**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ЛАБОРАТОРНАЯ РАБОТА №6**

по дисциплине: «Разработка приложений баз данных для информационных систем»

на тему: «Создание *Web*-сервиса с использованием *WEB API ASP.NET MVC* (*ASP.NET Core MVC*) для работы с информацией из базы данных»

Выполнил: студент гр. ИТП-31

Трацевский И. С.

Принял: доцент

Асенчик О. Д.

Гомель 2023

**Цель работы:** ознакомиться с возможностями .*NET Core* по получению, отображению и изменение данных при помощи технологии *WEB API*.

**Задание:**

Создать с использованием технологии WEB API Web-приложение, в котором организовать получение, отображение и изменение данных.

Для этого необходимо:

1. Создать (использовать ранее разработанные в предыдущих лабораторных работах):

a. Классы, моделирующие не менее чем три таблицы базы данных согласно вашему варианту. Одна из таблиц обязательно должна находиться на стороне отношения «многие» связи с другой таблицей в схеме базы данных.

b. Класс контекста данных для доступа к базе данных.

2. Создать контроллер для операций с данными таблицы, стоящей на стороне отношения «многие».

3. Задокументировать WEB API ASP.NET Core с помощью инструментов Swagger.

4. Клиентское приложение в виде HTML страницы для визуального выполнения всех операций с данными выбранной в п.2 таблицы, использующих запросы к методам разработанного контроллера.

**Ход работы:**

Для выполнения данной лабораторной работы было создано новое WEB API Web-приложение. За основу которого было взято приложение, разработанное в предыдущих лабораторных работах, а именно: были использованы ранее созданные модели, а также контекст данных. Отличительной особенностью же *API* приложений является отсутствие представлений для отображений данных.

Для отображения данных в *API* приложении служат запросы, что подразумевает наличие некого клиента, который будет выполнять эти запросы. В данной работе доступ к серверу осуществляется посредством *HTML* страницы и функций, разработанных на языке *JavaScript.*

*GET* запрос был реализован первым. Данный запросы позволяет получить данные с сервера. В этой работе получаются данные из таблицы вызовов такси *(Calls)*, а также дополнительно запрашиваются данные стоящие на стороне отношения «один» (автомобили, тарифы, диспетчеры).

*PUT* запрос позволил реализовать возможность обновления данных уже существующих записей. Для этого ему передается *id* записи, которую необходимо изменить, а также сами новые данные записи.

*DELETE* запрос отвечает за возможность удаления записей. Для удаления записи в запрос необходимо передать *id* нужной записи.

*POST* запрос необходим для реализации возможности создания новых записей. Для создания записи в запрос необходимо передать все требуемые данные для записи (время вызова, телефон и т.д.).

На рисунке 1 показана таблица, содержащая данные, полученные из запроса, а также кнопки для удаления записи или ее редактирования.

