LMS-API DOC

```
================================================================================
   _____                  __  __   ____                  _         __  __    _____
  / ____|     /\         |  \/  | |  _ \      /\        | |       |  \/  |  / ____|
 | (___      /  \        | \  / | | |_) |    /  \       | |       | \  / | | (___
  \___ \    / /\ \       | |\/| | |  _ <    / /\ \      | |       | |\/| |  \___ \
  ____) |  / ____ \      | |  | | | |_) |  / ____ \     | |____   | |  | |  ____) |
 |_____/  /_/    \_\     |_|  |_| |____/  /_/    \_\    |______|  |_|  |_| |_____/
 ================================================================================
 :: Spring Boot      v3.2.0 ::                                    :: 2024 ::
 :: api version      v1.0.0 ::
```

Dokumentacja do aplikacji rest api projektu LSM.

Spis treści

1. Baza danych
2. Instalacja i konfigurowanie systemu
    - Plik konfiguracyjny
3. Zadeklarowane stałe
    - Statusy
    - Role
    - Flagi
4. Autentykacja użytkownika
    - Rejestracja nowego użytkownika
    - Logowanie użytkownika
5. Użytkownicy
    - Pobieranie listy wszystkich użytkowników
    - Pobieranie pojedynczego użytkownika
    - Pobieranie pojedynczego użytkownika po loginie
    - Usuwanie użytkownika
    - Dodawanie nowego użytkownika
    - Aktualizacja danych użytkownika
6. Przedmioty
    - Pobieranie listy wszystkich przedmiotów
    - Pobieranie pojedynczego przedmiotu
    - Pobieranie pojedynczego przedmiotu po kodzie
    - Usuwanie przedmiotu
    - Dodawanie nowego przedmiotu
    - Aktualizacja danych przedmiotu
7. Okresy
    - Pobieranie listy wszystkich okresów
    - Pobieranie pojedynczego okresu
    - Usuwanie okresu

    - Dodawanie nowego okresu
    - Aktualizacja danych okresu
8. Rejestracja
    - Rejestracja ucznia na przedmiot
    - Pobieranie listy lub konkretnego powiązania
    - Pobieranie powiązania o konkretnym numerze ID
    - Wystawienie oceny uczniowi
    - Wyrejestrowanie ucznia z przedmiotu
9. Zadania
    - Rodzaje zadań
    - Pobieranie listy wszystkich zadań
    - Pobieranie pojedynczego zadania
    - Pobieranie aktywnych zadań
    - Usuwanie zadania
    - Dodawanie nowego zadania
    - Aktualizacja danych zadania
10. Odpowiedzi
    - Rodzaje odpowiedzi
    - Pobieranie listy wszystkich odpowiedzi
    - Pobieranie pojedynczej odpowiedzi
    - Dodawanie nowej odpowiedzi
    - Aktualizacja danych odpowiedzi
    - Usunięcie odpowiedzi
    - Wystawienie oceny dla zadnia
11. Materiały
    - Pobieranie listy wszystkich materiałów
    - Pobieranie pojednczego materiału
    - Usuwanie materiału
    - Dodawanie nowego materiału
    - Aktualizacja danych materiału
12. Powiadomienia
    - Pobieranie listy wszystkich powiadomień dla użytkownika
    - Pobieranie pojedynczego powiadomienia
    - Usuwanie powiadomienia
    - Aktualizacja flagi powiadomienia
13. Forum - wpisy
    - Pobieranie listy wszystkich wpisów na forum
    - Pobieranie pojedynczego wpisu na forum
    - Dodawanie wpisu na forum
    - Usuwanie wpisu z forum
    - Edycja wpisu na forum
14. Forum - odpowiedzi
    - Pobieranie listy wszystkich odpowiedzi na forum
    - Pobieranie pojedynczej odpowiedzi z forum
    - Dodawanie odpowiedzi do wpisu na forum
    - Usuwanie odpowiedzi z forum

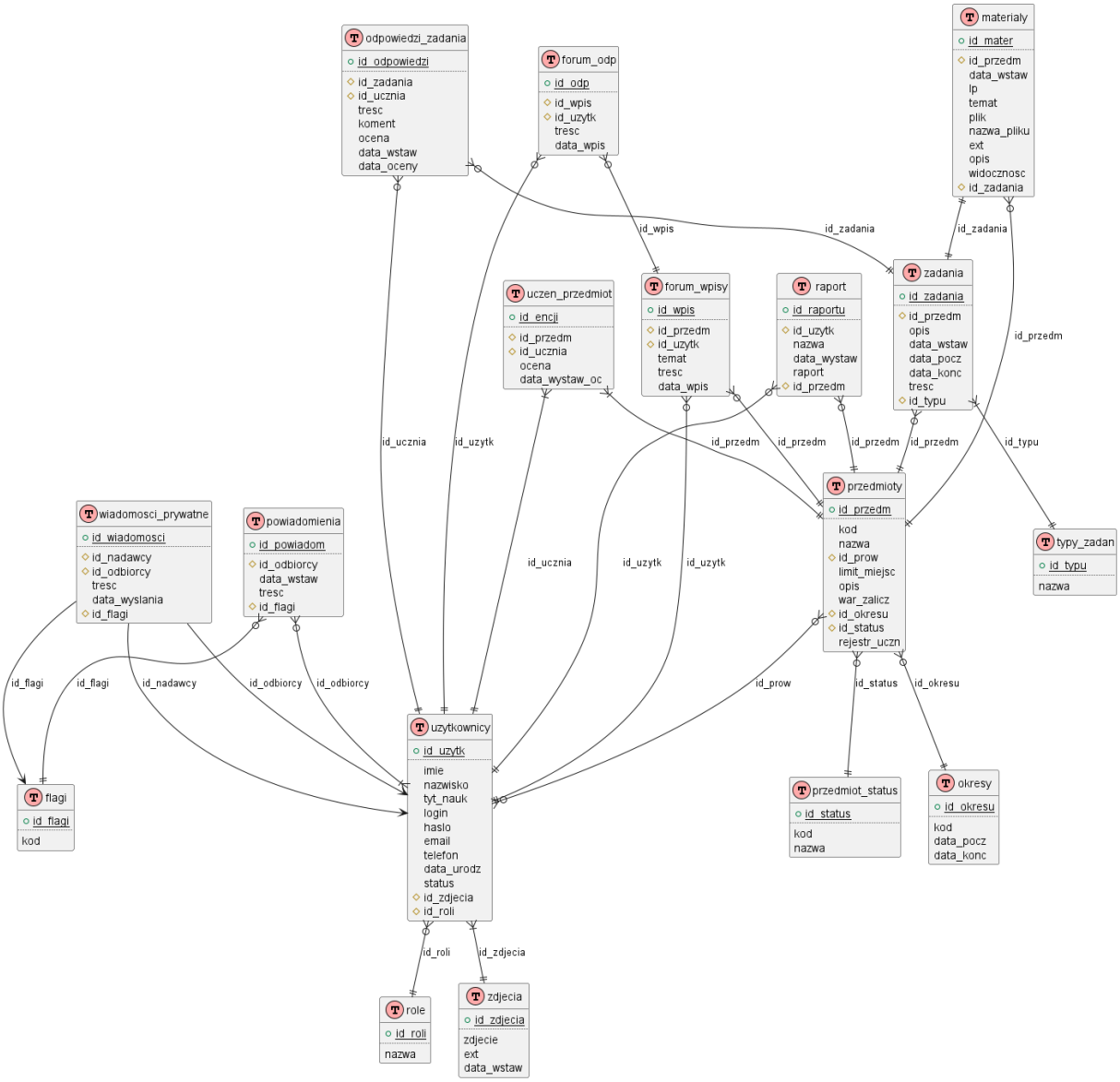
    - Edycja odpowiedzi na forum
15. Raporty
    - Generowanie raportu
16. Zaliczenie (wykaz ocen ucznia)
    - Pobranie wykazu ocen ucznia
17. Wiadomości Prywatne
    - Pobieranie listy wszystkich wiadomości
    - Pobieranie pojedynczej wiadomości
    - Pobieranie konwersacji
    - Dodawanie wiadomości
    - Edytowanie wiadomości
    - Usuwanie wiadomości

Baza danych

Opis bazy danych

Jest to relacyjna baza danych, zaimplementowana w języku MySQL. Została zaprojektowana do obsługi systemu zarządzania nauką online. Składa się z odpowiednich tabel, triggerów i procedur umożliwiających między innymi zarządzanie kursami, materiałami, użytkownikami oraz interakcjami między nimi.

