

Web Api MusicStoreApi back-end application

Used: C#, ASP.NET Core (.NET 6.0), REST, Entity Framework, MS_SQL, Swagger(homepage) documentation, Microsoft Azure Cloud, NLogger, JWT Token, Postman, CORS policy, Middleware.

POST -> api/account/register

A program that presents a music shop portal, where you can register as a new user with a role:

- **USER** -> you can view everything (**GET**), you cannot create (**POST**), delete (**DELETE**), update (**UPDATE**)
- **PREMIUM_USER** -> you can view everything (**GET**), create (**POST**) only unique artist names with albums and songs, and delete (**DELETE**) and update (**UPDATE**) only your created artists, albums, songs.
- **ADMIN** -> you can do everything,

We send the information via JSON: **firstName, lastName, email, password, confirmPassword, nationality, dateOfBirth, roleId**. By the **roleId** variable you know what permissions a user has: **1 -> USER, 2 -> PREMIUM_USER, 3 -> ADMIN**. Security:

- if wrong date is entered, error message
- if email is empty, error message -> "**email must not be empty**", if it is already in the database, error -> "**that email is taken**", if email is invalid, error -> "**is not a valid email address**"
- if **lastName** and **firstName** are empty, the message -> "**must not be empty**"
- if **password** is less than 6 characters, message -> "**The lenght of Password must be at least 6 characters. You entered 5 characters**"
- if you enter **confirmPassword** not equal to **password**, then error -> "**must be equal to value of password**"

When registering correctly, the new password is hashed and PasswordHash is saved to the database.

POST -> api/account/login

After that, you can log in to the music portal, using the sending of email information as login and password. Security:

- if you enter an empty or invalid email, a corresponding error message
- If you enter a wrong password or email, the error message -> "**invalid username or password**".
- If you enter a password with less than 6 characters, an error message will appear.

When logging into the music shop, the login (**email**) is checked to see if it is the same in the database, the password hash of the user (**PasswordHash**) is checked against the password hash (**PasswordHash**) which is in the database. If everything is ok, the server generates a JWT token containing information about the user(**CLAIMY**): **UserId, FirstName, LastName, RoleName, DateOfBirth**. To be able to use all the actions in the api as a logged in user, you need to send in the header **Key** -> **Authorization**, and **Value** -> **Bearer** {the generated **JWT Token** that was generated at login}.

Artist

GET -> api/artist

You can search for all artists with its albums and songs, you need to specify two values in the parameters: **PageSize** -> **5, 10, 15** and **PageNumber** -> **1, 2, 3** ... etc. In addition, 3 more values can be entered: **SearchWord** -> searches for a given word after given Name or Description records, **SortDirection** -> **1(asc), 2(desc)**, sorts ascending or descending, **SortBy** -> sorts after given Name, Description, KindOfMusic records. In addition, result pagination(pagination) information with values is shown: **TotalPages** -> all possible pages, **TotalItemsCount** -> all records, **ItemFrom** -> shows the beginning of the page where the item starts, **ItemTo** -> shows the end of the page where the item ends.

Security features:

-if the number of all records to be displayed is less than or equal to **PageSize * (PageNumber-1)**, then an error -> 400 Bad Request with the message "**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** "

-if you specify **PageSize** incorrectly, i.e. other than 5, 10, 15 , then error 400 Bad Request with the information -> "**must in [5, 10, 15]**".

-If you specify **PageNumber** incorrectly i.e. other than 1, 2, 3..., then error 400 with information -> "**must be greater than or equal to 1**"

GET -> api/artist/{id}

You can search by the id number of a particular artist.

Safeguards:

-if you specify an **artistId** that does not exist in the database, a 404 Not Found error with the message -> "**Artist {artistId} is not found**"

POST -> api/artist

You can create a new artist by specifying the values **Name, Description, KindOfMusic, ContactEmail, ContactNumber, Country, City**. Can only be used by a logged in user with the role: **PREMIUM_USER** and **ADMIN**.

Security features:

-if you enter **City, Name, Country, Description** as empty value, then error with message -> "**field is required**".

-if you enter **ContactEmail** and **ContactNumber** wrong, then error with message -> "**is not a valid {variable}**"

-if you enter an artist **name** that was already created by this user, error with info => "**Name: invalid value because there is already on artist created by this user(duplicate)**"

DELETE -> api/artist/{id}

You can delete a given album by specifying the album id. Only a user with the role: **USER**, who created the artist in question, or a user with the role: **ADMIN** can delete the album in question.

Security features:

-if you specify an **artistId** that does not exist in the database, an error with the message -> "**Artist {artistId} is not found**".

PUT -> api/artist/{id}

Artist can be updated, with values **name, description, kindOfMusic, contactEmail, contactNumber, Country, City**. Changes can only be done by a user with the role: **PREMIUM_USER** who created the artist in question or a user with the role: **ADMIN**.

Security features:

-if you specify an artist name already in the database, created by a given user, error 409 Conflict with the message -> "**Name : invalid value because there is already an artist created by this user (duplicate)**".

In albums and songs, additional possibility still to delete all albums or songs. Safeguards:

-if there are no albums to delete, or no albums to display, the information -> "**list of albums is empty**"

-if there are no songs to delete, or no songs to display, the message -> "**list of songs is empty**"

Albums and songs just like artists, can be created new, deleted, updated, displayed all or individually. In the search for all albums there is an additional option to add 3 values **SearchWord** ->

search name after the variable **Title**, **SortBy** -> Title sorts by this record, **SortDirection** -> **0** is ascending or **1** descending sorting. No additional filters in the song search.

