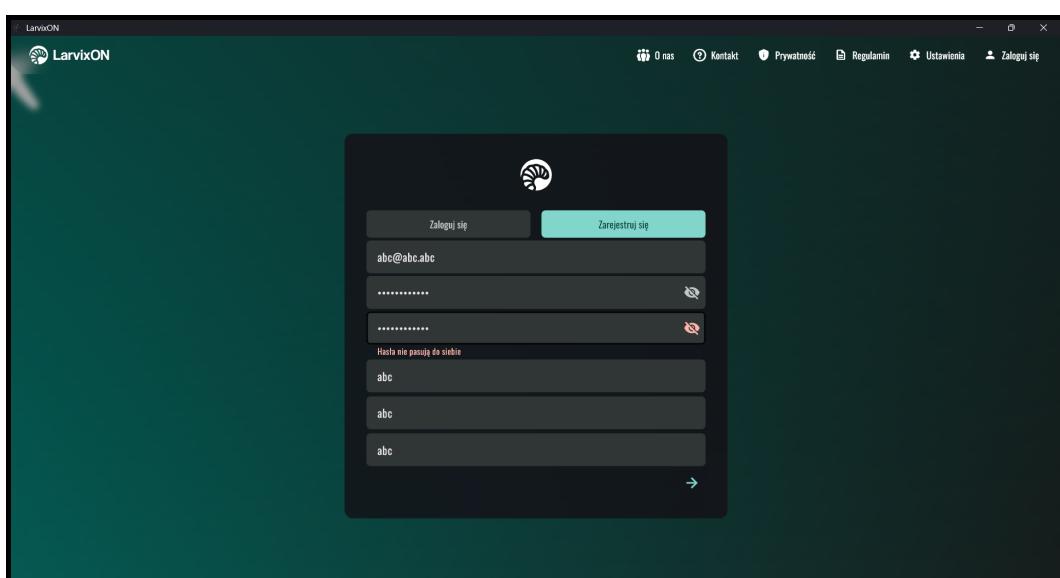
8.3.1.6 Moduł autentykacji

Moduł autentykacji jest odpowiedzialny za zarządzaniem stanem autoryzacji użytkownika aplikacji - stanowi punkt dostępu do chronionych zasobów aplikacji. Widok główny modułu

(rys. 31) obsługuje zarówno proces autoryzacji istniejących użytkowników (**US-1.2**), jak i rejestrację nowych profili (**US-1.1**) poprzez wybór trybu za pomocą przycisków. Podczas próby zatwierdzenia formularza, dane poddawane są wstępnej walidacji po stronie klienta (weryfikowana jest np. poprawność formatu adresu e-mail czy zgodność haseł).

W przypadku wykrycia nieprawidłowości, wyświetlane są adekwatne komunikaty błędów, a odpowiednie pola przechodzą w tryb ciągłej walidacji - każda kolejna modyfikacja wartości jest natychmiast sprawdzana pod kątem poprawności. System obsługuje również błędy zwracane przez serwer, informując użytkownika o takich zdarzeniach jak np. próba utworzenia konta na zajęty już adres e-mail.

Układ formularza jest niemal identyczny do prototypu (rys. 8) - zawiera odpowiednie pola w zależności od trybu oraz przyciski zmiany rejestracja/logowanie.



Rysunek 31: Formularz logowania

8.3.1.7 Moduł analiz

Kluczowym modułem aplikacji jest system zarządzania analizami. Widok główny (rys. 32) prezentuje listę wszystkich badań (**US-2.2**). Widok ten odzwierciedla w dużym stopniu prototyp (rys. 9), jednakże usprawniono wizualnie karty poszczególnych analiz aby przedstawiały najważniejsze informacje w skróconej formie (wyniki, data). Dodatkowo w ramach usprawnień nawigacyjnych wprowadzono możliwość filtrowania analiz (względem statusu, zakresu dat) oraz sortowania.



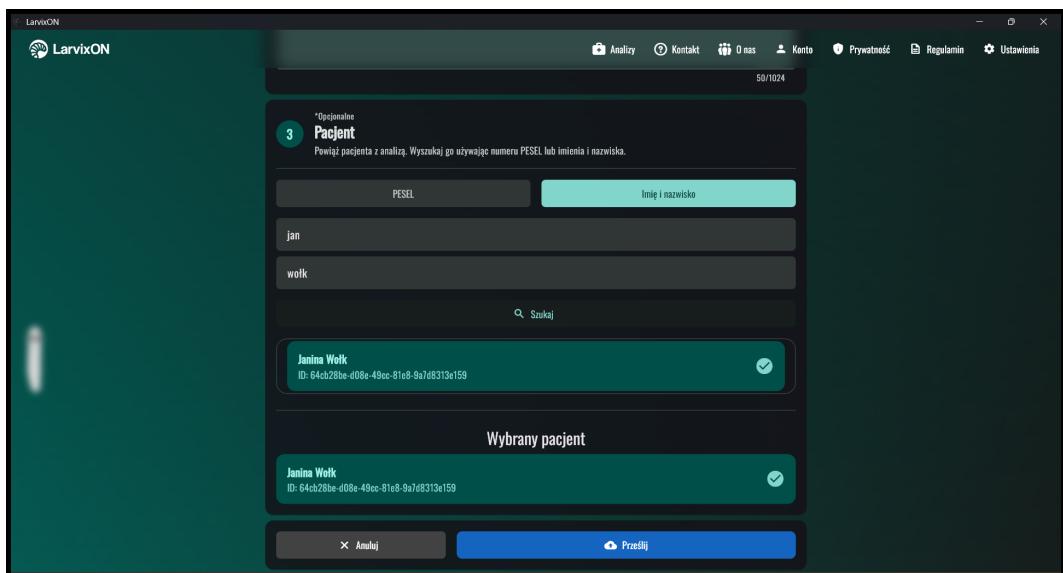
Rysunek 32: Widok główny siatki analiz

Proces zlecania nowej analizy (**US-2.1**) zrealizowano w formie wieloetapowego kreatora. W pierwszym kroku (rys. 33) użytkownik przesyła plik wideo, a następnie ma możliwość dodania opcjonalnego opisu klinicznego przypadku. Ostatnim etapem jest powiązanie badania z kartoteką pacjenta. Wbudowana wyszukiwarka (rys. 34) umożliwia szybką identyfikację osoby na podstawie numeru PESEL lub nazwiska. W odróżnieniu od pierwotnego prototypu (rys. 11), zdecydowano się na logiczną separację poszczególnych kroków procesu, co znacząco poprawiło czytelność interfejsu.

The screenshot shows the 'Nowa analiza' (New analysis) creation wizard:

- Wybór pliku**: Step 1. Maksymalny rozmiar pliku to 3.00 GB. File selected: Screen Recording 2025-03-11 101444.mp4.
- *Opcjonalne Opis**: Step 2. Description: Pacjent wiek 22, nieprzytomny, podejrzanego zatrucie.

