

## Aplikacja back-endowa Web Api MusicStoreApi

Wykorzystano: C#, ASP.NET Core (.NET 6.0), REST, Entity Framework, MS\_SQL, dokumentacje Swagger(główna strona), Chmura Microsoft Azure, NLogger, Token JWT, Postman, polityka CORS, Middleware.

### POST -> api/account/register

Program przedstawiający portal sklepu muzycznego, gdzie można się zarejestrować jako nowy użytkownik z rolą:

- **USER** -> można wszystko przeglądać (GET), nie można tworzyć (POST), usuwać (DELETE), aktualizować (UPDATE)
- **PREMIUM\_USER** -> można wszystko przeglądać (GET), tworzyć (POST) tylko unikatowe nazwy artystów z albumami i piosenkami, a usuwać (DELETE) i aktualizować (UPDATE) tylko swoich stworzonych artystów, albumów, piosenek.
- **ADMIN** -> można wszystko,

Wysyłamy informacje za pomocą JSONa: **firstName, lastName, email, password, confirmPassword, nationality, dateOfBirth, roleId**. Po zmiennej **roleId** wiadomo jakie ma uprawnienia dany użytkownik: **1 -> USER, 2 -> PREMIUM\_USER, 3 -> ADMIN**. Zabezpieczenia:

- jeśli poda się błędną **date**, informacja o błędzie
- jeśli **email** jest pusty, informacja o błędzie -> **"email must not be empty"**, jeśli już jest w bazie danych, to błąd -> **"that email is taken"**, jeśli **email** jest nieprawidłowy, to błąd -> **"is not a valid email address"**
- jeśli poda się **lastName** i **firstName** puste, to informacja -> **"must not be empty"**
- jeśli poda się **password** mniej niż 6 znaków, to informacja -> **"The lenght of Password must be at least 6 characters. You entered 5 characters"**
- jeśli poda się **confirmPassword** nie równe **password**, to błąd -> **"must be equal to value of password"**

Podczas prawidłowej rejestracji, nowe hasło jest haszowane i zapisywane **PasswordHash** do bazy danych.

### POST -> api/account/login

Potem można się zalogować do portalu muzycznego, za pomocą wysłania informacji **email** jako login i **password**. Zabezpieczenia:

- jeśli poda się **email** pusty lub nieprawidłowy, to stosowna informacja błędu
- jeśli poda się **password** lub **email** nie prawidłowe, to błąd -> **"invalid username or password"**
- jeśli poda się **password** mniej niż 6 znaków, to dany błąd

Podczas zalogowania do sklepu muzycznego, sprawdzany jest login(**email**) czy jest taki sam w bazie, sprawdzany jest hash hasła usera(**PasswordHash**) z hashem hasła(**PasswordHash**), który jest w bazie. Jeśli wszystko ok to server generuje **token JWT** zawierający informacje o danym

użytkownika(**CLAIMY**): **UserId, FirstName, LastName, RoleName, DateOfBirth**. Aby móc korzystać ze wszystkich akcji w api jako zalogowany użytkownik, trzeba przesłać w nagłówku **Key** -> **Authorization**, a **Value** -> **Bearer** {wygenerowany **Token JWT**, który został wygenerowany przy logowaniu}.

## Artist

### GET -> api/artist

Można wyszukać wszystkich artystów z jego albumami i piosenkami, trzeba podać w parametrach dwie wartości: **PageSize** -> **5, 10, 15** i **PageNumber** -> **1, 2, 3** ... itp. Dodatkowo można podać jeszcze 3 wartości: **SearchWord** -> szuka danego słowa po danych rekordach **Name** lub **Description**, **SortDirection** -> **0(asc), 1(desc)**, sortuje rosnąco lub malejąco, **SortBy** -> sortuje po danych rekordach **Name, Description, KindOfMusic**. Pokazują się dodatkowo informacje o stronicowaniu wyników(paginacja) z wartościami: **TotalPages** -> wszystkie możliwe strony, **TotalItemsCount** -> wszystkie rekordy, **ItemFrom** -> pokazuje początek strony od której pozycji się zaczyna, **ItemTo** -> pokazuje koniec strony na której pozycji się kończy.

Zabezpieczenia:

-jeśli liczba wszystkich rekordów do wyświetlenia jest mniejsza lub równa od **PageSize** \* (**PageNumber-1**), to błąd -> 400 **Bad Request** o informacji **"search result items of Artists: 15 is too small or equal, because the number of skip 15, change the values in PageSize = 5, PageNumber = 1, to see the result "**

-jeśli poda się **PageSize** nieprawidłowe czyli inną niż 5, 10, 15 , to błąd 400 **Bad Request** z informacją -> **"must in [5, 10, 15]"**

Jeśli poda się **PageNumber** nieprawidłowe czyli od 1, 2, 3...., to błąd 400 z informacją -> **"must be greater than or equal to 1"**

### GET -> api/artist/{id}

Można wyszukać po numerze id danego artystę.

Zabezpieczenia:

-jeśli poda się **artistId** nieistniejące w bazie danych, to błąd 404 **Not Found** z informacją -> **"Artist {artistId} is not found"**

### POST -> api/artist

Można stworzyć nowego artystę podając wartości **Name, Description, KindOfMusic, ContactEmail, ContactNumber, Country, City**. Może tylko korzystać zalogowany użytkownik z rolą: **PREMIUM\_USER** i **ADMIN**.

Zabezpieczenia:

-jeśli poda się **City, Name, Country, Description** jako pustą wartość, to błąd z informacją -> **"field is required"**

-jeśli poda się ContactEmail i ContactNumber błędny, to error z informacją -> "is not a valid {zmienna}"

-jeśli poda się nazwę artysty, którą już stworzył dany użytkownik, to błąd z informacją => "Name: invalid value because there is already an artist created by this user(duplicate)"

### **DELETE -> api/artist/{id}**

Można usunąć dany album podając nr id albumu. Usunąć dany album może jedynie użytkownik z rolą: **USER** , który stworzył danego artystę lub użytkownik z rolą: **ADMIN**.

Zabezpieczenia:

-jeśli poda się **artistId** nieistniejące w bazie danych, to error z informacją -> "Artist {artistId} is not found"

### **PUT -> api/artist/{id}**

Można aktualizować artystę, o wartościach **name, description, kindOfMusic, contactEmail, contactNumber, Country, City**. Zmiany może zrobić jedynie użytkownik z rolą: **PREMIUM\_USER**, który stworzył danego artystę lub użytkownik z rolą: **ADMIN**.