File

GET -> file

Ability to read the contents of a given **PrivateFiles/private-file.txt** text file that is located a server. You need to use the path **url + file/?fileName=private-file.txt**. Only logged-in users can use it.

If non-logged-in users try to use any action except the **GET** action, the message -> **401 Unauthorized** will occur.

If a logged-in user without credentials is authorized to the action **PUT, DELETE, POST** , the message -> **403 Forbidden** will occur.

Album

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

Ability to create a new album, you need to specify the values of **title**, **length** -> length of the whole album, **price**. You cannot specify the same title that is already in Artist.

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

Ability to update the album data, providing values like in the POST action.

GET -> api/artist/{artistId}/album -> possible to display all albums

GET -> api/artist/{artistId}/album/{albumId} -> displays one album

DELETE -> api/artist/{artistId}/album -> deletes all albums

DELETE -> api/artist/{artistId}/album/{albumId} -> deletes one album

Song

POST -> api/artist/{artistId}/{album/{albumId}}/song -> creates a new unique song with name value given

GET -> api/artist/{artistId}/{album/{albumId}}/song -> displays all songs

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

DELETE -> api/artist/{artistId}/album/{albumId}/song -> deletes all songs

DELETE -> api/artist/{artistId}/album/{albumId}/song/{songId} -> deletes only one song

PUT -> api/artist/{artistId}/album/{albumId}/song/{songId} -> updates song, name must be unique , not duplicate.

In the album, an additional variable **numberOfSongs** shows the current number of songs in the album. When a song is added or deleted, this variable changes the value.

Additionally created / changed things for frontend application: FrontEndStoreMusicAPI , on github at link : <https://github.com/PatrykPrusko2019/FrontendStoreMusicApiUsingWPF> :

1. Adding a new endpoint -> **GET api/login/user/{email}** -> to access the user details and the generated **JWT Token**, when logged into the main Music Store window.
2. Adding a new endpoint -> **GET api/login/user/{userId}/artist** -> to have access, after logging into the main Music Store window, to all the artists created by a given user and the details to display.
3. Adding a new endpoint -> **GET api/song** -> retrieves all songs from the database.
4. Adding a new endpoint -> **GET api/album** -> retrieves all albums from the database.
5. Changing data sorting, if you don't select **ASC (1)**-> ascending or **DESC (2)** descending and SearchWord, it doesn't sort.
6. Adding a new endpoint -> **GET api/artist/{artistId}/album/{albumId}/song/details/{songId}** -> to access the details of a given song with additional fields: **AlbumTitle, ArtistId, ArtistName**.
7. Add a new endpoint -> **GET api/artist/{artistId}/album/details/{albumId}** -> to access the details of a given album with additional fields: **ArtistName, ArtistId**.
8. Add a new endpoint -> **GET api/artist/details/{Id}** -> to access the details of a given artist with additional fields: **ContactEmail, ContactNumber**.

The ability to connect to the Backend MusicStore , which works online, at the link:

<https://musicstore-api-app9.azurewebsites.net/swagger/index.html> , then you can connect it to the FrontEnd application as follows:

```

namespace FrontEndStoreMusicAPI.Utilites
{
    63 references
    static class HelperHttpClient
    {
        private const string uri = @"https://localhost:7195";
    }
}

```

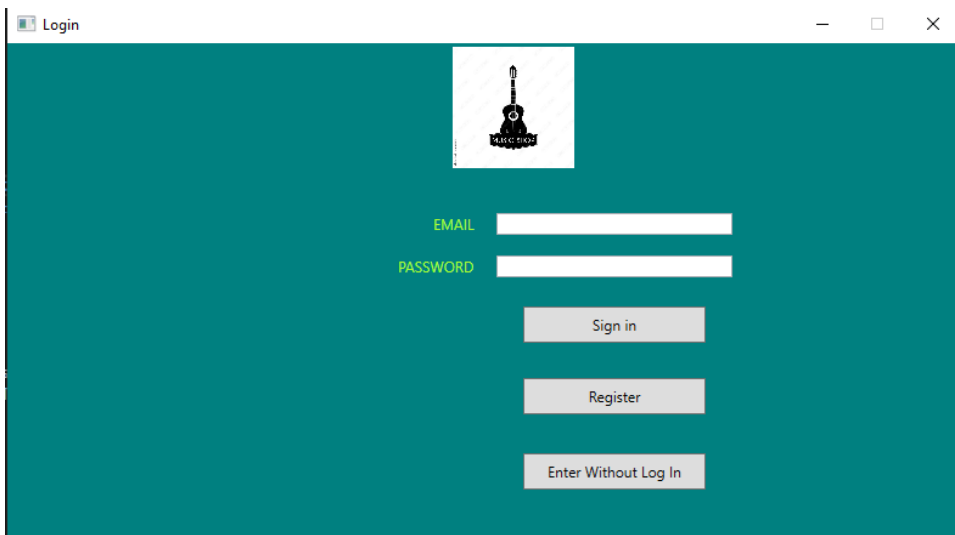
1. Go to Utilites/HelperHttpClient.cs directory -> in variable uri change from https://localhost:7195 to https://musicstore-api-app9.azurewebsites.net .

```

namespace FrontEndStoreMusicAPI.Utilites
{
    63 references
    static class HelperHttpClient
    {
        // private const string uri = @"https://localhost:7195"; // local connection
        private const string uri = @"https://musicstore-api-app9.azurewebsites.net/"; // global connection
    }
}
6 references