Rysunek 33: Kreator analizy, krok 1-2

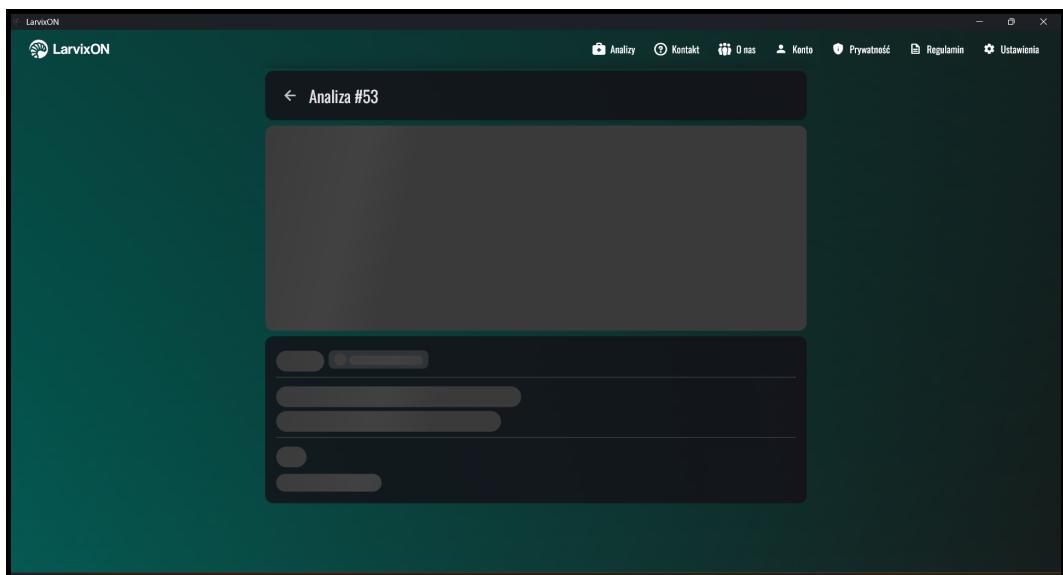


Rysunek 34: Kreator analizy, krok 3

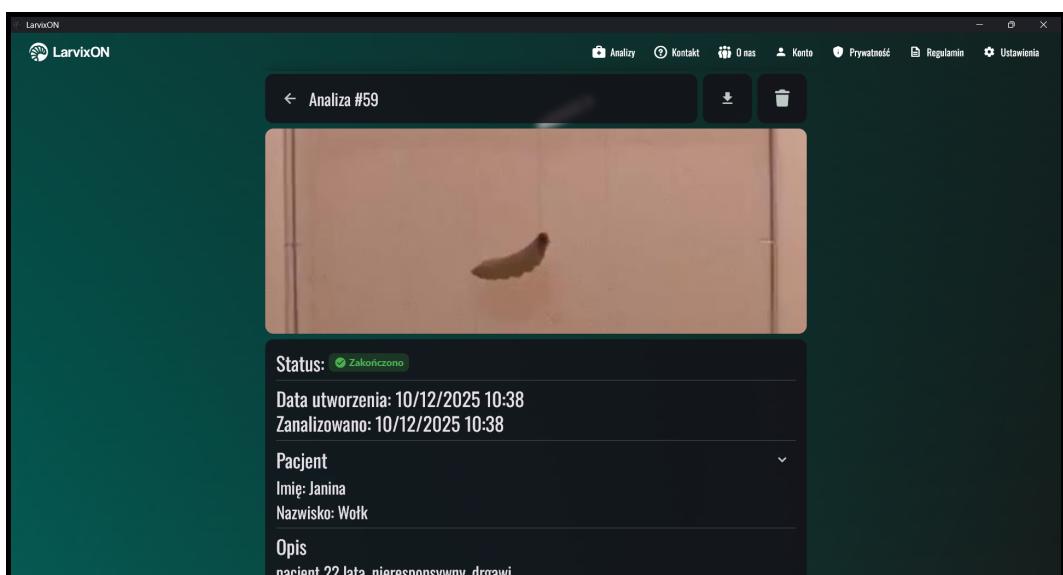
W trosce o User Experience, podczas pobierania szczegółowych danych z serwera, aplikacja wyświetla tzw. szkieletowy ekran ładowania (rys. 35). Informuje to użytkownika, że proces wczytywania jest w toku.

Po załadowaniu danych (rys. 36), zdjęcie pobrane z nagrania źródłowego oraz sekcję metadanych (**US-2.3**). Kluczowym elementem jest panel wyników (rys. 37), który przedstawia wykryte substancje.

Użytkownik ma możliwość usunięcia analizy (**US-2.4**), wyeksportowanie jej do pliku PDF zawierającego wszystkie szczegóły (rys. 39) (**US-2.6**) oraz ponowienia próby analizy jeśli nastąpił błąd (**US-2.5**). W przypadku wybrania opcji usunięcia, system prosi użytkownika o potwierdzenie akcji (rys. 38).



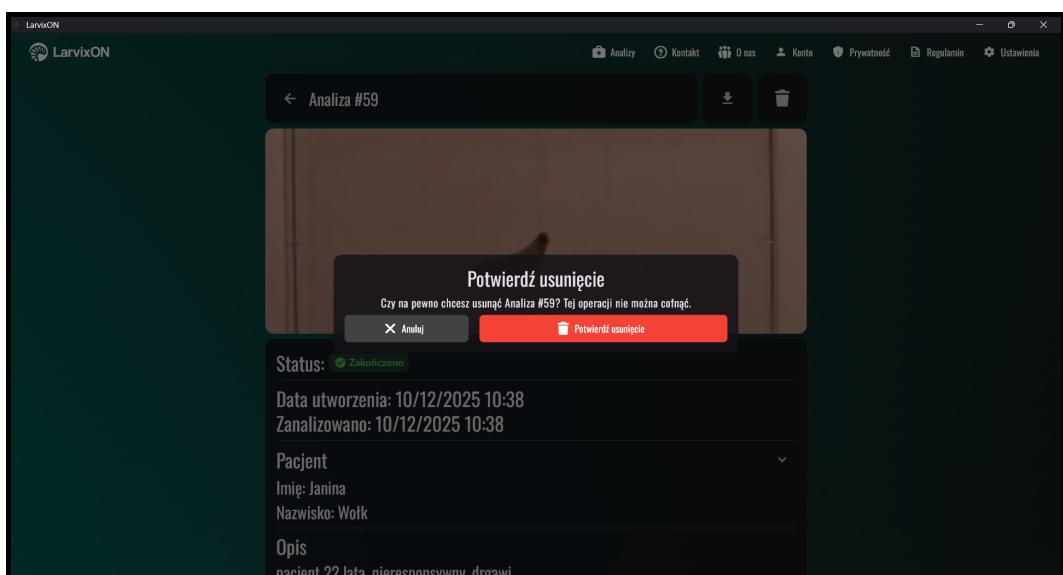
Rysunek 35: Wczytywanie szczegółów analizy



