МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование в среде dotNET»

Тема: «РАЗРАБОТКА ПРОГРАММНОГО ИНТЕРФЕЙСА ДЛЯ ВЗАИМОДЕЙСТВИЯ С ПРИЛОЖЕНИЕМ»

Студент гр. 6305 _____ Белоусов Е.О.

Преподаватель _____ Пешехонов К.А.

Санкт-Петербург

2020

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ И ЗАДАНИЕ

Цель работы: реализовать программный интерфейс для взаимодействия с приложением.

Постановка задания:

1. Ознакомиться с принципом работы Swagger и AutoMapper.
2. Реализовать интерфейс для взаимодействия с приложением.

ХОД РАБОТЫ

Swagger

Для того чтобы взаимодействовать с разработанным приложением, подключим Swagger – фреймворк для разработки и документации REST веб-сервисов. В частности, нам необходим такой инструмент как Swagger UI для визуализации нашего приложения. Добавим на слой представления данных (в нашем случае, API-слой) пакет Swashbuckle.AspNetCore последней версии. Чтобы сконфигурировать Swagger добавим следующий код в файл `Startup.cs` в метод `ConfigureServices()`:

```
            services.AddSwaggerGen(options =>
            {
                options.SwaggerDoc("v1", new OpenApiInfo { Title = "Music Applicati
on", Version = "v.1.0" });
            });
```

В метод `Configure()` добавим следующие строки:

```
            app.UseSwagger();
            app.UseSwaggerUI(c =>
            {
                c.RoutePrefix = "";
                c.SwaggerEndpoint("/swagger/v1/swagger.json", "Music Application v.
1.0");
            });
```

Контроллеры и AutoMapper

Создадим контроллеры `MusicsController.cs` и `ArtistsController.cs`. В контроллерах будем реализовывать Get (получение всех существующих объектов и получение отдельного объекта), Post (добавление нового объекта), Put (обновление информации отдельного объекта) и Delete (удаление существующего объекта) методы.

Создадим классы-ресурсы `MusicResource.cs` и `ArtistResource.cs`, которые имеют те же свойства, что и у доменных моделей, созданных ранее при разработке, с отличием в том, что `ArtistResource.cs` не будет ссылаться на `MusicResource.cs`.

Содержание `MusicResource.cs`:

```
namespace MusicApp.API.Resources
```

```
{
    public class MusicResource
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public ArtistResource Artist { get; set; }
    }
}
```

Содержание `ArtistResource.cs`:

```
namespace MusicApp.API.Resources
{
    public class ArtistResource
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

Также опишем классы-ресурсы для того, чтобы не запрашивать ненужные данные.

Содержание `SaveMusicResource.cs`:

```
namespace MusicApp.API.Resources
{
    public class SaveMusicResource
    {
        public string Name { get; set; }
        public int ArtistId { get; set; }
    }
}
```

Содержание `SaveArtistResource.cs`:

```
namespace MusicApp.API.Resources
{
    public class SaveArtistResource
    {
        public string Name { get; set; }
    }
}
```

Определим взаимное соответствие данных между доменными моделями и ресурсами (маппинг) с помощью такого инструмента как AutoMapper. Для этого создадим отдельный класс `MappingProfile.cs`:

```
using AutoMapper;
```

```
using MusicApp.API.Resources;
using MusicApp.Core.Models;

namespace MusicApp.API.Mapping
{
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            CreateMap<Music, MusicResource>().ReverseMap();
            CreateMap<Artist, ArtistResource>().ReverseMap();

            CreateMap<SaveMusicResource, Music>();
            CreateMap<SaveArtistResource, Artist>();
        }
    }
}
```

Для того чтобы быть уверенным в корректности данных, получаемых из запросов, опишем валидаторы при помощи популярной библиотеки FluentValidation.

Содержание `SaveMusicResourceValidator.cs`:

```
using FluentValidation;
using MusicApp.API.Resources;

namespace MusicApp.API.Validators
{
    public class SaveMusicResourceValidator : AbstractValidator<SaveMusicResource>
    {
        public SaveMusicResourceValidator()
        {
            const int maxLength = 50;
            const string errorMsg = "'Artist Id' must be greater than 0.";

            RuleFor(m => m.Name).NotEmpty().MaximumLength(maxLength);