Zabezpieczenia:

-jeśli poda się już nazwę artysty, która już w bazie danych, stworzona przez danego użytkownika, błąd **409 Conflict** z informacją -> "**Name : invalid value because there is already an artist created by this user (duplicate)**"

W albumach i piosenkach dodatkowa możliwość jeszcze usuwania wszystkich albumów lub piosenek. Zabezpieczenia:

-jeśli brak albumów do usunięcia, lub brak albumów do wyświetlenia, to informacja -> "**list of albums is empty**"

-jeśli brak piosenek do usunięcia, lub brak piosenek do wyświetlenia, to informacja -> "**list of songs is empty**"

Albumy i piosenki tak samo jak artystów, można tworzyć nowe, usuwać, aktualizować, wyświetlać wszystkie lub pojedynczo. W wyszukiwaniu wszystkich albumów jest dodatkowa opcja dodania 3 wartości **SearchWord** -> nazwy szukanej po zmiennej **Title, SortBy** -> Title sortuje po tym rekordzie, **SortDirection** -> 0 to rosnąco lub 1 malejąco sortowanie. W wyszukiwaniu piosenek brak dodatkowych filtrów.

## **File**

**GET -> file**

Możliwość odczytania zawartości danego pliku **PrivateFiles/private-file.txt** tekstowego który znajduje się a serverze. Trzeba użyć ścieżki **url + file/?fileName=private-file.txt**. Mogą używać tylko użytkownicy zalogowani.

Jeśli osoby niezalogowane próbują skorzystać z każdej akcji poza akcji **GET**, to wystąpi informacja -> **401 Unauthorized**.

Jeśli użytkownik zalogowany bez danych uprawnień do danej akcji **PUT, DELETE, POST**, to informacja -> **403 Forbidden**

## Album

**POST -> api/artist/{artistId}/album**

Możliwość stworzenia nowego albumu, trzeba podać wartości **title, length** -> długość całego albumu, **price**. Nie można podać takiej samej tytułu, która jest już w Artyście.

**PUT -> api/artist/{artistId}/album/{albumId}**

Możliwość aktualizacji danych albumu, podając wartości takie jak w akcji POST.

**GET -> api/artist/{artistId}/album** -> możliwe wyświetlić wszystkie albumy

**GET -> api/artist/{artistId}/album/{albumId}** -> wyświetla jeden album

**DELETE -> api/artist/{artistId}/album** -> usuwa wszystkie albumy

**DELETE -> api/artist/{artistId}/album/{albumId}** -> usuwa jeden album

## Song

**POST -> api/artist/{artistId}/album/{albumId}/song** -> tworzy nową unikatową piosenkę z podaniem wartości name

**GET -> api/artist/{artistId}/album/{albumId}/song** -> wyświetla wszystkie piosenki

**GET -> api/artist/{artistId}/album/{albumId}/song/{songId}** -> wyświetla jedną piosenkę

**DELETE -> api/artist/{artistId}/album/{albumId}/song** -> usuwa wszystkie piosenki

**DELETE -> api/artist/{artistId}/album/{albumId}/song/{songId}** -> usuwa tylko jedną piosenkę

**PUT -> api/artist/{artistId}/album/{albumId}/song/{songId}** -> aktualizuje piosenkę, nazwa musi być unikatowa, nie duplikat.

W albumie dodatkowa zmienna numberOfSongs pokazuje aktualną liczbę piosenek w danym albumie. Gdy piosenkę się dodaje lub usuwa, to zmienna zmienia wartość.

Opis działania aplikacji ze zdjęciami:

The screenshot shows the Swagger UI for the MusicStore API. The browser address bar indicates the URL is `localhost:7195/swagger/index.html`. The Swagger logo and "Supported by SMARTBEAR" are in the top left. The top right has a "Select a definition" dropdown with "MusicStore API" selected. The main title is "MusicStoreApi" with version "1.0" and "OAS3" tags. Below the title is the link `/swagger/v1/swagger.json`. The API is organized into sections: "Account" and "Album". Under "Account", there are two POST endpoints: `/api/account/register` and `/api/account/login`. Under "Album", there are six endpoints: a POST endpoint `/api/artist/{artistId}/album`, a DELETE endpoint `/api/artist/{artistId}/album`, a GET endpoint `/api/artist/{artistId}/album`, a PUT endpoint `/api/artist/{artistId}/album/{albumId}`, a DELETE endpoint `/api/artist/{artistId}/album/{albumId}`, and a GET endpoint `/api/artist/{artistId}/album/{albumId}`. Each endpoint is represented by a colored box with the HTTP method and the URL path.

Swagger  
Supported by SMARTBEAR

Select a definition **MusicStore API**

# MusicStoreApi 1.0 OAS3

</swagger/v1/swagger.json>

## Account

- POST** `/api/account/register`
- POST** `/api/account/login`

## Album

- POST** `/api/artist/{artistId}/album`
- DELETE** `/api/artist/{artistId}/album`
- GET** `/api/artist/{artistId}/album`
- PUT** `/api/artist/{artistId}/album/{albumId}`
- DELETE** `/api/artist/{artistId}/album/{albumId}`
- GET** `/api/artist/{artistId}/album/{albumId}`

Artist	
POST	/api/artist
GET	/api/artist
DELETE	/api/artist/{id}
PUT	/api/artist/{id}
GET	/api/artist/{id}
File	
GET	/file
POST	/file
Song	
POST	/api/artist/{artistId}/album/{albumId}/song
DELETE	/api/artist/{artistId}/album/{albumId}/song
GET	/api/artist/{artistId}/album/{albumId}/song
PUT	/api/artist/{artistId}/album/{albumId}/song/{songId}
DELETE	/api/artist/{artistId}/album/{albumId}/song/{songId}
GET	/api/artist/{artistId}/album/{albumId}/song/{songId}

Widok głównej strony aplikacji, dzięki wykorzystaniu **swaggera**.

SELECT TOP (1000) [Id]  
, [Name]  
, [Description]  
, [KindOfMusic]  
, [ContactEmail]  
, [ContactNumber]  
, [CreatedById]  
, [AddressId]  
FROM [MusicStoreDb].[dbo].[Artists]