Schemat bazy danych



Zarządzenie bazą danych

Dla każdej z tabel, z wyłączeniem tabel zawierających stałe (`flagi`, `role`, `przedmiot_status`, `typy_zadan`), utworzono procedury składowane, które ułatwiają wykonywanie operacji tworzenia, odczytu, aktualizacji oraz usuwania rekordów.

W celu walidacji danych wprowadzanych do bazy danych wykorzystano triggery, a także utworzono odpowiednie klucze unikalne dla kolumn oraz odpowiednie `check constraints`.

Triggery oprócz funkcji sprawdzania poprawności wprowadzanych danych, służą także do generowania powiadomień. Więcej o tym w podrozdziale *powiadomienia*.

Instalacja i konfigurowanie systemu

Aplikacja API dla projektu LMS (Learning Management System), zbudowana przy użyciu języka Java w wersji 17 i frameworku Spring Boot w wersji 3.2.0, reprezentuje nowoczesne podejście do tworzenia zaawansowanych aplikacji internetowych. Użycie Java 17 jako platformy programistycznej zapewnia dostęp do najnowszych ulepszeń języka, co przekłada się na lepszą wydajność, zwiększoną bezpieczeństwo oraz ułatwione zarządzanie zasobami. Spring Boot 3.2.0 z kolei oferuje znaczne udogodnienia w tworzeniu aplikacji

dzięki automatycznej konfiguracji, łatwej integracji z różnorodnymi bibliotekami oraz wsparciu dla mikroserwisów.

Dla efektywnego uruchomienia i działania aplikacji `lms-api`, konieczne jest zainstalowanie Java Runtime Environment (JRE) zgodnego z wersją 17. JRE jest niezbędne do uruchomienia skompilowanego kodu Java na różnych systemach operacyjnych, zapewniając jednolite środowisko uruchomieniowe.

Dodatkowo, kluczowym elementem konfiguracji aplikacji jest plik `application.properties`. Umożliwia on elastyczne dostosowanie aplikacji do specyficznych wymagań i warunków środowiska produkcyjnego, co gwarantuje odpowiednie dostosowanie aplikacji do potrzeb użytkownika końcowego.

Plik konfiguracyjny

Nazwa pliku: `application.properties`

```
### KONFIGURACJA SERWERA ###
# PORT SERWERA - Port, na którym aplikacja nasłuchuje
server.port=8080
# ADRES SERWERA - Adres IP, na którym działa serwer
server.address=127.0.0.1

### KONFIGURACJA BAZY DANYCH ###
# URL BAZY DANYCH
spring.datasource.url=jdbc:mysql://<data-base-url>
# NAZWA UŻYTKOWNIKA BAZODANOWEGO
spring.datasource.username=<user-name>
# HASŁO UŻYTKOWNIKA BAZODANOWEGO
spring.datasource.password=<password>

### KONFIGURACJA BEZPIECZEŃSTWA ###
spring.security.filter.order=10
# SEKRET BEZPIECZEŃSTWA - klucz, na podstawie którego generowane są tokeny JWT
security.auth.secret=~a?%B^"}i[xu}~IhA+BO'nGS8G(o5x
```

Zadeklarowane stałe

Statusy

Statusy przedmiotów:

1. `DO_ZATWIERDZENIA (2AP)` - Oczekuje na zatwierdzenie.
2. `ZATWIERDZONY (APR)` - Został zatwierdzony.
3. `ODRZUCONY (REJ)` - Został odrzucony.
4. `TRWAJACY (ACT)`- Jest aktualnie w trakcie.
5. `ZAKONCZONY (END)`- Zakończony.

Statusy użytkowników:

1. `AKTYWNY (1)`- Aktywny.

2. `NIEAKTYWNY (0)`- Nieaktywny.

Role

1. `ADMIN (1)`
2. `NAUCZYCIEL (3)`
3. `UCZEN (2)`

Flagi

1. `NIEPRZECZYTANA` - Oznacza, że obiekt nie został jeszcze przeczytany.
2. `PRZECZYTANA` - Oznacza, że obiekt został przeczytany.
3. `ZARCHIWIZOWANA`- Oznacza, że obiekt został zarchiwizowany.
4. `USUNIETA` - Oznacza, że obiekt został usunięty.
5. `NOWA` - Oznacza, że obiekt jest nowy.
6. `ROBOCZA` - Oznacza, że obiekt jest wersją roboczą lub tymczasową.

TypyZadan

1. `ZADANIE (1)`
2. `TEST (2)`
3. `EGZAMIN (3)`

Autentykacja użytkownika

Opis

Klasa `AuthController` odpowiada za obsługę end-pointów związanych z rejestracją i logowaniem użytkowników. W ramach dokumentacji przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Rejestracja nowego użytkownika

> **WAŻNE!**
>
> Login użytkownika nadawany jest automatycznie w konwencji <pierwsza_litera_imienia>.[?nr_porządkowy]
>
> W bazie danych przechowywane jest zaszyfrowane hasło przy użyciu algorytmu `BCrypt`
>
> Gdy nie zostanie przekazane zdjęcie, ustawione zostanie na domyślne (id 1).

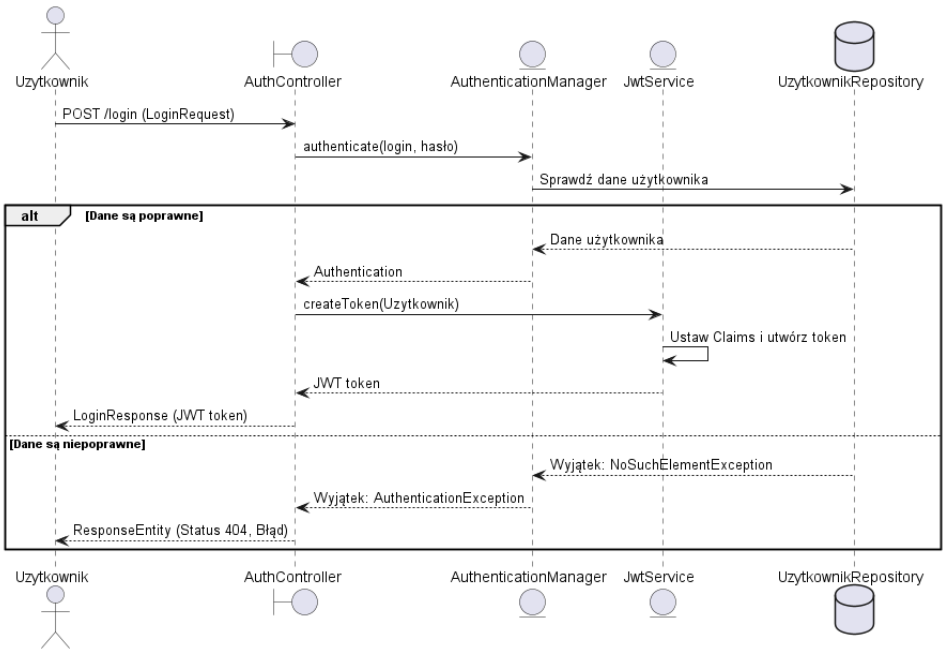
- **Ścieżka:** `/api/v1/auth/register`
- **Metoda:** `POST`
- **Parametry:**
  - `uzytkownik` (ciało zapytania, wymagane): Obiekt reprezentujący dane użytkownika.
- **Odpowiedź:**
  - `201 Created` - sukces, konto zostało utworzone.

- `500 Internal Server Error` - błąd utworzenia konta.

```
POST /api/v1/auth/register
Content-Type: application/json
```

```
{
    "imie": "Jan",
    "nazwisko": "Kowalski",
    "tytNauk": null,
    "email": "john.doe@example.com",
    "haslo": "haslo",
    "telefon": 123456789,
    "dataUrodz": "1990-01-01",
    "status": "AKTYWNY",
    "rola": "UCZEN",
    "zdjecie": {
        "plik": "<plik_binarny_base_64>"
    }
}
```

2. Logowanie użytkownika



- **Ścieżka:** `/api/v1/auth/login`
- **Metoda:** `POST`
- **Parametry:**
  - `request` (ciało zapytania, wymagane): Obiekt zawierający dane logowania (login, hasło).
- Odpowiedzi:
  - `200 OK` - sukces, logowanie udane, zwraca token JWT.
  - `400 Bad Request` - błędny login lub hasło.