            RuleFor(m => m.ArtistId).NotEmpty().WithMessage(errorMsg);
        }
    }
}
```

Содержание `SaveArtistResourceValidator.cs`:

```
using FluentValidation;
using MusicApp.API.Resources;

namespace MusicApp.API.Validators
{
```

```
    public class SaveArtistResourceValidator : AbstractValidator<SaveArtistResource
>
    {
        public SaveArtistResourceValidator()
        {
            const int maxLength = 50;

            RuleFor(a => a.Name).NotEmpty().MaximumLength(maxLength);
        }
    }
}
```

По итогу, контроллеры описаны следующим образом.

Содержание `MusicsController.cs`:

```
using System.Collections.Generic;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using MusicApp.API.Resources;
using MusicApp.API.Validators;
using MusicApp.Core.Models;
using MusicApp.Core.Services;

namespace MusicApp.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MusicsController : ControllerBase
    {
        private readonly IMusicService _musicService;
        private readonly IMapper _mapper;

        public MusicsController(IMusicService musicService, IMapper mapper)
        {
            _mapper = mapper;
            _musicService = musicService;
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<MusicResource>> GetMusicById(int id)
        {
            var music = await _musicService.GetMusicById(id);
            var musicResource = _mapper.Map<Music, MusicResource>(music);

            return Ok(musicResource);
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<MusicResource>>> GetAllMusics()
```

```
        {
            var musics = await _musicService.GetAllWithArtist();
            var musicResources = _mapper.Map<IEnumerable<Music>, IEnumerable<MusicR
esource>>(musics);

            return Ok(musicResources);
        }

        [HttpPost]
        public async Task<ActionResult<MusicResource>> CreateMusic([FromBody] SaveM
usicResource saveMusicResource)
        {
            var validator = new SaveMusicResourceValidator();
            var validationResult = await validator.ValidateAsync(saveMusicResource)
;

            if (!validationResult.IsValid)
                return BadRequest(validationResult.Errors);

            var musicToCreate = _mapper.Map<SaveMusicResource, Music>(saveMusicReso
urce);
            var newMusic = await _musicService.CreateMusic(musicToCreate);
            var musicResource = _mapper.Map<Music, MusicResource>(newMusic);

            return Ok(musicResource);
        }

        [HttpPut("{id}")]
        public async Task<ActionResult<MusicResource>> UpdateMusic(int id, [FromBod
y] SaveMusicResource saveMusicResource)
        {
            var validator = new SaveMusicResourceValidator();
            var validationResult = await validator.ValidateAsync(saveMusicResource)
;

            var requestIsInvalid = id == 0 || !validationResult.IsValid;
            if (requestIsInvalid)
                return BadRequest(validationResult.Errors);

            var music = _mapper.Map<SaveMusicResource, Music>(saveMusicResource);

            await _musicService.UpdateMusic(id, music);

            var updatedMusic = await _musicService.GetMusicById(id);
            var updatedMusicResource = _mapper.Map<Music, MusicResource>(updatedMus
ic);

            return Ok(updatedMusicResource);
        }
```

```
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteMusic(int id)
        {
            if (id == 0)
                return BadRequest();

            var music = await _musicService.GetMusicById(id);
            if (music == null)
                return NotFound();

            await _musicService.DeleteMusic(music);

            return NoContent();
        }
    }
}
```

Содержание `ArtistsController.cs`:

```
using System.Collections.Generic;
using System.Threading.Tasks;
using AutoMapper;
using FluentValidation;
using Microsoft.AspNetCore.Mvc;
using MusicApp.API.Resources;
using MusicApp.Core.Models;
using MusicApp.Core.Services;

namespace MusicApp.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ArtistsController : ControllerBase
    {
        private readonly IArtistService _artistService;
        private readonly IMapper _mapper;
        private readonly AbstractValidator<SaveArtistResource> _validator;

        public ArtistsController(IArtistService artistService, IMapper mapper, Abst
ractValidator<SaveArtistResource> validator)
        {
            _mapper = mapper;
            _artistService = artistService;
            _validator = validator;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<ArtistResource>>> GetAllArtists(
)
        {
```

