**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

**отчет**

**по лабораторной работе №4**

**по дисциплине «Программирование в среде .Net»**

**Тема: РАЗРАБОТКА ПРОГРАММНОГО ИНТЕРФЕЙСА ДЛЯ ВЗАИМОДЕЙСТВИЯ С ПРИЛОЖЕНИЕМ**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6306 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Бохан Ю.А. |
|  |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пешехонов К. А. |

Санкт-Петербург

2020

**Цель работы**

Реализовать программный интерфейс для взаимодействия с приложением.

**Задание**

Реализовать WebAPI-слой для приложения (для проверки можно использовать SoapUI, Postman и т.д.)

**Код программы**

Примеры кода программы даны для сущности «гонщик», другая сущность («команда») реализованы похожим образом.

1. Пример объекта для передачи данных между слоями (RiderDTO)

namespace Formula1.Client.DTO.Read  
{  
 public class RiderDTO  
 {  
 //Идентификатор  
 public int Id { get; set; }  
   
 //Имя гощника  
 public string Name { get; set; }  
   
 //Национальность  
 public string Country { get; set; }  
   
 //Сезоны  
 public int Seasons { get; set; }  
   
 //Чемпион мира  
 public int Champion { get; set; }  
   
 //Гран-при(старты)  
 public int Starts { get; set; }  
   
 //Победы  
 public int Wins { get; set; }  
  
 //Поулы  
 public int Poles { get; set; }  
  
 public F1TeamDTO F1Team { get; set; }  
 }  
}

1. Пример модели DTO для запроса (RiderCreateDTO)

using System.ComponentModel.DataAnnotations;  
  
namespace Formula1.Client.Requests.Create  
{  
 public class RiderCreateDTO  
 {  
  
 //Имя гощника  
 [Required(ErrorMessage = "Name is required")]  
 public string Name { get; set; }  
   
 //Национальность  
 [Required(ErrorMessage = "Country is required")]  
 public string Country { get; set; }  
   
 //Сезоны  
 [Required(ErrorMessage = "Seasons is required")]  
 public int Seasons { get; set; }  
   
 //Чемпион мира  
 [Required(ErrorMessage = "Champion is required")]  
 public int Champion { get; set; }  
   
 //Гран-при(старты)  
 [Required(ErrorMessage = "Starts is required")]  
 public int Starts { get; set; }  
   
 //Победы  
 [Required(ErrorMessage = "Wins is required")]  
 public int Wins { get; set; }  
  
 //Поулы  
 [Required(ErrorMessage = "Poles is required")]  
 public int Poles { get; set; }  
  
 public int? F1TeamId { get; set; }  
 }  
}

1. Пример контроллера WebAPI (RiderController)

using System;  
using System.Collections.Generic;  
using System.Diagnostics;  
using System.Linq;  
using System.Threading.Tasks;  
using AutoMapper;  
using Formula1.BLL.Contracts;  
using Formula1.Client.DTO.Read;  
using Formula1.Client.Requests.Create;  
using Formula1.Client.Requests.Update;  
using Formula1.Domain.Models;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.Extensions.Logging;  
namespace Formula1.WebAPI.Controllers  
{  
 [ApiController]  
 [Route("api/rider")]  
 public class RiderController  
 {  
 private ILogger<RiderController> Logger { get; }  
 private IRiderCreateService RiderCreateService { get; }  
 private IRiderGetService RiderGetService { get; }  
 private IRiderUpdateService RiderUpdateService { get; }  
 private IMapper Mapper { get; }  
  
 public RiderController(ILogger<RiderController> logger, IMapper mapper, IRiderCreateService riderCreateService, IRiderGetService riderGetService, IRiderUpdateService riderUpdateService)  
 {  
 this.Logger = logger;  
 this.RiderCreateService = riderCreateService;  
 this.RiderGetService = riderGetService;  
 this.RiderUpdateService = riderUpdateService;  
 this.Mapper = mapper;  
 }  
  
