**[Harmonogram](#harmon)**

**Temat pracy dyplomowej inżynierskiej: "****Tworzenie dokumentacji projektu z wykorzystaniem metod sztucznej inteligencji."**

Celem tego projektu jest stworzenie inteligentnego systemu, który automatycznie generuje dokumentację dla API na podstawie kodu źródłowego oraz komentarzy zawartych w kodzie. Projekt będzie koncentrował się na generowaniu czytelnej dokumentacji technicznej oraz na personalizacji poziomu szczegółowości w zależności od potrzeb i poziomu doświadczenia użytkownika. Użytkownicy będą mogli przeglądać dokumentację przez intuicyjny interfejs i dostosować jej szczegółowość zgodnie ze swoimi preferencjami.

**To właśnie etapy które chciałabym zrealizować w ramach tej pracy:**

**1. Analiza kodu i generowanie podstawowej dokumentacji**

• Stworzenie parsera kodu, który będzie analizować kod źródłowy i identyfikować funkcje, klasy, endpointy itp.

• Integracja z systemem Swagger/OpenAPI do wygenerowania szkieletu dokumentacji na podstawie struktury kodu.

Rezultat: Podstawowa wersja dokumentacji API bez zaawansowanych opisów.

Zbudowałam pełną infrastrukturę backendową:

* java-api (Spring Boot) – serwer główny,
* python-nlp (FastAPI) – mikroserwis do przetwarzania języka naturalnego,
* web (Nginx) – reverse proxy, łączący wszystko pod localhost:8080.

Stworzyłam system uploadu projektu (.zip):

* Endpoint /api/projects/upload rozpakowuje projekt i zapisuje go w /uploads/<ID>.
* Obsługuje walidację, błędy i tworzy unikalny identyfikator projektu.

Dodałam detekcję pliku openapi.yaml lub openapi.yml:

Klasa SpecDetector analizuje strukturę ZIP i odnajduje specyfikację.

Jeśli spec nie istnieje, system oferuje generację dokumentacji z kodu.

Zintegrowałam system z OpenAPI / Swaggerem:

EnrichmentService potrafi wczytać istniejący openapi.yaml i wzbogacić go o opisy wygenerowane przez NLP.

Działa endpoint /api/projects/{id}/spec/enriched.

Dodałam parser kodu źródłowego (JavaParser):

JavaSpringParser analizuje pliki .java, wykrywa klasy z @RestController, ich metody i adnotacje (@GetMapping, @PostMapping itd.).

Tworzy pośrednią strukturę EndpointIR, która opisuje endpointy, parametry i typy zwracane.

Zbudowałam moduł „Code → OpenAPI”:

Klasa CodeToDocsService generuje kompletny plik openapi.generated.yaml na podstawie kodu źródłowego.

Integracja z NLP dodaje opis do każdej metody i parametru.

Działa endpoint /api/projects/{id}/docs/from-code.

**2. Implementacja NLP do analizy komentarzy i generowania opisów**

• Zastosowanie NLP do analizy komentarzy, aby tworzyć jasne, zrozumiałe opisy funkcji i parametrów.

• Użycie modeli NLP do interpretacji kontekstu i generowania opisów na podstawie komentarzy.

Rezultat: Automatycznie generowane, czytelne opisy dla każdej funkcji, co znacznie zwiększa czytelność dokumentacji.

Stworzyłam osobny mikroserwis NLP (python-nlp), który:

przyjmuje strukturę endpointu (symbol, comment, params, returns),

analizuje komentarze i typy parametrów, generuje automatyczne opisy w trzech poziomach szczegółowości:

* shortDescription
* mediumDescription
* longDescription
* dodaje także paramDocs (opis każdego parametru) i returnDoc.

Zintegrowałam NLP z backendem (Spring Boot):

* EnrichmentService wysyła do /nlp/describe dane z kodu i odbiera opisy.
* Wyniki są automatycznie wstawiane do dokumentacji OpenAPI lub pliku YAML.

Zaimplementowałam personalizowany poziom szczegółowości (short/medium/long):

Użytkownik może wybrać poziom, a system automatycznie dopasowuje długość i szczegółowość opisów.

**3. Stworzenie systemu personalizacji dokumentacji**

Umożliwiłam ręczny wybór poziomu szczegółowości dokumentacji (short, medium, long) – użytkownik decyduje, jak rozbudowane mają być opisy.

• Implementacja mechanizmów śledzenia interakcji użytkownika, aby rozpoznać wzorce zachowań. (Śledzenie kliknięć i wyborów, czas spędzony na poszczególnych sekcjach, śledzenie wyszukiwań, interakcje z poziomem szczegółowości)

• Zastosowanie uczenia maszynowego do klasyfikacji użytkowników jako początkujących lub zaawansowanych.

• Tworzenie personalizowanych wersji dokumentacji, w zależności od poziomu doświadczenia użytkownika.

Rezultat: Dokumentacja dostosowana do poziomu wiedzy użytkownika, z możliwością wyboru poziomu szczegółowości.

4. Budowa interaktywnego interfejsu użytkownika

• Stworzenie dynamicznego interfejsu użytkownika, który umożliwia przeglądanie dokumentacji, filtrowanie i przeszukiwanie.

• Integracja interfejsu z backendem oraz systemem personalizacji.

Rezultat: Funkcjonalny interfejs użytkownika, który umożliwia wygodne przeglądanie dokumentacji i dostosowywanie poziomu szczegółowości.

5. Testowanie i optymalizacja

• Przeprowadzenie testów użyteczności i optymalizacji pod kątem wydajności.

• Testowanie algorytmów personalizacji i dopasowywanie ich do realnych potrzeb użytkowników.

Rezultat: Stabilna i zoptymalizowana wersja systemu gotowa do wdrożenia.

*07.10.2025:*

**Java:** Spring Boot, springdoc-openapi

**Python:** FastAPI, do NLP: spaCy / Hugging Face

**Frontend:** React + TypeScript

**Wspólne:** Docker

**Co działa teraz:**

• java-api - serwis backendowy (Spring Boot),

• python-nlp - mikroserwis AI (FastAPI),

• web - serwer Nginx (reverse proxy), który spina wszystko razem i wystawia publiczny adres <http://localhost:8080>.

1. **Środowisko uruchomieniowe (Docker + Nginx)**

• Trzy serwisy odpalane razem: java-api (Spring Boot), python-nlp (FastAPI), web (Nginx reverse proxy).

• Jeden punkt dostępu: http://localhost:8080 (Nginx przekazuje /api, /v3, /swagger-ui, /nlp do właściwych serwisów).

2. **Java API – szkielety i dokumentacja**

* springdoc-openapi podłączony: automatyczna specyfikacja OpenAPI:

/v3/api-docs (JSON), /v3/api-docs.yaml (YAML),

* Swagger UI: /swagger-ui/index.html.
* OpenApiConfig: ładny tytuł, opis, contact, license (MIT)

Endpointy demo (do dokumentowania i testów)

* GET /api/hello?name=: szybki test.
* GET /api/users/{id}: przykładowy odczyt (DTO w odpowiedzi).
* POST /api/users (JSON body + walidacja): pełny przepływ request body: response:
* 400 Bad Request z czytelnymi błędami walidacji, gdy dane są niepełne.