- `400 Bad Request` - inny błąd podczas logowania.

```
POST /api/v1/auth/login
Content-Type: application/json
```

```
{
    "login":"j.doe",
    "haslo":"haslo"
}
```

Odpowiedź:

```
{
    "id": 1,
    "login": "j.doe",
    "token": "<jwt-token>"
}
```

Pozostałe zapytania wymagają uwierzytelnienia, dlatego należy dołączać do zapytania:

```
Authorization: Bearer <token>
```

Użytkownicy

Opis

Klasa `UzytkownicyController` odpowiada za obsługę end-pointów związanych z zarządzaniem użytkownikami w systemie. W ramach dokumentacji przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Pobieranie listy wszystkich użytkowników

- **Ścieżka:** `/api/uzytkownik/all`
- **Metoda:** `GET`
- **Parametry:**
  - `size` (opcjonalny): liczba elementów na stronie
  - `page` (opcjonalny): numer strony (liczony od 0)
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca listę użytkowników w formacie JSON
  - `404 Not Found` - brak użytkowników

```
GET /api/uzytkownik/all
Authorization: Bearer <token>
```

2. Pobieranie pojedynczego użytkownika

- **Ścieżka:** `/api/uzytkownik/{id}`
- **Metoda:** `GET`
- **Parametry:**
  - `id` (ścieżka): identyfikator użytkownika
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca dane użytkownika w formacie JSON
  - `404 Not Found` - użytkownik o podanym identyfikatorze nie istnieje

```
GET /api/uzytkownik/1
Authorization: Bearer <token>
```

3. Pobieranie pojedynczego użytkownika po loginie

- **Ścieżka:** `/api/uzytkownik`
- **Metoda:** `GET`
- **Parametry:**
  - `login` (parametr zapytania): login użytkownika (zakodowany w Base64)
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca dane użytkownika w formacie JSON
  - `404 Not Found` - użytkownik o podanym loginie nie istnieje

```
GET /api/uzytkownik?login=<login_base_64>
Authorization: Bearer <token>
```

4. Usuwanie użytkownika

- **Ścieżka:** `/api/uzytkownik/{id}`
- **Metoda:** `DELETE`
- **Parametry:**
  - `id` (ścieżka): identyfikator użytkownika
- **Odpowiedź:**
  - `204 No Content` - sukces, użytkownik został usunięty
  - `404 Not Found` - użytkownik o podanym identyfikatorze nie istnieje

```
DELETE /api/uzytkownik/<nr_id>
Authorization: Bearer <token>
```

5. Dodawanie nowego użytkownika

- **Ścieżka:** `/api/uzytkownik`
- **Metoda:** `POST`
- **Parametry:**
  - Ciało żądania zawiera dane nowego użytkownika w formacie JSON
- **Odpowiedź:**
  - `201 Created` - sukces, użytkownik został dodany, zwraca link do nowo utworzonego użytkownika
  - `400 Bad Request` - błąd w danych wejściowych

> **WAŻNE!**
>
> Login użytkownika nadawany jest automatycznie w konwencji <pierwsza_litera_imienia>.[?nr_porządkowy]
>
> W bazie danych przechowywane jest zaszyfrowane hasło przy użyciu algorytmu `BCrypt`
>
> Gdy nie zostanie przekazane zdjęcie, ustawione zostanie na domyślne (id 1).

```
POST /api/uzytkownik
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "imie": "Jan",
    "nazwisko": "Kowalski",
    "tytNauk": null,
    "email": "john.doe@example.com",
    "haslo": "haslo",
    "telefon": 123456789,
    "dataUrodz": "1990-01-01",
    "status": "AKTYWNY",
    "rola": "UCZEN",
    "zdjecie": {
        "plik": "<plik_binarny_base_64>"
    }
}
```

6. Aktualizacja danych użytkownika

- **Ścieżka:** `/api/uzytkownik/{id}`
- **Metoda:** `PATCH`
- **Parametry:**
  - `id` (ścieżka): identyfikator użytkownika
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON -**Odpowiedź:**

- `200 OK` - sukces, użytkownik został zaktualizowany, zwraca link do zaktualizowanego użytkownika
- `400 Bad Request` - błąd w danych wejściowych
- `404 Not Found` - użytkownik o podanym identyfikatorze nie istnieje

Przykład:

```
PATCH /api/uzytkownik/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
  "nazwisko": "Smith",
  "email": "john.smith@example.com"
}
```

Przedmioty

Opis

Klasa `PrzedmiotyController` odpowiada za obsługę end-pointów związanych z zarządzaniem przedmiotami w systemie. W ramach dokumentacji przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Pobieranie listy wszystkich przedmiotów

- **Ścieżka:** `/api/przedmiot/all`
- **Metoda:** `GET`
- **Parametry:**
  - `idUcznia` (opcjonalny): id ucznia, dla którego mają zostać pobrane przedmioty, na które jest zapisany
  - `size` (opcjonalny): liczba elementów na stronie
  - `page` (opcjonalny): numer strony (liczony od 0) (liczony od 0)
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca listę przedmiotów w formacie JSON
  - `404 Not Found` - brak przedmiotów

```
GET /api/przedmiot/all
Authorization: Bearer <token>
```

2. Pobieranie pojedynczego przedmiotu

- **Ścieżka:** `/api/przedmiot/{id}`
- **Metoda:** `GET`
- **Parametry:**
  - `id` (ścieżka): identyfikator przedmiotu
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca dane przedmiotu w formacie JSON
  - `404 Not Found` - przedmiot o podanym identyfikatorze nie istnieje

```
GET /api/przedmiot/1
Authorization: Bearer <token>
```

3. Pobieranie pojedynczego przedmiotu po kodzie

- **Ścieżka:** `/api/przedmiot`
- **Metoda:** `GET`
- **Parametry:**
  - `kod` (parametr zapytania): kod przedmiotu (zakodowany w Base64)
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca dane przedmiotu w formacie JSON
  - `404 Not Found` - przedmiot o podanym kodzie nie istnieje

```
GET /api/przedmiot/?kod=<kod_base_64>
Authorization: Bearer <token>
```

4. Usuwanie przedmiotu

- **Ścieżka:** `/api/przedmiot/{id}`
- **Metoda:** `DELETE`
- **Parametry:**
  - `id` (ścieżka): identyfikator przedmiotu
- **Odpowiedź:**
  - `204 No Content`- sukces, przedmiot został usunięty
  - `404 Not Found` - przedmiot o podanym identyfikatorze nie istnieje

```
DELETE /api/przedmiot/<nr_id>
Authorization: Bearer <token>
```

5. Dodawanie nowego przedmiotu

- **Ścieżka:** `/api/przedmiot`
- **Metoda:** `POST`
- **Parametry:**
  - Ciało żądania zawiera dane nowego przedmiotu w formacie JSON

- **Odpowiedź:**
  - `201 Created` - sukces, przedmiot został dodany, zwraca link do nowo utworzonego przedmiotu
  - `400 Bad Request` - błąd w danych wejściowych

> WAŻNE!