 [HttpPut]  
 [Route("")]  
 public async Task<RiderDTO> PutAsync(RiderCreateDTO rider)  
 {  
 this.Logger.LogTrace($"{nameof(this.PutAsync)} called");  
  
 var result = await this.RiderCreateService.CreateAsync(this.Mapper.Map<RiderUpdateModel>(rider));  
  
 return this.Mapper.Map<RiderDTO>(result);  
 }  
  
 [HttpPatch]  
 [Route("")]  
 public async Task<RiderDTO> PatchAsync(RiderUpdateDTO rider)  
 {  
 this.Logger.LogTrace($"{nameof(this.PutAsync)} called");  
  
 var result = await this.RiderUpdateService.UpdateAsync(this.Mapper.Map<RiderUpdateModel>(rider));  
  
 return this.Mapper.Map<RiderDTO>(result);  
 }  
  
 [HttpGet]  
 [Route("")]  
 public async Task<IEnumerable<RiderDTO>> GetAsync()  
 {  
 this.Logger.LogTrace($"{nameof(this.GetAsync)} called");  
  
 return this.Mapper.Map<IEnumerable<RiderDTO>>(await this.RiderGetService.GetAsync());  
 }  
  
 [HttpGet]  
 [Route("{riderId}")]  
 public async Task<RiderDTO> GetAsync(int riderId)  
 {  
 this.Logger.LogTrace($"{nameof(this.GetAsync)} called for {riderId}");  
  
 return this.Mapper.Map<RiderDTO>(await this.RiderGetService.GetAsync(new RiderIdentityModel(riderId)));  
 }  
 }  
}

**Результаты работы приложения**

Приложение тестировалось с помощью Postman. Результаты работы некоторых запросов представлены на рисунках ниже.

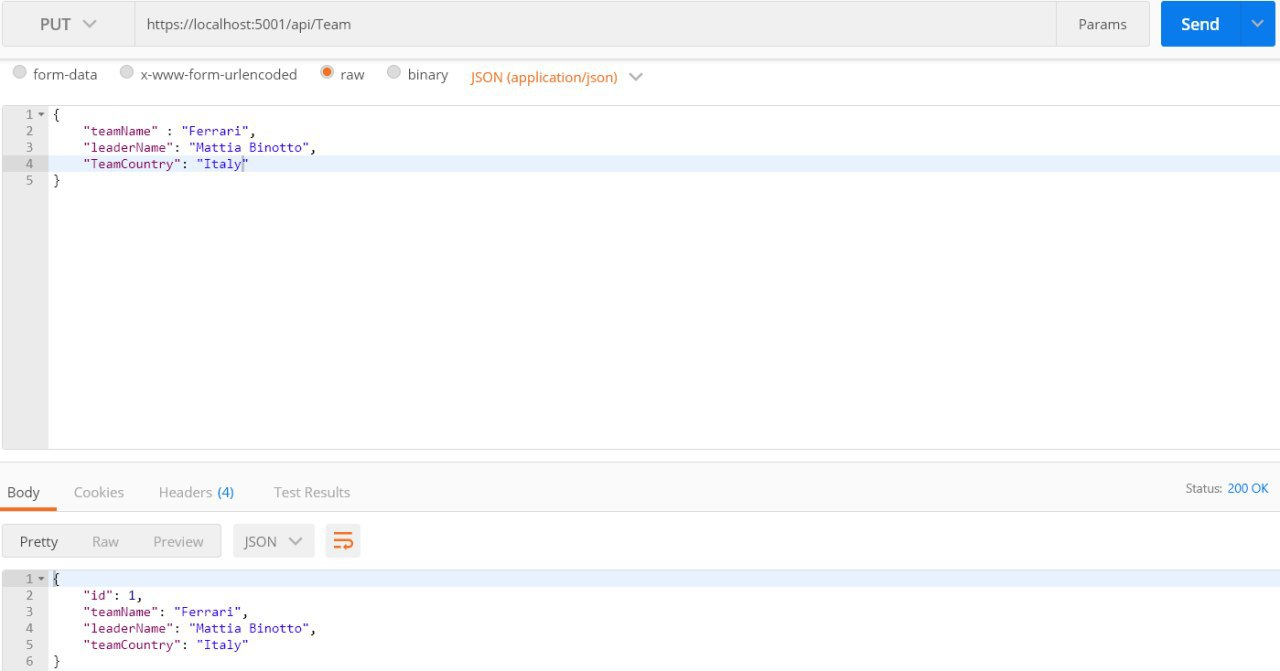


Рисунок 1. Put-запрос для добавления команды.