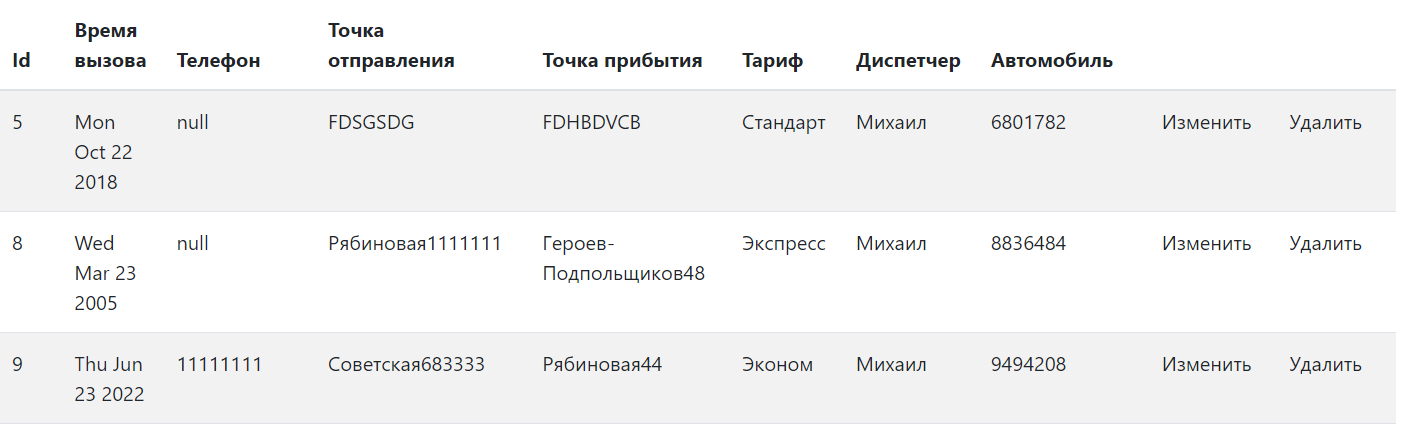


Рисунок 1 – Таблица вызовов такси

Также была создана форма, для возможности внесения данных при редактировании или создании записи, она показана на рисунке 2.

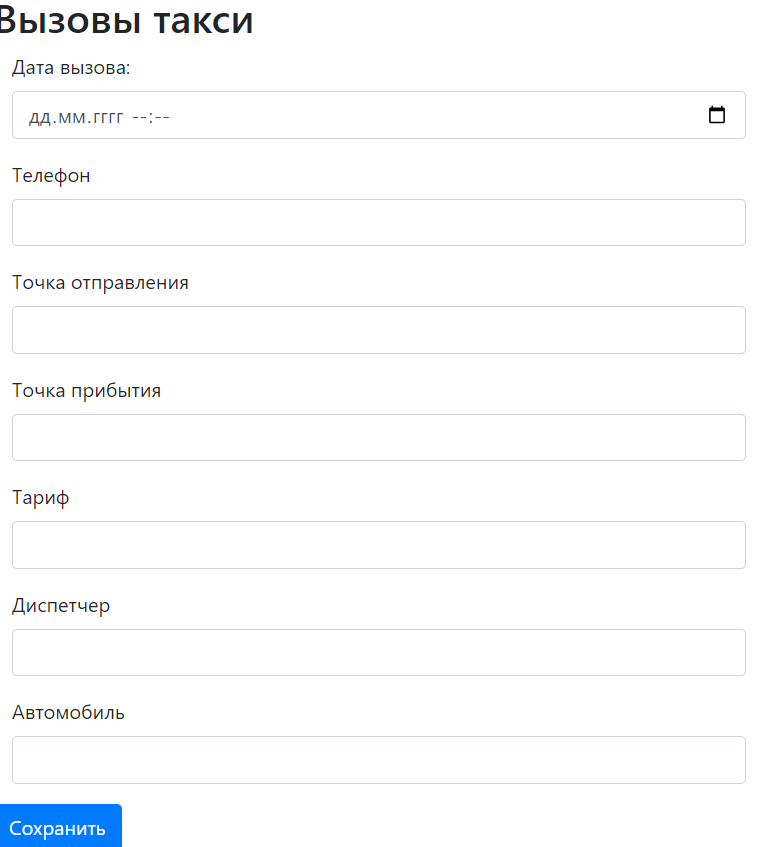


Рисунок 1 – Форма для создания и редактирования записи

Листинг разработанного приложения представлен в приложении А.

Также в данном приложении была реализована документация *Swagger.* *Swagger* — это набор инструментов, который позволяет автоматически описывать *API* на основе его кода. *API* — интерфейс для связи между разными программными продуктами, и у каждого проекта он свой. Документация, автоматически созданная через *Swagger*, облегчает понимание *API* для компьютеров и людей.

В приложении Б показаны копии экранов, демонстрирующие возможности *Swagger.*

*GitHub:* https://github.com/IvanTratsevski/Lab6TaxiGomel

**Вывод**: в ходе выполнения лабораторной работы был разработан контроллер, который выступает в виде обработчика запросов, которые поступают от клиентов. Был изучен механизм Swagger для удобного документирования и тестирования *WEB API*. Было разработано *web*-приложение, которое получает доступ к данным через *WEB API.*

**ПРИЛОЖНИЕ А**

**(обязательное)**

Листинг класса *Program*

using Lab6TaxiGomel.Data;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using System.Text.Json;

using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

string conStr = builder.Configuration.GetConnectionString("SqlServerConnection");

builder.Services.AddDbContext<TaxiGomelContext>(options => options.UseSqlServer(conStr));

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

/\*app.UseHttpsRedirection();\*/

app.UseStaticFiles();

app.UseAuthorization();

app.MapControllers();

app.Run();