```

2. Now run FrontEnd applications -> **CTRL + F5** in **VISUAL STUDIO Community 2022**.



Login

EMAIL

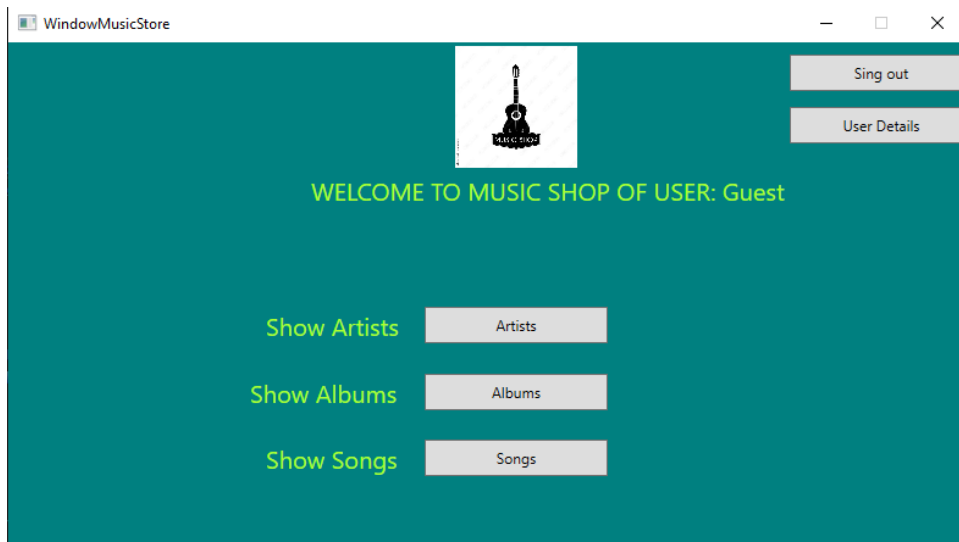
PASSWORD

Sign in

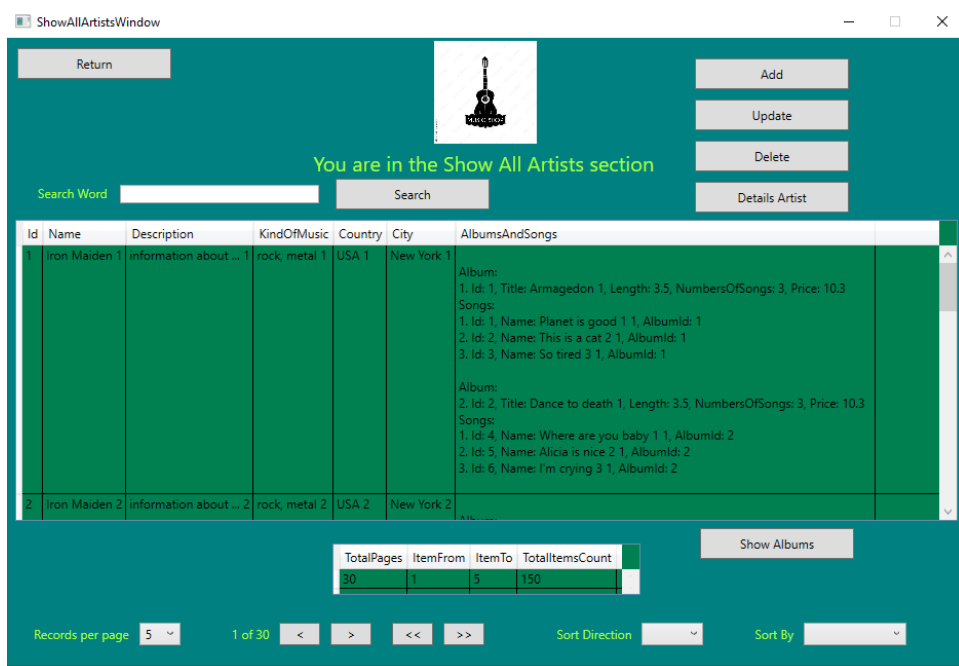
Register

Enter Without Log In

3. A login window for this application displays.



4. Music Shop main page.



5. The Show All Artists page.

6. That's it, a detailed description of how the Frontend application works on GitHub.

Description of how the application works with photos:

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}

View of the main page of the application, thanks to the use of **swagger**.