Rysunek 36: Szczegóły analizy



Rysunek 37: Szczegóły analizy - lista wyników



Rysunek 38: Komunikat potwierdzający usunięcie analizy



Analysis #59 Report

Description: pacjent 22 lata, nieresponsywny, drgawi
Status: Completed
Commissioned by: Lavixon Test
Created: 2025-12-10 10:38
Completed: 2025-12-10 10:38

Patient Details:

Name: Janina Wołk
Gender: Female
Birth Date: 1935-07-09
Phone: +48 516 375 744
Email: mmieszala@example.net

Detected Substances

Substance	Confidence Score
Redbull	88.71%
Nothing	77.58%
Water (H ₂ O)	27.04%
Ethanol	18.54%

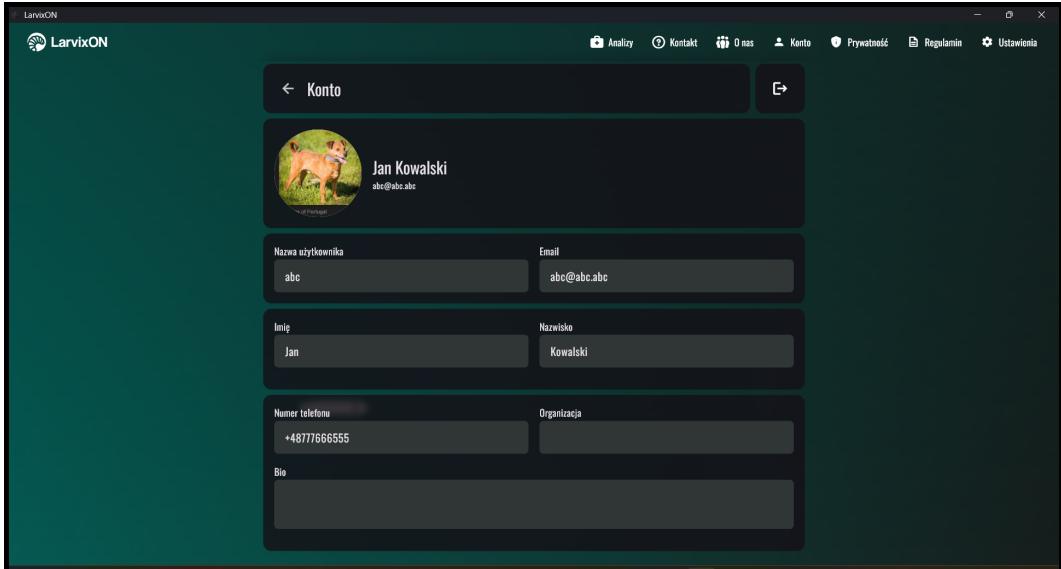
Generated on 2025-12-10 10:40

Rysunek 39: Wygenerowany raport

8.3.1.8 Moduł użytkownika

Moduł użytkownika stanowi centrum zarządzania profilem w aplikacji. Główny widok został przedstawiony na rysunku 40. W sekcji nagłówkowej wyeksponowano kluczowe dane

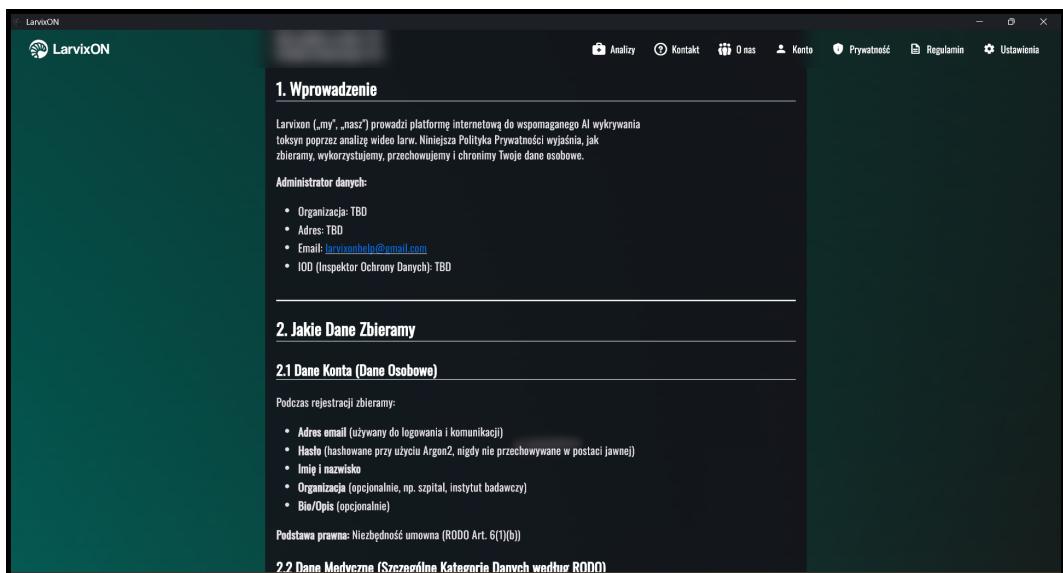
identyfikacyjne, takie jak imię, nazwisko, adres e-mail oraz awatar. Poniżej umieszczono formularz edycyjny, pozwalający na aktualizację szczegółowych informacji: numeru telefonu, nazwy organizacji oraz biogramu (**US-1.3**). Choć ostateczny układ interfejsu bazuje na pierwotnym prototypie (rys. 12), wprowadzono w nim usprawnienia wizualne i funkcjonalne, w tym obsługę zdjęcia profilowego oraz przycisk umożliwiający wylogowanie z serwisu.



