

Harmonogram

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) – mikroservis 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.

Dział 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 mikroservis 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)

Endpoints 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 trafiają 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 źródłowym w sposób zautomatyzowany i inteligentny, bez konieczności pisania tekstów przez człowieka.

Test	Heurystyki (bez AI)	NLP (z AI)
GET /api/users/{id}	„Zwraca zasób po ID.”	„Zwraca użytkownika o podanym identyfikatorze. Jeśli nie istnieje, zwraca 404.”
POST /api/users	„Tworzy nowy zasób.”	Tworzy nowego użytkownika z danymi name i ↓ i1, walidując poprawność adresu e-mail.”

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.

3. Dla każdego 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

```

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

- Integracja modelu mT5 w mikrosерwisie 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.

3. 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	1. 2. Tak 3. 4.
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
2. Dodać odczyt komentarzy (//, /* */ , /** */) i zapisywać ich do EndpointIR
3. 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 ...t.-'kcident困 り acärsk...ärskkcident (ärsk...
      description: ๐๐๐๐lytteniu꺈lytte –lytte꺈๐๐๐꺈lytteja๐๐๐๐lytte zda vulneru eplowniem
        zda.
      operationId: OrderController_getOrder
      parameters:
        - name: id
          in: path
```

description: Identyfikator zamówienia.

content:

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

```
/api/users:
```

get:

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

[illegible]

koskelytter:lytteㄹㄹㄹㄹ(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 strongy.

required: false

schema:

type: integer

format: int32

- *name*: size

in: query

description: Rozmiar strony.

required: false

schema:

type: integer

format: int32

responses:

"200":

description: waiуланྐlyttelytteྐуланlyttelytelytteཆྷྐlytteེུྐlyttelytteེུྐྐྐྐnie.

content:

application/json:

schema:

type: object

post:

summary: moji;">.. z.

description: ㅇ koske꺾꺾꺾lytteje꺾lytte:꺾lytte)rodo꺾꺾ᄇᆞᆫ꺾꺾مشخصاتlytte꺾lytteулан꺾lytte

operationId: UserController_create

requestBody:

description: Dane użytkownika.

content:

application/json:

schema:

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

required: **true**

responses:

"200":

description: wavз○○○ўўланlyttelytteнукле lyttelyttelytteуланр ©lyttelytte○○ўўланнуклеเขียนเมื่อ.

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

PROBLEM

1. Model wpłata <extra_id_0>

To <extra_id_0> to klasyczne „sentinel tokens” T5/mT5. Model czasem je wpłata, 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")
```

```

# ----- utils -----

def _sentences(txt: str) -> List[str]:
    if not txt:
        return []

    parts = re.split(r'[.!?]+|\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: DescriberIn) -> 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: DescriberIn) -> 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 + "\nInstrukcja 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 + "\nInstrukcja 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 + "\nInstrukcja 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"\nInstrukcja: 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 regut
                    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