МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

ОТЧЁТ

по лабораторной работе №4

по дисциплине «Программирование в среде dotNET»

Тема: “Разработка программного интерфейса для взаимодействия с приложением”

Студент гр. 6305 _____ Силинский А.С.

Преподаватель _____ Пешехонов К.А.

Санкт-Петербург

2020

Оглавление

Цель работы

Целью данной лабораторной работы является разработка программного интерфейса для взаимодействия с приложением

Задание

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

Ход работы

Для удобства взаимодействия с приложением можно подключить библиотеку Swagger. Она обеспечивает интерфейс (по сути, обычная страница html) для работы с HTTP запросами. Благодаря ей отпадает необходимость в совершении запросов через Postman (и другие подобные сервисы).

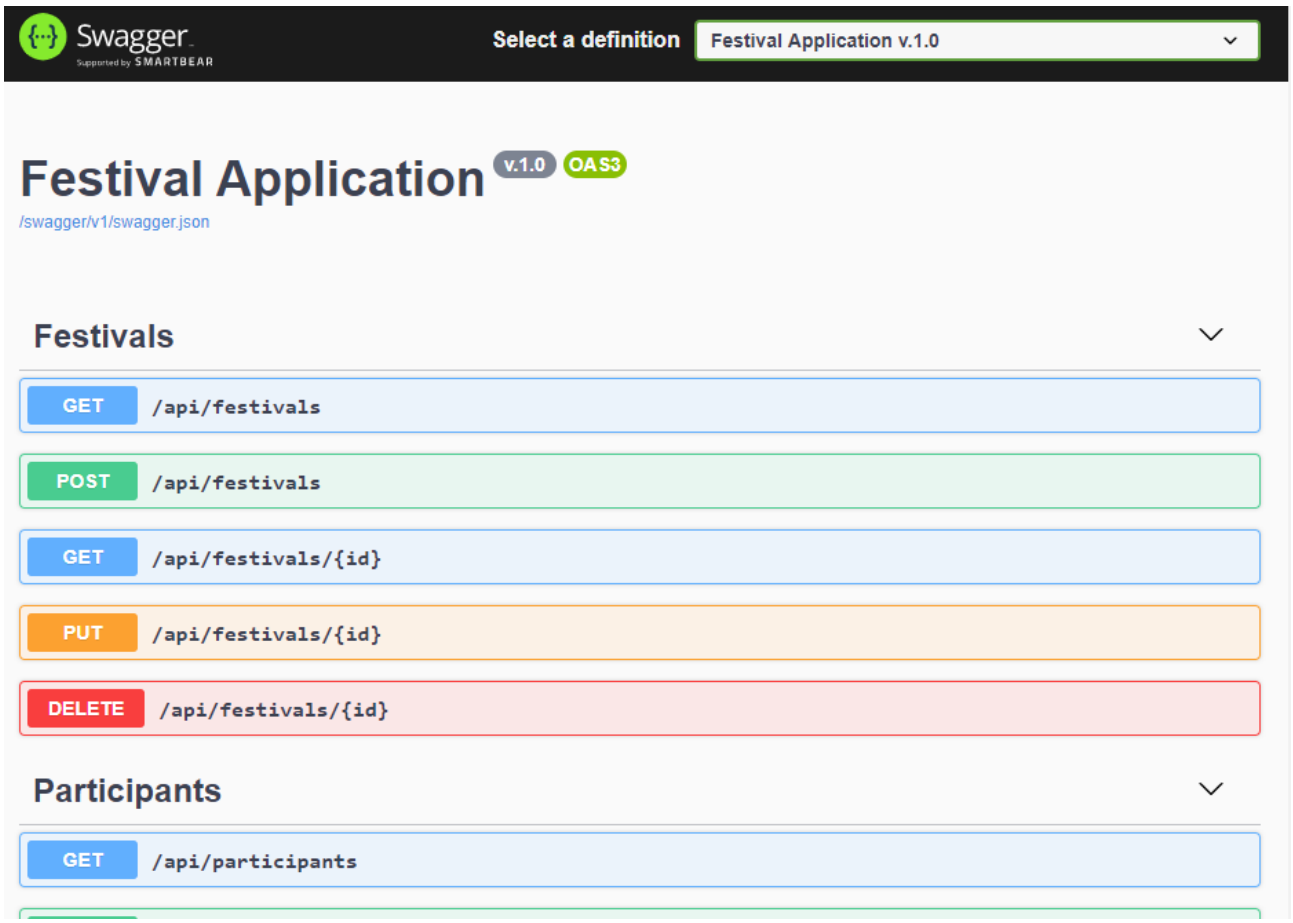

Рис. 1. Сгенерированный html-файл для работы с API.