Листинг класса *Call*

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace Lab6TaxiGomel.Models

{

public partial class Call

{

public int CallId { get; set; }

[Display(Name = "Время вызова")]

[Required(ErrorMessage = "Время вызова не указано")]

public DateTime CallTime { get; set; }

[Display(Name = "Телефон клиента")]

[StringLength(11, MinimumLength = 7)]

public string? Telephone { get; set; }

[Display(Name = "Адрес отправления")]

[StringLength(25, MinimumLength = 7)]

public string? StartPosition { get; set; }

[Display(Name = "Адрес доставки")]

[StringLength(25, MinimumLength = 7)]

public string? EndPosition { get; set; }

[Display(Name = "Тариф")]

public int? RateId { get; set; }

[Display(Name = "Автомобиль")]

public int? CarId { get; set; }

[Display(Name = "Диспетчер")]

public int? DispatcherId { get; set; }

public virtual Car? Car { get; set; }

public virtual Employee? Dispatcher { get; set; }

public virtual Rate? Rate { get; set; }

}

}

Листинг класса *Car*

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Text.Json.Serialization;

namespace Lab6TaxiGomel.Models

{

public partial class Car

{

public int CarId { get; set; }

[Display(Name = "Регистрационный номер")]

[Required(ErrorMessage = "Регистрационный номер не указан")]

[StringLength(11, MinimumLength = 7)]

public string? RegistrationNumber { get; set; }

[Display(Name = "Марка")]

[Required(ErrorMessage = "Марка автомобиля не указан")]

public int? CarModelId { get; set; }

[Display(Name = "Номер корпуса")]

[Required(ErrorMessage = "Номер корпуса не указан")]

[StringLength(11, MinimumLength = 7)]

public string? CarcaseNumber { get; set; }

[Display(Name = "Номер двигателя")]

[Required(ErrorMessage = "Номер двигателя не указан")]

[StringLength(11, MinimumLength = 7)]

public string? EngineNumber { get; set; }

[Display(Name = "Дата производства")]

[Required(ErrorMessage = "Дата производства не указана")]

[DataType(DataType.Date)]

public DateTime? ReleaseYear { get; set; }

[Display(Name = "Пробег")]

[Required(ErrorMessage = "Пробег не указан")]

[Range(1, 500000)]

public int? Mileage { get; set; }

[Display(Name = "Последний тех. осмотр")]

[Required(ErrorMessage = "Дата тех. осмотра не указана")]

[DataType(DataType.Date)]

public DateTime? LastTi { get; set; }

[Display(Name = "Особые отметки")]

public string? SpecialMarks { get; set; }

[JsonIgnore]

public virtual ICollection<Call> Calls { get; set; }

}

}

Листинг класса *Employee*

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Text.Json.Serialization;

namespace Lab6TaxiGomel.Models

{

public partial class Employee

{

public int EmployeeId { get; set; }

[Display(Name = "Имя")]

[Required(ErrorMessage = "Имя не указано")]

[StringLength(20, MinimumLength = 2)]

public string? FirstName { get; set; }

[Display(Name = "Фамилия")]

[Required(ErrorMessage = "Фамилия не указана")]

[StringLength(20, MinimumLength = 2)]

public string? LastName { get; set; }

[Display(Name = "Возраст")]

[Required(ErrorMessage = "Возраст не указан")]

[Range(18, 100)]

public int? Age { get; set; }

[Display(Name = "Должность")]

[Required(ErrorMessage = "Должность не указана")]

public int? PositionId { get; set; }

[Display(Name = "Стаж")]

[Required(ErrorMessage = "Стаж не указана")]

[Range(0, 100)]

public int? Experience { get; set; }

[JsonIgnore]

public virtual ICollection<Call> Calls { get; set; }

}

}

Листинг класса *Rate*

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Text.Json.Serialization;

namespace Lab6TaxiGomel.Models

{

public partial class Rate

{

public int RateId { get; set; }

[Display(Name = "Тариф")]

[Required(ErrorMessage = "Тариф не указан")]

[StringLength(20, MinimumLength = 1)]

public string? RateDescription { get; set; }

[Display(Name = "Стоимость")]

[Required(ErrorMessage = "Стоимость не указана")]