Java API będzie wysyłać surowe dane (nazwy funkcji, parametry, komentarze) do serwisu python-nlp, żeby otrzymać opisy w języku naturalnym.

3. **Python NLP – gotowy mikroserwis**

**•** GET /nlp/healthz (przez Nginx jako /nlp/healthz) — healthcheck.

• POST /nlp/describe — zwraca short/medium/long (szkielet pod późniejsze NLP).

• Nginx ma poprawne proxy dla /nlp/\*, więc UI/Java mogą go wołać bez CORS.

• Java API będzie wysyłać do niego „surowe dane z parsera” (nazwy metod, komentarze),

• on będzie zwracał czytelne opisy,

• dane te trafią z powrotem do dokumentacji OpenAPI.

**Pliki/elementy, które powstały:**

• java-api/pom.xml — zależności: springdoc-openapi-starter-webmvc-ui, walidacja.

• java-api/src/main/java/.../config/OpenApiConfig.java — tytuł/opis/contact/license.

• java-api/src/main/java/.../controller/HelloController.java — prosty endpoint.

• java-api/src/main/java/.../controller/UsersController.java — GET/POST z JSON body.

• java-api/src/main/java/.../dto/CreateUserRequest.java i UserResponse.java — DTO (walidacja + schematy w OpenAPI).

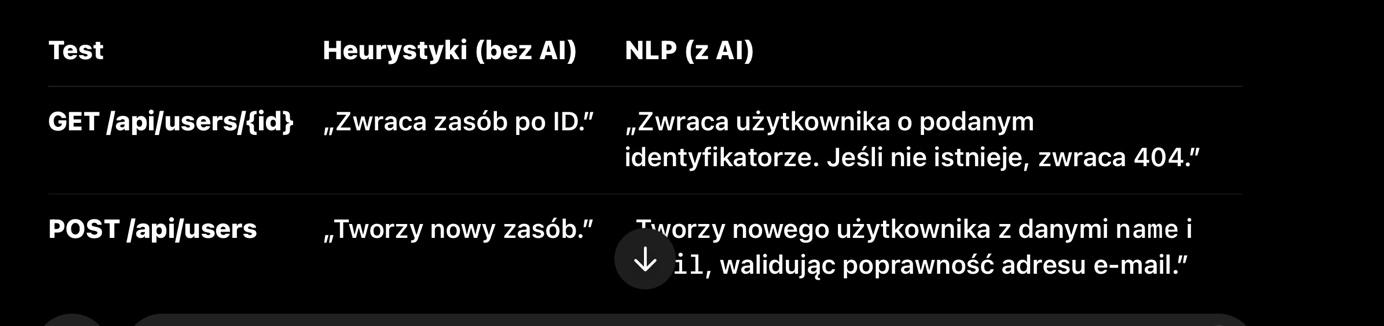
• web-ui/nginx.conf — proxy do /api, /v3, /swagger-ui, /nlp.

• docker-compose.yml — definicje trzech kontenerów i ich sieci.

Zastosowany mikroserwis python-nlp będzie wykorzystywać model językowy mT5 (Multilingual Text-to-Text Transfer Transformer), opracowany przez Google Research.

Model ten przetwarza dane wejściowe w postaci komentarzy i nazw metod, a następnie generuje opisy w języku naturalnym w kilku wariantach (krótki, średni, szczegółowy).

Dzięki temu możliwe jest tworzenie dokumentacji technicznej opartej na kodzie żródłowym w sposób zautomatyzowany i inteligentny, bez konieczności pisania tekstów przez człowieka.



|  |
| --- |
| google/mt5-small |

Co się dzieje pod spodem:

1. Plik trafia do backendu (java-api - / api/upload).
2. Mój system rozpakowuje ZIP-a, analizuje kod:

* ﻿﻿wykrywa klasy, kontrolery, funkcje, parametry, adnotacje, komentarze;
* ﻿﻿tworzy surowy opis kodu.

1. Dla kazdego endpointu (np. GET /api/ users/{id}) wysyła zapytanie do mikroserwisu python-nlp, który analizuje komentarze i generuje teksty opisowe (short, medium, long).

Swagger daje strukturę, a NLP daje semantykę i naturalny język

Przed: surowe dane

/api/hello:

get:

responses:

"200":

description: OK

Po:

/api/hello:

get:

summary: Zwraca powitanie użytkownika.

description: Endpoint zwraca powitanie z imieniem przekazanym w parametrze `name`.

responses:

"200":

description: Poprawna odpowiedź z wiadomością powitalną.

*14.10.2025:*

2. Implementacja NLP do analizy opisów w specyfikacji OpenAPI i generowania rozszerzonej dokumentacji

W ramach tego etapu wdrożono mikroserwis NLP, który analizuje istniejące opisy i komentarze w pliku OpenAPI (openapi.yaml) oraz automatycznie generuje bardziej rozbudowane, naturalne i zrozumiałe opisy funkcji, parametrów i odpowiedzi.

W odróżnieniu od klasycznego podejścia, gdzie analiza odbywa się bezpośrednio na kodzie źródłowym, system wykorzystuje strukturę OpenAPI jako pośrednią warstwę semantyczną. Dzięki temu możliwe jest automatyczne wzbogacanie dokumentacji wygenerowanej z dowolnego projektu zawierającego specyfikację API, niezależnie od języka programowania.

Mikroserwis NLP, oparty na frameworku FastAPI i modelach językowych, generuje opisy w trzech poziomach szczegółowości (short, medium, long). Wyniki są automatycznie wstawiane do sekcji description w obiektach paths, parameters i responses specyfikacji OpenAPI.

Rezultat: dokumentacja API staje się pełniejsza, spójna i bardziej zrozumiała dla użytkownika końcowego, bez konieczności ręcznego uzupełniania opisów w kodzie.

Do 21.10:

**Następnym krokiem w rozwoju systemu** będzie

1. **dodanie pełnej obsługi generowania dokumentacji na podstawie kodu źródłowego i komentarzy w kodzie – w sytuacji, gdy projekt nie zawiera pliku openapi.yaml.**

Jeśli użytkownik wgra projekt bez gotowej specyfikacji OpenAPI, system:

* automatycznie wykryje brak pliku openapi.yaml,
* przeanalizuje kod źródłowy (Java, a w przyszłości także Python),
* odczyta komentarze, typy danych i endpointy,
* wygeneruje kompletną dokumentację API przy użyciu NLP,
* zapisując ją jako openapi.generated.yaml.

Dzięki temu użytkownik nie musi samodzielnie pisać pliku OpenAPI, dokumentacja zostanie stworzona na podstawie kodu i komentarzy.

UPDATE *18.10.25*:

**Co zostało zrobione:**

Zaimplementowałam mechanizm automatycznego generowania dokumentacji API w formacie OpenAPI na podstawie kodu źródłowego projektu (Java) w sytuacji, gdy użytkownik nie dostarcza własnego pliku openapi.yaml.

System analizuje kod, odczytuje komentarze (Javadoc), typy danych oraz adnotacje kontrolerów Springa, a następnie generuje kompletny plik openapi.generated.yaml.

**Jak to działa:**

1. Użytkownik wysyła projekt jako archiwum ZIP.

2. System sprawdza, czy w projekcie znajduje się plik openapi.yaml. Jeśli go brak — uruchamiany jest moduł Code -> OpenAPI.