При взаимодействии между слоями часто приходится “мапить” объекты, например, из сущности домена в модель вида и наоборот. Но часто эти модели совпадают (или отличаются на несколько полей), поэтому приходится писать много бойлерплейтного кода.

К счастью, существует библиотека AutoMapper, которая позволяет с легкостью создать мапперы и использовать их. Для работы библиотеки необходимо две вещи:

1. Написать файлы-конфигураторы (профайлы);
2. Внедрить зависимость через инъекцию в классе Startup.cs.

Код исходных файлов данного модуля можно посмотреть в приложении к данной лабораторной работе.

Выводы

В ходе выполнения данной лабораторной работы был разработан программный интерфейс для взаимодействия с приложением.

Приложение

FestivalController.cs

```
using System.Threading.Tasks;
using AutoMapper;
using FestivalWebApp.API.Models;
using FestivalWebApp.Core.Models;
using FestivalWebApp.Core.Services;
using FluentValidation;
using Microsoft.AspNetCore.Mvc;

namespace FestivalWebApp.API.Controllers
{
   [Route("api/festivals")]
   [ApiController]
   public class FestivalsController : ControllerBase
   {
      private readonly IValidator<FestivalCreateRequestBody> _createValidator;
      private readonly IMapper _mapper;
      private readonly IFestivalService _service;
      private readonly IValidator<FestivalUpdateRequestBody> _updateValidator;

      public FestivalsController(IMapper mapper, IFestivalService service,
         IValidator<FestivalCreateRequestBody> validator,
         IValidator<FestivalUpdateRequestBody> updateValidator)
      {
         _mapper = mapper;
         _service = service;
         _createValidator = validator;
         _updateValidator = updateValidator;
      }

      [HttpGet]
      public async Task<ActionResult> GetAllFestivals()
      {
         var festivals = await _service.GetAllFestivals();
         return Ok(festivals);
      }

      [HttpGet("{id}")]
      public async Task<ActionResult> GetFestivalById(int id)
```

```
        {
            var festival = await _service.GetFestivalById(id);
            return Ok(festival);
        }

        [HttpPost]
        public async Task<ActionResult> CreateFestival([FromBody]
FestivalCreateRequestBody createRequestBody)
        {
            var validationResult = await
_createValidator.ValidateAsync(createRequestBody);
            if (!validationResult.IsValid) return BadRequest(validationResult.Errors);

            var mappedValue = _mapper.Map<Festival>(createRequestBody);
            await _service.AddFestival(mappedValue);
            return Ok(mappedValue);
        }

        [HttpPut("{id}")]
        public async Task<ActionResult> UpdateFestival(int id,
            [FromBody] FestivalUpdateRequestBody updateRequestBody)
        {
            var validationResult = await
_updateValidator.ValidateAsync(updateRequestBody);
            if (!validationResult.IsValid) return BadRequest(validationResult.Errors);


            var festival = _mapper.Map<Festival>(updateRequestBody);
            if (id != festival.Id) return BadRequest();
            await _service.UpdateFestival(festival);
            return Ok(festival);
        }

        [HttpDelete("{id}")]
        public async Task<ActionResult> DeleteFestival(int id)
        {
            await _service.RemoveFestival(id);
            return NoContent();
        }
```

```
    }
}
```