> Kod przedmiotu nadawany jest automatycznie w konwencji:
>
> < 3 znaki okresu>/< 4 znaki nazwy przedmiotu>/<numer porządkowy>

```
POST /api/przedmiot
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "nazwa": "Matematyka",
    "idProwadzacego": 123,
    "limit": 30,
    "opis": "Przedmiot z zakresu matematyki",
    "warunkiZaliczenia": "Egzamin końcowy",
    "idOkresu": 1,
    "status": "DO_ZATWIERDZENIA",
    "czyRejestrUczn": true
}
```

6. Aktualizacja danych przedmiotu

- **Ścieżka:** `/api/przedmiot/{id}`
- **Metoda:** `PATCH`
- **Parametry:**
  - `id` (ścieżka): identyfikator przedmiotu
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON
- **Odpowiedź:**
  - `200 OK` - sukces, przedmiot został zaktualizowany, zwraca link do zaktualizowanego przedmiotu
  - `400 Bad Request` - błąd w danych wejściowych
  - `404 Not Found` - przedmiot o podanym identyfikatorze nie istnieje

```
PATCH /api/przedmiot/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "nazwa": "Fizyka",
```

```
    "opis": "Przedmiot z zakresu fizyki"
}
```

Okresy

Opis

Klasa `OkresyController` odpowiada za obsługę end-pointów związanych z zarządzaniem okresami przedmiotów w systemie. W ramach dokumentacji przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Pobieranie listy wszystkich okresów

- **Ścieżka:** `/api/przedmiot/okres/all`
- **Metoda:** `GET`
- **Parametry:**
  - `size` (opcjonalny): liczba elementów na stronie
  - `page` (opcjonalny): numer strony (liczony od 0)
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca listę okresów w formacie JSON
  - `404 Not Found` - brak okresów

```
GET /api/przedmiot/okres/all
Authorization: Bearer <token>
```

2. Pobieranie pojedynczego okresu

- **Ścieżka:** `/api/przedmiot/okres/{id}`
- **Metoda:** `GET`
- **Parametry:** `id` (ścieżka): identyfikator okresu
- **Odpowiedź:** `200 OK` - sukces, zwraca dane okresu w formacie JSON `404 Not Found` - okres o podanym identyfikatorze nie istnieje

```
GET /api/przedmiot/okres/1
Authorization: Bearer <token>
```

3. Usuwanie okresu

- **Ścieżka:** `/api/przedmiot/okres/{id}`
- **Metoda:** `DELETE`
- **Parametry:**
  - `id` (ścieżka): identyfikator okresu

- **Odpowiedź:**
  - `204 No Content` - sukces, okres został usunięty
  - `404 Not Found` - okres o podanym identyfikatorze nie istnieje

```
DELETE /api/przedmiot/okres/<nr_id>
Authorization: Bearer <token>
```

4. Dodawanie nowego okresu

- **Ścieżka:** `/api/przedmiot/okres`
- **Metoda:** `POST`
- **Parametry:**
  - Ciało żądania zawiera dane nowego okresu w formacie JSON
- **Odpowiedź:**
  - `201 Created` - sukces, okres został dodany, zwraca link do nowo utworzonego okresu
  - `400 Bad Request` - błąd w danych wejściowych

```
POST /api/przedmiot/okres
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "kod": "SEM01",
    "dataPoczatku": "2023-01-01T00:00:00",
    "dataKonca": "2023-02-28T00:00:00"
}
```

5. Aktualizacja danych okresu

- **Ścieżka:** `/api/przedmiot/okres/{id}`
- **Metoda:** `PATCH`
- **Parametry:**
  - `id` (ścieżka): identyfikator okresu
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON
- **Odpowiedź:**
  - `200 OK` - sukces, okres został zaktualizowany, zwraca link do zaktualizowanego okresu
  - `400 Bad Request` - błąd w danych wejściowych
  - `404 Not Found` - okres o podanym identyfikatorze nie istnieje

```
PATCH /api/przedmiot/okres/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "dataPoczatku": "2023-02-01T00:00:00",
    "dataKonca": "2023-03-31T00:00:00"
}
```

Rejestracja

Opis

Klasa `UczenPrzedmiotController` odpowiada za obsługę end-pointów związanych z rejestracją ucznia na przedmiot w systemie i wystawianiem mu oceny. W ramach dokumentacji przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Rejestracja ucznia na przedmiot

- **Ścieżka:** `/api/przedmiot/uczen/rejestruj`
- **Metoda:** `POST`
- **Parametry:**
  - `nick` (parametr zapytania, wymagany): nick (login) ucznia w formacie Base64
  - `kod` (parametr zapytania, wymagany): kod przedmiotu w formacie Base64
- **Odpowiedź:**
  - `201 Created` - sukces, uczniowi został przypisany przedmiot
  - `400 Bad Request` - błąd w danych wejściowych

```
POST /api/przedmiot/uczen/rejestruj?nick=<nick_base64>&kod=<kod_base64>
Authorization: Bearer <token>
```

2. Pobieranie listy lub konkretnego powiązania:

Powiązanie można pobrać po:

- **nicku (loginie)** ucznia wtedy jest to lista wszystkich przedmiotow ucznia
- **kodzie przedmiotu**, wtedy jest to lista wszystkich uczniow w danym przedmiocie
- **nicku i kodzie**, wtedy pobiera konkretny poziazanie ucznia i przedmiotu
- **bez żadnych parametrów**, wtedy pobiera wszytko

- **Ścieżka:** `/api/przedmiot/uczen`
- **Metoda:** `GET`
- **Parametry:**
  - `nick` (parametr zapytania, opcjonalny): nick (login) ucznia w formacie Base64
  - `kod` (parametr zapytania, opcjonalny): kod przedmiotu w formacie Base64
  - `size` (parametr zapytania, opcjonalny): liczba wyników na stronie
  - `page` (parametr zapytania, opcjonalny): numer strony, liczony od 0

- **Odpowiedź:**
  - `200 OK` - sukces, zwraca kolekcję modeli powiązań w formacie JSON
  - `404 Not Found` - brak powiązań

```
GET /api/przedmiot/uczen?nick=<nick_base64>&kod=<kod_base64>&size=<size>&page=
<page>
Authorization: Bearer <token>
```

3. Pobieranie powiązania o konkretnym numerze ID

- **Ścieżka:** `/api/przedmiot/uczen/{id}`
- **Metoda:** `GET`
- **Parametry:**
  - `id` (ścieżka): numer ID powiązania
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca model powiązania w formacie JSON
  - `404 Not Found` - brak powiązania o podanym ID

```
GET /api/przedmiot/uczen/{id}
Authorization: Bearer <token>
```

4. Wystawienie oceny uczniowi

- **Ścieżka:** `/api/przedmiot/uczen/ocena`
- **Metoda:** `PATCH`
- **Parametry:**
  - `ocena` (parametr zapytania, wymagany): ocena do wystawienia
  - `nick` (parametr zapytania, wymagany): nick (login) ucznia w formacie Base64
  - `kod` (parametr zapytania, wymagany): kod przedmiotu w formacie Base64
- **Odpowiedź:**
  - `200 OK` - sukces, ocena została wystawiona
  - `404 Not Found` - brak powiązania o podanym nicku i kodzie

```
PATCH /api/przedmiot/uczen/ocena?ocena=<ocena>&nick=<nick_base64>&kod=<kod_base64>
Authorization: Bearer <token>
```

5. Wyrejestrowanie ucznia z przedmiotu

- **Ścieżka:** `/api/przedmiot/uczen/wyrejestruj`
- **Metoda:** `DELETE`
- **Parametry:**
  - `nick` (parametr zapytania, wymagany): nick (login) ucznia w formacie Base64
  - `kod` (parametr zapytania, wymagany): kod przedmiotu w formacie Base64

- **Odpowiedź:**
  - `204 No Content` - sukces, ucznia został wyrejestrowany z przedmiotu
  - `404 Not Found` - brak powiązania o podanym nicku i kodzie

```
DELETE /api/przedmiot/uczen/wyrejestruj?nick=<nick_base64>&kod=<kod_base64>
Authorization: Bearer <token>
```

Zadania

Rodzaje zadań

Zadania przekazywane w `tresc` mają format listy obiektów json. To pole jest typu `String`, więc wszystkie `"` muszą być poprzedzone znakiem *escape*: `\` . Poniżej typy zadań:

1. Zadania otwarte:

```
{
    "typ":"OTWARTE",
    "pytanie":"Bardzo trudne pytanie wymagające rozbudowanej odpowiedzi",
    "punkty": 10
}
```

2. Zadania zamknięte:

```
{
    "typ":"ZAMKNIETE",
    "pytanie":"Pytanie testowe z czterema odpowiedziami do wyboru, gdzie dwie są
prawdziwe",
    "odpowiedz":["odpowiedź1","odpowiedź2","odpowiedź3","odpowiedź3"],
    "poprawneOdp":[1,3],
    "punkty": 2
}
```

3. Zadania prawda-fałsz:

```
{
    "typ":"PRAWDA_FALSZ",
    "pytanie":"Bardzo proste pytanie wymagające odpowiedzi prawda albo fałsz",
    "odpowiedz":"true",
    "punkty": 1
}
```