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

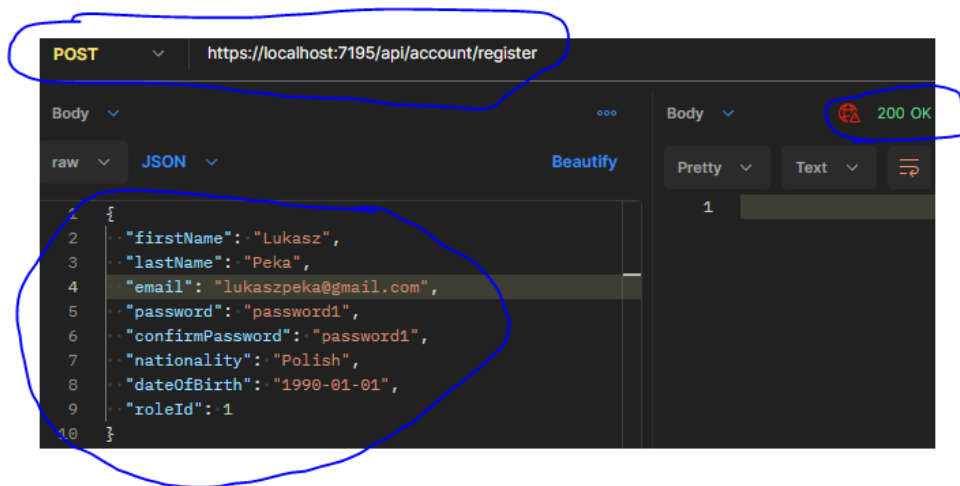
100 %

Results

Messages

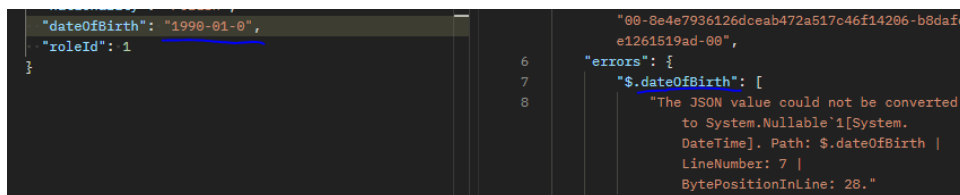
	Id	Name	Description	KindOfMusic	ContactEmail	ContactNumber	CreatedById	AddressId
1	1	Iron Maiden 1	information about ... 1	rock, metal 1	contact@email1.com	987423851	2	1
2	2	Iron Maiden 2	information about ... 2	rock, metal 2	contact@email2.com	987423852	2	2
3	3	Iron Maiden 3	information about ... 3	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

When the application is first started , if there are no records in the database, it fills in the sample data.

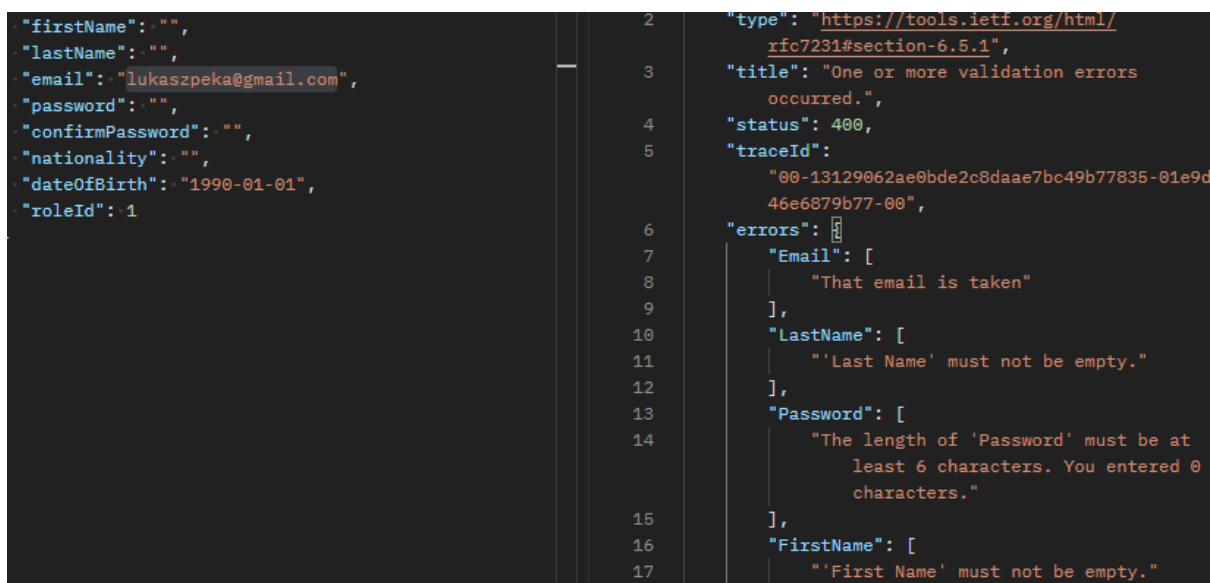


	Id	Email	FirstName	LastName	DateOfBirth	Nationality
1	1	user@gmail.com	Adam	Ciska	1990-01-01 00:00:00.0000000	Polish
2	2	premiumuser@gmail.com	Blazej	Mana	1987-05-06 00:00:00.0000000	Polish
3	3	admin@gmail.com	Grzegorz	Rybka	2000-02-03 00:00:00.0000000	Greek
4	4	lukaszpeka@gmail.com	Lukasz	Peka	1990-01-01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

First we register on the music shop portal : **api/account/register** , send new information via json , reply back came -> **200 OK** . I created a new user , with role 1, that is a normal user.



Safeguards against bad data, here an incorrect date and feedback about it.



<pre>"email": "lukaszpekagmail.com", "password": "password1", "confirmPassword": "password1", "nationality": "Polish", "dateOfBirth": "1990-01-01", "roleId": 1</pre>	<pre>3 "title": "One or more validation errors occurred.", 4 "status": 400, 5 "traceId": "00-c5c89eac5dcf2a74aac800291bfa93b-a273cc 9966e2715e-00", 6 "errors": { 7 "Email": [8 "'Email' is not a valid email address.</pre>
---	---

```
"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"
}

```

[illegible]

We then go to the portal login page : **api/account/login** , send email and password with Json, if the data match then a response of **200 OK** comes back and **hashPassword** is generated.

```

{
  "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."
]

```

If the email is empty or incorrect, if the password has less than 6 characters there are notifications.