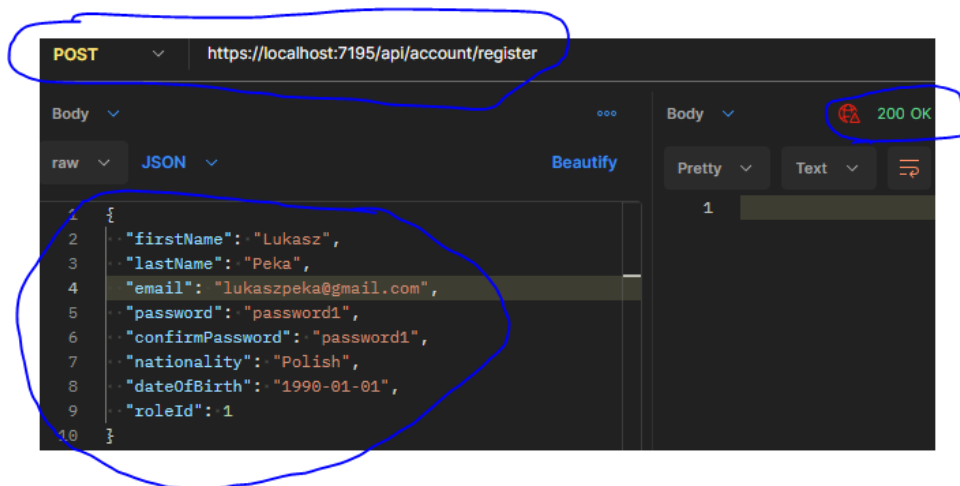
100 %

Results

Messages

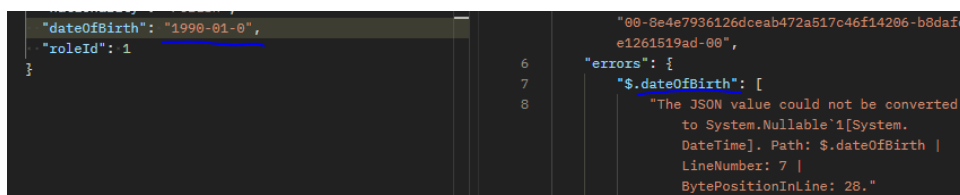
	Id	Name	Description	KindOfMusic	ContactEmail	ContactNumber	CreatedById	AddressId
1	1	Iron Maiden 1	information about ... 1	rock, metal 1	contact@email1.com	987423851	2	1
2	2	Iron Maiden 2	information about ... 2	rock, metal 2	contact@email2.com	987423852	2	2
3	3	Iron Maiden 3	information about ... 3	rock, metal 3	contact@email3.com	987423853	2	3
4	4	Iron Maiden 4	information about ... 4	rock, metal 4	contact@email4.com	987423854	2	4
5	5	Iron Maiden 5	information about ... 5	rock, metal 5	contact@email5.com	987423855	2	5
6	6	Iron Maiden 6	information about ... 6	rock, metal 6	contact@email6.com	987423856	2	6
7	7	Iron Maiden 7	information about ... 7	rock, metal 7	contact@email7.com	987423857	2	7

Podczas pierwszego uruchomienia aplikacji , gdy w bazie danych brak rekordów, to wypełnia się przykładowymi danymi.

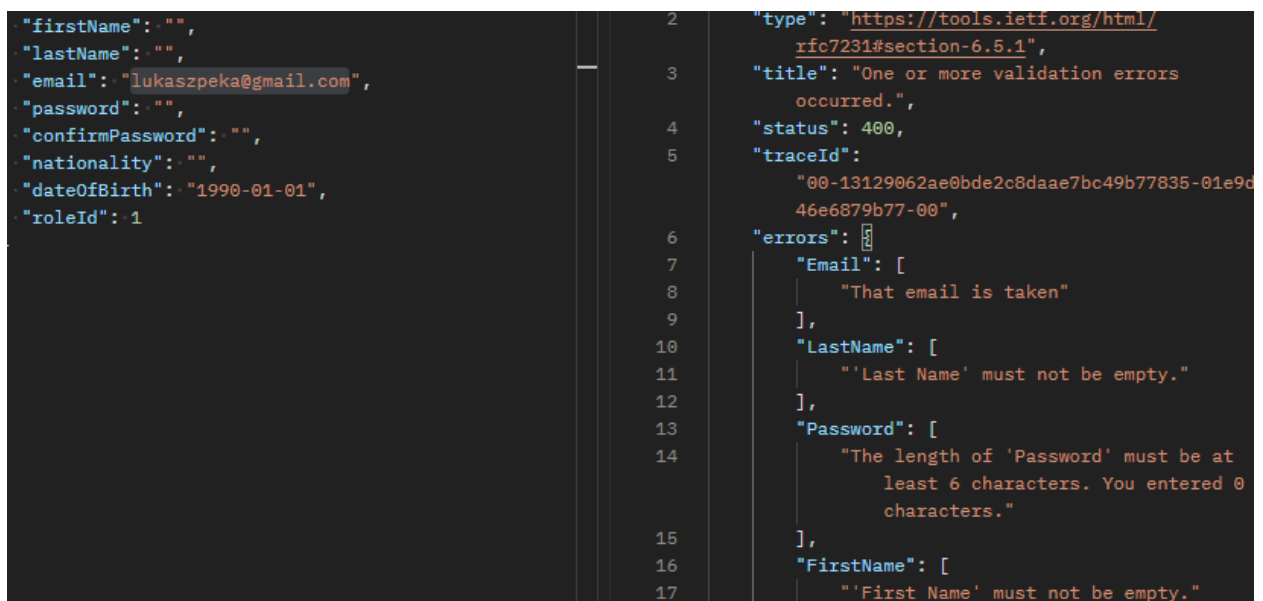


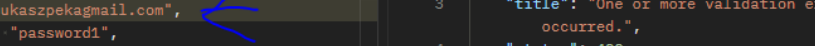
	Id	Email	FirstName	LastName	DateOfBirth	Nationality
1	1	user@gmail.com	Adam	Ciska	1990-01-01 00:00:00.0000000	Polish
2	2	premiumuser@gmail.com	Blażej	Mana	1987-05-06 00:00:00.0000000	Polish
3	3	admin@gmail.com	Grzegorz	Rybka	2000-02-03 00:00:00.0000000	Greek
4	4	lukaszpeka@gmail.com	Lukasz	Peka	1990-01-01 00:00:00.0000000	Polish

Najpierw się rejestrujemy na portalu sklepu muzycznego : **api/account/register**, wysyłamy nowe informacje za pomocą Jsona, odpowiedź zwrotna przyszła -> **200 OK** . Stworzyłem nowego usera , z rolą 1, czyli zwykły user.



Zabezpieczenia przed złymi danymi, tutaj błędna data i informacja zwrotna o tym.





```

{
  "email": "lukaszpekagmail.com",
  "password": "password1",
  "confirmPassword": "password1",
  "nationality": "Polish",
  "dateOfBirth": "1990-01-01",
  "roleId": 1
}

```

3 "title": "One or more validation errors occurred.",

4 "status": 400,

5 "traceId": "00-c5c89eac5dcf2a74aac8000291bfa93b-a273cd9966e2715e-00",

6 "errors": {

7 "Email": [

8 "'Email' is not a valid email address."

```
"password": "password1",
"confirmPassword": "password",
"errors": {
  "ConfirmPassword": [
    "'Confirm Password' must be equal to 'password1'."
  ]
}
```

**POST** <https://localhost:7195/api/account/login>

```
{
  "email": "lukaszpeka@gmail.com",
  "password": "password1"
}
```

Potem wchodzimy na strone logowania do portalu : **api/account/login** , wysyłamy Jsonem **email** i **password**, jeśli dane się zgadzają to wraca odpowiedź 200 OK i **hashPassword** się generuje.



```

{
  "email": "",
  "password": ""
}


```




```

"Email": [
  "'Email' must not be empty.",
  "'Email' is not a valid email address.",
  ",
  "Invalid username or password"
],
"Password": [
  "The length of 'Password' must be at
  least 6 characters. You entered 0
  characters."
]

```

Jeśli email pusty lub błędny, jeśli **password** ma mniej niż 6 znaków są powiadomienia.