3. Klasa JavaSpringParser analizuje wszystkie pliki .java:

* wykrywa klasy oznaczone adnotacjami @RestController lub @Controller,
* rozpoznaje metody z adnotacjami @GetMapping, @PostMapping, @RequestMapping itd.,
* odczytuje ścieżki, typy metod HTTP, parametry oraz komentarze Javadoc (@param, @return).

Wynik zapisywany jest jako struktura pośrednia EndpointIR.

4. Klasa CodeToDocsService przetwarza te dane i generuje gotową specyfikację OpenAPI 3.0:

* dodaje sekcje paths, parameters, requestBody, responses,
* uzupełnia opisy metod i parametrów przy pomocy NLP,
* zapisuje wynik jako plik openapi.generated.yaml.

5. Użytkownik może pobrać wygenerowany plik.

**Efekt:** Dzięki temu system automatycznie tworzy pełną dokumentację API nawet wtedy, gdy projekt nie zawiera gotowego pliku openapi.yaml. Użytkownik nie musi jej pisać ręcznie — dokumentacja jest generowana dynamicznie na podstawie kodu i komentarzy.

*openapi*: 3.0.1

*info*:

*title*: Project bb587ae5001842b3aa59a8623c9ee7a8-API

*version*: 1.0.0

*paths*:

*/@GetMapping("/hello")*:

*get*:

*summary*: "Zwraca obiekt `Map<String,String>`. Typowe kody odpowiedzi: 200."

*description*: "Zwraca obiekt `Map<String,String>`. Typowe kody odpowiedzi: 200."

*operationId*: HelloController\_hello

*parameters*:

- *name*: name

*in*: query

*description*: Parametr name.

*required*: ***false***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: "Zwraca obiekt `Map<String,String>`."

*content*:

*application/json*:

*schema*:

*type*: object

*/@RequestMapping("/api/orders")/@GetMapping("/id")*:

*get*:

*summary*: "Pobiera zamówienie po ID. Typowe kody odpowiedzi: 200."

*description*: "Pobiera zamówienie po ID. Typowe kody odpowiedzi: 200."

*operationId*: OrderController\_getOrder

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*/@RequestMapping("/api/orders")/id*:

*delete*:

*summary*: "Usuwa zamówienie (przykład użycia RequestMapping z metodą). Typowe\

\ kody odpowiedzi: 200."

*description*: "Usuwa zamówienie (przykład użycia RequestMapping z metodą). Typowe\

\ kody odpowiedzi: 200."

*operationId*: OrderController\_delete

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*/@RequestMapping("/api/orders")/@PostMapping("/orderId/items")*:

*post*:

*summary*: "Dodaje pozycję do zamówienia. Typowe kody odpowiedzi: 200."

*description*: "Dodaje pozycję do zamówienia. Typowe kody odpowiedzi: 200."

*operationId*: OrderController\_addItem

*parameters*:

- *name*: orderId

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

- *name*: sku

*in*: query

*description*: Kod produktu.

*required*: ***false***

*schema*:

*type*: string

- *name*: qty

*in*: query

*description*: Ilość.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*/@RequestMapping("/api/users")/@GetMapping("/id")*:

*get*:

*summary*: "Zwraca użytkownika po ID. Typowe kody odpowiedzi: 200."

*description*: "Zwraca użytkownika po ID. Typowe kody odpowiedzi: 200."

*operationId*: UserController\_getById

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator użytkownika.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: Zwraca obiekt `UserResponse`.

*content*:

*application/json*:

*schema*:

*type*: object

*/@RequestMapping("/api/users")/*:

*get*:

*summary*: "Wyszukuje użytkowników. Typowe kody odpowiedzi: 200."

*description*: "Wyszukuje użytkowników. Typowe kody odpowiedzi: 200."

*operationId*: UserController\_search

*parameters*:

- *name*: q

*in*: query

*description*: Fraza wyszukiwania.

*required*: ***false***

*schema*:

*type*: string

- *name*: page

*in*: query

*description*: Numer strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

- *name*: size

*in*: query

*description*: Rozmiar strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*post*:

*summary*: "Tworzy nowego użytkownika. Typowe kody odpowiedzi: 200, 400, 409."

*description*: "Tworzy nowego użytkownika. Typowe kody odpowiedzi: 200, 400, 409."

*operationId*: UserController\_create

*requestBody*:

*description*: Dane użytkownika.

*content*:

*application/json*:

*schema*:

*type*: object

*required*: ***true***

*responses*:

"*200*":

*description*: Zwraca obiekt `UserResponse`.

*content*:

*application/json*:

*schema*:

*type*: object

Dodać:

Dodać prosty parser klas DTO.

Wydobyć z każdej klasy pola (String name, int age, itp.) i dodać je do components/schemas.

Zamiast schema: object używać $ref: '#/components/schemas/NazwaKlasy'.

UPDATE *20.10.25*:

*openapi*: 3.0.1

*info*:

*title*: Project f76baebc5cc443f9a84dc3713598fcc9-API

*version*: 1.0.0

*paths*:

*/hello*:

*get*:

*summary*: "Zwraca obiekt `Map<String,String>`. Typowe kody odpowiedzi: 200."

*description*: "Zwraca obiekt `Map<String,String>`. Typowe kody odpowiedzi: 200."

*operationId*: HelloController\_hello

*parameters*:

- *name*: name

*in*: query

*description*: Parametr name.

*required*: ***false***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: "Zwraca obiekt `Map<String,String>`."

*content*:

*application/json*:

*schema*:

*type*: object

*additionalProperties*:

*type*: string

*/api/orders/{id}*:

*get*:

*summary*: "Pobiera zamówienie po ID. Typowe kody odpowiedzi: 200."

*description*: "Pobiera zamówienie po ID. Typowe kody odpowiedzi: 200."

*operationId*: OrderController\_getOrder

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*delete*:

*summary*: "Usuwa zamówienie (przykład użycia RequestMapping z metodą). Typowe\

\ kody odpowiedzi: 200."

*description*: "Usuwa zamówienie (przykład użycia RequestMapping z metodą). Typowe\

\ kody odpowiedzi: 200."

*operationId*: OrderController\_delete

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*/api/orders/{orderId}/items*:

*post*:

*summary*: "Dodaje pozycję do zamówienia. Typowe kody odpowiedzi: 200."

*description*: "Dodaje pozycję do zamówienia. Typowe kody odpowiedzi: 200."

*operationId*: OrderController\_addItem

*parameters*:

- *name*: orderId

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

- *name*: sku

*in*: query

*description*: Kod produktu.

*required*: ***false***

*schema*:

*type*: string

- *name*: qty

*in*: query

*description*: Ilość.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*/api/users/{id}*:

*get*:

*summary*: "Zwraca użytkownika po ID. Typowe kody odpowiedzi: 200."

*description*: "Zwraca użytkownika po ID. Typowe kody odpowiedzi: 200."

*operationId*: UserController\_getById

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator użytkownika.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: Zwraca obiekt `UserResponse`.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/UserResponse"

*/api/users*:

*get*:

*summary*: "Wyszukuje użytkowników. Typowe kody odpowiedzi: 200."

*description*: "Wyszukuje użytkowników. Typowe kody odpowiedzi: 200."

*operationId*: UserController\_search

*parameters*:

- *name*: q

*in*: query

*description*: Fraza wyszukiwania.