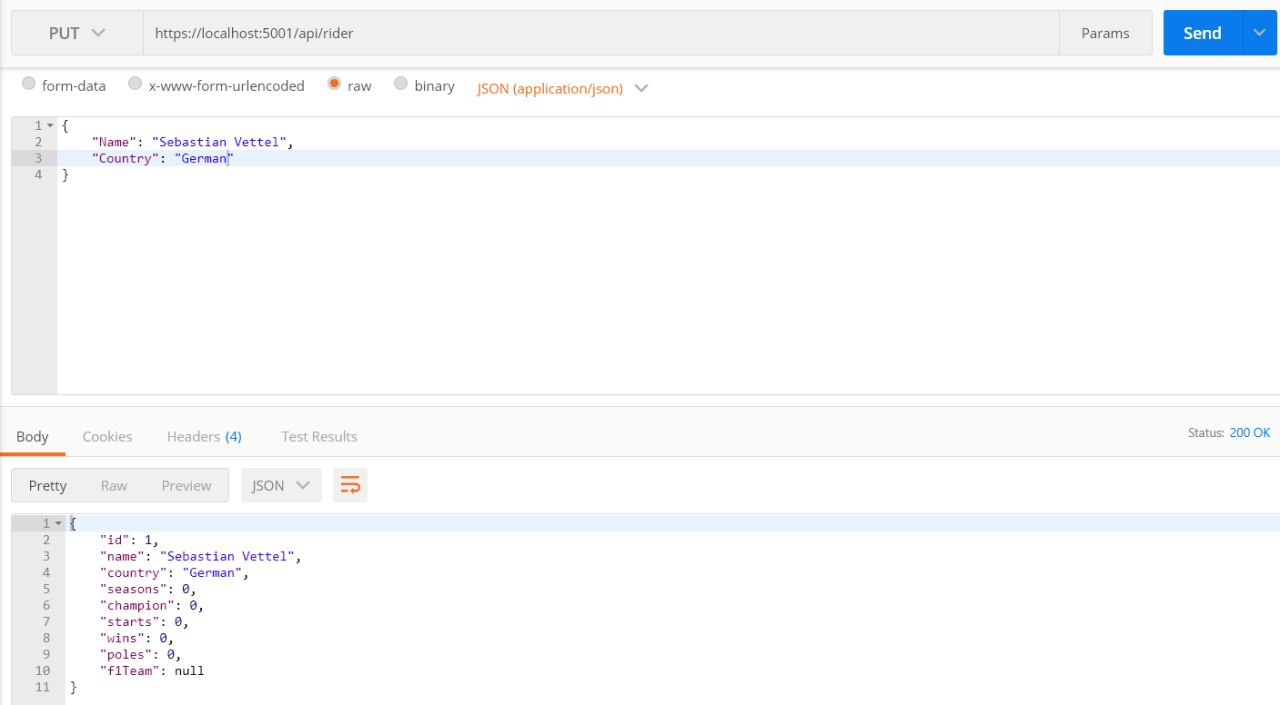


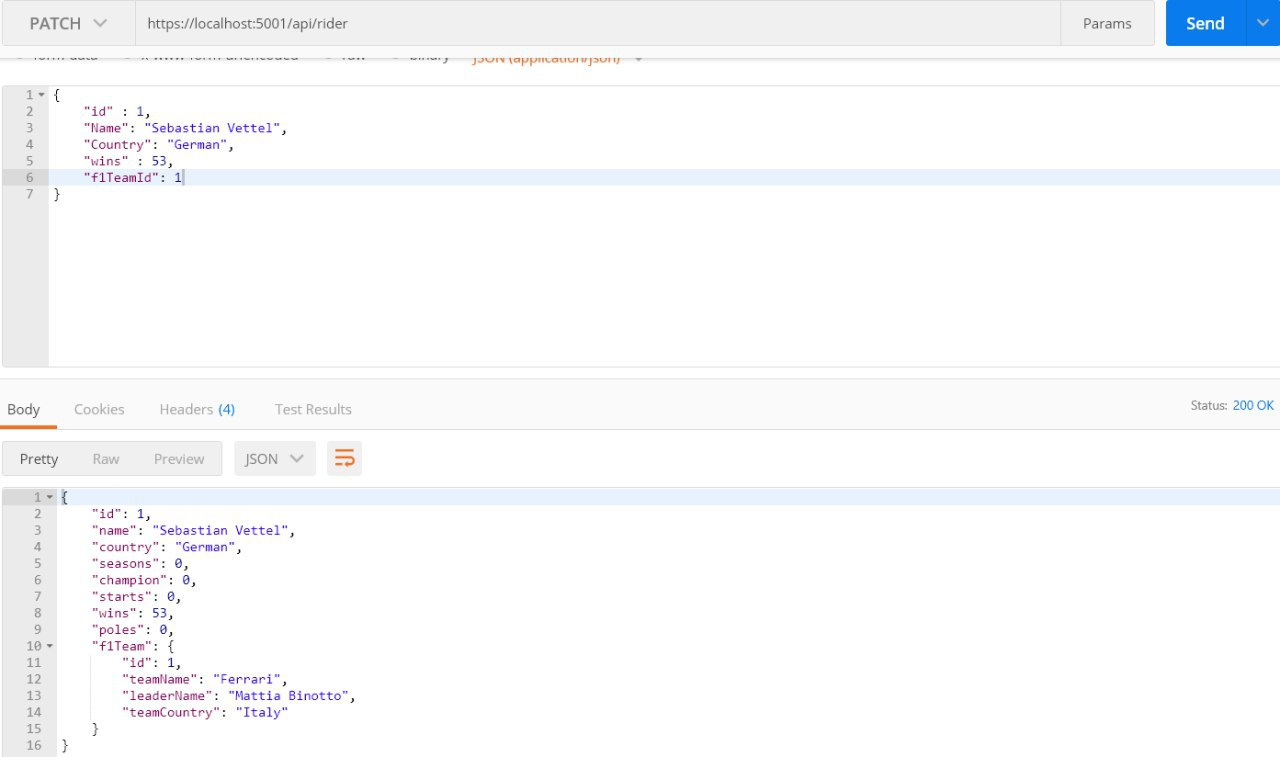
Рисунок 2. Put-запрос для добавления гонщика. 