Body  400 Bad Request 17 ms 138 B

Pretty  Text  

1 Invalid username or password

Zabezpieczenia jeśli poda się emaila lub błędne hasło to informacja zwrotna **400 Bad Request** i info: **Invalid username or password**. Ze względów bezpieczeństwa, nie podaje się dokładnie czy to błędne hasło czy email.

Potem po prawidłowym zalogowaniu możemy korzystać z naszego portalu muzycznego.

<https://localhost:7195/api/artist/?pagesize=5&pagenumber=2>

```

"items": [
  {
    "id": 6,
    "name": "Iron Maiden 6",
    "description": "information about ...
      6",
    "kindOfMusic": "rock, metal 6",
    "country": "USA 6",
    "city": "New York 6",
    "albums": [
      {
        "id": 11,
        "title": "Armagedon 6",
        "length": 3.5,
        "numberOfSongs": 3,
        "price": 10.3,
        "songs": [
          {
            "id": 31,
            "name": "Planet is
              good 1 6",
            "albumId": 11
          }
        ]
      }
    ]
  }
]

```

```

"totalPages": 30,
"itemFrom": 6,
"itemTo": 10,
"totalItemsCount": 150

```

Potem wchodzimy na strone : **api/artist?pageSize=5&pageNumber=2** , która zwraca nam 5 artystów, z jego albumami i piosenkami. Trzeba podać 2 warunki **pageSize**: 5, 10, 15 -> czyli po ile rekordów ma pokazać na stronie, **pageNumber**: 1,2,3.... którą strone aktualnie chce się przeglądać. Wyświetla dodatkowo informacje o stronicowaniu wyników (użycie paginacji) -> jeżeli **pageSize** = 5 i **pageNumber** = 2 to wartość **itemFrom** pokazuje początek strony od której pozycji się zaczyna , a **itemTo** na której się kończy. **TotalPages** pokazuje wszystkie możliwe strony. A **totalItemsCount** -> to wszystkie rekordy.

```

GET https://localhost:7195/api/artist?PageSize=5&pageNumber=3&SortDirection=asc&sortBy=Name&searchWord=korn

```

☒ PageSize 5
 ☒ pageNumber 3
 ☒ SortDirection asc
 ☒ sortBy Name
 ☒ searchWord korn

```

      "items": [
        {
          "id": 41,
          "name": "Korn 41",
          "description": "information about ...
            41",
          "kindOfMusic": "rock, metal 41",
          "country": "USA 41",
          "city": "New York 41",
          "albums": [
            {
              "id": 81,
              "title": "Armagedon 41",
              "length": 3.5,
              "numberOfSongs": 3,
              "price": 10.3,
              "songs": [
                {
                  "id": 241,
                  "name": "Planet is
                    good 1 41",
                  "albumId": 81
                }
              ]
            }
          ]
        }
      ]
    
```

```

    },
    "totalPages": 3,
    "itemFrom": 11,
    "itemTo": 15,
    "totalItemsCount": 15
  }
}
  
```

Są jeszcze 3 dodatkowe opcje filtrowania danych: **SortDirection** -> sortuje asc (rosnąco) lub desc (malejąco) lub 0, 1, **SortBy** -> sortuje po danych rekordach **Name**, **Description**, **KindOfMusic**, **SearchWord** -> wyszukuje po słowie korn.

☒ sortBy Kin

```

      "errors": {
        "SortBy": [
          "Sort by is optional, or must be in
            [Name,Description,KindOfMusic]"
        ]
      }
    
```

Jeśli błędna nazwa opcji **SortBy**, to dana informacja o prawidłowych rodzajach rekordów.

<input checked="" type="checkbox"/>	PageSize	5
<input checked="" type="checkbox"/>	pageNumber	4
<input checked="" type="checkbox"/>	SortDirection	asc
<input checked="" type="checkbox"/>	sortBy	Name
<input checked="" type="checkbox"/>	searchWord	korn

400 Bad Request 6 ms 272 B Save as example

Text

search result items of Artists: 15 is too small  
or equal, because the number of skip: 15 ,  
change the values in 'PageSize = 5,  
PageNumber = 1' , to see the result

Jeżeli liczba wszystkich rekordów do wyświetlenia jest mniejsza lub równa od **PageSze** \* (**PageNumber**-1) , to informacja zwrotna **400 Bad Request** i info o zbyt małej liczbie rekordów do wyświetlenia oraz aby zmienić **PageNumber** = 1 .

GET https://localhost:7195/api/artist/35/album?SearchWord=&SortBy=Title&SortDirection=1

Params Auth Headers (7) Body Pre-req Tests Settings


```

{
  "id": 70,
  "title": "Dance to death 35",
  "length": 3.5,
  "numberOfSongs": 3,
  "price": 10.3,
  "songs": [
    {
      "id": 208,
      "name": "Where are you baby 1 35",
      "albumId": 70
    },
    {
      "id": 209,
      "name": "Alicia is nice 2 35",
      "albumId": 70
    },
    {
      "id": 210,
      "name": "I'm crying 3 35",
      "albumId": 70
    }
  ]
}

{
  "id": 69,
  "title": "Armagedon 35",
  "length": 3.5,
  "numberOfSongs": 3,
  "price": 10.3,

```

Wyszukiwanie albumów danego artysty można wykorzystać 3 dodatkowe funkcje **SearchWord**: wyszukuje po danym tytule, **SortBy**: sortuje po danej nazwie Title, **SortDirection**: sortuje rosnąco, malejąco -> 0 (asc), 1 (desc). Trzeba użyć w **SortBy** -> Title, a w **SortDirection** -> 0, 1 lub asc, desc.



Headers 8 hidden

	Key	Value
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1...