```
            var artists = await _artistService.GetAllArtists();
            var artistResources = _mapper.Map<IEnumerable<Artist>, IEnumerable<Arti
stResource>>(artists);

            return Ok(artistResources);
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<ArtistResource>> GetArtistById(int id)
        {
            var artist = await _artistService.GetArtistById(id);
            var artistResource = _mapper.Map<Artist, ArtistResource>(artist);

            return Ok(artistResource);
        }

        [HttpPost]
        public async Task<ActionResult<ArtistResource>> CreateArtist([FromBody] Sav
eArtistResource saveArtistResource)
        {
            var validationResult = await _validator.ValidateAsync(saveArtistResourc
e);
            if (!validationResult.IsValid)
                return BadRequest(validationResult.Errors);

            var artistToCreate = _mapper.Map<SaveArtistResource, Artist>(saveArtist
Resource);
            var newArtist = await _artistService.CreateArtist(artistToCreate);
            var artistResource = _mapper.Map<Artist, ArtistResource>(newArtist);

            return Ok(artistResource);
        }

        [HttpPut("{id}")]
        public async Task<ActionResult<ArtistResource>> UpdateArtist(int id, [FromB
ody] SaveArtistResource saveArtistResource)
        {
            var validationResult = await _validator.ValidateAsync(saveArtistResourc
e);
            if (!validationResult.IsValid)
                return BadRequest(validationResult.Errors);

            var artist = _mapper.Map<SaveArtistResource, Artist>(saveArtistResource
);

            await _artistService.UpdateArtist(id, artist);

            var updatedArtist = await _artistService.GetArtistById(id);
            var updatedArtistResource = _mapper.Map<Artist, ArtistResource>(updated
Artist);
```

```
            return Ok(updatedArtistResource);
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteArtist(int id)
        {
            var artist = await _artistService.GetArtistById(id);

            await _artistService.DeleteArtist(artist);

            return NoContent();
        }
    }
}
```

Проверка работы приложения

В процессе разработки работа приложения осуществлялась при помощи ПО Postman. Теперь проверим работоспособность с помощью подключенного Swagger UI. Для этого запустим приложение и автоматически перейдём по адресу `https://localhost:5001`.

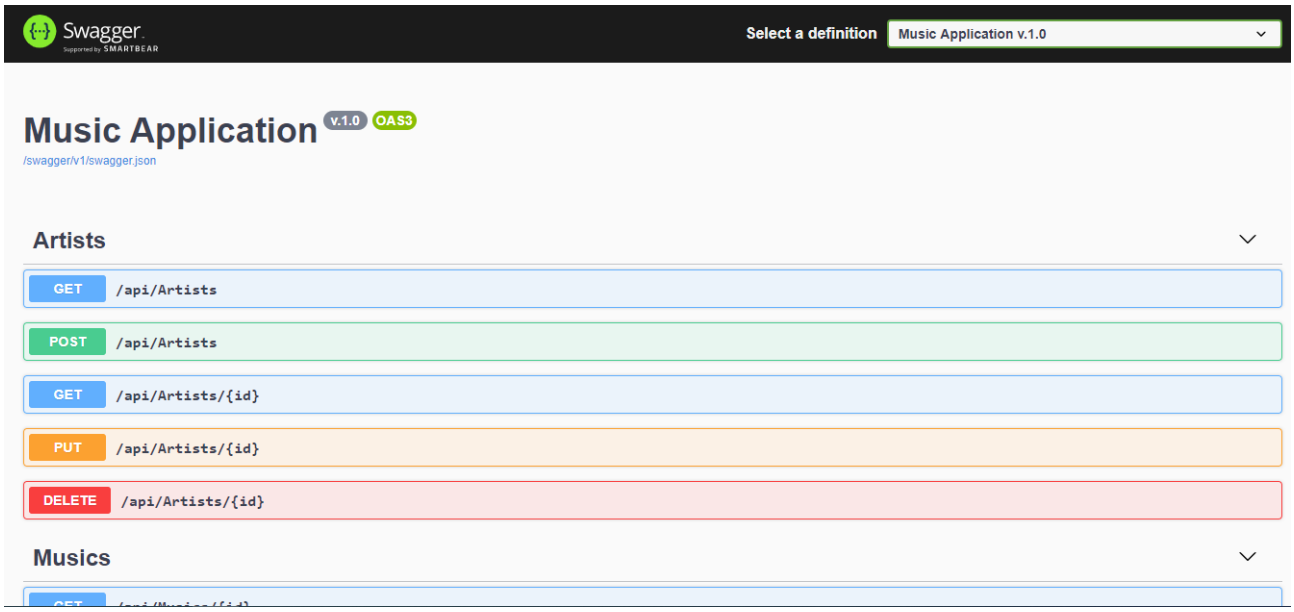

Рисунок 1. Приложение с Swagger UI

Request URL