Rysunek 40: Implementacja widoku konta użytkownika

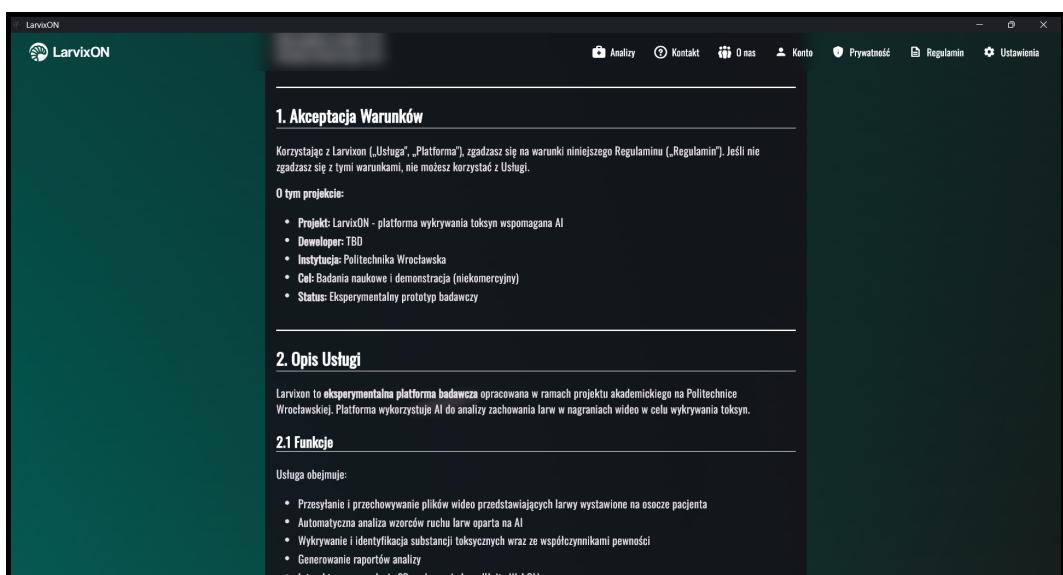
8.3.1.9 Dodatkowe widoki

Polityka prywatności. Ze względu na wrażliwość danych przesyłanych do systemu, zdecydowano się wprowadzić widok Polityki Prywatności (rys. 41). Realizuje on obowiązek informacyjny względem użytkowników, zgodnie z wymogami ochrony danych osobowych (RODO). Dokument ten prezentuje szczegółowy zakres zbieranych informacji, obejmujący m.in. dane konta czy sposób przetrzymywania przesyłanych nagrań.



Rysunek 41: Polityka prywatności

Regulamin korzystania z systemu. Sekcja Regulaminu (rys. 42) definiuje zasady korzystania z platformy, kładąc szczególny nacisk na jej charakter badawczy i niekomercyjny. Użytkownik jest informowany, że aplikacja stanowi eksperimentalny prototyp rozwijany w ramach projektu akademickiego. Tekst reguluje kwestie związane z akceptacją warunków świadczenia usługi oraz opisuje główne funkcje systemu.



Rysunek 42: Regulamin

8.3.2 Obsługa błędów przy komunikacji z API

W celu ujednolicenia mechanizmu detekcji i obsługi błędów w warstwie prezentacji, zastosowano wzorzec kreacyjny **Metoda wytwórcza**. Wprowadzono hierarchię klas, której podstawą jest abstrakcyjna klasa **Failure**. Pozwala ona w sposób jednoznaczny zidentyfikować przyczynę niepowodzenia operacji i podjąć odpowiednią akcję w interfejsie użytkownika (np. wyświetlenie komunikatu, wylogowanie użytkownika).

Błędy wynikające z komunikacji sieciowej zostały zgrupowane w klasie **ApiFailure**, dziedziczącej po **Failure**. Klasa ta przechowuje kod statusu HTTP, wiadomość dla użytkownika oraz opcjonalne, dedykowane pola. W celu dokładnej obsługi zdarzeń, zdefiniowano podklasy reprezentujące konkretne kategorie błędów. Przykładowe błędy:

- **Błędy klienta (4xx)** (rys.43): BadRequestFailure (400), ValidationFailure (400 oraz wykryte błędy pól), UnauthorizedFailure (401) itp.
- **Błędy serwera (5xx)**: InternalServerErrorFailure (500), BadGatewayFailure (502) itp.
- **Nieznane błędy**: UnknownApiFailure (np. w przypadku braku kodu statusu).

```

/// 400 Bad Request
base class BadRequestFailure extends ApiFailure {
| const BadRequestFailure({required super.message}) : super(statusCode: 400);
}

/// 400 Bad Request – with field validation errors
base class ValidationFailure extends BadRequestFailure {
    final Map<String, String> fieldErrors;
    final dynamic responseData;

    const ValidationFailure({
        required this.fieldErrors,
        this.responseData,
        required super.message,
    });
    @override
    String toString() {
        return "${super.toString()} ($fieldErrors)";
    }
}

/// 401 Unauthorized
final class UnauthorizedFailure extends ApiFailure {
    const UnauthorizedFailure({required super.message}) : super(statusCode: 401);
}

/// 403 Forbidden
final class ForbiddenFailure extends ApiFailure {
    const ForbiddenFailure({required super.message}) : super(statusCode: 403);
}

/// 404 Not Found
final class NotFoundFailure extends ApiFailure {
    const NotFoundFailure({required super.message}) : super(statusCode: 404);
}

```

Rysunek 43: Przykładowe błędy 4xx

Bazowa klasa `ApiFailure` wykorzystuje konstruktor fabryczny (rys. 44) zdefiniowany jako `factory ApiFailure.fromResponse`. Słowo kluczowe **factory** w języku Dart pozwala na to, aby konstruktor nie tworzył zawsze nowej instancji tej samej klasy, lecz mógł zwracać instancje klas pochodnych. Konstruktor ten przyjmuje jako argument obiekt typu `Response?` (gdzie znak „?” oznacza typ, który może również być typu null). Na podstawie kodu statusu HTTP zawartego w odpowiedzi, fabryka instancjonuje odpowiedni typ błędu domenowego.

```

base class ApiFailure extends Failure {
  final int statusCode;
  const ApiFailure({required this.statusCode, required super.message});
  factory ApiFailure.fromResponse(Response? response) {
    final statusCode = response?.statusCode ?? -1;
    final data = response?.data;
    if (response == null) {
      return const UnknownApiFailure(statusCode: 0, message: 'Unknown error');
    }
    switch (statusCode) {
      case 400:
        final fieldErrors = _extractFieldErrors(data);
        if (data != null) {
          return ValidationFailure(
            fieldErrors: fieldErrors,
            message: fieldErrors.isNotEmpty
              ? 'validation_failed'
              : 'bad_request',
            responseData: data,
          );
        }
        return const BadRequestFailure(message: 'bad_request');
      case 401:
        return const UnauthorizedFailure(message: 'unauthorized');
      case 403:
        return const ForbiddenFailure(message: 'forbidden');
      case 404:
        return const NotFoundFailure(message: 'not_found');
      case 405:
        return const MethodNotAllowedFailure(message: 'method_not_allowed');
    }
  }
}

```

Rysunek 44: Konstruktor fabryczny ApiFailure

W celu integracji z biblioteką klienta HTTP, wykorzystano mechanizm rozszerzeń, dostępny w języku Dart. Zaimplementowano rozszerzenie **ApiFailureMapperX** na klasie **DioException** (rys. 45). Klasa DioException jest wyjątkiem zgłaszanym przez bibliotekę dio w momencie niepowodzenia żądania sieciowego. Dzięki zastosowaniu rozszerzenia, techniczny wyjątek biblioteki jest mapowany bezpośrednio na opisany wyżej domenowy obiekt **ApiFailure**, co pozwala na zachowanie czystości kodu.

```

extension ApiFailureMapperX on DioException {
    ApiFailure toApiFailure() {
        if (response == null) {
            switch (type) {
                case DioExceptionType.connectionTimeout:
                case DioExceptionType.receiveTimeout:
                case DioExceptionType.sendTimeout:
                    return const RequestTimeoutFailure(message: 'timeout');
                case DioExceptionType.connectionError:
                    return const ServiceUnavailableFailure(message: 'no_connection');
                default:
                    return const UnknownApiFailure(
                        statusCode: 0,
                        message: 'Unknown error',
                    );
            }
        }
        return ApiFailure.fromResponse(response);
    }
}