[Range(1, 100)]

public decimal? RatePrice { get; set; }

[JsonIgnore]

public virtual ICollection<Call> Calls { get; set; }

}

}

Листинг класса TaxiGomelContext

using System;

using System.Collections.Generic;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata;

using Lab6TaxiGomel.Models;

namespace Lab6TaxiGomel.Data

{

public partial class TaxiGomelContext : DbContext

{

public TaxiGomelContext()

{

}

public TaxiGomelContext(DbContextOptions<TaxiGomelContext> options)

: base(options)

{

}

public virtual DbSet<Call> Calls { get; set; } = null!;

public virtual DbSet<Car> Cars { get; set; } = null!;

public virtual DbSet<CarModel> CarModels { get; set; } = null!;

public virtual DbSet<Employee> Employees { get; set; } = null!;

public virtual DbSet<Position> Positions { get; set; } = null!;

public virtual DbSet<Rate> Rates { get; set; } = null!;

}

}

Листинг класса *CallsController*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Lab6TaxiGomel.Data;

using Lab6TaxiGomel.Models;

namespace Lab6TaxiGomel.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class CallsController : ControllerBase

{

private readonly TaxiGomelContext \_context;

public CallsController(TaxiGomelContext context)

{

\_context = context;

}

[HttpGet]

public IEnumerable<Call> Get()

{

var data = \_context.Calls

.Include(e => e.Car)

.Include(e => e.Dispatcher)

.Include(e => e.Rate)

.ToList();

return data;

}

[HttpGet("{callId}")]

public IActionResult Get(int callId)

{

Call call = \_context.Calls.FirstOrDefault(e => e.CallId == callId);

if (call == null)

return NotFound();

return new ObjectResult(call);

}

[HttpPost]

public IActionResult Post([FromBody] Call call)

{

if (call == null)

{

return BadRequest();

}

\_context.Calls.Add(call);

\_context.SaveChanges();

return Ok(call);

}

[HttpPut]

public IActionResult Put([FromBody] Call call)

{

if (call == null)

{

return BadRequest();

}

if (!\_context.Calls.Any(e => e.CallId == call.CallId))

{

return NotFound();

}

\_context.Update(call);

\_context.SaveChanges();

return Ok(call);

}

[HttpDelete("{callId}")]

public IActionResult Delete(int callId)

{

Call call = \_context.Calls.FirstOrDefault(e => e.CallId == callId);

if (call == null)

{

return NotFound();

}

\_context.Calls.Remove(call);

\_context.SaveChanges();

return Ok(call);

}

}

}

Листинг файла *index.html*

﻿

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width" />

<title>Вызовы такси</title>

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.0/css/bootstrap.min.css" rel="stylesheet" />

</head>

<body>

<h2>Вызовы такси</h2>

<form name="callForm">

<input type="hidden" name="callId" value="0" />

<div class="form-group col-md-5">

<label for="callTime">Дата вызова:</label>

<input class="form-control" name="callTime" type="datetime-local" />

</div>

<div class="form-group col-md-5">

<label for="telephone">Телефон</label>

<input class="form-control" name="telephone" />

</div>

<div class="form-group col-md-5">

<label for="startPosition">Точка отправления</label>

<input class="form-control" name="startPosition" />

</div>

<div class="form-group col-md-5">

<label for="endPosition">Точка прибытия</label>

<input class="form-control" name="endPosition" />

</div>

<div class="form-group col-md-5">

<label for="rateId">Тариф</label>

<input class="form-control" name="rateId" type="number" />

</div>

<div class="form-group col-md-5">

<label for="dispatcherId">Диспетчер</label>

<input class="form-control" name="dispatcherId" type="number" />

</div>

<div class="form-group col-md-5">

<label for="carId">Автомобиль</label>

<input class="form-control" name="carId" type="number" />

</div>

<div class="panel-body">

<button type="submit" id="submit" class="btn btn-primary">Сохранить</button>

</div>

</form>

<button id="reset" onclick="reset()" class="btn btn-primary">Сбросить</button>

<table class="table table-condensed table-striped col-md-6">

<thead>

<tr>

<th>Id</th>

<th>Время вызова</th>

<th>Телефон</th>

<th>Точка отправления</th>

<th>Точка прибытия</th>

<th>Тариф</th>

<th>Диспетчер</th>

<th>Автомобиль</th>

<th></th>

</tr>

</thead>

<tbody>

</tbody>

</table>

<script>

// Получение всех пользователей

async function GetCalls() {

const response = await fetch("/api/Calls", {

method: "GET",

headers: { "Accept": "application/json" }

});

if (response.ok === true) {

const calls = await response.json();

let rows = document.querySelector("tbody");

calls.forEach(call => {

rows.append(row(call));