ParticipantController.cs

```
using System.Threading.Tasks;
using AutoMapper;
using FestivalWebApp.API.Models;
using FestivalWebApp.Core.Models;
using FestivalWebApp.Core.Services;
using FluentValidation;
using Microsoft.AspNetCore.Mvc;

namespace FestivalWebApp.API.Controllers
{
    [Route("api/participants")]
    [ApiController]
    public class ParticipantsController : ControllerBase
    {
        private readonly IValidator<ParticipantCreateRequestBody> _createValidator;
        private readonly IMapper _mapper;
        private readonly IParticipantService _service;
        private readonly IValidator<ParticipantUpdateRequestBody> _updateValidator;

        public ParticipantsController(IMapper mapper, IParticipantService service,
            IValidator<ParticipantCreateRequestBody> validator,
            IValidator<ParticipantUpdateRequestBody> updateValidator)
        {
            _mapper = mapper;
            _service = service;
            _createValidator = validator;
            _updateValidator = updateValidator;
        }

        [HttpGet]
        public async Task<ActionResult> GetAllParticipants()
        {
            var participants = await _service.GetAllParticipants();
            return Ok(participants);
        }

        [HttpGet("{id}")]
```

```
    public async Task<ActionResult> GetParticipantById(int id)
    {
        var participant = await _service.GetParticipantById(id);
        return Ok(participant);
    }

    [HttpGet("festival/{id}")]
    public async Task<ActionResult> GetParticipantsByFestivalId(int id)
    {
        var participants = await _service.GetParticipantsByFestivalId(id);
        return Ok(participants);
    }

    [HttpPost]
    public async Task<ActionResult> CreateParticipant([FromBody]
ParticipantCreateRequestBody createRequestBody)
    {
        var validationResult = await
_createValidator.ValidateAsync(createRequestBody);
        if (!validationResult.IsValid) return BadRequest(validationResult.Errors);

        var mappedValue = _mapper.Map<Participant>(createRequestBody);
        await _service.AddParticipant(mappedValue);
        return Ok(mappedValue);
    }

    [HttpPut("{id}")]
    public async Task<ActionResult> UpdateParticipant(int id,
        [FromBody] ParticipantUpdateRequestBody updateRequestBody)
    {
        var validationResult = await
_updateValidator.ValidateAsync(updateRequestBody);
        if (!validationResult.IsValid) return BadRequest(validationResult.Errors);

        var participant = _mapper.Map<Participant>(updateRequestBody);
        if (id != participant.Id) return BadRequest();

        await _service.UpdateParticipant(participant);
        return Ok(participant);
    }
```

```
        [HttpDelete("{id}")]
        public async Task<ActionResult> DeleteParticipant(int id)
        {
            await _service.RemoveParticipant(id);
            return NoContent();
        }
    }
}
```

MappingProfile.cs

```
using AutoMapper;
using FestivalWebApp.API.Models;
using FestivalWebApp.Core.Models;

namespace FestivalWebApp.API.Mappings
{
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            CreateMap<FestivalUpdateRequestBody, Festival>();
            CreateMap<FestivalCreateRequestBody, Festival>();

            CreateMap<ParticipantUpdateRequestBody, Participant>();
            CreateMap<ParticipantCreateRequestBody, Participant>();
        }
    }
}
```

FestivalCreateRequestBody.cs

```
using System;

namespace FestivalWebApp.API.Models
{
    public class FestivalCreateRequestBody
    {
        public string Name { get; set; }
        public string Description { get; set; }
        public DateTime Date { get; set; }
    }
}
```

FestivalUpdateRequestBody.cs

```
using System;

namespace FestivalWebApp.API.Models
{
   public class FestivalUpdateRequestBody
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public string Description { get; set; }
      public DateTime Date { get; set; }
   }
}
```

ParticipantCreateRequestBody.cs

```
namespace FestivalWebApp.API.Models
{
   public class ParticipantCreateRequestBody
   {
      public string Name { get; set; }
      public string SecondName { get; set; }
      public int Age { get; set; }
      public int FestivalId { get; set; }
   }
}
```

ParticipantUpdateRequestBody.cs

```
namespace FestivalWebApp.API.Models
{
   public class ParticipantUpdateRequestBody
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public string SecondName { get; set; }
      public int Age { get; set; }
      public int FestivalId { get; set; }
   }
}
```

FestivalCreateValidator.cs

```
using FestivalWebApp.API.Models;
using FluentValidation;
```

```
namespace FestivalWebApp.API.Validators
{
   public class FestivalCreateValidator :
AbstractValidator<FestivalCreateRequestBody>
   {
      private const int FestivalNameMaxSize = 50;
      private const int FestivalDescriptionMaxSize = 200;

      public FestivalCreateValidator()
      {
         RuleFor(f => f.Name).MaximumLength(FestivalNameMaxSize);
         RuleFor(f => f.Description).MaximumLength(FestivalDescriptionMaxSize);
      }
   }
}
```