```

Rysunek 45: Rozszerzenie klasy DioException

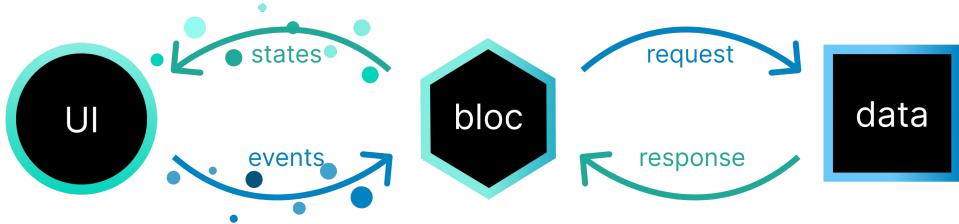
Tak przygotowany, otypowany obiekt błędu trafia następnie do warstwy zarządzania stanem (komponentu BLoC). Dzięki precyzyjnej hierarchii klas błędów, logika biznesowa może podjąć decyzję adekwatną do zaistniałej sytuacji. System może zareagować w sposób jawnny, wyświetlając użytkownikowi komunikat, lub podjąć próbę ponowienia operacji bez zwracania uwagi użytkownika (np. przy chwilowej utracie połączenia).

8.3.3 Zarządzanie stanem

W celu zarządzania stanem aplikacji w warstwie prezentacji postanowiono wykorzystać popularną bibliotekę flutter_bloc², która stanowi implementację wzorca BLoC (*Business Logic Component*).

Wzorzec BLoC opiera się na reaktywnym paradygmacie programowania i wykorzystaniu strumieni. Przepływ danych (rys.46) odbywa się w następujący sposób: warstwa prezentacji emituje **zdarzenia**, które są przetwarzane przez BLoC, skutkując emisją nowego **stanu**, który wymusza odświeżenie wybranych widoków.

²Oficjalna dokumentacja biblioteki flutter_bloc: <https://bloclibrary.dev/>



Rysunek 46: Schemat przepływu danych w architekturze BLoC. Źródło: [1]

8.3.3.1 Struktura stanu

W projekcie przyjęto podejście oparte na pojedynczej, niemutowalnej klasie stanu, która aggrezuje wszystkie niezbędne dane dla danego widoku. Kluczowym elementem tej struktury jest:

- **Status operacji** – zrealizowany za pomocą typu wyliczeniowego enum (np. `initial`, `loading`, `success`, `error`), który determinuje, co aktualnie dzieje się w obrębie danego komponentu.
- **Przechowywanie danych** – stan zawiera pola dla danych wynikowych oraz potencjalnych komunikatów błędów.
- **Niemutowalność i metoda copyWith** – ze względu na wymogi biblioteki BLoC, każda zmiana stanu wiąże się z utworzeniem nowej instancji obiektu stanu. W tym celu zaimplementowano metodę `copyWith`, która pozwala na skopiowanie obecnego stanu z nadpisaniem tylko wybranych pól.

8.3.3.2 Optymalizacja odświeżania widoku

Aby zapobiec zbędnym przebudowom interfejsu, klasa stanu rozszerza bibliotekę **Equatable**³. Pozwala ona na porównywanie obiektów przez wartość, a nie przez referencję. Dzięki temu, jeśli nowa instancja stanu zawiera identyczne dane co poprzednia, mechanizm BLoC nie wymusi odświeżenia komponentów prezentacji, co pozytywnie wpływa na wydajność aplikacji.

8.3.3.3 Przykład - moduł ustawień

Jako przykład zastosowania zarządzania stanem wybrano moduł ustawień. Pozwala on użytkownikowi na modyfikację preferencji, takich jak język interfejsu czy motyw (jasny/ciemny).

Za logikę biznesową odpowiada w tym przypadku klasa `SettingsCubit` (będąca uproszczoną wersją BLoC, rys.48), która operuje na stanie `SettingsState` (rys. 47). Stan ten przechowuje dwie kluczowe informacje:

³Oficjalna dokumentacja biblioteki equatable: <https://pub.dev/documentation/equatable/latest/>

- theme: aktualnie wybrany tryb wyświetlania,
- locale: aktualnie wybrany język aplikacji.

```
class SettingsState extends Equatable {
  final Locale locale;
  final ThemeMode theme;
  const SettingsState({required this.locale, required this.theme});

  @override
  List<Object> get props => [locale, theme];

  SettingsState copyWith({Locale? locale, ThemeMode? theme}) {
    return SettingsState(
      locale: locale ?? this.locale,
      theme: theme ?? this.theme,
    );
  }
}
```

Rysunek 47: Definicja klasy stanu SettingsState

Przepływ sterowania w przypadku zmiany motywu wygląda następująco:

1. Użytkownik w widoku ustawień wybiera nowy motyw. Skutkuje to wywołaniem metody `setTheme` w `SettingsCubit`.
2. Metoda ta komunikuje się z warstwą repozytorium, aby trwale zapisać wybór użytkownika w pamięci urządzenia.
3. Po poprawnym zapisie, Cubit emituje nową instancję stanu metodą `emit(..)`.

```
class SettingsCubit extends Cubit<SettingsState> {
  final SettingsRepository repository;
  final List<Locale> supportedLocales;
  SettingsCubit({required this.repository, required this.supportedLocales})
    : super(const SettingsState(locale: Locale('en'), theme: ThemeMode.system));

  Future<void> setTheme({required ThemeMode theme}) async {
    final result = await repository
      .setThemeMode(themeMode: theme)
      .flatMap(_ => repository.getThemeMode())
      .run();
    result.match((e) {}, (theme) {
      emit(state.copyWith(theme: theme));
    });
  }
}
```

Rysunek 48: Fragment definicji klasy SettingsCubit

Reakcja interfejsu jest natychmiastowa dzięki zastosowaniu widgetu BlocBuilder, który nasłuchuje na zmiany stanu podanego komponentu BLoC, na najwyższym poziomie drzewa widgetów. Jak przedstawiono na rys. 49, każda emisja nowego stanu powoduje przebudowanie głównego widgetu aplikacji z nowymi parametrami themeMode oraz locale, co dynamicznie zmienia wygląd całej aplikacji.

```
BlocBuilder<SettingsCubit, SettingsState>(
    builder: (context, state) {
        return MaterialApp.router(
            theme: AppTheme.appThemeLight,
            darkTheme: AppTheme.appThemeDark,
            themeMode: state.theme,
            locale: state.locale,
            routerConfig: _appRouter.router,
            localizationsDelegates: AppLocalizations.localizationsDelegates,
            supportedLocales: AppLocalizations.supportedLocales,
            debugShowCheckedModeBanner: false,
        ); // MaterialApp.router
    },
), // BlocBuilder
```

Rysunek 49: Wykorzystanie stanu ustawień do zmiany preferencji na najwyższym poziomie aplikacji

8.3.4 Automatyczna dokumentacja kodu

W ekosystemie języka Dart standardem jest wykorzystanie narzędzia dartdoc⁴, które umożliwia automatyczne generowanie dokumentacji API bezpośrednio z kodu źródłowego.