403 Forbidden

Jeśli spróbuje zalogowany user z rolą : **USER** to ma komunikat -> **403 Forbidden**.

```
{
  "name": "OffSpring",
  "description": "description",
  "kindOfMusic": "Rock, metal",
  "contactEmail": "offspring@example.com",
  "contactNumber": "565485222",
  "country": "USA",
  "city": "Miami"
}
```

201 Created 44 ms

/api/artist/151

Id	Name	Description	KindOfMusic	ContactEmail	ContactNumber	CreatedById	AddressId
151	Off Spring	description	Rock, metal	offspring@example.com	565485222	5	151

Jeżeli spróbuje zalogowany user z rolą: **PREMIUM\_USER** to ma odpowiedź -> **200 Ok** i w nagłówku informacja że pod ścieżką **/api/artist/151** został stworzony nowy rekord.

```
"errors": {
  "City": [
    "The City field is required."
  ],
  "Name": [
    "The Name field is required."
  ],
  "Country": [
    "The Country field is required."
  ],
  "Description": [
    "The Description field is required."
  ],
  "ContactEmail": [
    "The ContactEmail field is not a valid e-mail address."
  ],
  "ContactNumber": [
    "The ContactNumber field is not a valid phone number."
  ]
}
```

```
{
  "name": "",
  "description": "",
  "kindOfMusic": "",
  "contactEmail": "",
  "contactNumber": "",
  "country": "",
  "city": ""
}
```

Jeśli poda się błędne dane, to otrzymuje się odpowiedź , że **City, Name, Country Description** nie mogą być puste i **ContactNumber ,ContactEmail** sprawdza poprawność.

```
GET https://localhost:7195/api/artist/151

{
  "id": 151,
  "name": "OffSpring",
  "description": "description",
  "kindOfMusic": "Rock, metal",
  "country": "USA",
  "city": "Miami",
  "albums": []
}
```

200 OK 4 m

Potem wchodzimy na akcje wyszukania artysty po id, którego utworzyliśmy pod numerem id 151 -> OffSpring.

```
GET https://localhost:7195/api/artist/1002

404 Not Found 3
```

```
Artist 1002 is not found
```

Jeżeli będzie próba szukania rekordu nieistniejącego w bazie danych, to odpowiedź jest **404 Not Found**, i info: **Artist 1002 is not found**.

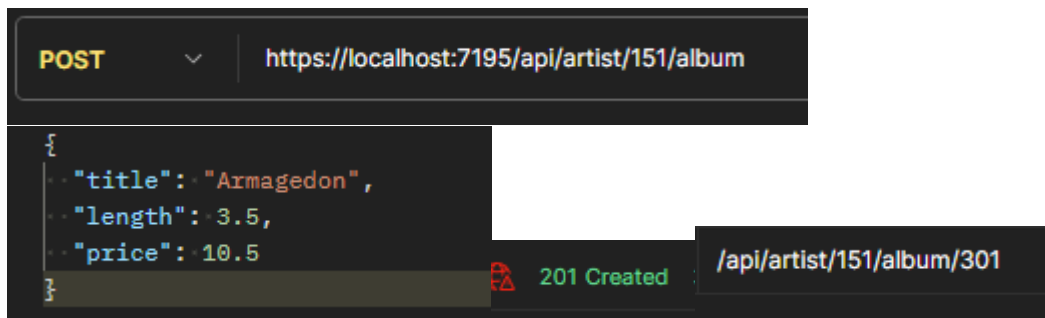
```
POST https://localhost:7195/api/artist

{
  "name": "OffSpring",
  "description": "description",
  "kindOfMusic": "Rock, metal",
  "contactEmail": "offspring@example.com",
  "contactNumber": "565485222",
  "country": "USA",
  "city": "Miami"
}
```

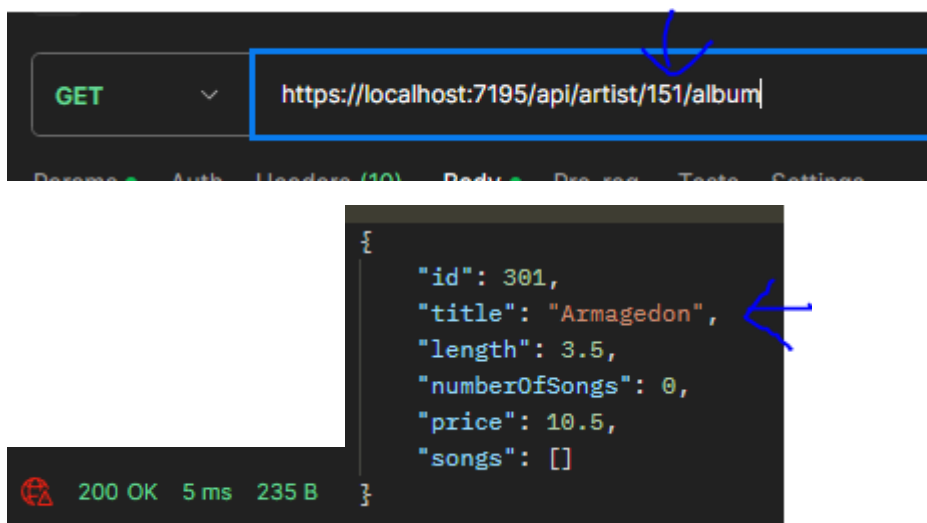
```
Name : invalid value because there is already an artist
created by this user (duplicate)

409 Conflict 11 ms
```

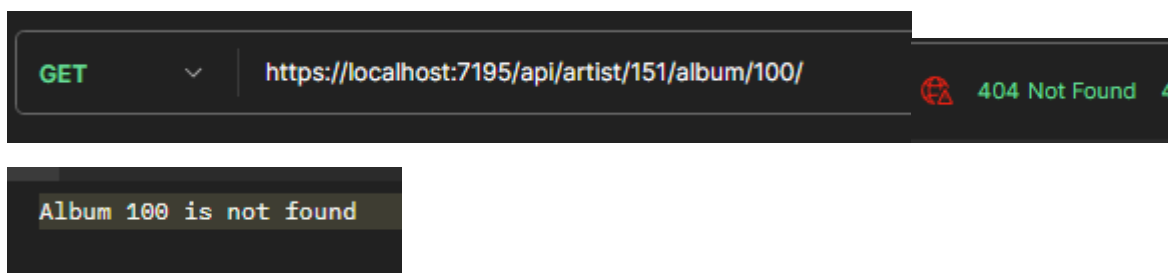
Jeżeli podczas próby stworzenia nowego Artysty, lecz o takiej samej nazwie, która występuje w bazie, która już została przez tego usera stworzona, to wyskoczy błąd **409 Conflict**, że jest nieprawidłowa wartość **Name**.



Następnie tworzy się nowy album, informacja zwrotna **201 Created** i ścieżka pod którą stworzono nowy album **api/artist/151/album/301**.