*required*: ***false***

*schema*:

*type*: string

- *name*: page

*in*: query

*description*: Numer strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

- *name*: size

*in*: query

*description*: Rozmiar strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: Zwraca obiekt `Object`.

*content*:

*application/json*:

*schema*:

*type*: object

*post*:

*summary*: "Tworzy nowego użytkownika. Typowe kody odpowiedzi: 200, 400, 409."

*description*: "Tworzy nowego użytkownika. Typowe kody odpowiedzi: 200, 400, 409."

*operationId*: UserController\_create

*requestBody*:

*description*: Dane użytkownika.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/CreateUserRequest"

*required*: ***true***

*responses*:

"*200*":

*description*: Zwraca obiekt `UserResponse`.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/UserResponse"

*components*:

*schemas*:

*UserResponse*:

*type*: object

*properties*:

*id*:

*type*: string

*name*:

*type*: string

*email*:

*type*: string

*CreateUserRequest*:

*type*: object

*properties*:

*name*:

*type*: string

*email*:

*type*: string

1. **wdrożenie modelu mT5 (text-to-text) do inteligentnego generowania dokumentacji**

* Integracja modelu mT5 w mikroserwisie python-nlp przy użyciu biblioteki transformers (Hugging Face).
* Model mT5 będzie przetwarzać dane w formacie:

Wejście:

"Komentarz: Zwraca użytkownika po ID. Parametr: id - identyfikator użytkownika."

Wyjście:

"Endpoint służy do pobierania danych użytkownika na podstawie jego identyfikatora.

Jeśli użytkownik nie zostanie znaleziony, zwracany jest kod 404."

UPDATE *21.10.2025:*

Po objawach w YAML (podpisy typu “GET /…”, “Operacja …”, “Typowe kody odpowiedzi: .”) do finalnej specyfikacji trafia **fallback rule-based**, a nie teksty z mT5. Dzieją się dwa rzeczy naraz:

1. mT5 zwraca treści, ale „sanityzacja” przycina je zbyt agresywnie i robi z nich null, więc Java bierze fallback.

Funkcja „czyszcząca” (sanityzacja) po stronie Javy odrzucała całe zdania wygenerowane przez mT5, więc w kodzie lądowało null, a potem logika brała fallback rule-based.

2. W niektórych przebiegach mT5 potrafi dorzucić metakomentarz (np. „Instrukcja: …”), który wcześniej wycinam całkowicie, zamiast tylko posprzątać początek.

Poprawić żeby **wymusić użycie mT5** (gdy jest włączony) i nie „zjadać” jego wynik.

Błąd ładowania modelu w PyTorch/Transformers:

Cannot copy out of meta tensor; no data!

Please use torch.nn.Module.to\_empty() instead of torch.nn.Module.to()

when moving module from meta to a different device.

1. **Gotowy harmonogram Pracy Inżynierskiej**

**Harmonogram**

**Aplikacja:**

|  |  |  |
| --- | --- | --- |
| **Data** | **Zadanie** | **Wykonane** |
| **07.10.2025** | Analiza kodu i generowanie podstawowej dokumentacji | **Tak** |
| **14.10.2025** | Implementacja NLP do analizy opisów w specyfikacji OpenAPI i generowania rozszerzonej dokumentacji | **Tak** |
| **21.10.2025** | 1. Dodanie pełnej obsługi generowania dokumentacji na podstawie kodu źródłowego– w sytuacji, gdy projekt nie zawiera pliku openapi.yaml.  2. Wdrożenie modelu mT5 (text-to-text) do inteligentnego generowania dokumentacji  3. Gotowy harmonogram Pracy Inżynierskiej | **1. Tak**  **2. Nie działa**  **3. Tak** |
| **28.10.2025** | 1. Naprawienie i końcowa implementacja mT5  2. Zrobić 3 osobne pliki openapi.generated: 1. Bez opisów, 2. Z fallback base-rules, 3. Z użyciem modelu mT5  3. Dodać odczyt komentarzy (//, /\* \*/, /\*\* \*/) i zapisywać ich do EndpointIR  4. mT5+komentarzy  UPDATE:  5. Analiza i badania innych metod  6. implementacja modelu | **1. Nie**  **2. Tak**  **3. Tak**  **4. Nie**  **5.**  **6.** |
| **04.11.2025** | 1. Generowanie 3 poziomy opisu (short, medium, long  2. przegląd dokumentacji web + pdf dokumentacja |  |
| **12.11.2025** | 1. Stworzenie dynamicznego interfejsu  2. Generowanie opisu całego projektu do dokumentacji, inerface dokumentacji |  |
| **18.11.2025** | 1. Personalizację bez logowania i bazy – całkowicie „anonimowo”, per-przeglądarka, z wykorzystaniem localStorage + cookie/sessionId i płaskich logów NDJSON |  |
| **25.11.2025** |  |  |

**Praca:**

|  |  |  |
| --- | --- | --- |
| **Data** | **Zadanie** | **Wykonane** |
| **02.12.2025** |  |  |
| **09.12.2025** |  |  |
| **16.12.2025** |  |  |
| **08.01.2026** |  |  |
| **15.01.2026** |  |  |

Do 28.10:

1. Naprawienie i końcowa implementacja mT5

2. Zrobić 3 osobne pliki openapi.generated: 1. Bez opisów, 2. Z fallback base-rules, 3. Z użyciem modelu mT5

3. Dodać odczyt komentarzy (//, /\* \*/, /\*\* \*/) i zapisywać ich do EndpointIR

4. mT5+komentarzy, Generowanie opisu komentarze za pomocą mT5

UPDATE *23.10.2025*

1. Aktualna implementacja działa technicznie (pipeline uruchamia model i zapisuje wyniki), jednak wygenerowane opisy są w dużej mierze niespójne i niezrozumiałe. Konieczne jest dalsze strojenie / dobór promptów, filtrowanie wyjść i walidacja jakości przed uznaniem tego wariantu za produkcyjny.

2. Zaimplementowano generowanie archiwum .zip z trzema osobnymi plikami dokumentacji:

1. openapi.plain.yaml – dokumentacja bez opisów.

2. openapi.rules.yaml – dokumentacja z opisami tworzonymi przez rules-base.

3. openapi.mt5.yaml – dokumentacja z opisami generowanymi przez mT5.

*openapi*: 3.0.1

*info*:

*title*: Project 9ef7280395e44a82b506daac96baee82-API

*version*: 1.0.0

*paths*:

*/hello*:

*get*:

*summary*: "moji,sじみ: -zostałymi o /,zostałotymi i o. a..."

*description*: '....”Example.. undefined>.. »cychdytuj."/>. ....'

*operationId*: HelloController\_hello

*parameters*:

- *name*: name

*in*: query

*description*: Parametr name.

*required*: ***false***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: "....”: Parametr: name. Example."

*content*:

*application/json*:

*schema*:

*type*: object

*additionalProperties*:

*type*: string

*/api/orders/{id}*:

*get*:

*summary*: mojiärsk...akcident ...ł.-⸢kcident困りaćärsk...ärskkcident (ärsk...

*description*: იෝරlytteniu껍lytte –lytte껍ෝර껍lyttejaෝරlytte zda vulneru epłowniem

zda.