4. Zadania z plikiem:

```
{
    "typ":"PLIK",
    "pytanie":"Pytanie wymagajace wgrania pliku lub jego edycję i ponowne wganie",
    "plik":"<base64_plik>",
    "punkty": 40
}
```

Przykład

```
[
  {\"typ\":\"OTWARTE\",\"pytanie\":\"Bardzo trudne pytanie wymagające rozbudowanej
odpowiedzi\",\"punkty\":2},
  {\"typ\":\"PRAWDA_FALSZ\",\"pytanie\":\"Bardzo proste pytanie wymagające
odpowiedzi prawda albo fałsz\",\"odpowiedz\":\"true\"\"punkty\":30},
  {\"typ\":\"PLIK\",\"pytanie\":\"Pytanie wymagajace wgrania pliku lub jego edycję
i ponowne wganie\",\"plik\":\"Ym5Wc2JBPT0=\"\"punkty\":20},
  {\"typ\":\"ZAMKNIETE\",\"pytanie\":\"Pytanie testowe z czterema odpowiedziami do
wyboru, gdzie dwie są prawdziwe\",\"odpowiedz\":
[\"odpowiedź1\",\"odpowiedź2\",\"odpowiedź3\",\"odpowiedź3\"],\"poprawneOdp\":
[1,3]\"punkty\":1}
]
```

Opis

Klasa `ZadaniaController` odpowiada za obsługę end-pointów związanych z zarządzaniem zadaniami przedmiotów w systemie. Poniżej przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Pobieranie listy wszystkich zadań

- Ścieżka: `/api/przedmiot/zadanie/all`
- Metoda: `GET`
- Parametry:
  - `kod` (wymagany): Kod przedmiotu zakodowany w base64.
  - `size` (opcjonalny): Liczba elementów na stronie.
  - `page` (opcjonalny): Numer strony (liczony od 0).
- Odpowiedź:
  - `200 OK` - sukces, zwraca listę zadań w formacie JSON.
  - `404 Not Found` - brak zadań.

```
GET /api/przedmiot/zadanie/all?kod=aGFwcHk=
Authorization: Bearer <token>
```

2. Pobieranie pojedynczego zadania

- **Ścieżka**: `/api/przedmiot/zadanie/{id}`
- **Metoda**: `GET`
- **Parametry:**
  - `id` (ścieżka): Identyfikator zadania.
- **Odpowiedź**:
  - `200 OK` - sukces, zwraca dane zadania w formacie JSON.
  - `404 Not Found` - zadanie o podanym identyfikatorze nie istnieje.

```
GET /api/przedmiot/zadanie/1
Authorization: Bearer <token>
```

3. Pobieranie aktywnych zadań

Poniższe zapytanie zwraca zdania o określonym typie, które spełniają warunek: `NOW() BETWEEN z.data_pocz AND z.data_konc`. Można pobrać zadania aktywne dla danego użytkownika, w danym przedmiocie lub wszystkie aktywne.

- Ścieżka: `/api/przedmiot/zadanie/aktywne`
- Metoda: `GET`
- Parametry:
  - `login` (opcjonalny): login użytkownika zakodowany w base64
  - `kod` (opcjonalny): Kod przedmiotu zakodowany w base64.
  - `typ` (wymagany): Typ zadania (zadanie, test, egzamin) zakodowany w base64.
- Odpowiedź:
  - `200 OK` - sukces, zwraca listę zadań w formacie JSON.
  - `404 Not Found` - brak zadań.

4. Usuwanie zadania

- **Ścieżka**: `/api/przedmiot/zadanie/{id}`
- **Metoda**: `DELETE`
- **Parametry**:
  - `id` (ścieżka): Identyfikator zadania.
- **Odpowiedź**:
  - `204 No Content` - sukces, zadanie zostało usunięte.
  - `404 Not Found` - zadanie o podanym identyfikatorze nie istnieje.

```
DELETE /api/przedmiot/zadanie/<nr_id>
Authorization: Bearer <token>
```

5. Dodawanie nowego zadania

- **Ścieżka:** `/api/przedmiot/zadanie`

- **Metoda:** `POST`
- **Parametry**:
  - Ciało żądania zawiera dane nowego zadania w formacie JSON.
- **Odpowiedź**:
  - `201 Created`- sukces, zadanie zostało dodane, zwraca link do nowo utworzonego zadania.
  - `400 Bad Request` - błąd w danych wejściowych.

```
POST /api/przedmiot/zadanie
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
  "idPrzedmiotu": 4,
  "typZadania":"EGZAMIN",
  "opis":"<opis_zadania>",
  "dataWstawienia": "2023-12-28T19:11:04",
  "dataPoczatku": "2024-02-28T00:00:00",
  "dataKonca": "2024-02-28T00:00:00",
  "tresc": "<tresc_json>"
}
```

6. Aktualizacja danych zadania

- **Ścieżka**: `/api/przedmiot/zadanie/{id}`
- **Metoda**: `PATCH`
- **Parametry**:
  - `id` (ścieżka): Identyfikator zadania.
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON.
- **Odpowiedź**:
  - `200 OK` - sukces, zadanie zostało zaktualizowane, zwraca link do zaktualizowanego zadania.
  - `400 Bad Request` - błąd w danych wejściowych.
  - `404 Not Found` - zadanie o podanym identyfikatorze nie istnieje.

```
PATCH /api/przedmiot/zadanie/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "dataPoczatku": "2023-02-01T00:00:00",
    "dataKonca": "2023-03-31T00:00:00",
    "tresc": "<tresc_json>"
}
```

Odpowiedzi

Rodzaje Odpowiedzi

Rodzaje odpowiedzi są analogiczne dla typów zadań. Pola zawierają jedynie odpowiedź oraz punktację. Odpowiedzią dla zadań zamkniętych jest lista numerów odpowiedzi.

Opis

Klasa `OdpowiedzRepository` odpowiada za interakcję z bazą danych w kontekście odpowiedzi na zadania przedmiotów w systemie. Poniżej przedstawione są wszystkie dostępne metody wraz z ich opisem, parametrami i możliwymi odpowiedziami.

Podczas dodawania nowej odpowiedzi można ustawiać wartości wszystkich pól (`komentarz`, `ocena`, `dataOcenienia`) jednak zakłada się, że wstawianie nowej odpowiedzi wykonywane jest przez ucznia, który nie ocenia.

Podczas wstawiania i uaktualniania odpowiedzi następuje jej sprawdzenie dla zadań typu zamkniętego oraz prawda-fałsz. Ręcznie muszą zostać ocenione zadania otwarte.

End-pointy

1. Pobieranie listy wszystkich odpowiedzi

- **Ścieżka**: `/api/przedmiot/zadanie/odpowiedz/all`
- **Metoda**: `GET`
- **Parametry**:
  - `login` (parametr zapytania, opcjonalny): login ucznia w formacie Base64
  - `kod` (parametr zapytania, opcjonalny): kod przedmiotu w formacie Base64
  - `size` (parametr zapytania, opcjonalny): liczba wyników na stronie
  - `page` (parametr zapytania, opcjonalny): numer strony, liczony od 0
- **Odpowiedź**:
  - `200 OK` - sukces, zwraca listę odpowiedzi w formacie JSON
  - `404 Not Found` - brak odpowiedzi

```
GET /api/przedmiot/zadanie/odpowiedz/all?requestParams=size;page;kod;login
Authorization: Bearer <token>
```