Body  400 Bad Request 17 ms 138 B

Pretty Text 

1 Invalid username or password

Security If you enter an email or wrong password, the feedback is **400 Bad Request** and info: **Invalid username or password**. For security reasons, you do not specify whether it is an incorrect password or email.

Once correctly logged in, we can then use our music portal.

<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

```

Then we go to : **api/artist?pagesize=5&pagenumber=2** , which returns us 5 artists, with their albums and songs. You have to specify 2 conditions **pagesize**: 5, 10, 15 -> that is how many records it should show on the page, **pagenumber**: 1,2,3.... which page you currently want to view. In addition, it displays information about the pagination of the results (**use of pagination**) -> if pagesize = 5 and **pagenumber** = 2, the value **itemFrom** shows the start of the page from which the item starts , and **itemTo** on which it ends. **TotalPages** shows all possible pages. And **totalItemsCount** -> is all the records.

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

☒ PageSize
 ☒ pageNumber
 ☒ SortDirection
 ☒ sortBy
 ☒ searchWord

5
 3
 asc
 Name
 korn

200 OK 37 ms 3.34 KB Save as example

JSON

```

      "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
  }

```

There are 3 additional options for filtering the data: **SortDirection** -> sorts asc (ascending) or desc (descending) or 1, 2, **SortBy** -> sorts by the given records Name, Description, KindOfMusic, **SearchWord** -> searches by the word korn.

☒ sortBy

Kin

```

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

If the wrong name of the **SortBy** option, the given information about the correct record types.

<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

If the number of all records to be displayed is less than or equal to $\text{PageSize} * (\text{PageNumber} - 1)$, the feedback is **400 Bad Request** and info about too few records to be displayed and to change $\text{PageNumber} = 1$.

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

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

```

10 : 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,

```

You can use 3 additional functions **SearchWord**: searches by a given title, **SortBy**: sorts by a given Title name, **SortDirection**: sorts ascending, descending -> 1 (asc), 2 (desc). You have to use in **SortBy** -> Title, and in SortDirection -> 1, 2 or asc, desc.



Search for all artists, by artistId, albums and songs, all can be searched by customers -> not logged in and logged out users. After that, to be able to use all the possibilities of the music shop portal, you need to be logged in.

When registering to the portal, the user role is selected:


1. **USER** -> possibility to only browse all artists, albums, songs.
2. **PREMIUM_USER** -> possibility to use all options -> view, create, delete, update data (DELETE, UPDATE only for your artists, albums, songs)
3. **ADMIN** -> everything

[illegible]

Then, when logging in, the login(email) of the user in question and the **hashPassword** are checked against the hash from the server. If everything is ok then the server generates a **JWT token**, containing **claims** -> information about the user in question: **UserId, FirstName, LastName, RoleName, DateOfBirth**.

POST  <https://localhost:7195/api/artist>  401 Unauthorized 5

Then we create a new Artist, when trying to send a request by a user who is not logged in , this will pop up a message -> **401 Unauthorised**.



Headers 8 hidden

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

403 Forbidden

If a logged-in user with the role : USER tries it, he has the message -> **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

If a logged-in user with the role : USER tries it, he has the message -> **403 Forbidden**.

```
"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": ""
}
```

If incorrect data is entered, the response is that **City, Name, Country Description** cannot be empty and **ContactNumber, ContactEmail** checks for correctness.

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

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

We then enter the action to search for the artist by id we created at id 151 -> OffSpring.

```
GET https://localhost:7195/api/artist/1002 404 Not Found 3

Artist 1002 is not found
```

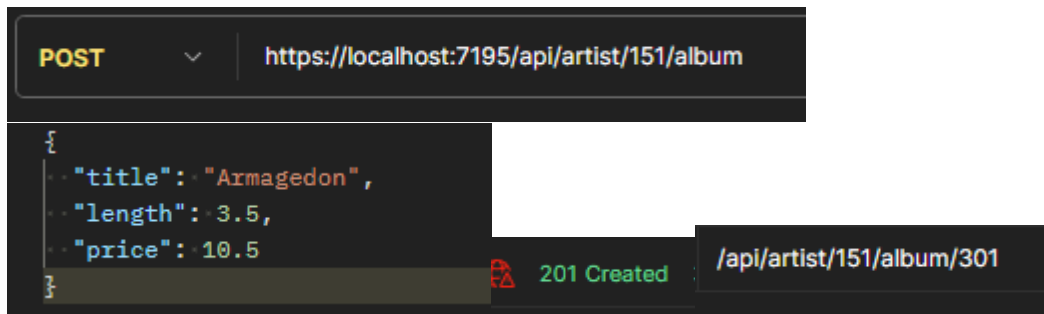
If there is an attempt to search for a record that does not exist in the database, the response is 404 Not Found, and info: **Artist 1002 is not found**.