Kluczowym elementem są tutaj tzw. *komentarze dokumentujące*, oznaczane w składni języka za pomocą potrójnego ukośnika (///). Pozwalają one na opisywanie klas, metod oraz zmiennych z wykorzystaniem formatowania **Markdown**. Dzięki temu możliwe jest nie tylko dodawanie opisów tekstowych, ale również:

- tworzenie list i tabel,
- umieszczanie fragmentów przykładowego kodu,
- tworzenie aktywnych odnośników do innych elementów kodu (poprzez ujęcie nazwy klasy lub metody w nawiasy kwadratowe, np. [ApiFailure]).

Wynikiem działania narzędzia jest zestaw statycznych plików HTML, prezentujący strukturę projektu w czytelnej formie. Przykład został przedstawiony na rysunku 50.

⁴Oficjalna dokumentacja narzędzia dartdoc: <https://dart.dev/tools/dart-doc>

Domain entity representing a video analysis request and its results.

Tracks the lifecycle of a larva detection analysis from upload through ML processing to completion.

Part of Clean Architecture domain layer.

Example:

```
final analysis = Analysis(
  id: 123,
  uploadedAt: DateTime.now(),
  status: AnalysisProgressStatus.processing,
  patient: patient,
);
```

Inheritance
Object > Equatable > Analysis

Constructors

```
Analysis({required int id, required DateTime uploadedAt, String? description, AnalysisProgressStatus status = AnalysisProgressStatus.pending,
          DateTime? analysedAt, String? thumbnailUrl, AnalysisResults? results, Patient? patient, String? errorMessage})
const
```

Rysunek 50: Dokumentacja klasy Analysis

8.3.5 Testy akceptacyjne i z udziałem użytkowników

Celem testów z udziałem użytkowników była weryfikacja użyteczności interfejsu oraz potwierdzenie, że zaimplementowane funkcjonalności pokrywają zdefiniowane wymagania biznesowe.

Testy przeprowadzono w oparciu o przygotowane scenariusze testowe, odwzorowujące typowe ścieżki postępowania użytkownika w systemie. W testach aktywnie brali udział członkowie zespołu deweloperskiego oraz (sporadycznie) interesariusze.

Szczególną uwagę zwrócono na:

- **Intuicyjność nawigacji** – czy użytkownik potrafi samodzielnie poruszać się po aplikacji bez uprzedniego szkolenia.
- **Obsługę błędów** – czy system czytelnie komunikuje napotkane problemy.
- **Responsywność** – weryfikacja czytelności interfejsu na urządzeniach mobilnych oraz desktopowych.

Poniższa tabela (tab. 5) prezentuje przykładowe scenariusze testowe, które zostały podane weryfikacji manualnej.

ID Scenariusza	Opis czynności	Oczekiwany rezultat
TC-01	Rejestracja nowego użytkownika z podaniem niepoprawnego formatu adresu e-mail.	System blokuje wysyłkę formularza i wyświetla komunikat weryfikacji pod odpowiednim polem.

ID Scenariusza	Opis czynności	Oczekiwany rezultat
TC-02	Logowanie przy użyciu poprawnych danych uwierzytelniających.	Użytkownik zostaje przekierowany do widoku listy analiz.
TC-03	Przesłanie pliku wideo w formacie o dopuszczonym rozmiarze (<3 GB).	Plik zostaje przesłany, pojawia się pasek postępu, a następnie analiza zmienia status na "Przetwarzanie".
TC-04	Odczyt wyników analizy i generowanie raportu PDF.	Wyświetlają się wykryte tokeny. Po kliknięciu "Eksportuj" generowany jest plik zgodny z szablonem.

Tabela 5: Zestawienie wybranych manualnych scenariuszy testowych

Przeprowadzone testy manualne pozwoliły na wykrycie i wyeliminowanie szeregu błędów interfejsu na wczesnym etapie prac. Najważniejsze poprawki wdrożone na podstawie informacji zwrotnej od testerów to:

- Rozdzielenie procesu zlecania analizy na kroki:** Pierwotnie formularz znajdował się na jednym długim ekranie, co testerzy uznali za przytłaczające. Wprowadzono "Kreator analizy".
- Wskaźniki ładowania:** Dodano ekrany szkieletowe m.in. przy pobieraniu szczegółów analizy, ponieważ użytkownicy nie mieli pewności, czy aplikacja się zawiesiła, czy pobiera dane.
- Komunikaty błędów:** Zastąpiono techniczne kody błędów API (np. "500 Internal Server Error") czytelnymi dla człowieka komunikatami (np. "Wystąpił problem z serwerem, spróbuj ponownie później").

Ostateczna wersja systemu przeszła pomyślnie testy akceptacyjne, realizując założone wymagania funkcjonalne.

8.4 Implementacja modelu uczenia maszynowego (Larvixon ML)

Implementacja modelu *ML* (PyTorch) była zorientowana na wydajne przetwarzanie czasowo-przestrzenne danych wideo i zapewnienie środowiska gotowego do serwowania predykcji.

8.4.1 Architektura Modelu i Warstwy

Wybrano architekturę hybrydową **CNN+LSTM** zoptymalizowaną do klasyfikacji krótkich sekwencji wideo.

1. Warstwa Ekstrakcji Cech (CNN):

- Wykorzystano gotową, wstępnie wytrenowaną architekturę **ResNet18** (dostępna w PyTorch), co znaczco przyspieszyło zbieżność i zmniejszyło wymaganą ilość danych treningowych.
- Ostatnie warstwy klasyfikacyjne sieci *ResNet18* zostały usunięte. Rolą *CNN* jest działanie jako **extractor cech**, redukujący każdą klatkę o wymiarach [3, 112, 112] do wektora cech wysokiego poziomu (tzw. *embedding*).

2. Warstwa Modelowania Czasowego (LSTM):

- Wektory cech, sekwencyjnie generowane przez warstwę *CNN*, są podawane do sieci **LSTM** (Long Short-Term Memory).
- *LSTM* skutecznie uczy się zależności i dynamiki czasowej, tj. w jaki sposób ewolucja wzorców ruchu (modyfikowanych przez *CurrentMovementModifier* w symulacji) koreluje z klasą substancji.