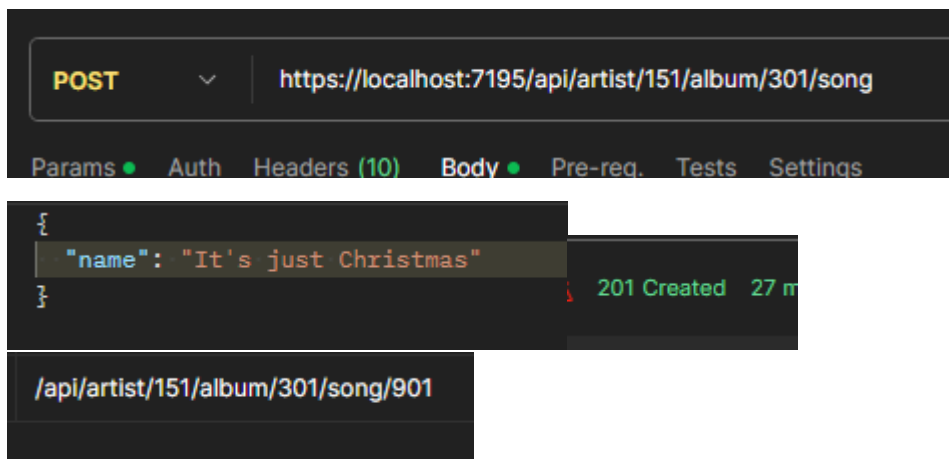


Później możliwość wyszukiwania wszystkich albumów po numerze id artysty lub tylko danego albumu po nr id albumu -> **api/artist/151/album/301**

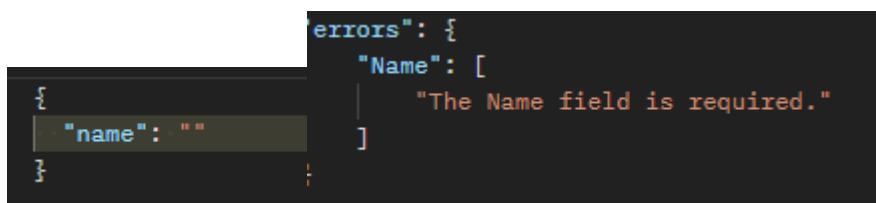


Jeżeli będzie próba wyszukania nieistniejącego albumu, to informacja zwrotna **404 Not Found** i info: **Album 100 is not found**.

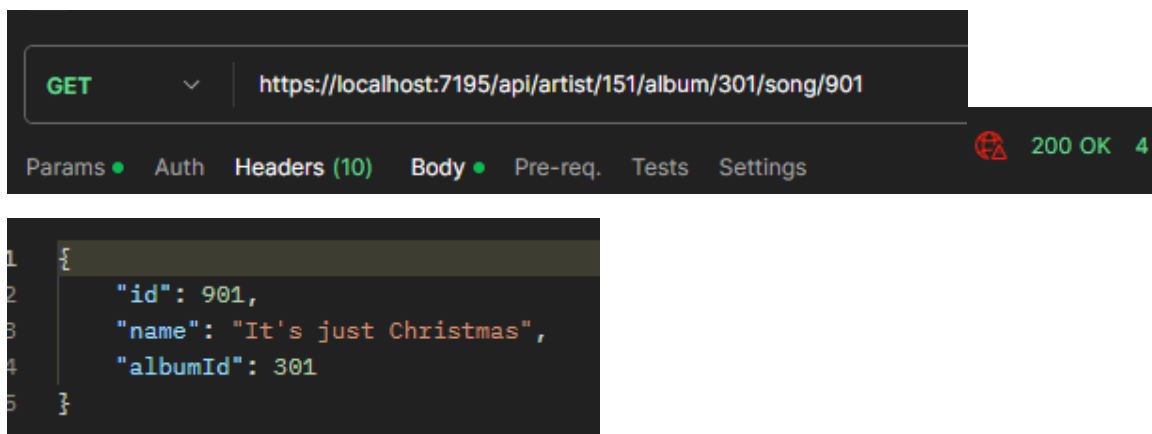




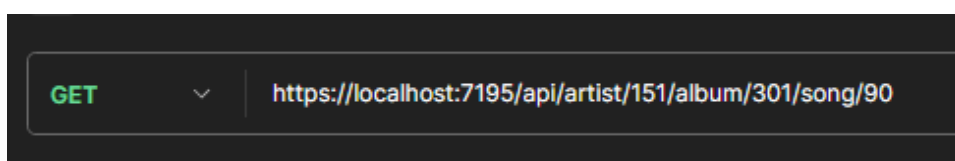
Później tworzymy nowe piosenki, gdy wszystko ok, to otrzymujemy odpowiedź **201 Created** i ścieżkę pod którą została nowa piosenka utworzona -> **api/artist/151/album/301/song/901**



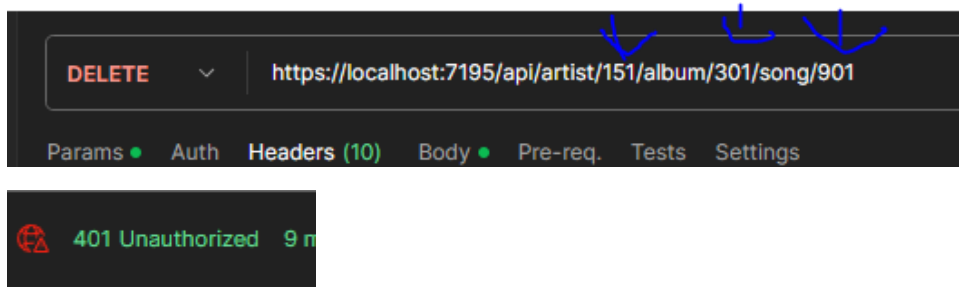
Jeżeli będzie próba podania pustej nazwy Name, to wystąpi error z informacją, że dane pole nie może być puste.



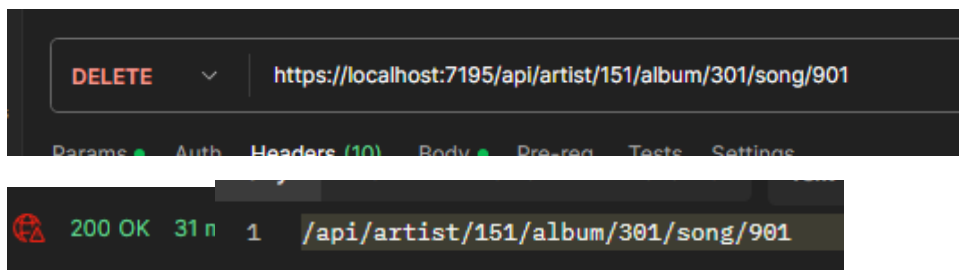
Potem możliwość wyświetlenia wszystkich piosenek lub jednej piosenki po numerze id.



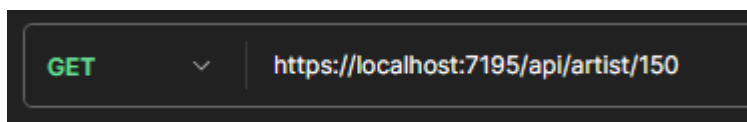
Jeżeli poda się nieprawidłowy nr id piosenki, to informacja, że **Song 90 is not found** i kod **404 Not Found**.