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

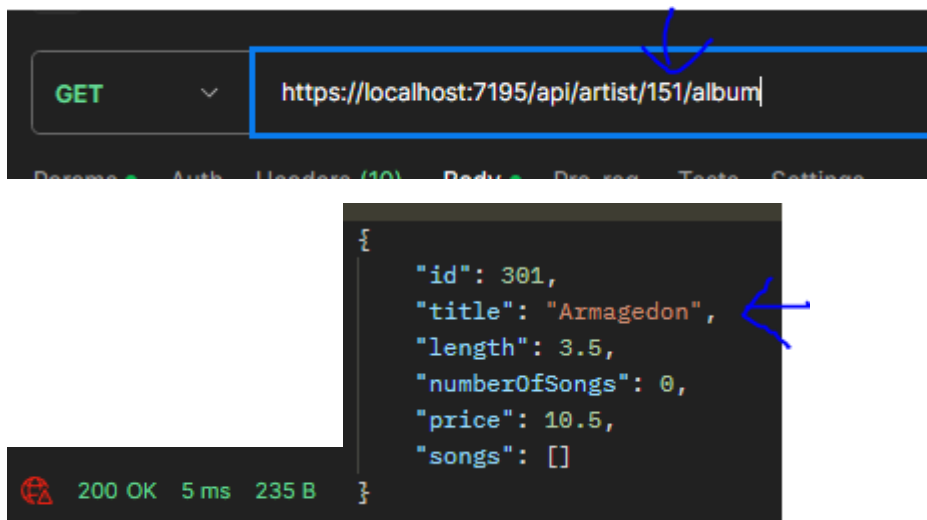
{
  "name": "OffSpring",
  "description": "description",
  "kindOfMusic": "Rock, metal",
  "contactEmail": "offspring@example.com",
  "contactNumber": "565485222",
  "country": "USA",
  "city": "Miami"
}
```

```
409 Conflict 11 ms Name : invalid value because there is already an artist
created by this user (duplicate)
```

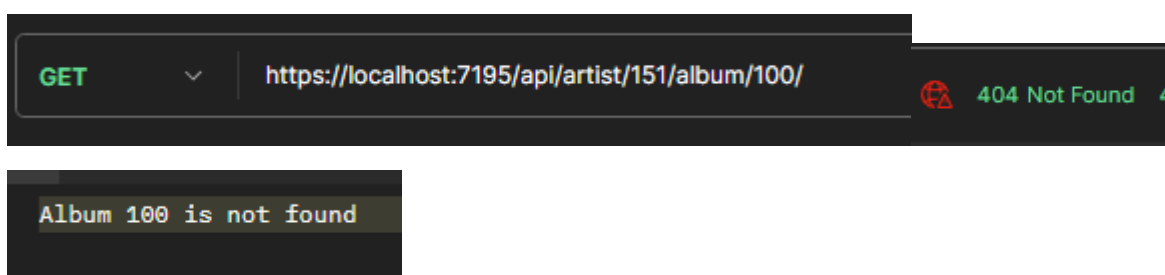
If, when attempting to create a new Artist, but with the same name that appears in a database that has already been created by this user , a **409 Conflict** error pops up that there is an invalid Name value.



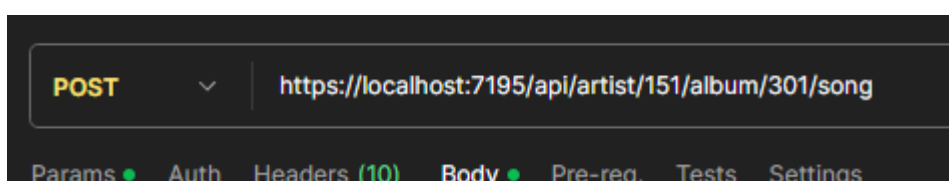
A new album is then created, the feedback **201 Created** and the path under which the new album was created **api/artist/151/album/301**.



Later, the possibility to search for all albums by artist id number or only a particular album by album id number -> **api/artist/151/album/301**



If there is an attempt to search for a non-existent album , the feedback is 404 Not Found and info: **Album 100 is not found**.



```
{
  "name": "It's just Christmas"
}
```

201 Created 27 m

/api/artist/151/album/301/song/901

Then you create new songs, when everything is ok, you get a response **201 Created** and the path under which the new song was created -> **api/artist/151/album/301/song/901**

```
{
  "name": ""
}
```

```
errors": {
  "Name": [
    "The Name field is required."
  ]
}
```

If there is an attempt to enter an empty Name, an error will occur stating that the field cannot be empty.

GET https://localhost:7195/api/artist/151/album/301/song/901

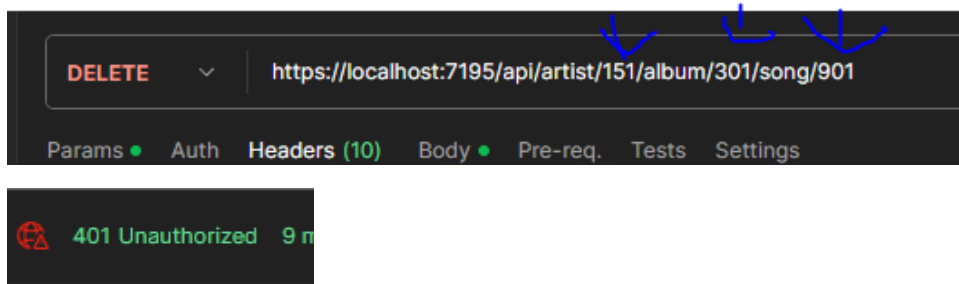
Params • Auth Headers (10) Body • Pre-req. Tests Settings 200 OK 4