});

}

}

async function GetCall(callId) {

const response = await fetch("/api/Calls/" + callId, {

method: "GET",

headers: { "Accept": "application/json" }

});

if (response.ok === true) {

const call = await response.json();

const form = document.forms["callForm"];

let date = new Date(call.callTime);

let year = date.getFullYear();

let month = ("0" + (date.getMonth() + 1)).slice(-2);

let day = ("0" + date.getDate()).slice(-2);

let hours = ("0" + date.getHours()).slice(-2);

let minutes = ("0" + date.getMinutes()).slice(-2);

let formattedDateString = year + "-" + month + "-" + day + "T" + hours + ":" + minutes;

form.elements["callId"].value = call.callId;

form.elements["callTime"].value = formattedDateString;

form.elements["telephone"].value = call.telephone;

form.elements["startPosition"].value = call.startPosition;

form.elements["endPosition"].value = call.endPosition;

form.elements["rateId"].value = call.rateId;

form.elements["dispatcherId"].value = call.dispatcherId;

form.elements["carId"].value = call.carId;

}

}

async function CreateCall(callTime, telephone, startPosition, endPosition, carId, rateId, dispatcherId) {

console.log(JSON.stringify({

callTime: callTime,

telephone: telephone,

startPosition: startPosition,

endPosition: endPosition,

carId: carId,

rateId: rateId,

dispatcherId: dispatcherId

}))

const response = await fetch("api/Calls", {

method: "POST",

headers: { "Accept": "application/json", "Content-Type": "application/json" },

body: JSON.stringify({

callTime: callTime,

telephone: telephone,

startPosition: startPosition,

endPosition: endPosition,

carId: carId,

rateId: rateId,

dispatcherId: dispatcherId

})

});

if (response.ok === true) {

await response.json();

reset();

}

}

async function EditCall(callId, callTime, telephone, startPosition, endPosition, carId, rateId, dispatcherId) {

console.log(JSON.stringify({

callId: callId,

callTime: callTime,

telephone: telephone,

startPosition: startPosition,

endPosition: endPosition,

carId: carId,

rateId: rateId,

dispatcherId: dispatcherId

}))

const response = await fetch("api/Calls", {

method: "PUT",

headers: { "Accept": "application/json", "Content-Type": "application/json" },

body: JSON.stringify({

callId: callId,

callTime: callTime,

telephone: telephone,

startPosition: startPosition,

endPosition: endPosition,

carId: carId,

rateId: rateId,

dispatcherId: dispatcherId

})

});

if (response.ok === true) {

await response.json();

reset();

}

}

async function DeleteCall(callId) {

const response = await fetch("/api/Calls/" + callId, {

method: "DELETE",

headers: { "Accept": "application/json" }

});

if (response.ok === true) {

const call = await response.json();

document.querySelector("tr[data-rowid='" + call.callId + "']").remove();

}

}

function reset() {

document.location.reload();

}

function row(call) {

const tr = document.createElement("tr");

tr.setAttribute("data-rowid", call.callId);

const callIdTd = document.createElement("td");

callIdTd.append(call.callId);

tr.append(callIdTd);

const calllTimeTd = document.createElement("td");

calllTimeTd.append(new Date(call.callTime).toDateString());

tr.append(calllTimeTd);

const telephoneTd = document.createElement("td");

telephoneTd.append(call.telephone);

tr.append(telephoneTd);

const startPositionTd = document.createElement("td");

startPositionTd.append(call.startPosition);

tr.append(startPositionTd);

const endPositionTd = document.createElement("td");

endPositionTd.append(call.endPosition);

tr.append(endPositionTd);

const rateTd = document.createElement("td");

rateTd.append(call.rate.rateDescription);

tr.append(rateTd);

const dispatcherTd = document.createElement("td");

dispatcherTd.append(call.dispatcher.firstName);

tr.append(dispatcherTd);

const carTd = document.createElement("td");

carTd.append(call.car.registrationNumber);

tr.append(carTd);

const linksTd = document.createElement("td");

const editLink = document.createElement("a");

editLink.setAttribute("data-callId", call.callId);

editLink.setAttribute("style", "cursor:pointer;padding:15px;");

editLink.append("Изменить");

editLink.addEventListener("click", e => {

e.preventDefault();