3. Warstwa Klasyfikacyjna (Fully Connected):

- Finalny wektor stanu z *LSTM* jest przekazywany przez warstwy gęste (FC), które mapują cechy behawioralne na zdefiniowaną przestrzeń klas (substancji).
- Na wyjściu stosowana jest funkcja aktywacji *softmax* do generowania rozkładu prawdopodobieństwa predykcji.

8.4.2 Trening, Walidacja i Wnioskowanie

Faza implementacji obejmowała kluczowe aspekty związane z zarządzaniem danymi i środowiskiem.

- **Zbiór Danych:** W początkowej fazie wykorzystano dane wygenerowane przez **La-rivixon Simulation**. Dzięki precyzyjnemu modelowaniu modyfikatorów ruchu (*np. SpeedMultiplier, RandomnessMultiplier*) oraz ich dynamiki czasowej (*onsetTime, duration, comedownTime*), dane syntetyczne miały wysoką wartość wczesnowalidacyjną dla architektury.
- **Funkcja Straty:** Zastosowano **Cross-Entropy Loss** jako funkcję straty dla problemu klasyfikacji wieloklasowej.
- **Wnioskowanie (Inference):** Moduł *ML* został skonteneryzowany (**Docker**) wraz z zależnościami *PyTorch*. Wnioskowanie jest serwowane za pośrednictwem dedykowanego API opartego na **FastAPI**, co zapewnia minimalne opóźnienia i wysoką przepustowość.

- **Walidacja:** Proces implementacji był ściśle konsultowany z dr Piórkowską w celu walidacji podejścia, prawidłowości implementacji modelu oraz badania jego celności na danych spoza treningowego zbioru danych, co przyczyniło się do osiągnięcia stabilnej dokładności w zakresie 60%-80%.

8.5 Implementacja symulacji

8.5.1 Algorytm fizycznego ruchu larw

Nietrywialnym algorytmem jest sposób poruszania się larw w symulacji. Każda larwa to tak naprawdę 5 punktów, gdzie punkt nr 1 odpowiada jej głowie, a punkt nr 5 jej odwłokowi (reszta po kolei między tymi punktami). Dla każdej pary sąsiadujących punktów jest zdefiniowana preferowana odległość, która może się zmieniać. Ruch larwy składa się z trzech faz:

1. Zwiększenie preferowanej odległości między punktami 1. a 2. Oznacza to, że głowa larwy podróżuje do przodu.
2. Reset preferowanej odległości między punktami 1. a 2. oraz zmniejszenie preferowanej odległości między punktami 4. i 5. Oznacza to, że odwłok larwy przybliża się do reszty ciała.
3. Reset wszystkich preferowanych odległości. Oznacza to, że przez 1/3 czasu punkty larwy dążą do pozycji, jakby była w spoczynku.

Po wyliczeniu w danej fazie preferowanej odległości, w każdej klatce do czasu następnej fazy logika obsługująca fizykę zwiększa przyspieszenie punktów tak, aby znalazły się w preferowanych odległościach do sąsiadów. Na przykład w fazie pierwszej, dzięki wzajemnym oddziaływaniom punktów na siebie, punkt 1. jest bardziej popchany do przodu niż punkt 2. do tyłu, a to dlatego, że punkt 2. ma za sobą 3 inne punkty, które ciężej mu “popchnąć”.

Na ruch larw wpływają narkotyki - dla larwy wyliczany jest jej *CurrentMovementModifier*. Algorytm jego wyliczenia jest opisany w sekcji 8.5.2. *MovementModifier* posiada następujące parametry:

1. **SpeedMultiplier** - wpływa głównie na zwiększanie prędkości larwy
2. **SegmentCoordinationMultiplier** - wpływa głównie na wzmacnianie/osłabienie efektów wynikających z faz larwy
3. **RandomnessMultiplier** - wpływa głównie na ciągłe zmienianie kierunku głowy larwy
4. **DirectionStability** - wpływa na możliwość mocniejszej zmiany kierunku i zmienia losowo pozycję punktu, do którego larwa dąży na nieco inny
5. **SegmentSyncMultiplier** - może sprawić, że następna faza ruchu zostanie wybrana losowo

6. **HeadForceMultiplier** - wpływa na to, z jaką siłą larwa dąży do wybranego kierunku
7. **RestoreForceMultiplier** - wpływa na to, z jaką siłą wszystkie punkty kierują się w swoje preferowane kierunki

8.5.2 Wyliczenie wartości modyfikatorów narkotyków

Dla zwiększenia realistyczności, narkotyki nie działają od razu. Narkotyki aplikowane są z określonym *Intensity*, które rośnie z 0 do podanego celu. Dodatkowo narkotyki mają parametry:

1. **onsetTime**, który definiuje czas etapu zwiększania *intensity*.
2. **duration**, który definiuje czas etapu utrzymania *intensity*.
3. **comedownTime**, który definiuje czas etapu zmniejszania *intensity* do 0 (jeśli larwa dożyje ostatniej fazy).

Dzięki wykorzystaniu wzorca projektowego *NullObject* do opisu parametrów ruchu dla larwy w domyślnym stanie (*MovementModifier.Normal*), wyliczenie finalnego modyfikatora ruchu larwy polega na liniowej interpolacji między *MovementModifier.Normal* i wszystkimi *MovementModifier* wyliczonymi dla narkotyków na podstawie ich intensywności.

9 Demonstracja produktu programowego

9.1 Przykładowy workflow wykonywania analizy

W celu zaprezentowania kluczowych funkcjonalności systemu, poniżej przedstawiono kompletny scenariusz realizacji zadania diagnostycznego przez zalogowanego użytkownika. Proces ten obejmuje autoryzację, zlecenie analizy wideo, monitorowanie jej statusu oraz interpretację otrzymanych wyników.

Krok 1: Autoryzacja i dostęp do systemu

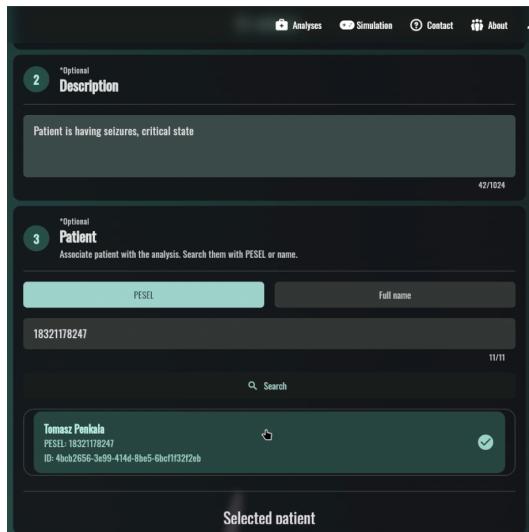
Proces rozpoczyna się na ekranie powitalnym (rys. 25), skąd użytkownik przechodzi do panelu logowania. Po wprowadzeniu poprawnych danych uwierzytelniających następuje przekierowanie do głównego widoku aplikacji. W przypadku nowych użytkowników dostępna jest ścieżka rejestracji.