Potem usuwamy daną piosenkę jako niezalogowany user , to informacja jest **401 Unauthorized** -> czyli brak autoryzacji. Przy próbie zalogowania usera z rolą : **USER**, to informacja zwrotna **403 Forbidden**.



User który stworzył daną piosenkę , to może ją tylko usunąć lub user z rolą **ADMIN**. Gdy się udało to informacja **200 OK** i ścieżka piosenki , z której została usunięta.



```

{id": 150,
"name": "Venflon 150",
"description": "information about ...
150",
"kindOfMusic": "rock, metal 150",
"country": "USA 150",
"city": "New York 150",
"albums": [
  {
    "id": 299,
    "title": "Armagedon 150",
    "length": 3.5,
    "numberOfSongs": 3,
    "price": 10.3,
    "songs": [
      {
        "id": 895,
        "name": "Planet is good 1
150",
        "albumId": 299
      },
      {
        "id": 896,
        "name": "This is a cat 2
150",
        "albumId": 299
      },
      {
        "id": 897,
        "name": "So tired 3 150",
        "albumId": 299
      }
    ]
  }
]
}

```

DELETE <https://localhost:7195/api/artist/150> 1 Artist 150 is not found

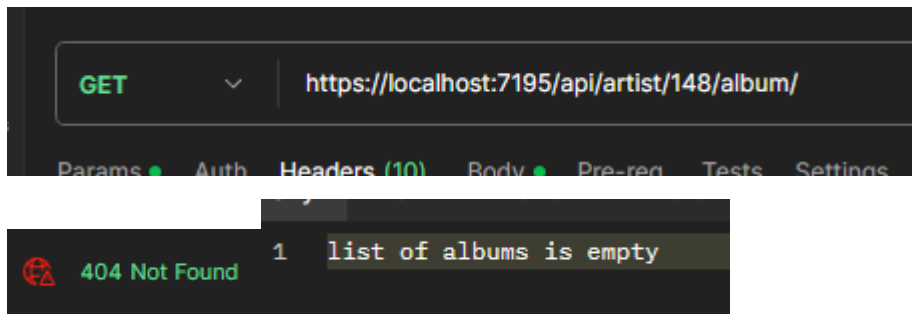
Teraz usuniemy całego Artyste z albumami i piosenkami pod nr id 150, później sprawdzamy czy jest , lecz dostajemy odpowiedź **Artist 150 not found**.

GET <https://localhost:7195/api/artist/148/album/295/song/>

1 list of songs is empty

404 Not Found 5 ms

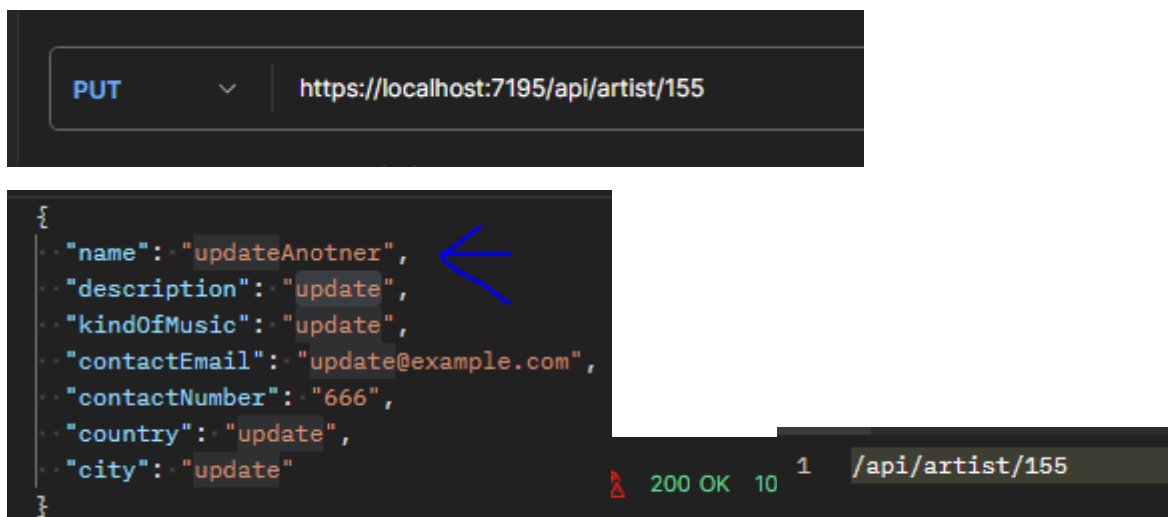
Jeżeli próbujemy wyszukać wszystkie piosenki z danego albumu, a jest brak piosenek, to jest odpowiedź: **list of songs is empty**.



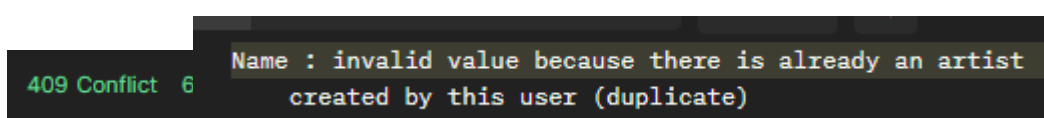
Jeżeli próbujemy wyszukać wszystkie albumy z danego artysty, a jest brak albumów, to jest odpowiedź: **list of albums is empty**.

Podsumowanie działania autoryzacji na podstawie danego **Claima** roli usera, zawartym w Tokenie JWT:

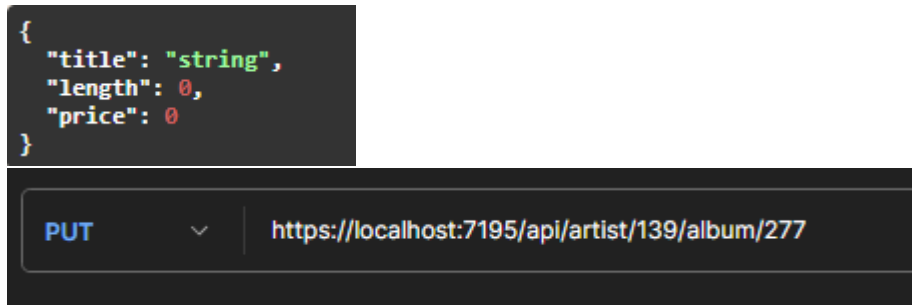
- Jeżeli zalogowany user z rolą **USER** -> może tylko przeglądać wszystko -> czasownik GET
- Jeżeli zalogowany user z rolą **PREMIUM\_USER** -> może wszystko, lecz gdy chce coś usunąć, zaktualizować dane, to jedynie swoich artystów, albumy, piosenki.
- Jeżeli zalogowany user z rolą **ADMIN** -> może wszystko.