*operationId*: OrderController\_getOrder

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: waaaaульт껍⸢08(mærkkcident06(уланෝරуланෝර⸢lytteулан vulner vulner06(.

*content*:

*application/json*:

*schema*:

*type*: object

*delete*:

*summary*: "wa』(уланಿಪулан-улан (:gachοστάrzy. [...],..."

*description*: ი koske껍lytte –껍껍껍ෝරlytteулан껍껍lytte vulnerlytte - 06(껍ෝරෝර껍lytte

o ...

*operationId*: OrderController\_delete

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: walangsungaෝරуланулан pomar08(08( pomar pomarуланෝර⸢ vulner껍⸢ärsklytte:.

*content*:

*application/json*:

*schema*:

*type*: object

*/api/orders/{orderId}/items*:

*post*:

*summary*: moji -i - - z- opisz zwrot się inicznych. Zobaczy.

*description*: იуланlytte⸢уланෝරෝරෝර껍ෝරෝරக்கீෝරlytteෝරෝරlyttelyttelytte껍ෝරlytte껍lytteෝරlytte

koskelytte.

*operationId*: OrderController\_addItem

*parameters*:

- *name*: orderId

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

- *name*: sku

*in*: query

*description*: Kod produktu.

*required*: ***false***

*schema*:

*type*: string

- *name*: qty

*in*: query

*description*: Ilość.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: "waaaaулан껍⸢улан⸢улан vulner vulner08(08(уланlytte1)}08( 1)}к\

омпресс."

*content*:

*application/json*:

*schema*:

*type*: object

*/api/users/{id}*:

*get*:

*summary*: "moji -amiek - którego użytkownika po polsku...,.,. ,."

*description*: ი koskeෝරlyttenem困りlytte koskelytter idrулан id id...껍lytte.

*operationId*: UserController\_getById

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator użytkownika.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: waa–~$(уланулан08(-⸢улануланಿಪуланಿಪ08(-уланಿಪ⸢уланಿಪismissулан.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/UserResponse"

*/api/users*:

*get*:

*summary*: "moji;\"> . , ..cecece ..."

*description*: იулан vulnerуланෝර (ෝරෝරෝරlytte08(lyttelyttelytteෝරуланlyttelytte

koskelytter:lytteෝර08( vulnerlytte e困りlytte g.

*operationId*: UserController\_search

*parameters*:

- *name*: q

*in*: query

*description*: Fraza wyszukiwania.

*required*: ***false***

*schema*:

*type*: string

- *name*: page

*in*: query

*description*: Numer strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

- *name*: size

*in*: query

*description*: Rozmiar strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: waiулан껍lyttelytte껍уланlyttelyttelytteக்கீ껍lytteෝරlyttelytteෝර껍껍껍nie.

*content*:

*application/json*:

*schema*:

*type*: object

*post*:

*summary*: moji;">.. z.

*description*: ი koske껍껍껍lytteje껍lytte:껍lytte)rodo껍껍ෝර껍껍مشخصاتlytte껍lytteулан껍lytte

-.

*operationId*: UserController\_create

*requestBody*:

*description*: Dane użytkownika.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/CreateUserRequest"

*required*: ***true***

*responses*:

"*200*":

*description*: waвзෝරуланlyttelytteнукле lyttelyttelytteуланр ©lyttelytteෝරуланнуклеเขียนเมื่อ.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/UserResponse"

*components*:

*schemas*:

*UserResponse*:

*type*: object

*properties*:

*id*:

*type*: string

*name*:

*type*: string

*email*:

*type*: string

*CreateUserRequest*:

*type*: object

*properties*:

*name*:

*type*: string

*email*:

*type*: string

UPDATE *24.10.2025*

3. Dodać odczyt komentarzy (//, /\* \*/, /\*\* \*/) i zapisywać ich do EndpointIR

Obecnie brane pod uwagę:

**głównie Javadoc (/\*\* ... \*/) bezpośrednio nad klasą/metodą/polem:**

* + @param -> parameters[].description
  + opis metody -> operation.description (czasem skrót do summary)
  + Javadoc klasy/pól DTO -> components.schemas.\*.description / properties.\*.description

Ignorowane: zwykłe komentarze // i /\* ... \*/.

Efekt w plikach:

* 1. openapi.plain.yaml – bez opisów.
  2. openapi.rules.yaml – opisy z Javadoc + reguły uzupełniające.
  3. openapi.mt5.yaml – opisy z mT5 (na razie „bełkot”, ale używa treści z IR/Javadoc).

Co uwzględnia generowanie:

* Adnotacje Spring (@GetMapping, @RequestParam, itp.) -> ścieżki, metody, parametry, required.
* Javadoc przy deklaracjach -> opisy operacji/parametrów/DTO.
* Bez Javadoc -> opisy są generyczne („Parametr q.”), kody odpowiedzi nie są wylistowane.

**Po implementacji zapisywania innych komentarze // oraz /\* ... \*/:**

Jeśli zebrać komentarz bezpośrednio nad deklaracją (np. nad metodą) i zapisać do IR:

* można użyć go jako fallback opisu, gdy brak Javadoc
* trafi do operation.description / parameters[].description analogicznie jak Javadoc

Komentarze // wpływają łagodniej: trafiają do pól pomocniczych, mogą być streszczane (rules/mT5), żeby nie „zalać” dokumentacji.

**Przepływ:**

1. Parser (JavaParser/Spoon) zbiera:

javadoc nad klasami/metodami/polami,

* 1. leadingComments (komentarze tuż nad deklaracją),
  2. wszystkie inlineComments z ciał metod (z lineNo, type).
  3. 2. IR wypełnia pola jak wyżej.
  4. 3. RULES:
  5. główny opis = javadoc || leadingComments
  6. inlineComments → streszczenie do notes, surowe → x-impl-notes
  7. TODO/FIXME → x-todos

// walidacja ilości

/\* sprawdź dostępność SKU w katalogu \*/

// TODO: dodać audit log

paths***:***

/api/orders/{orderId}/items***:***

post***:***

description***:*** "Dodaje pozycję do zamówienia. Typowe kody: 200."

x-impl-notes***:***

- "walidacja ilości"

- "sprawdź dostępność SKU w katalogu"

x-todos***:***

- "dodać audit log"

1 && 4. Implementacja modelu mT5

Dlaczego surowy mT5 okazał się złym dopasowaniem do mojego programu oraz jakie podejścia AI dadzą stabilny, ludzki opis API (z przykładami) przy zachowaniu jakości i kontroli:

**1. Dlaczego mT5 „nie gra” w moim use-case:**

**1. Nie jest instruction-tuned**

mT5 (google/mt5-small/base) był uczony głównie na rekonstrukcji brakujących fragmentów (span-corruption). Nie był szkolony na pary polecenie-> odpowiedź.

Efekt: na prośbę „opisz endpoint po polsku, dodaj przykłady” model nie ma silnej „intuicji” formatu/tonu — w logach widzisz gibberish zamiast spójnego opisu.

**2. Wspólny, wielojęzyczny słownik != gwarancja jednego języka**

mT5 używa jednego SentencePiece dla wielu języków. Gdy sygnał „po polsku” jest za słaby, dekoder potrafi „zeskoczyć” w inne skrypty (cyrylica, znaki obce). Dlatego w logach widzimy mieszankę alfabetów. Instruction-tuned modele są zwykle bardziej „posłuszne” instrukcji dot. języka.