```
1 {
2   "id": 901,
3   "name": "It's just Christmas",
4   "albumId": 301
5 }
```

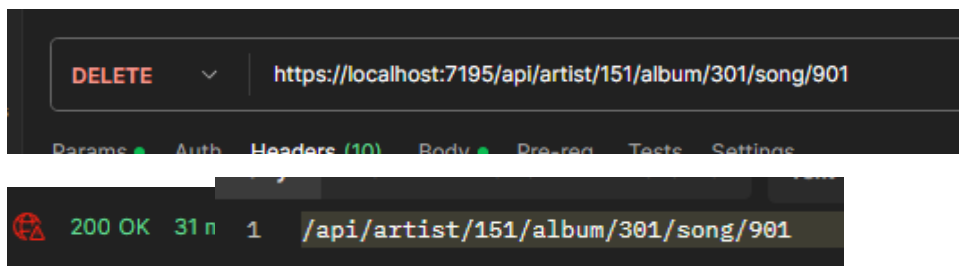
Then the option to display all songs or one song by id number.

GET https://localhost:7195/api/artist/151/album/301/song/90

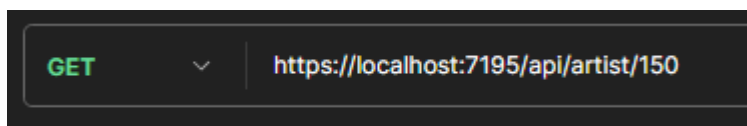
If an invalid song id is entered, it says **Song 90 is not found** and code **404 Not Found**.



Then you delete a song as a non-logged-in user, the message is **401 Unauthorized** ->. When trying to log in as a user with the role : **USER**, the feedback is **403 Forbidden**.



The user who created the song can only delete it or a user with the **ADMIN** role. When successful, the message **200 OK** and the track of the song from which it was deleted.



```

{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
      }
    ]
  }
]
}

```

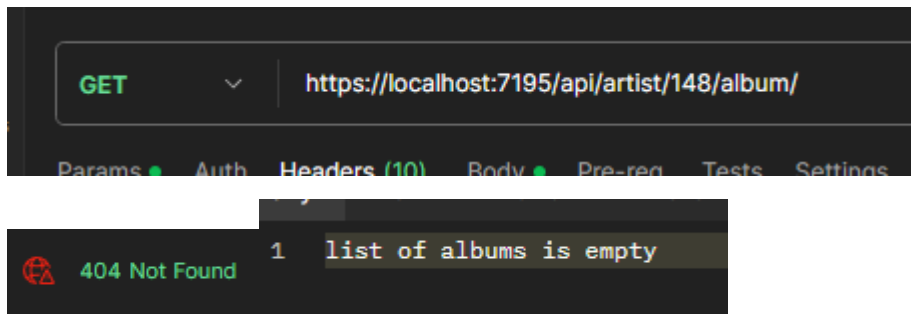
DELETE <https://localhost:7195/api/artist/150> 1 Artist 150 is not found

Now we will delete the whole Artist with albums and songs under id 150, then we check if it is , but we get the response **Artist 150 not found**.

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

404 Not Found 5 ms 1 list of songs is empty

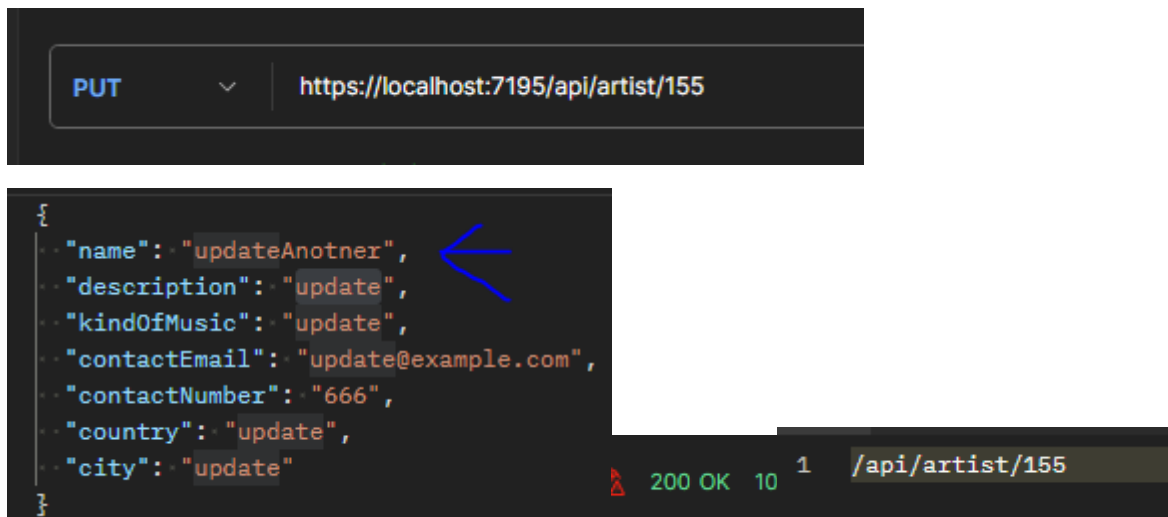
If you try to search for all the songs on an album and there are no songs, the answer is: **list of songs is empty**.



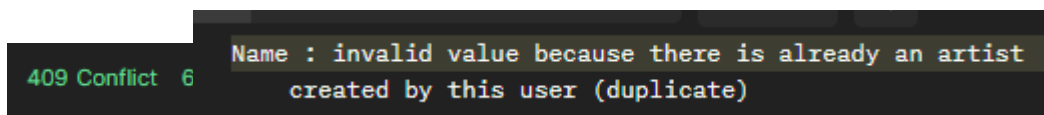
If we try to search for all albums from a given artist and there are no albums, the answer is: **list of albums is empty**.

Summary of how the authentication works based on the given USER role **Claim** contained in the **JWT Token**:

- If logged in user with **USER** role -> can only browse everything -> **GET** verb.
- If logged in user with the role **PREMIUM_USER** -> can view everything, but if he wants to delete something, update data, then only his artists, albums, songs.
- If logged in user with the role **ADMIN** -> can do everything.

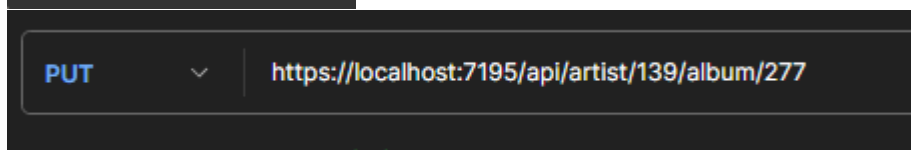


We will now proceed to the data update action -> PUT verb. You can change several values of a given artist: name, description, kindOfMusic, contactEmail, contactNumber, Country, City.



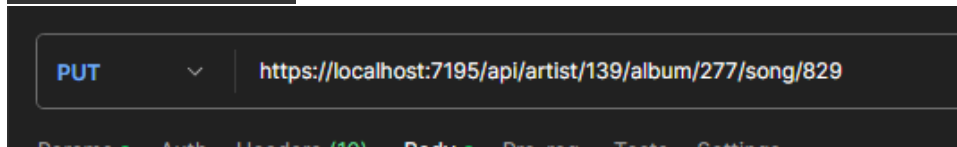
If there is an attempt to change the name of an artist, album or song to an existing name in the database which has already been created by this user, there will be an error -> **409 Conflict**, with info: Name : **invalid value because there is already an artist created by this user (duplicate)**.