```
https://localhost:5001/api/Musics
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 2,
    "name": "Tranquility Base Hotel & Casino",
    "artist": {
      "id": 1,
      "name": "Arctic Monkeys"
    }
  },
  {
    "id": 3,
    "name": "Why Did You Only Call Me When You Are High?",
    "artist": {
      "id": 1,
      "name": "Arctic Monkeys"
    }
  },
  {
    "id": 4,
    "name": "505",
    "artist": {
      "id": 1,
      "name": "Arctic Monkeys"
    }
  },
  {
    "id": 5,
    "name": "Crying Lightning",
```

Response headers

```
content-type: application/json; charset=utf-8
```

Рисунок 2. GET-запрос на получение списка всех композиций

Request URL

```
https://localhost:5001/api/Musics/15
```

Server response

Code	Details
200	Response body

```
{
  "id": 15,
  "name": "Break Stuff",
  "artist": {
    "id": 3,
    "name": "Limp Bizkit"
  }
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Wed, 15 Apr 2020 21:56:52 GMT
server: Kestrel
status: 200
```

Рисунок 3. GET-запрос на получение композиции с конкретным идентификатором

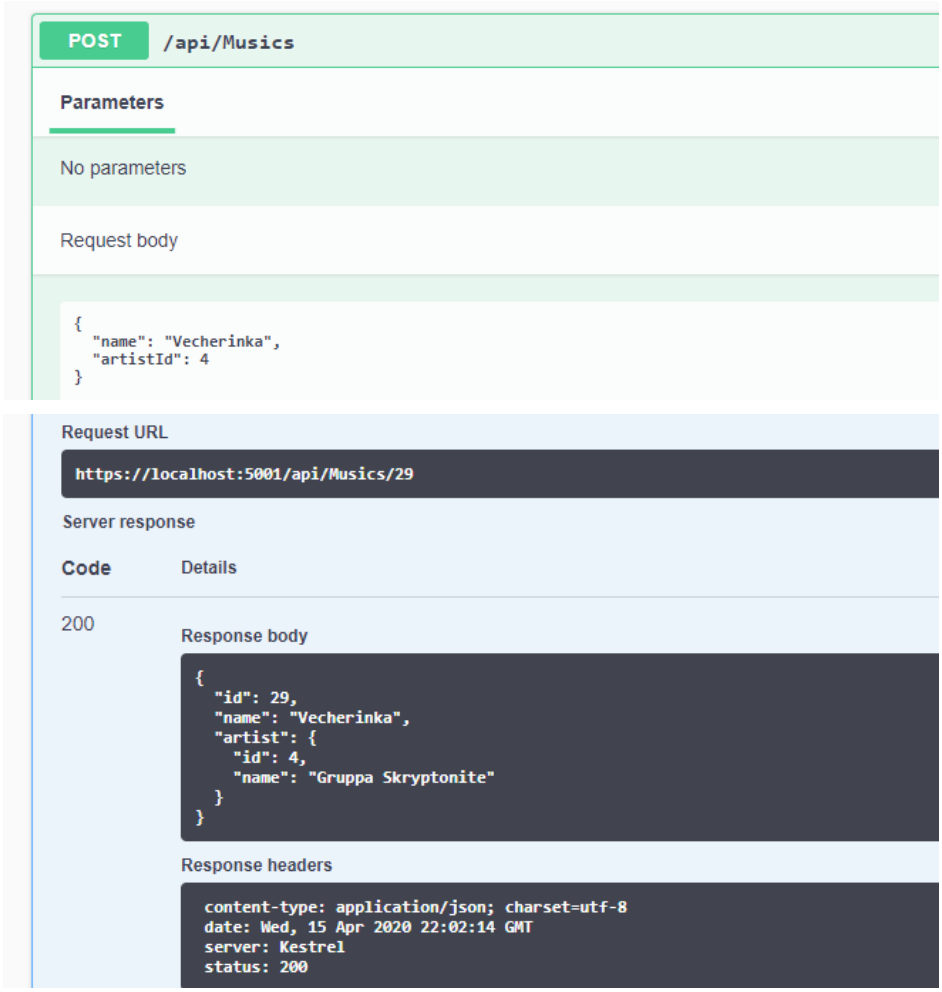

Рисунок 4. POST-запрос на создание композиции и проверка при помощи GET-запроса новой композиции по её идентификатору

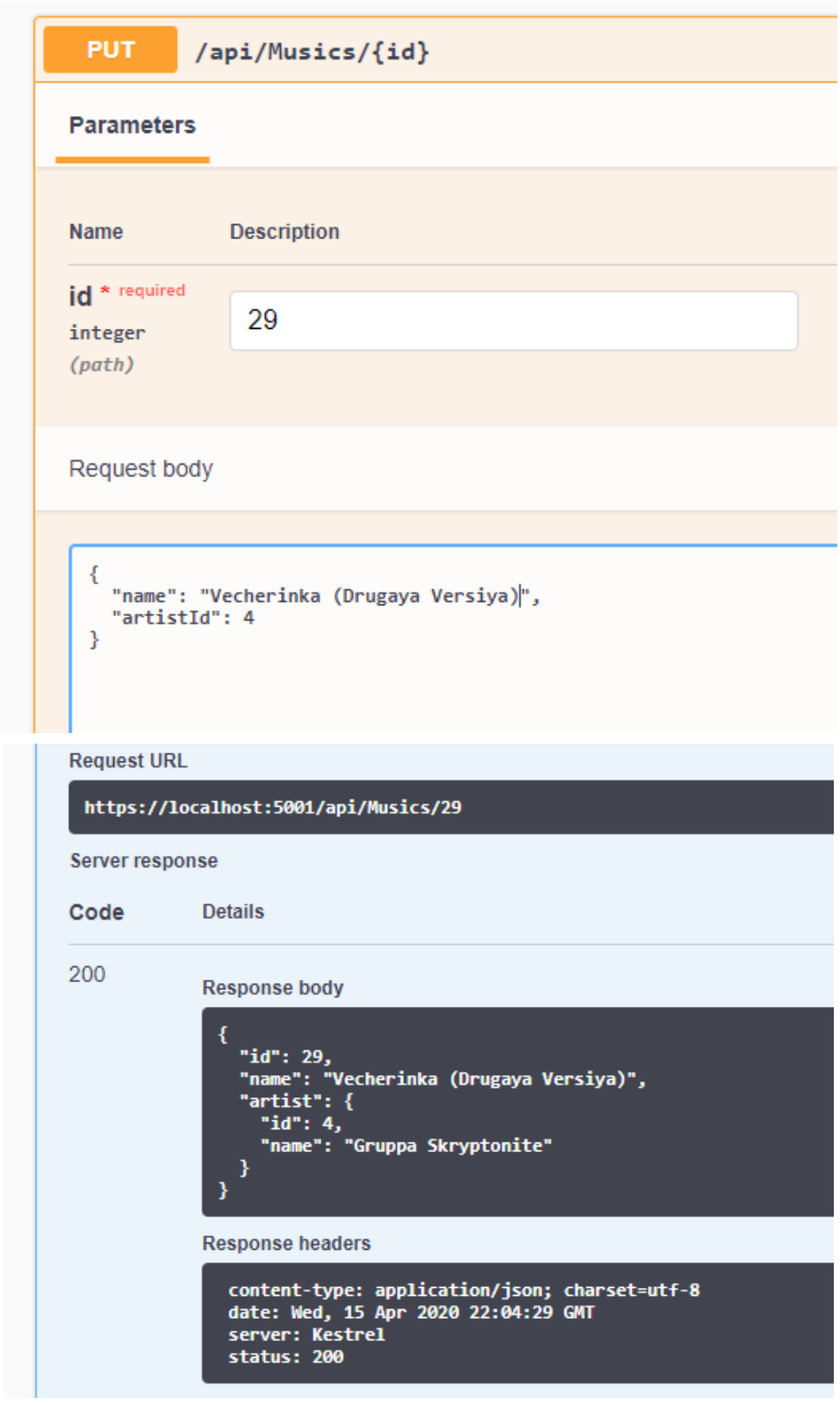

Рисунок 5. PUT-запрос на обновление информации о композиции с конкретным идентификатором и проверка при помощи GET-запроса данной композиции