**3. Brak dopasowania domenowego (API/REST)**

Mój cel to: krótkie, zrozumiałe opisy operacji REST + przykłady request/response + „notes” z komentarzy. To wymaga **konsekwentnego formatu** i precyzji domenowej. Surowy mT5 nie zna mojej konwencji; bez fine-tune’u generuje losową narrację.

**4. Format wyjścia (JSON) i kontrola**

Chcę strukturalny output (opis, implNotes, examples). mT5, bez specjalnych ograniczeń, często łamie format. W moim serwisie włączyłam walidację - ta odcina śmieci -> zostaje pusty opis.

**Wniosek:** surowy mT5 nie daje gwarancji krótkiego, ludzkiego, polskiego opisu w stałym formacie. Dokładnie tego potrzebuję.

**Co się sprawdzi lepiej i dlaczego:**

**Model:** Mistral-7B-Instruct **albo** Llama-3.1-8B-Instruct

**Dlaczego:**

1. Wyższa jakość parafraz i przykładów

2. Bardzo dobre „instruction following”, sensowny polski

3. Dają krótkie, spójne „ludzkie” opisy i poprawne przykłady JSON.

**Rezygnacja z mT5**

* 1. Usuwam cały kod/konfigurację związaną z mT5 i Transformers w python-nlp.
  2. W CodeToDocsService enum: DescribeMode { PLAIN, RULES, AI } (zamiast MT5).
  3. Zmiana wywołania NLP: POST /describe?mode=ollama&strict=true.

**Nowy tryb „AI” (Ollama)**

* 1. python-nlp:/describe?mode=ollama woła **Ollamę** (/api/generate) i zwraca: mediumDescription, notes, examples (po walidacji JSON).
  2. **Brak fallbacku** do reguł w trybie AI (żeby widać było „czyste” AI).

openapi.ai.yaml – nowy plik wyjściowy

**Reguły bez zmian**

* 1. openapi.rules.yaml generowane jak dotąd
  2. openapi.plain.yaml bez opisów – bez zmian

**Docker**

Ollama uruchamiana **natywnie na macOS** (GPU/Metal), nie w Dockerze

* 1. python-nlp dostaje ENV: OLLAMA\_BASE\_URL=http://host.docker.internal:11434,
  2. OLLAMA\_MODEL=mistral:instruct (lub llama3.1:8b-instruct).
  3. Healthcheck python-nlp sprawdza /healthz.

**PROBLEM**

**1. Model wplata** <extra\_id\_0>

To <extra\_id\_0> to klasyczne „sentinel tokens” T5/mT5. Model czasem je wplata, bo był trenowany na **text infilling**.

**app.py**

***import*** os

***import*** re

***from*** typing ***import*** List, Optional, Dict, Any

***from*** fastapi ***import*** FastAPI

***from*** models ***import*** DescribeIn, DescribeOut, ParamDoc

# ====== Konfiguracja mT5 ======

ENABLE\_MT5: ***bool*** = os.***getenv***("***NLP\_ENABLE\_MT5***","***true***").***lower***() == "true"

MT5\_MODEL\_NAME: ***str*** = os.***getenv***("***MT5\_MODEL\_NAME***","***google/mt5-small***")

NLP\_WARMUP: ***bool*** = os.***getenv***("***NLP\_WARMUP***","***true***").***lower***() == "true"

# Regulacja jakości/szybkości (można nadpisać env)

MT5\_NUM\_BEAMS: ***int*** = ***int***(***os***.***getenv***("***MT5\_NUM\_BEAMS***","***4***"))

MT5\_MAX\_TOK\_SHORT: ***int*** = ***int***(***os***.***getenv***("***MT5\_MAX\_TOK\_SHORT***","***48***"))

MT5\_MAX\_TOK\_MED: ***int*** = ***int***(***os***.***getenv***("***MT5\_MAX\_TOK\_MED***","***96***"))

MT5\_MAX\_TOK\_LONG: ***int*** = ***int***(***os***.***getenv***("***MT5\_MAX\_TOK\_LONG***","***140***"))

MT5\_MAX\_TOK\_RET: ***int*** = ***int***(***os***.***getenv***("***MT5\_MAX\_TOK\_RET***","***64***"))

# Lazy-load zasobów HF

\_tokenizer: Any = ***None***

\_model: Any = ***None***

\_device: ***str*** = "cpu"

\_warmed\_up: ***bool*** = ***False***

\_mt5\_error: Optional[***str***] = ***None***

app = ***FastAPI***(title="***NLP Describe Service***",version="***0.4.0***")

# ---------- utilsy ----------

def ***\_sentences***(txt: ***str***) -> List[***str***]:

***if*** not txt:

***return*** []

parts = re.***split***(***r***'[.;]***\s***+|\n+', ***txt***.***strip***())

***return*** [p.***strip***().***rstrip***('***.***') ***for*** p ***in*** parts ***if*** p.***strip***()]

def ***\_detect\_statuses***(txt: ***str***) -> List[***str***]:

***if*** not txt:

***return*** []

found = ***set***()

***for*** code ***in*** ["400", "401", "403", "404", "409", "422", "500"]:

***if*** re.***search***(***rf***"***\b***{***code***}***\b***", ***txt***):

found.***add***(***code***)

***return*** ***sorted***(***found***)

def ***\_type\_to\_words***(t: Optional[***str***]) -> ***str***:

***if*** not t:

***return*** "odpowiedź"

t\_clean = t.***replace***("***java.lang.***","")

low = t\_clean.***lower***()

***if*** low in {"string"}: ***return*** "napis (string)"

***if*** ***any***(***x in low for x in*** ["***int***","***long***","***integer***"]): ***return*** "liczba całkowita"

***if*** ***any***(***x in low for x in*** ["***double***","***float***","***bigdec***"]): ***return*** "liczba"

***if*** "boolean" in low: ***return*** "wartość logiczna (true/false)"

***if*** low.***endswith***("***response***") or low.***endswith***("***dto***"):

***return*** f"obiekt `{t\_clean}`"

***return*** f"obiekt `{t\_clean}`"

def ***\_build\_param\_docs***(params) -> List[ParamDoc]:

out: List[ParamDoc] = []

***if*** not params:

***return*** out

***for*** p ***in*** params:

base = (p.description or "").***strip***()

***if*** not base:

n = (p.name or "").***lower***()

***if*** n in {"id", "userid", "user\_id"}:

base = "Identyfikator zasobu."

***elif*** n in {"page", "limit", "size"}:

base = "Parametr paginacji."

***elif*** n in {"q", "query", "search"}:

base = "Fraza wyszukiwania."