```
{
  "title": "string",
  "length": 0,
  "price": 0
}
```

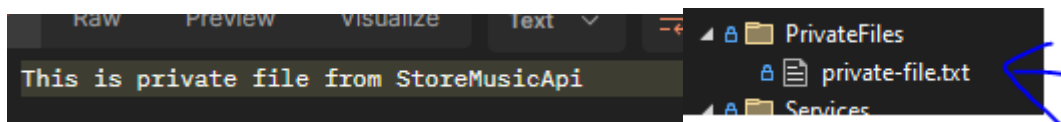
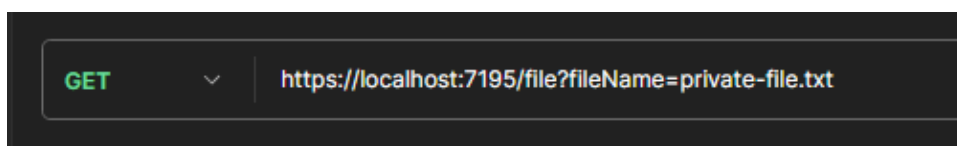


Here, the data of the respective album is updated , the 3 values **title**, **length** (total length of the album), **price** can be changed.

```
{
  "name": "string"
}
```



Here you can update the data of a song , you can change the name value.



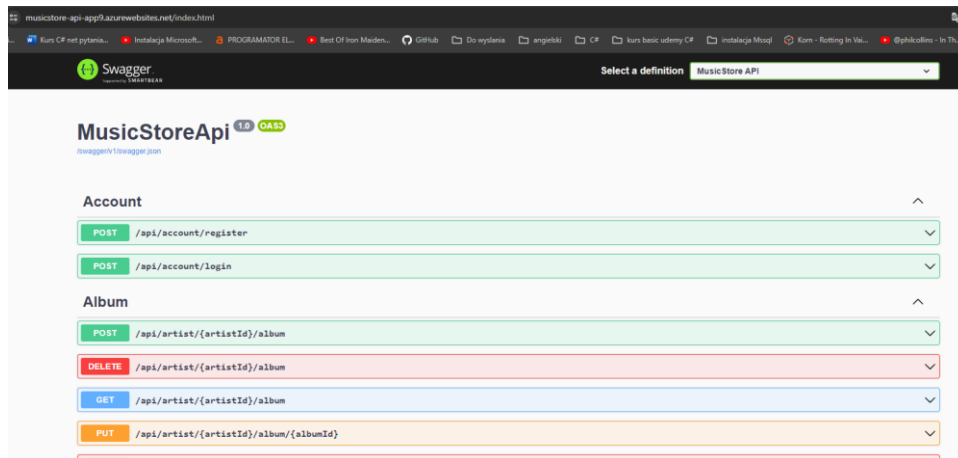
Afterwards, we also have the possibility of accessing a text file that is located on the server called : **private-file.txt**. Only logged-in users have access.

```
<!-- the targets to write to -->
<targets>
  <!-- write logs to file -->
  <target xsi:type="File" name="request-time" fileName="Environment.GetFolderPath(Environment.SpecialFolder.CommonDocuments)
    layout="{0:longdate}|{event-properties:item=EventId_Id}|{uppercase:${level}}|{logger}|{message} ${ex
```

There is still the NLogger library used, for creating logs in the **MusicStoreAPILogs** directory, which is configured in the nlog.config file . It is best to change the path yourself afterwards, where you want to have the logs -> **all**, **exceptions**, **request-time** -> occurs when the request to the server takes more than 4 seconds.

Use of **middleware** as **ErrorHandlingMiddleware** class, which is responsible for analysing queries that come into the API and when an exception occurs, adds information about it to the exceptions logger.

CORS policy added, so that in future queries can be sent to a particular api, from another frontend domain.



Azure cloud application deployed : <https://musicstore-api-app9.azurewebsites.net/swagger>

Possible actions to be performed in a given music shop:

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}