Scenariusz alternatywny Jeżeli użytkownik korzystał już wcześniej z aplikacji, system automatycznie weryfikuje ważność przechowywanego lokalnie tokenu dostępu. W przypadku pozytywnej walidacji sesji, etap wprowadzania danych logowania jest pomijany, a użytkownik uzyskuje natychmiastowy dostęp do panelu analiz.

Krok 2: Zlecenie nowej analizy

Użytkownik inicjuje proces badawczy, wybierając opcję dodania nowej analizy. Uruchamia to wieloetapowy kreator (rys. 51), który prowadzi użytkownika przez proces konfiguracji badania:

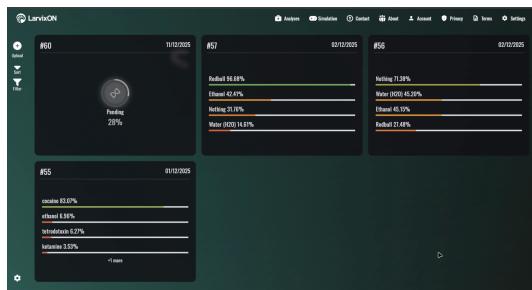
1. **Przesłanie materiału wideo:** Użytkownik wgrywa plik z nagraniem larw.
2. **(opcjonalne) Wprowadzenie opisu klinicznego:** Użytkownik wprowadza opis przypadku klinicznego.
3. **(opcjonalne) Powiązanie z pacjentem:** W kolejnym kroku następuje identyfikacja pacjenta. Dzięki zintegrowanej wyszukiwarce (rys. 34) użytkownik może odnaleźć kartotekę pacjenta po numerze PESEL, bez konieczności ręcznego przepisywania danych osobowych.



Rysunek 51: Widok ekranu kreatora nowej analizy

Krok 3: Monitorowanie postępu

Po zatwierdzeniu formularza, nowe zlecenie pojawia się na liście analiz (rys. 52). Użytkownik ma możliwość śledzenia statusu przetwarzania (np. *Oczekujące*, *Przetwarzanie*, *Zakończono*)



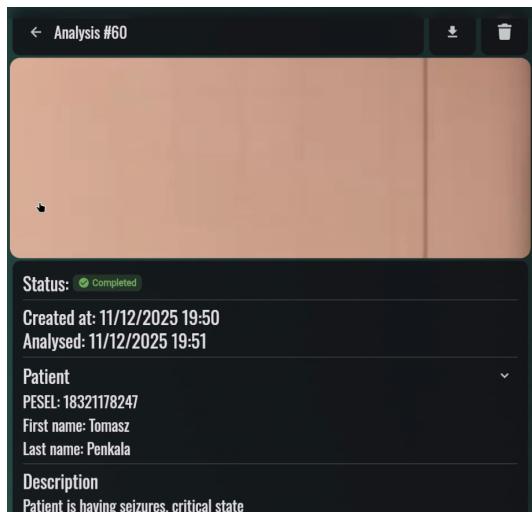
Rysunek 52: Widok gridu analiz, wraz z nową analizą oczekującą na przetworzenie

Krok 4: Odczyt wyników i raportowanie

Po zakończeniu przetwarzania, użytkownik uzyskuje dostęp do szczegółowych wyników. Widok ten prezentuje:

- Grafikę będącą miniaturką nagrania (rys. 53).
- Metadane dot. analizy, takie jak data zlecenia, status, informacje o pacjencie, opis kliniczny (rys. 53).
- Wykryte substancje toksyczne wraz ze wskaźnikami pewności predykcji (rys. 54)

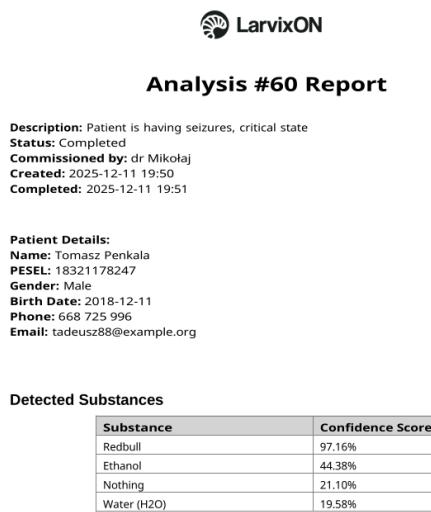
Umożliwia to podjęcie decyzji medycznej w oparciu o dostarczone dane. Opcjonalnie użytkownik może wygenerować sformalizowany raport (rys. 55), który może zostać dołączony do zewnętrznej dokumentacji medycznej pacjenta.



Rysunek 53: Widok tytułu analizy, miniaturki, danych pacjenta



Rysunek 54: Widok rezultatu obliczonego przez model maszynowy



Rysunek 55: Wyeksportowany raport z analizy

10 Dodatkowe materiały online

1. nasz board w GitHub Projects
2. larvixon-backend
3. larvixon-frontend
4. larvixon-model
5. larvixon-patients-service
6. larvixon-simulation
7. larvixon-documentation

Bibliografia

- [1] Felix Angelov. *Bloc Library Documentation - Architecture*. URL: <https://bloclibrary.dev/architecture/> (term. wiz. 11. 12. 2025).
- [2] Sebastián Ramírez Crespo. *FastAPI Documentation*. <https://fastapi.tiangolo.com/>. Dostęp: Grudzień 2025. 2024.
- [3] J. Donahue i et al. “Long-term Recurrent Convolutional Networks for Visual Recognition and Description”. W: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2015, s. 2625–2634.
- [4] K. He i in. “Deep Residual Learning for Image Recognition”. W: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), s. 770–778.
- [5] S. Hochreiter i J. Schmidhuber. “Long Short-Term Memory”. W: *Neural Computation* 9.8 (1997), s. 1735–1780.
- [6] K. Kavanagh i E.P. Reeves. “*Galleria mellonella* as a Model Host for the Study of Infectious Disease and Antimicrobial Development”. W: *Virulence* 6.6 (2015), s. 524–529.
- [7] A. Lange i et al. “High-throughput screening for xenobiotics using the behavioural response of *Galleria mellonella* larvae”. W: *Analytical and Bioanalytical Chemistry* 411.3 (2019), s. 729–737.
- [8] Robert C. Martin. *The Clean Architecture*. 2012. URL: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html> (term. wiz. 09. 12. 2025).
- [9] PyTorch Team. *PyTorch Documentation*. <https://pytorch.org/docs/stable/index.html>. Dostęp: Grudzień 2025. 2024.