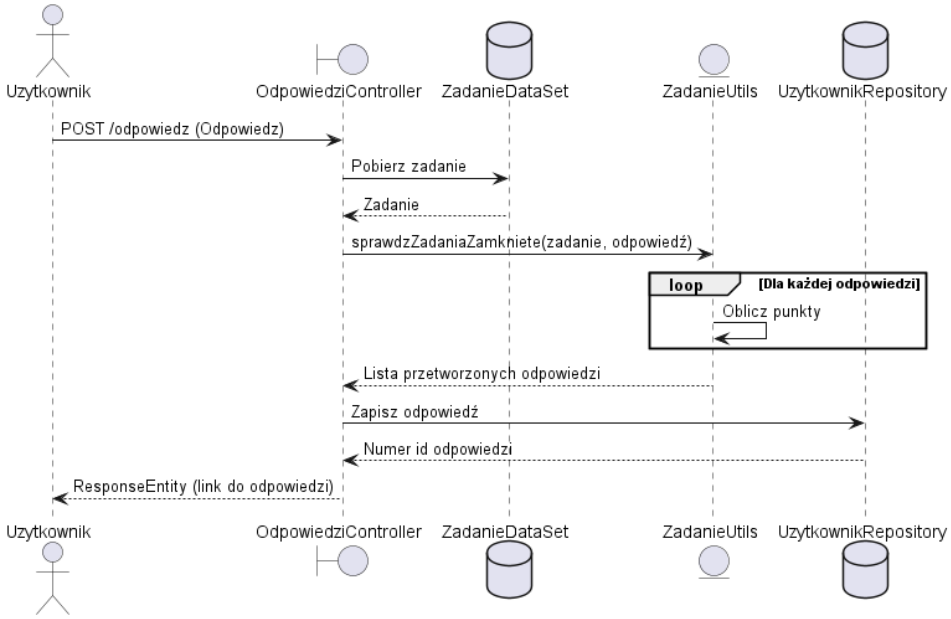
2. Pobieranie pojedynczej odpowiedzi

Ścieżka: `/api/przedmioty/zadania/odpowiedz/{id}` **Metoda**: `GET` **Parametry**: - `id` (ścieżka): Identyfikator odpowiedzi. **Odpowiedź**:

- `200 OK` - sukces, zwraca dane pojedynczej odpowiedzi w formacie JSON
- `404 Not Found` - odpowiedź o podanym identyfikatorze nie istnieje

```
GET /api/przedmiot/zadanie/odpowiedz/1
Authorization: Bearer <token>
```

3. Dodawanie nowej odpowiedzi



Ścieżka: `/api/przedmiot/zadanie/odpowiedz` **Metoda**: `POST` **Parametry**: Ciało żądania zawiera dane nowej odpowiedzi w formacie JSON. **Odpowiedź**: `201 Created` - sukces, odpowiedź została dodana, zwraca identyfikator nowo utworzonej odpowiedzi `400 Bad Request` - błąd w danych wejściowych

```
POST /api/przedmiot/zadanie/odpowiedz
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "idZadania": 1,
    "idUcznia": 2,
    "tresc": "<tresc_json>"
}
```

4. Aktualizacja danych odpowiedzi

- **Ścieżka**: `/api/przedmiot/zadanie/odpowiedz/{id}`
- **Metoda**: `PATCH`
- **Parametry**:
  - `id` (ścieżka): Identyfikator odpowiedzi.
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON.
- **Odpowiedź**:
  - `200 OK` - sukces, odpowiedź została zaktualizowana

    - `400 Bad Request` - błąd w danych wejściowych
    - `404 Not Found` - odpowiedź o podanym identyfikatorze nie istnieje

```
PATCH /api/przedmiot/zadanie/odpowiedz/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "tresc": "<tresc_json>",
    "komentarz": "Bardzo dobra praca!"
}
```

5. Usunięcie odpowiedzi

- **Ścieżka**: `/api/przedmiot/zadanie/odpowiedz/{id}`
- **Metoda**: `DELETE`
- **Parametry**:
    - `id` (ścieżka): Identyfikator zadania.
- **Odpowiedź**:
    - `204 No Content` - sukces, zadanie zostało usunięte.
    - `404 Not Found` - zadanie o podanym identyfikatorze nie istnieje.

```
DELETE /api/przedmiot/zadanie/odpowiedz/<nr_id>
Authorization: Bearer <token>
```

6. Wystawianie oceny

- **Ścieżka**: `/api/przedmiot/zadanie/odpowiedz/{id}/ocen`
- **Metoda**: `PATCH`
- **Parametry**:
    - `id` (ścieżka): Identyfikator odpowiedzi.
    - Ciało żądania zawiera dane do aktualizacji w formacie JSON.
- **Odpowiedź**:
    - `200 OK` - sukces, odpowiedź została zaktualizowana
    - `400 Bad Request` - błąd w danych wejściowych

```
PATCH /api/przedmiot/zadanie/odpowiedz/1/ocen
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "tresc": "<tresc_json>",
    "komentarz": "Bardzo dobra praca!",
    "ocena": 5
}
```

Materiały

Opis

Klasa `MaterialyController` obsługuje end-pointy związane z zarządzaniem materiałami przedmiotów w systemie. Poniżej przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Pobieranie listy wszystkich materiałów

- **Ścieżka**: `/api/przedmiot/material/all`
- **Metoda**: `GET`
- **Parametry**:
  - `kod` (parametr zapytania, wymagany): Kod przedmiotu w base64.
  - `lp` (parametr zapytania, opcjonalny): Numer porządkowy materiału.
  - `size` (parametr zapytania, opcjonalny): Liczba wyników na stronie.
  - `page` (parametr zapytania, opcjonalny): Numer strony, liczony od 0.
- **Odpowiedź**:
  - `200 OK` - sukces, zwraca listę materiałów w formacie JSON
  - `404 Not Found` - brak materiałów

```
GET /api/przedmiot/material/all?kod=<kod>&lp=<lp>&size=<size>&page=<page>
Authorization: Bearer <token>
```

2. Pobieranie pojedynczego materiału

- **Ścieżka**: `/api/przedmiot/materialy/{id}`
- **Metoda**: `GET`
- **Parametry**:
  - `id` (ścieżka): Identyfikator materiału.
- **Odpowiedź**:
  - `200 OK` - sukces, zwraca dane pojedynczego materiału w formacie JSON
  - `404 Not Found` - materiał o podanym identyfikatorze nie istnieje

```
GET /api/przedmiot/material/1
Authorization: Bearer <token>
```

3. Usuwanie materiału

Po usunięciu materiału automatycznie zostaje poprawiona liczba porządkowa pozostałych materiałów,

- **Ścieżka**: `/api/przedmiot/materialy/{id}`
- **Metoda**: `DELETE`
- **Parametry**:
  - `id` (ścieżka): Identyfikator materiału.
- **Odpowiedź**:
  - `204 No Content` - sukces, materiał został usunięty
  - `404 Not Found` - materiał o podanym identyfikatorze nie istnieje

```
DELETE /api/przedmiot/material/1
Authorization: Bearer <token>
```

4. Dodawanie nowego materiału

Automatycznie jest nadawany liczba porządkowa oraz ustawiana data wstawienia.

Można dodać zadanie do danego materiału przedmiotu. Wówczas należy najpierw utworzyć owe zadanie i przekazać jego `id` w tym zapytaniu w obiekcie `json`.

- **Ścieżka**: `/api/przedmiot/material`
- **Metoda**: `POST`
- **Parametry**:
  - Ciało żądania zawiera dane nowego materiału w formacie JSON
- **Odpowiedź**:
- `201 Created` - sukces, materiał został dodany, zwraca link do nowo utworzonego materiału
- `400 Bad Request` - błąd w danych wejściowych

```
POST /api/przedmiot/material
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "idPrzedmiotu":1,
    "temat": "Nowy temat",
    "plik": "<Zawartość pliku zakodowana w base64>",
    "nazwaPliku": "nazwa_pliku.txt",
    "ext": "txt",
    "opis": "Opis materiału",
    "idZadania": 2,
    "widocznosc": 1
}
```

5. Aktualizacja danych materiału

- **Ścieżka**: `/api/przedmiot/material/{id}`
- **Metoda**: `PATCH`
- **Parametry**:
  - `id` (ścieżka): Identyfikator materiału.
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON
- **Odpowiedź**:
  - `200 OK` - sukces, materiał został zaktualizowany, zwraca link do zaktualizowanego materiału
  - `400 Bad Request` - błąd w danych wejściowych
  - `404 Not Found` - materiał o podanym identyfikatorze nie istnieje

```
PATCH /api/przedmiot/material/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "temat": "Nowy temat"
}
```

Powiadomienia

Opis

Klasa `PowiadomieniaController` obsługuje end-pointy związane z zarządzaniem powiadomieniami użytkowników w systemie. Poniżej przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

Powiadomienia są dodawane automatycznie po stronie bazy danych za pomocą odpowiednich *triggerów*:

- `tai_materialy` - Trigger dodaje powiadomienie wszystkim uczestnikom przedmiotu, gdy zostanie dodany material do przedmiotu.
  - Treść: `'Dostępny nowy materiał z przedmiotu \'<nazwa_przedmiotu\'!'`
- `tbu_materialy` - Trigger dodaje powiadomienie wszystkim uczestnikom przedmiotu, gdy zostanie uwidoczniony materiał.
  - Treść: `'Dostępny nowy materiał z przedmiotu \'<nazwa_przedmiotu\'!'`
- `tai_odpowiedzi_zadania` - Trigger dodaje nowe powiadomienie nauczycielowi, gdy uczeń doda odpowiedz do zadania.
  - Treść: `'Użytkownik <imie i nazwisko> dodał nową odpowiedź do zadania z przedmiotu \'<nazwa_przedmiotu\'.'`
- `tbu_odpowiedzi_zadania` - Trigger dodaje nowe powiadomienie gdy wystawiono ocenę uczniowi za zadanie.
  - Treść: `'Dostałeś nową ocenę za zadanie z przedmiotu \'<nazwa_przedmiotu\'!'`
- `tai_zadania` - Trigger dodaje powiadomienie wszystkim uczestnikom przedmiotu, gdy zostanie dodane nowe zadanie, które jest aktywne danego dnia.