***else***:

base = f"Parametr `{p.name}`."

out.***append***(***ParamDoc***(name=***p***.***name***,doc=***base***))

***return*** out

def ***\_add\_dot***(s: ***str***) -> ***str***:

s = s.***strip***()

***return*** s ***if*** not s or s.***endswith***('***.***') ***else*** s + '.'

# ---------- Fallback (rule-based) ----------

def ***generate\_descriptions\_rule\_based***(payload: DescribeIn) -> DescribeOut:

sentences = ***\_sentences***(***payload***.***comment or*** "")

statuses = ***\_detect\_statuses***(***payload***.***comment or*** "")

***if*** "200" not in statuses:

statuses = ["200"] + statuses

***if*** payload.signature:

short = payload.signature

***elif*** payload.kind == "endpoint":

short = f"Operacja {payload.symbol}"

***else***:

short = f"Funkcja {payload.symbol}"

parts\_med = []

***if*** sentences:

parts\_med.***append***(***\_add\_dot***(***sentences***[0]))

***else***:

ret = ***\_type\_to\_words***(***payload***.***returns***.***type if payload***.***returns else None***)

parts\_med.***append***(***\_add\_dot***(***f***"***Zwraca*** {***ret***}"))

parts\_med.***append***(***f***"***Typowe kody odpowiedzi:*** {'***,*** '.***join***(***statuses***)}***.***")

medium = " ".***join***(***parts\_med***)

long\_parts: List[***str***] = []

***if*** sentences:

long\_parts.***append***(***\_add\_dot***(***sentences***[0]))

***if*** ***len***(***sentences***) > 1:

long\_parts.***append***(***\_add\_dot***("".***join***(***sentences***[1:2])))

***else***:

long\_parts.***append***(***parts\_med***[0])

pdocs = ***\_build\_param\_docs***(***payload***.***params***)

***if*** pdocs:

param\_lines = "; ".***join***([***f***"***`***{***p***.***name***}***` –*** {***p***.***doc***}" ***for p in pdocs***])

long\_parts.***append***(***\_add\_dot***(***f***"***Parametry:*** {***param\_lines***}"))

ret\_doc = ***None***

***if*** payload.returns:

ret\_phrase = payload.returns.description or f"Zwraca {***\_type\_to\_words***(***payload***.***returns***.***type***)}"

ret\_doc = ***\_add\_dot***(***ret\_phrase***)

long\_parts.***append***(***ret\_doc***)

long\_parts.***append***(***f***"***Typowe kody odpowiedzi:*** {'***,*** '.***join***(***dict***.***fromkeys***(***statuses***))}***.***")

long\_text = " ".***join***(***long\_parts***)

***return*** ***DescribeOut***(

shortDescription=***short***,

mediumDescription=***medium***,

longDescription=***long\_text***,

paramDocs=***pdocs***,

returnDoc=***ret\_doc***

)

# ---------- mT5 (transformers) ----------

def ***\_lazy\_load\_mt5***():

"""Ładuje tokenizer/model przy pierwszym użyciu; zapisuje ewentualny błąd do \_mt5\_error."""

***global*** \_tokenizer, \_model, \_device, \_mt5\_error

***if*** \_tokenizer is not ***None*** and \_model is not ***None***:

***return***

***if*** not ENABLE\_MT5:

***return***

***try***:

***import*** torch

***from*** packaging ***import*** version

# Twardy wymóg HF (CVE) – Torch >= 2.6

***if*** version.***parse***(***torch***.\_\_version\_\_) < version.***parse***("***2.6***"):

***raise*** ***RuntimeError***(***f***"***PyTorch*** {***torch***.\_\_version\_\_} ***< 2.6 (wymagane ≥ 2.6)***")

***from*** transformers ***import*** AutoTokenizer, AutoModelForSeq2SeqLM

\_device = "cuda" ***if*** torch.cuda.***is\_available***() ***else*** "cpu"

dtype = torch.float16 ***if*** \_device == "cuda" ***else*** torch.float32 # używamy 'dtype=', nie 'torch\_dtype'

\_tokenizer = AutoTokenizer.***from\_pretrained***(***MT5\_MODEL\_NAME***)

\_model = AutoModelForSeq2SeqLM.***from\_pretrained***(***MT5\_MODEL\_NAME***,dtype=***dtype***)

\_model.***to***(***\_device***)

\_model.***eval***()

***except*** ***Exception*** ***as*** e:

\_mt5\_error = f"{e}"

***print***(***f***"***[WARN] mT5 load error:*** {***e***}")

\_tokenizer = ***None***

\_model = ***None***

def ***\_prompt\_from\_payload***(payload: DescribeIn) -> ***str***:

lines = []

***if*** payload.signature:

lines.***append***(***f***"***Sygnatura:*** {***payload***.***signature***}")

***if*** payload.comment:

lines.***append***(***f***"***Komentarz:*** {***payload***.***comment***}")

***if*** payload.params:

param\_str = "; ".***join***([***f***"{***p***.***name***} ***(***{***p***.***type*** or'***unknown***'}***):*** {(***p***.***description*** or'').***strip***()or'***—***'}" ***for p in payload***.***params***])

lines.***append***(***f***"***Parametry:*** {***param\_str***}")

***if*** payload.returns and payload.returns.type:

lines.***append***(***f***"***Zwracany typ:*** {***payload***.***returns***.***type***}")

instr = (

"Zadanie: Na podstawie informacji o endpointzie wygeneruj zrozumiały opis działania. "

"Pisz po polsku, jasno i rzeczowo, bez marketingu."

)

***return*** f"{instr}\n" + " ".***join***(***lines***).***strip***()

def ***\_mt5\_generate***(text: ***str***, max\_new\_tokens=120, num\_beams=4, do\_sample=***False***, temperature=0.8) -> ***str***:

"""Jednorazowa generacja z mT5."""

***from*** transformers ***import*** GenerationConfig

***import*** torch

inputs = ***\_tokenizer***(***text***,return\_tensors="***pt***",truncation=***True***,max\_length=512)

inputs = {k: v.***to***(***\_device***) ***for*** k, v ***in*** inputs.***items***()}

gen\_cfg = ***dict***(max\_new\_tokens=***max\_new\_tokens***,num\_beams=***num\_beams***)

***if*** do\_sample:

gen\_cfg.***update***(***dict***(do\_sample=***True***,temperature=***temperature***,top\_p=0.95))

***with*** torch.***no\_grad***():

out\_ids = \_model.***generate***(\*\****inputs***,\*\****gen\_cfg***)

***return*** \_tokenizer.***decode***(***out\_ids***[0],skip\_special\_tokens=***True***).***strip***()

def ***generate\_descriptions\_mt5***(payload: DescribeIn) -> Dict[***str***, ***str***]:

"""Zwraca tylko pola tekstowe z mT5; brakujące pola pozostaw puste – zostaną scalone z fallbackiem."""

base = ***\_prompt\_from\_payload***(***payload***)

out: Dict[***str***, ***str***] = {}

***try***:

out["shortDescription"] = ***\_mt5\_generate***(

***base*** +"\n***Instrukcja długości: Napisz 1–2 krótkie zdania podsumowania.***",

max\_new\_tokens=***MT5\_MAX\_TOK\_SHORT***,num\_beams=***MT5\_NUM\_BEAMS***

)

***except*** ***Exception***: ***pass***

***try***:

out["mediumDescription"] = ***\_mt5\_generate***(

***base*** +"\n***Instrukcja długości: Napisz 2–4 zdania. Uwzględnij kontekst i typowe kody odpowiedzi.***",

max\_new\_tokens=***MT5\_MAX\_TOK\_MED***,num\_beams=***MT5\_NUM\_BEAMS***

)

***except*** ***Exception***: ***pass***

***try***:

out["longDescription"] = ***\_mt5\_generate***(

***base*** +"\n***Instrukcja długości: Napisz 4–6 zdań. Uwzględnij walidację parametrów i przypadek 404.***",

max\_new\_tokens=***MT5\_MAX\_TOK\_LONG***,num\_beams=***MT5\_NUM\_BEAMS***

)