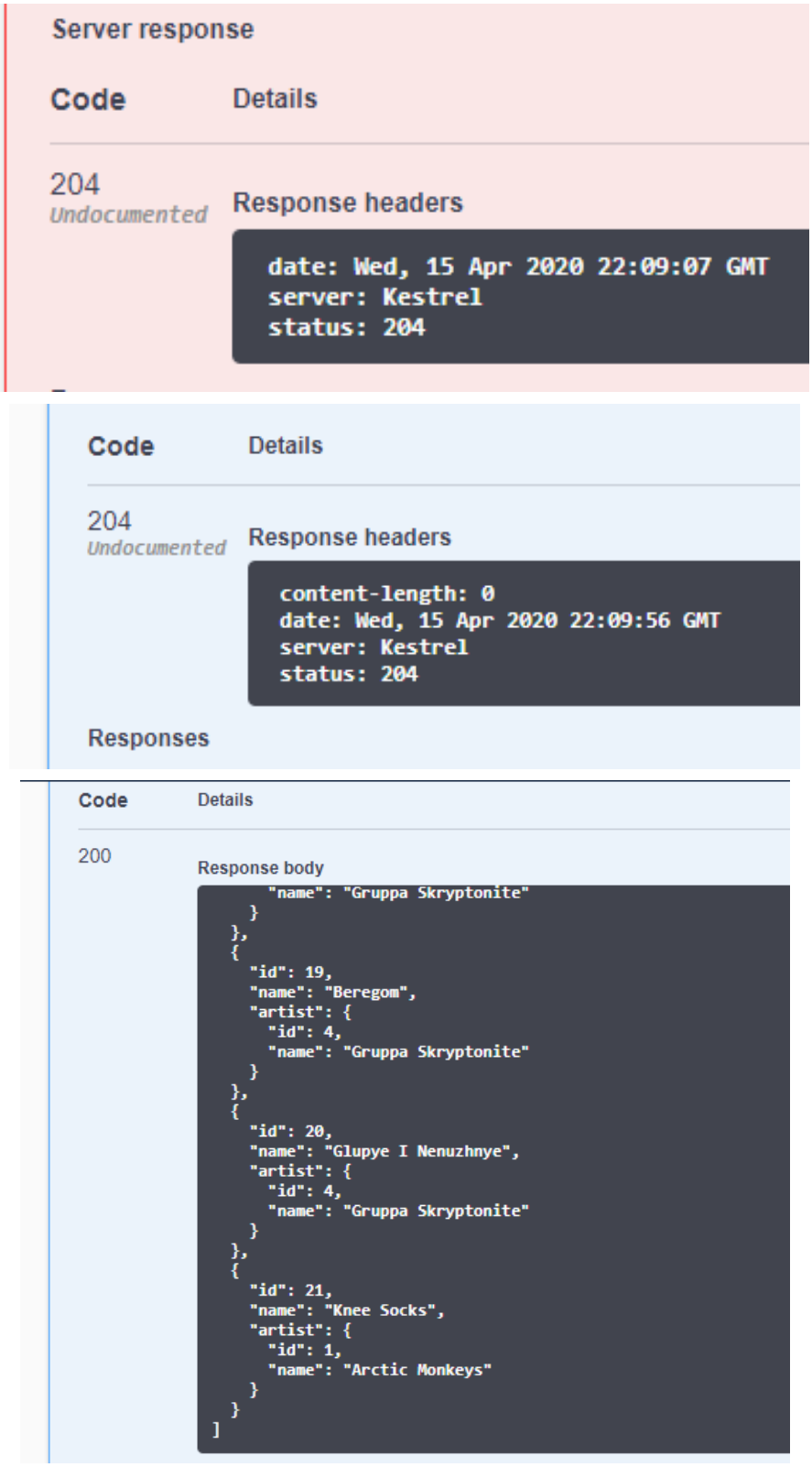

Рисунок 6. DELETE-запрос на удаление композиции с конкретным идентификатором и проверка при помощи GET-запроса для удалённой композиции и для всех композиций

Аналогично работают запросы для контроллера артистов.

ВЫВОДЫ

В результате выполнения данной лабораторной работы была закончена разработка ASP.NET Core 3.1 WebAPI приложения. Были получены знания и навыки проектирования RESTful веб-приложений с многослойной архитектурой, работы с ORM EntityFramework Code First, маппинга различных объектов, валидации данных с помощью FluentValidation, использования Postman для проверки работы запросов, использования Swagger для простой визуализации интерфейса взаимодействия с приложением.