    - Treść: `'Dostępne nowe zadanie dla przedmiotu \'<nazwa_przedmiotu\'! Zadanie będzie dostępne do dnia: <data_konca>.'`
- `tbu_uczen_przedmiot` - Trigger dodaje nowe powiadomienie, gdy wystawiono ocenę uczniowi z przedmiotu.
    - Treść: `'Dostałeś nową ocenę z przedmiotu \'<nazwa_przedmiotu\'!'`
- `tai_forum_wpisy` - Trigger dodaje powiadomienie nauczycielowi o nowym wpisie na forum, o ile nie on jest jego autorem.
    - Treść: `Dostępny nowy wpis na forum z przedmiotu \'<nazwa_przedmiotu\'!'`
- `tai_forum_odp` - Trigger dodaje powiadomienie autorowi wpisu na forum, gdy pojawi się nowy komentarz, o ile nie dodał go autor tego wpisu.
    - Treść: `'Dostępny nowy komentarz do wpisu \'<temat_wpisu>\' na forum z przedmiotu \'<nazwa_przedmiotu\'!'`

Dla zadań utworzony jest także *event* (`e_zadania_aktywne`), który wysyła powiadomienia użytkownikom, zarejestrowanym do danego przedmiotu, jeżeli danego dnia zadanie się otwiera.

End-pointy

1. Pobieranie listy wszystkich powiadomień dla użytkownika

- **Ścieżka**: `/api/powiadomienie/all`
- **Metoda**: `GET`
- **Parametry**:
    - `login` (parametr zapytania, wymagany): Login użytkownika w base64.
    - `size` (parametr zapytania, opcjonalny): Liczba wyników na stronie.
    - `page` (parametr zapytania, opcjonalny): Numer strony, liczony od 0.
- **Odpowiedź**:
    - `200 OK` - sukces, zwraca listę powiadomień w formacie JSON
    - `404 Not Found` - brak powiadomień

```
GET /api/powiadomienie/all?login=<login>&size=<size>&page=<page>
Authorization: Bearer <token>
```

2. Pobieranie pojedynczego powiadomienia

- **Ścieżka**: `/api/powiadomienie/{id}`
- **Metoda**: `GET`
- **Parametry**:
    - `id` (ścieżka): Identyfikator powiadomienia.
- **Odpowiedź**:
    - `200 OK` - sukces, zwraca dane pojedynczego powiadomienia w formacie JSON
    - `404 Not Found` - powiadomienie o podanym identyfikatorze nie istnieje

```
GET /api/powiadomienie/1
Authorization: Bearer <token>
```

3. Usuwanie powiadomienia

- **Ścieżka**: `/api/powiadomienie/{id}`
- **Metoda**: `DELETE`
- **Parametry**:
  - `id` (ścieżka): Identyfikator powiadomienia.
- **Odpowiedź**:
  - `204 No Content` - sukces, powiadomienie zostało usunięte
  - `404 Not Found` - powiadomienie o podanym identyfikatorze nie istnieje

```
DELETE /api/powiadomienie/1
Authorization: Bearer <token>
```

4. Aktualizacja flagi powiadomienia

- **Ścieżka**: `/api/powiadomienie/{id}`
- **Metoda**: `PATCH`
- **Parametry**:
  - `id` (ścieżka): Identyfikator powiadomienia.
  - `flaga` (parametr zapytania, obowiązkowy): nazwa flagi według: Flagi (obsługiwane jest uppercase i lowercase),
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON
- **Odpowiedź**:
  - `200 OK` - sukces, flaga powiadomienia została zaktualizowana, zwraca link do zaktualizowanego powiadomienia
  - `404 Not Found` - powiadomienie o podanym identyfikatorze nie istnieje

```
PATCH /api/powiadomienie/1?flaga=nowa
Content-Type: application/json
Authorization: Bearer <token>
```

Forum-wpisy

Opis

Klasa `ForumWpisyController` odpowiada za obsługę end-pointów związanych z wpisami na forum. W ramach dokumentacji przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Pobieranie listy wszystkich wpisów na forum

- **Ścieżka:** `/api/forum/wpisy/all`

- **Metoda:** `GET`
- **Parametry:**
  - `size` (opcjonalny): liczba elementów na stronie
  - `page` (opcjonalny): numer strony (liczony od 0) (liczony od 0)
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca listę wpisów w formacie JSON
  - `404 Not Found` - brak wpisów

```
GET /api/forum/wpisy/all
Authorization: Bearer <token>
```

2. Pobieranie pojedynczego wpisu z forum

- **Ścieżka:** `/api/forum/wpisy{id}`
- **Metoda:** `GET`
- **Parametry:**
  - `id` (ścieżka): identyfikator wpisu
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca dane wpisu w formacie JSON
  - `404 Not Found` - wpis o podanym identyfikatorze nie istnieje

```
GET /api/forum/wpisy/1
Authorization: Bearer <token>
```

3. Dodawanie wpisu na forum

- **Ścieżka:** `/api/forum/wpisy`
- **Metoda:** `POST`
- **Parametry:**
  - Ciało żądania zawiera dane nowego wpisu w formacie JSON
- **Odpowiedź:**
  - `201 Created` - sukces, wpis został dodany, zwraca link do nowo utworzonego wpisu
  - `400 Bad Request` - błąd w danych wejściowych

```
POST /api/forum/wpisy
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "idWpisu": 1,
    "idPrzedmiotu": 1,
    "idUzytkownika": 1,
```

```
    "temat": "<temat wpisu>",
    "tresc": "<tresc wpisu>",
    "DataWpis": "2023-12-28T19:11:04"
}
```

4. Usuwanie wpisu z forum

- **Ścieżka:** `/api/forum/wpisy/{id}
- **Metoda:** `DELETE`
- **Parametry:**
  - `id` (ścieżka): identyfikator wpisu
- **Odpowiedź:**
  - `204 No Content`- sukces, wpis został usunięty
  - `404 Not Found` - wpis o podanym identyfikatorze nie istnieje

```
DELETE /api/forum/<nr_id>
Authorization: Bearer <token>
```

5. Edycja wpisu na forum

- **Ścieżka:** `/api/forum/wpisy/{id}
- **Metoda:** `PATCH`
- **Parametry:**
  - `id` (ścieżka): identyfikator wpisu
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON -**Odpowiedź:**
  - `200 OK` - sukces, wpis został zaktualizowany, zwraca link do zaktualizowanego wpisu
  - `400 Bad Request` - błąd w danych wejściowych
  - `404 Not Found` - wpis o podanym identyfikatorze nie istnieje

```
PATCH /api/forum/wpisy/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "temat": "<temat wpisu>",
    "tresc": "<tresc wpisu>"
}
```

Forum-odpowiedzi

Opis

Klasa `ForumOdpController` odpowiada za obsługę end-pointów związanych z odpowiedziami do wpisów na forum. W ramach dokumentacji przedstawione są wszystkie dostępne end-pointy wraz z ich opisem, parametrami i możliwymi odpowiedziami.