GetCall(call.callId);

});

linksTd.append(editLink);

const removeLink = document.createElement("a");

removeLink.setAttribute("data-callId", call.callId);

removeLink.setAttribute("style", "cursor:pointer;padding:15px;");

removeLink.append("Удалить");

removeLink.addEventListener("click", e => {

e.preventDefault();

DeleteCall(call.callId);

});

linksTd.append(removeLink);

tr.appendChild(linksTd);

return tr;

}

document.forms["callForm"].addEventListener("submit", e => {

e.preventDefault();

const form = document.forms["callForm"];

const callId = form.elements["callId"].value;

const callTime = form.elements["callTime"].value;

const telephone = form.elements["telephone"].value;

const startPosition = form.elements["startPosition"].value;

const endPosition = form.elements["endPosition"].value;

const carId = form.elements["carId"].value;

const rateId = form.elements["rateId"].value;

const dispatcherId = form.elements["dispatcherId"].value;

if (callId == 0)

CreateCall(callTime, telephone, startPosition, endPosition, carId, rateId, dispatcherId);

else

EditCall(callId, callTime, telephone, startPosition, endPosition, carId, rateId, dispatcherId);

});

GetCalls();

</script>

</body>

</html>

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

Копии экранов *Swagger*

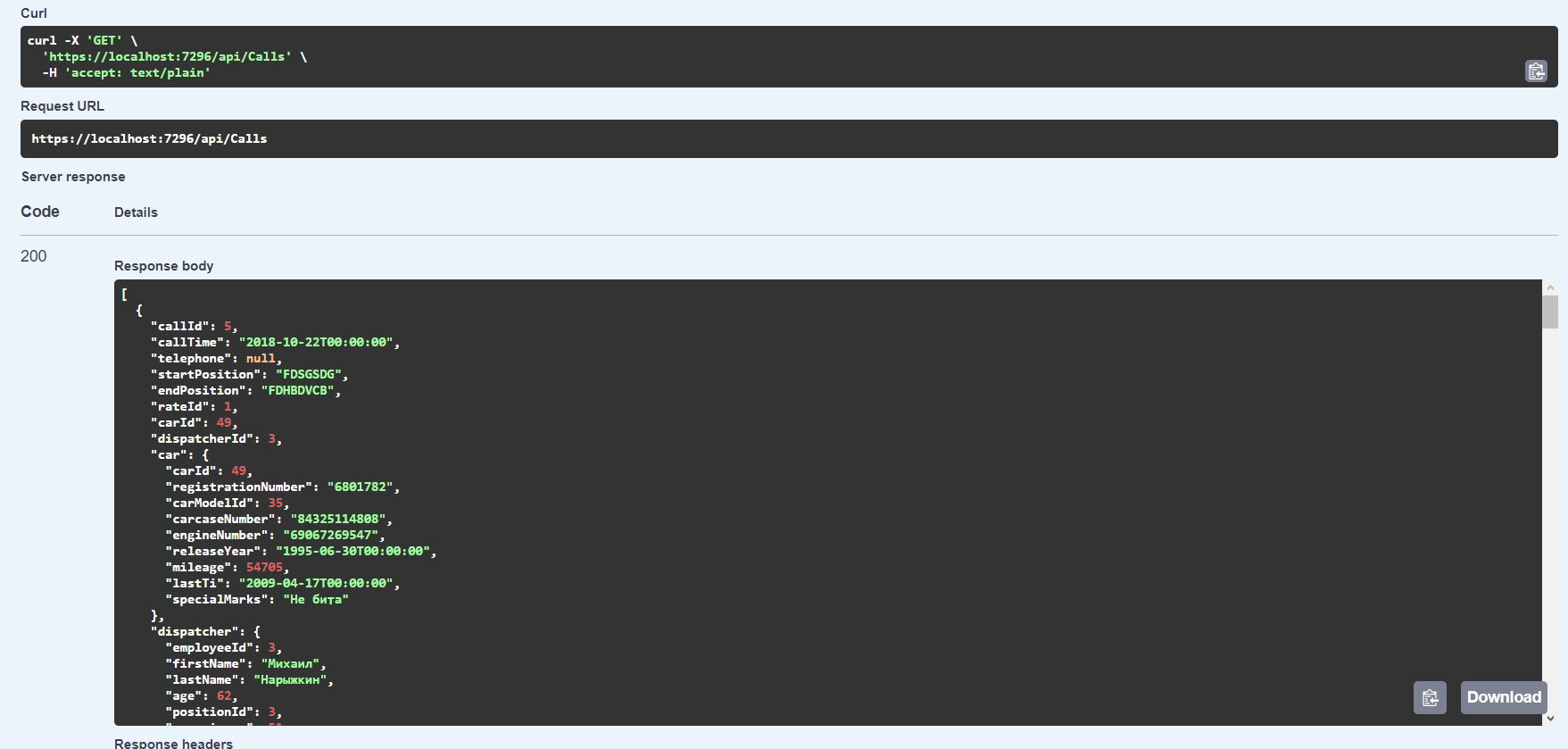


Рисунок А.1 – *GET* запрос

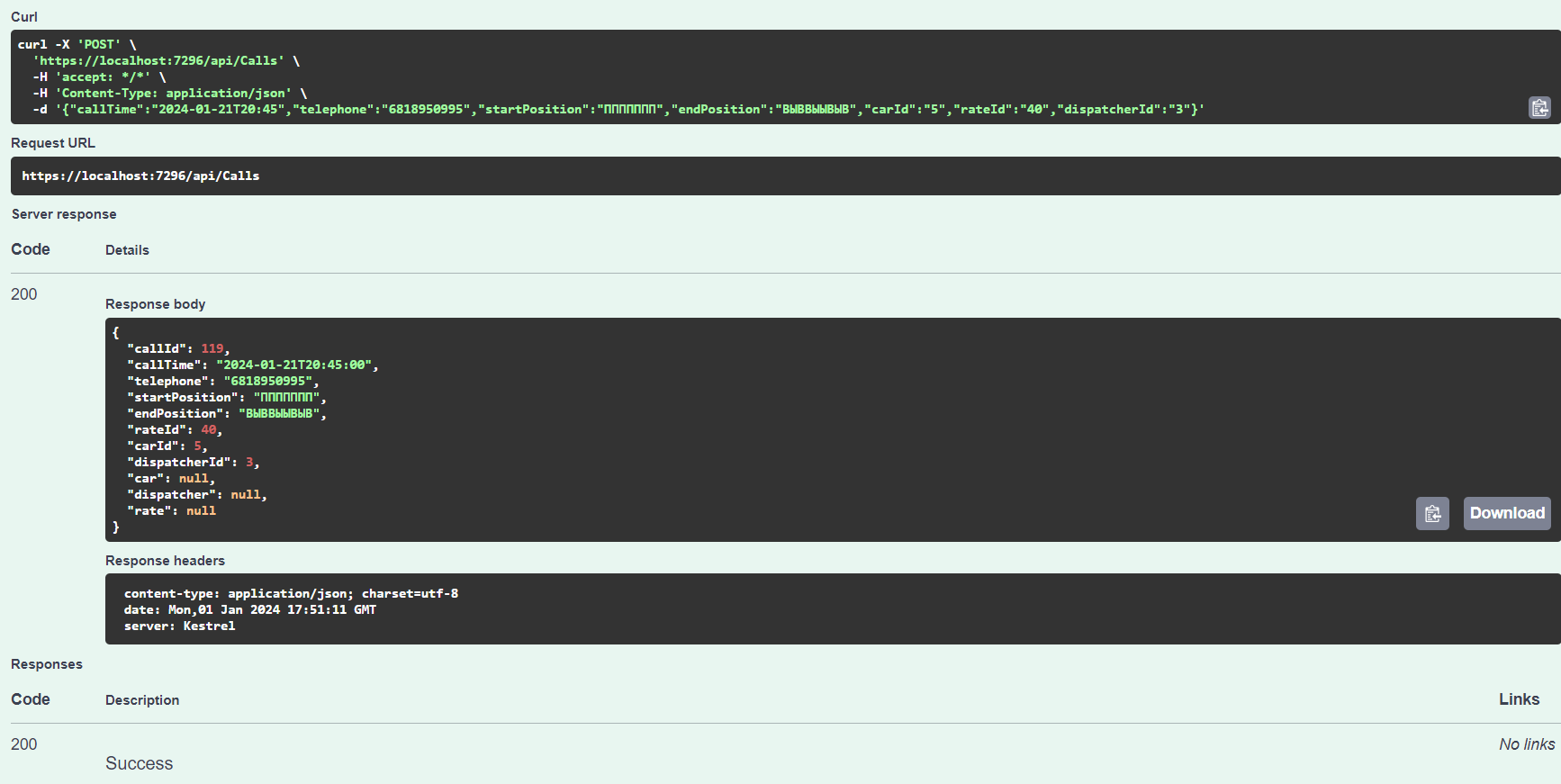


Рисунок А.2 – *POST* запрос

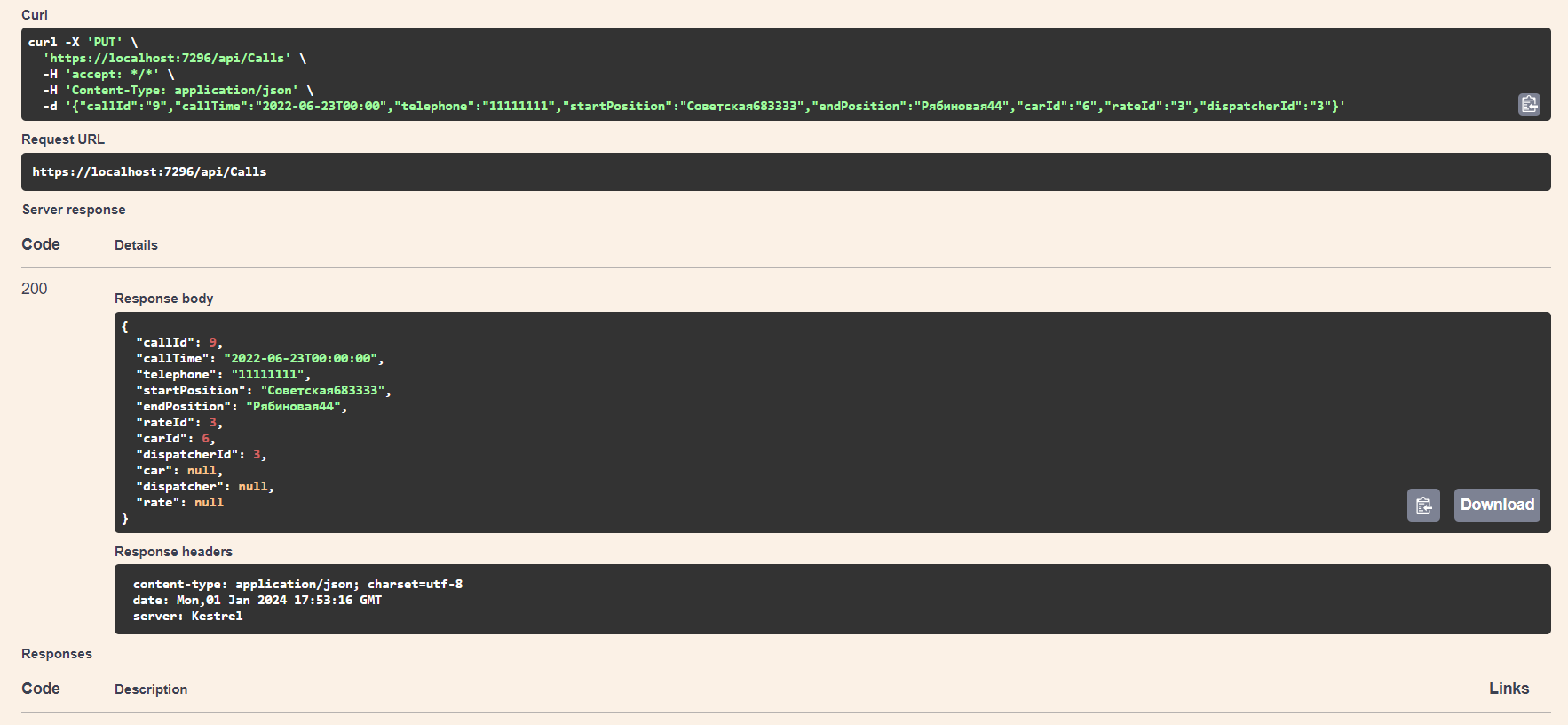


Рисунок А.3 – *POST* запрос



Рисунок А.4 – *DELETE* запрос