Рисунок 3. Get-запрос для получения списка гонщиков

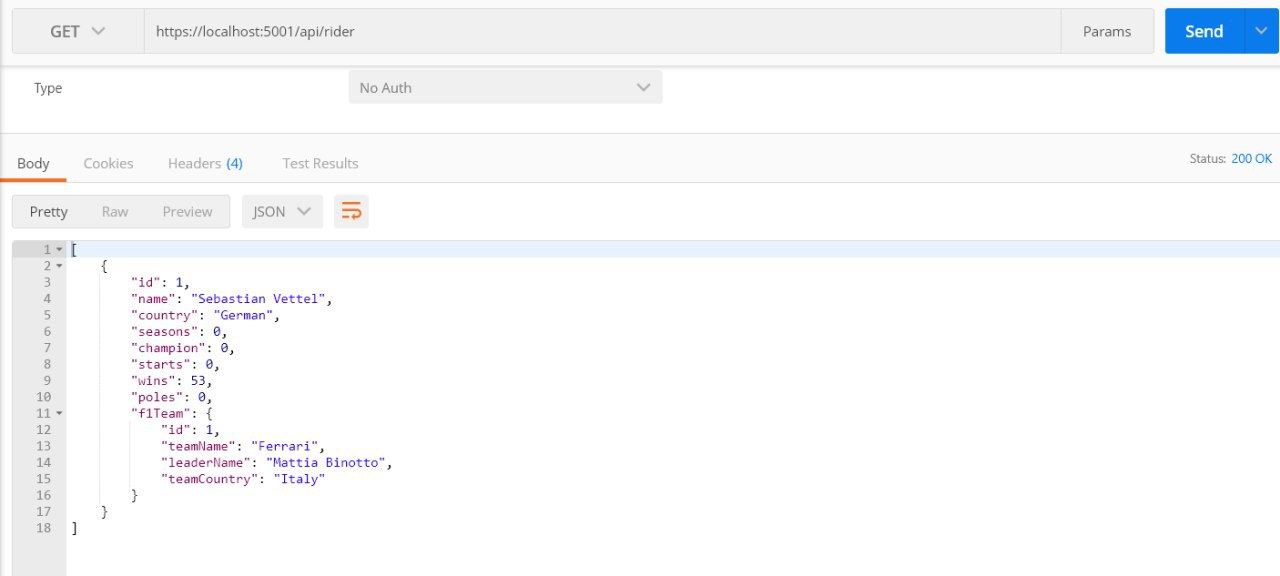


Рисунок 4. Patch-запрос для изменения данных о гонщике с id 1

**Выводы:**

В процессе выполнения лабораторной работы был реализован слой WebAPI приложения в среде .NET. Были получены навыки по работе с WebAPI и реализации передачи данных с помощью DTO.