***except*** ***Exception***: ***pass***

***try***:

ret\_type = payload.returns.type ***if*** (payload.returns and payload.returns.type) ***else*** "obiekt"

out["returnDoc"] = ***\_mt5\_generate***(

***base*** + ***f***"\n***Instrukcja: Jednym zdaniem opisz, co zwraca endpoint. Skup się na strukturze odpowiedzi. Zwracany typ:*** {***ret\_type***}***.***",

max\_new\_tokens=***MT5\_MAX\_TOK\_RET***,num\_beams=***MT5\_NUM\_BEAMS***

)

***except*** ***Exception***: ***pass***

# postprocess – przytnij spacje/kropki gdzie trzeba

***for*** k, v ***in*** ***list***(***out***.***items***()):

***if*** ***isinstance***(***v***, ***str***):

out[k] = v.***strip***()

***if*** "returnDoc" in out and out["returnDoc"]:

rd = out["returnDoc"].***strip***()

out["returnDoc"] = rd ***if*** rd.***endswith***("***.***") ***else*** rd + "."

***return*** out

# ---------- lifecycle ----------

@***app***.***on\_event***("***startup***")

def ***\_warmup***():

***global*** \_warmed\_up

***if*** not ENABLE\_MT5 or not NLP\_WARMUP:

***return***

***try***:

***\_lazy\_load\_mt5***()

***if*** \_tokenizer and \_model:

# mały dry-run żeby rozgrzać graf

\_ = ***\_mt5\_generate***("***Zadanie: Krótkie zdanie testowe o endpointach.***",max\_new\_tokens=16,num\_beams=2)

\_warmed\_up = ***True***

***print***("***[warmup] mT5 ready***")

***except*** ***Exception*** ***as*** e:

***print***(***f***"***[warmup] skipped:*** {***e***}")

# ---------- endpoints ----------

@***app***.***get***("***/healthz***")

def ***healthz***():

status = "ok"

m = "disabled" ***if*** not ENABLE\_MT5 ***else*** "unavailable"

***try***:

***if*** ENABLE\_MT5:

***\_lazy\_load\_mt5***()

***if*** \_tokenizer and \_model:

m = MT5\_MODEL\_NAME

***elif*** \_mt5\_error:

m = f"error: {\_mt5\_error}"

***except*** ***Exception*** ***as*** e:

m = f"error: {e.\_\_class\_\_.\_\_name\_\_}"

***return*** {"status": status, "mt5": m, "device": \_device, "warmed": \_warmed\_up}

@***app***.***post***("***/describe***",response\_model=***DescribeOut***)

def ***describe***(payload: DescribeIn):

# 1) zawsze zbuduj rule-based (gwarantowany komplet pól)

rb = ***generate\_descriptions\_rule\_based***(***payload***)

# 2) spróbuj mT5; scal wyniki (mT5 nadpisuje, fallback uzupełnia)

***if*** ENABLE\_MT5:

***try***:

***\_lazy\_load\_mt5***()

***if*** \_tokenizer and \_model:

mt5 = ***generate\_descriptions\_mt5***(***payload***)

***return*** ***DescribeOut***(

shortDescription= ***mt5***.***get***("***shortDescription***", ***rb***.***shortDescription***),

mediumDescription= ***mt5***.***get***("***mediumDescription***", ***rb***.***mediumDescription***),

longDescription= ***mt5***.***get***("***longDescription***", ***rb***.***longDescription***),

paramDocs= ***rb***.***paramDocs***,# stabilnie z reguł

returnDoc= ***mt5***.***get***("***returnDoc***", ***rb***.***returnDoc***),

)

***except*** ***Exception*** ***as*** e:

***print***(***f***"***[WARN] mT5 error:*** {***e***}")

# lecimy dalej do fallbacku

# 3) fallback

***return*** rb

**openapi.generate**

*openapi*: 3.0.1

*info*:

*title*: Project 29b05f2123ae4bc797d8f6da67833fe7-API

*version*: 1.0.0

*paths*:

*/hello*:

*get*:

*summary*: <extra\_id\_0>.

*description*: <extra\_id\_0>.

*operationId*: HelloController\_hello

*parameters*:

- *name*: name

*in*: query

*description*: Parametr name.

*required*: ***false***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: <extra\_id\_0>.

*content*:

*application/json*:

*schema*:

*type*: object

*additionalProperties*:

*type*: string

*/api/orders/{id}*:

*get*:

*summary*: <extra\_id\_0>.

*description*: <extra\_id\_0>.

*operationId*: OrderController\_getOrder

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: <extra\_id\_0>.

*content*:

*application/json*:

*schema*:

*type*: object

*delete*:

*summary*: <extra\_id\_0>.

*description*: <extra\_id\_0>.

*operationId*: OrderController\_delete

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: <extra\_id\_0>.

*content*:

*application/json*:

*schema*:

*type*: object

*/api/orders/{orderId}/items*:

*post*:

*summary*: <extra\_id\_0>.

*description*: <extra\_id\_0>.

*operationId*: OrderController\_addItem

*parameters*:

- *name*: orderId

*in*: path

*description*: Identyfikator zamówienia.

*required*: ***true***

*schema*:

*type*: string

- *name*: sku

*in*: query

*description*: Kod produktu.

*required*: ***false***

*schema*:

*type*: string

- *name*: qty

*in*: query

*description*: Ilość.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: <extra\_id\_0>.

*content*:

*application/json*:

*schema*:

*type*: object

*/api/users/{id}*:

*get*:

*summary*: <extra\_id\_0>.

*description*: <extra\_id\_0>.

*operationId*: UserController\_getById

*parameters*:

- *name*: id

*in*: path

*description*: Identyfikator użytkownika.

*required*: ***true***

*schema*:

*type*: string

*responses*:

"*200*":

*description*: <extra\_id\_0>.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/UserResponse"

*/api/users*:

*get*:

*summary*: <extra\_id\_0>.

*description*: <extra\_id\_0>.

*operationId*: UserController\_search

*parameters*:

- *name*: q

*in*: query

*description*: Fraza wyszukiwania.

*required*: ***false***

*schema*:

*type*: string

- *name*: page

*in*: query

*description*: Numer strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

- *name*: size

*in*: query

*description*: Rozmiar strony.

*required*: ***false***

*schema*:

*type*: integer

*format*: int32

*responses*:

"*200*":

*description*: <extra\_id\_0>.

*content*:

*application/json*:

*schema*:

*type*: object

*post*:

*summary*: <extra\_id\_0>.

*description*: <extra\_id\_0>.

*operationId*: UserController\_create

*requestBody*:

*description*: Dane użytkownika.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/CreateUserRequest"

*required*: ***true***

*responses*:

"*200*":

*description*: <extra\_id\_0> nowego użytkownika.

*content*:

*application/json*:

*schema*:

*$ref*: "#/components/schemas/UserResponse"

*components*:

*schemas*:

*UserResponse*:

*type*: object

*properties*:

*id*:

*type*: string

*name*:

*type*: string

*email*:

*type*: string

*CreateUserRequest*:

*type*: object

*properties*:

*name*:

*type*: string

*email*:

*type*: string