FestivalUpdateValidator.cs

```
using FestivalWebApp.API.Models;
using FluentValidation;

namespace FestivalWebApp.API.Validators
{
   public class FestivalUpdateValidator :
AbstractValidator<FestivalUpdateRequestBody>
   {
      private const int FestivalNameMaxSize = 50;
      private const int FestivalDescriptionMaxSize = 200;

      public FestivalUpdateValidator()
      {
         RuleFor(f => f.Name).MaximumLength(FestivalNameMaxSize);
         RuleFor(f => f.Description).MaximumLength(FestivalDescriptionMaxSize);
      }
   }
}
```

ParticipantCreateValidator.cs

```
using FestivalWebApp.API.Models;
using FluentValidation;

namespace FestivalWebApp.API.Validators
{
```

```
   public class ParticipantCreateValidator :
AbstractValidator<ParticipantCreateRequestBody>
   {
      private const int NameMaxSize = 50;
      private const int SecondNameMaxSize = 50;
      private const string ErrorMessage = "Id must be greater than 0.";

      public ParticipantCreateValidator()
      {
         RuleFor(p => p.FestivalId).GreaterThan(0).WithMessage(ErrorMessage);
         RuleFor(p => p.Name).MaximumLength(NameMaxSize);
         RuleFor(p => p.SecondName).MaximumLength(SecondNameMaxSize);
      }
   }
}
```

ParticipantUpdateValidator.cs

```
using FestivalWebApp.API.Models;
using FluentValidation;

namespace FestivalWebApp.API.Validators
{
   public class ParticipantUpdateValidator :
AbstractValidator<ParticipantUpdateRequestBody>
   {
      private const int NameMaxSize = 50;
      private const int SecondNameMaxSize = 50;
      private const string ErrorMessage = "Id must be greater than 0.";

      public ParticipantUpdateValidator()
      {
         RuleFor(p => p.FestivalId).GreaterThan(0).WithMessage(ErrorMessage);
         RuleFor(p => p.Name).MaximumLength(NameMaxSize);
         RuleFor(p => p.SecondName).MaximumLength(SecondNameMaxSize);
      }
   }
}
```

Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
```

```
namespace FestivalWebApp.API
{
   public class Program
   {
      public static void Main(string[] args)
      {
         CreateHostBuilder(args).Build().Run();
      }

      public static IHostBuilder CreateHostBuilder(string[] args)
      {
         return Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder => {
webBuilder.UseStartup<Startup>(); });
      }
   }
}
```

Startup.cs

```
using AutoMapper;
using FestivalWebApp.API.Mappings;
using FestivalWebApp.API.Models;
using FestivalWebApp.API.Validators;
using FestivalWebApp.BLL;
using FestivalWebApp.Core.Repositories;
using FestivalWebApp.Core.Services;
using FestivalWebApp.DAL;
using FestivalWebApp.DAL.Repositories;
using FluentValidation;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;

namespace FestivalWebApp.API
{
   public class Startup
   {
```

```
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the
container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers();

        services.AddScoped<IFestivalService, FestivalService>();
        services.AddScoped<IParticipantService, ParticipantService>();

        services.AddTransient<IFestivalRepository, FestivalRepository>();
        services.AddTransient<IParticipantRepository, ParticipantRepository>();

        services.AddTransient<IValidator<FestivalUpdateRequestBody>,
FestivalUpdateValidator>();
        services.AddTransient<IValidator<FestivalCreateRequestBody>,
FestivalCreateValidator>();

        services.AddTransient<IValidator<ParticipantUpdateRequestBody>,
ParticipantUpdateValidator>();
        services.AddTransient<IValidator<ParticipantCreateRequestBody>,
ParticipantCreateValidator>();

        services.AddDbContext<FestivalDatabaseContext>(options =>
            options.UseInMemoryDatabase("FestivalWebApp"));

        services.AddSwaggerGen(options =>
        {
            options.SwaggerDoc("v1",
                new OpenApiInfo {Title = "Festival Application", Version = "v.1.0"});
        });

        services.AddAutoMapper(typeof(MappingProfile));
    }
```

```
        // This method gets called by the runtime. Use this method to configure the
HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment()) app.UseDeveloperExceptionPage();

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

            app.UseSwagger();

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