End-pointy

1. Pobieranie listy wszystkich odpowiedzi na forum

- **Ścieżka:** `/api/forum/odpowiedzi/all`
- **Metoda:** `GET`
- **Parametry:**
  - `size` (opcjonalny): liczba elementów na stronie
  - `page` (opcjonalny): numer strony (liczony od 0) (liczony od 0)
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca listę odpowiedzi w formacie JSON
  - `404 Not Found` - brak odpowiedzi

```
GET /api/forum/odpowiedzi/all
Authorization: Bearer <token>
```

2. Pobieranie pojedynczej odpowiedzi z forum

- **Ścieżka:** `/api/forum/odpowiedzi/{id}`
- **Metoda:** `GET`
- **Parametry:**
  - `id` (ścieżka): identyfikator odpowiedzi
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca dane odpowiedzi w formacie JSON
  - `404 Not Found` - odpowiedź o podanym identyfikatorze nie istnieje

```
GET /api/odpowiedzi/1
Authorization: Bearer <token>
```

3. Dodawanie odpowiedzi do wpisu na forum

- **Ścieżka:** `/api/forum/odpowiedzi`
- **Metoda:** `POST`
- **Parametry:**
  - Ciało żądania zawiera dane nowej odpowiedzi w formacie JSON
- **Odpowiedź:**
  - `201 Created` - sukces, odpowiedź została dodana, zwraca link do nowo utworzonej odpowiedzi
  - `400 Bad Request` - błąd w danych wejściowych

```
POST /api/forum/odpowiedzi
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "idOdpowiedzi": 1,
    "idWpisu": 1,
    "idUzytkownika": 1,
    "tresc": "<tresc odpowiedzi>",
    "DataWpis": "2023-12-28T19:11:04"
}
```

4. Usuwanie odpowiedzi z forum

- **Ścieżka:** `/api/forum/odpowiedzi/{id}`
- **Metoda:** `DELETE`
- **Parametry:**
  - `id` (ścieżka): identyfikator odpowiedzi
- **Odpowiedź:**
  - `204 No Content`- sukces, odpowiedz została usunięta
  - `404 Not Found` - odpowiedz o podanym identyfikatorze nie istnieje

```
DELETE /api/forum/odpowiedzi/<nr_id>
Authorization: Bearer <token>
```

5. Edycja odpowiedzi na forum

- **Ścieżka:** `/api/forum/odpowiedzi/{id}`
- **Metoda:** `PATCH`
- **Parametry:**
  - `id` (ścieżka): identyfikator odpowidzi
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON -**Odpowiedź:**
  - `200 OK` - sukces, odpowiedź została zaktualizowana, zwraca link do zaktualizowanej odpowiedzi
  - `400 Bad Request` - błąd w danych wejściowych
  - `404 Not Found` - odpowiedź o podanym identyfikatorze nie istnieje

```
PATCH /api/forum/odpowiedzi/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "tresc": "<tresc odpowiedzi>"
}
```

Raporty

Opis

`Raport` jest zasobem służącym tylko do odczytu dlatego posiada tylko jeden end-point: `GET`. Klasa `RaportController` odpowiada za obsługę tego end-pointu. Poniżej znajduje się jego opis.

1. Generowanie raportu

- **Ścieżka:** `/api/raport/generuj?idPrzedmiotu=<id_przedmiotu>`
- **Metoda:** `GET`
- **Parametry:**
  - `id_przedmiotu`: id przedmiotu dla którego chcemy utworzyć i otrzymać raport
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca raport z listą uczniów uczęszczających na dany przedmiot, ich oceny cząstkowe i oceny końcowe w postaci tabeli w formacie PDF
  - `400 Bad Request` - błędny format zapytania
  - `500 Internal Server Error` - inny błąd

```
GET /api/raport/generuj?idPrzedmiotu=1
Authorization: Bearer <token>
```

Zaliczenie (wykaz ocen ucznia)

Klasa `ZaliczenieCOntroller` służy do pobierania aktualnych ocen (cząstkowych i końcowych) z poszczególnych przedmiotów dla konkretnego ucznia.

1. Pobranie wykazu ocen ucznia

- **Ścieżka:** `/api/uczen/oceny/{id}`
- **Metoda:** `GET`
- **Parametry:**
  - `id` (ścieżka): identyfikator ucznia, dla którego chcemy pobrać wykaz ocen
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca listę ocen końcowych i cząstkowych
  - `400 Bad Request` - błędny format zapytania
  - `500 Internal Server Error` - inny błąd

```
GET /api/uczen/oceny/{idUcznia}
Authorization: Bearer <token>
```

Wiadomości Prywatne

Klasa `WiadomosciPrywatneController` odpowiada za obsługę endpointów związanych z wiadomościami prywatnymi w ramach dokumentacji przedstawiono wszystkie dostępne end-pointy.

1. Pobieranie listy wszystkich wiadomości

- **Ścieżka:** `/api/wiadomosci/prywatne/all`
- **Metoda:** `GET`
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca listę wszystkich wiadomości prywatnych
  - `400 Bad Request` - błędny format zapytania
  - `500 Internal Server Error` - inny błąd

```
GET /api/wiadomosci/prywatne/all
Authorization: Bearer <token>
```

2. Pobieranie pojedynczej wiadomości

- **Ścieżka:** `/api/wiadomosci/prywatne/{id}`
- **Metoda:** `GET`
- **Parametry:**
  - `id` (ścieżka): identyfikator wiadomości
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca dane wiadomości w formacie JSON
  - `404 Not Found` - wiadomość o podanym identyfikatorze nie istnieje
  - `500 Internal Server Error` - inny błąd

```
GET /api/wiadomosci/prywatne/1
Authorization: Bearer <token>
```

3. Pobieranie konwersacji

- **Ścieżka:** `/api/wiadomosci/prywatne/between-users?idUser1=<id_uzytkownik1>?idUser2=<id_uzytkownik2>'
- **Metoda:** `GET`
- **Parametry:**
  - `id_uzytkownik1`: id jednego z użytykowników
  - `id_uzytkownik2`: id drugiego z użytykowników
- **Odpowiedź:**
  - `200 OK` - sukces, zwraca liste wiadomości w formacie JSON pomiędzy dwoma użytkownikami posortowaną według daty wysłania

- `404 Not Found` - wiadomość o podanym identyfikatorze nie istnieje
- `500 Internal Server Error` - inny błąd
-

```
GET /api/wiadomosci/prywatne/between-users?idNadawcy=5&idOdbiorcy=4
Authorization: Bearer <token>
```

4. Dodawanie wiadomości

- **Ścieżka:** `/api/wiadomosci/prywatne`
- **Metoda:** `POST`
- **Parametry:**
  - Ciało żądania zawiera dane nowej wiadomości w formacie JSON
- **Odpowiedź:**
  - `201 Created` - sukces, wiadomość została dodana, zwraca link do nowo utworzonej wiadomości
  - `400 Bad Request` - błąd w danych wejściowych

```
POST /api/wiadomosci/prywatne
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "idNadawcy": 4,
    "idOdbiorcy": 5,
    "tresc": "<tresc wiadomości>"
}
```

5. Edytowanie wiadomości

- **Ścieżka:** `/api/wiadomosci/prywatne/{id}`
- **Metoda:** `PATCH`
- **Parametry:**
  - `id` (ścieżka): identyfikator wiadomosci
  - Ciało żądania zawiera dane do aktualizacji w formacie JSON -**Odpowiedź:**
  - `200 OK` - sukces, odpowiedź została zaktualizowana, zwraca link do zaktualizowanej odpowiedzi
  - `400 Bad Request` - błąd w danych wejściowych
  - `404 Not Found` - wiadomość o podanym identyfikatorze nie istnieje

```
PATCH /api/wiadomosci/prywatne/1
Content-Type: application/json
Authorization: Bearer <token>
```

```
{
    "tresc": "<treść_wiadomości>",
    "idFlagi": 2
}
```

6. Usuwanie wiadomości

- **Ścieżka:** `/api/wiadomosci/prywatne/{id}`
- **Metoda:** `DELETE`
- **Parametry:**
  - `id` (ścieżka): identyfikator wiadomości
- **Odpowiedź:**
  - `204 No Content`- sukces, wiadomość została usunięta
  - `404 Not Found` - wiadomość o podanym identyfikatorze nie istnieje

```
DELETE /api/wiadmomosci/prywatne/1
Authorization: Bearer <token>
```