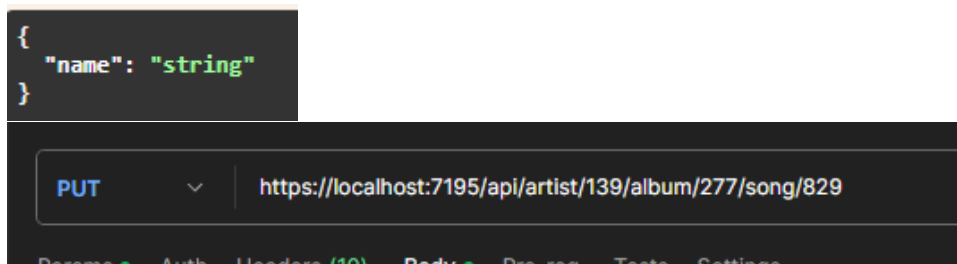
Teraz przejdziemy do akcji aktualizacji danych -> czasownik **PUT**. Można zmienić kilka wartości danego artysty: **name, description, kindOfMusic, contactEmail, contactNumber, Country, City**.



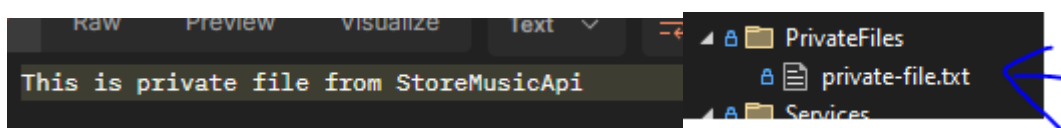
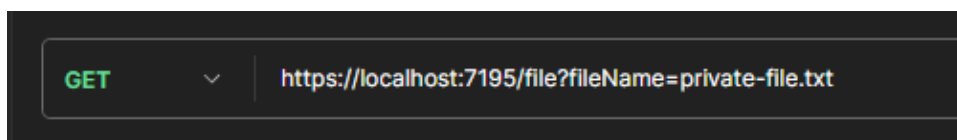
Jeżeli będzie próba zmiany nazwy artysty, albumu lub piosenki na już istniejącą nazwę w bazie, która została już przez danego usera stworzona to będzie błąd -> **409 Conflict**, z info: **Name : invalid value because there is already an artist created by this user (duplicate).**



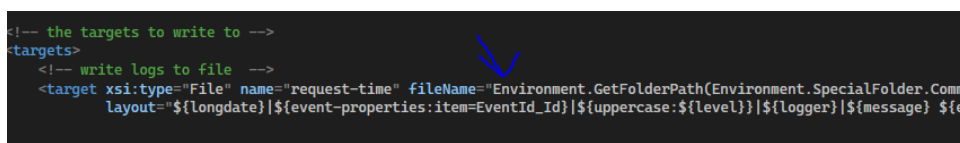
Tutaj aktualizacja danych danego albumu, można zmienić 3 wartości **title**, **length** (długość całkowita albumu), **price**.



Tutaj aktualizacja danych danej piosenki, można zmienić wartość **name**.



Później mamy też możliwość wglądu do pliku tekstowego, który znajduje się na serverze o nazwie : **private-file.txt**. Dostęp tylko mają userzy zalogowani.

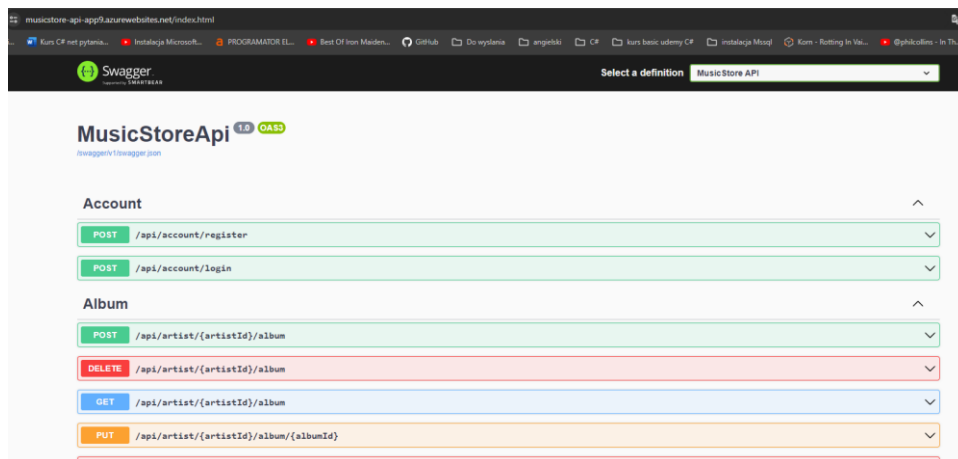


Jeszcze jest wykorzystana biblioteka **NLogger**, do tworzenia logów w katalogu **MusicStoreAPILogs**, która jest skonfigurowana w pliku `nlog.config`. Najlepiej samemu potem zmienić ścieżkę gdzie się

chce mieć dane logi -> **allLogs, exceptions, request-time** -> występuje gdy zapytanie do servera trwa powyżej 4 sekund.

Wykorzystanie **middleware** jako klasy **ErrorHandlingMiddleware**, która odpowiedzialna za analizowanie zapytań, które przychodzą do API i gdy wystąpi wyjątek, dodaje do loggera exceptions o tym informacji.

Dodano politykę **CORS**, aby w przyszłości można było wysyłać zapytania do danego api, z innej domeny frontendowej.



Wdrożona aplikacja na chmurze Azure : <https://musicstore-api-app9.azurewebsites.net/swagger>

Możliwe akcje do wykonania w danym sklepie muzycznym:

Account

**POST -> api/account/register**

**POST -> api/account/login**

File

**GET -> file**

Artist

**POST -> api/artist**

**GET -> api/artist**

**GET -> api/artist/{id}**

**DELETE -> api/artist/{id}**

**PUT -> api/artist/{id}**

Album

**POST -> api/artist/{artistId}/album**

**GET -> api/artist/{artistId}/album**

**GET -> api/artist/{artistId}/album/{albumId}**

**DELETE -> api/artist/{artistId}/album**

**DELETE -> api/artist/{artistId}/album/{albumId}**

**PUT -> api/artist/{artistId}/album/{albumId}**

Song

**POST -> api/artist/{artistId}/album/{albumId}/song**

**GET -> api/artist/{artistId}/album/{albumId}/song**

**GET -> api/artist/{artistId}/album/{albumId}/song/{songId}**

**DELETE -> api/artist/{artistId}/album/{albumId}/song**

**DELETE -> api/artist/{artistId}/album/{albumId}/song/{songId}**

**PUT -> api/artist/{artistId}/album/{albumId}/song/{songId}**