**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

Бондарев Е.Ю.

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**

В процессе выполнения лабораторной работы были применены классы моделей из предыдущих заданий, в которых одна из моделей (*Subscription*) выступает в роли стороны отношения "многие". Затем был создан класс контекста для взаимодействия с данными из базы данных.

Затем был разработан контроллер, представляющий интерфейс для взаимодействия с *API.* Весь процесс обработки в контроллере был построен в соответствии с принципами *REST API*. Здесь метод *GET* отвечает за операцию получения всей коллекции данных или конкретного элемента коллекции, метод *DELETE* отвечает за удаление элемента, метод *POST* отвечает за добавление элемента, а метод *PUT* отвечает за обновление данных. На основе этих шаблонов был разработан контроллер для работы с одной из таблиц базы данных, расположенной на стороне отношения "многие". Пример данного контроллера указан в приложении А. Пример *GET* запроса указан на рисунке 1.

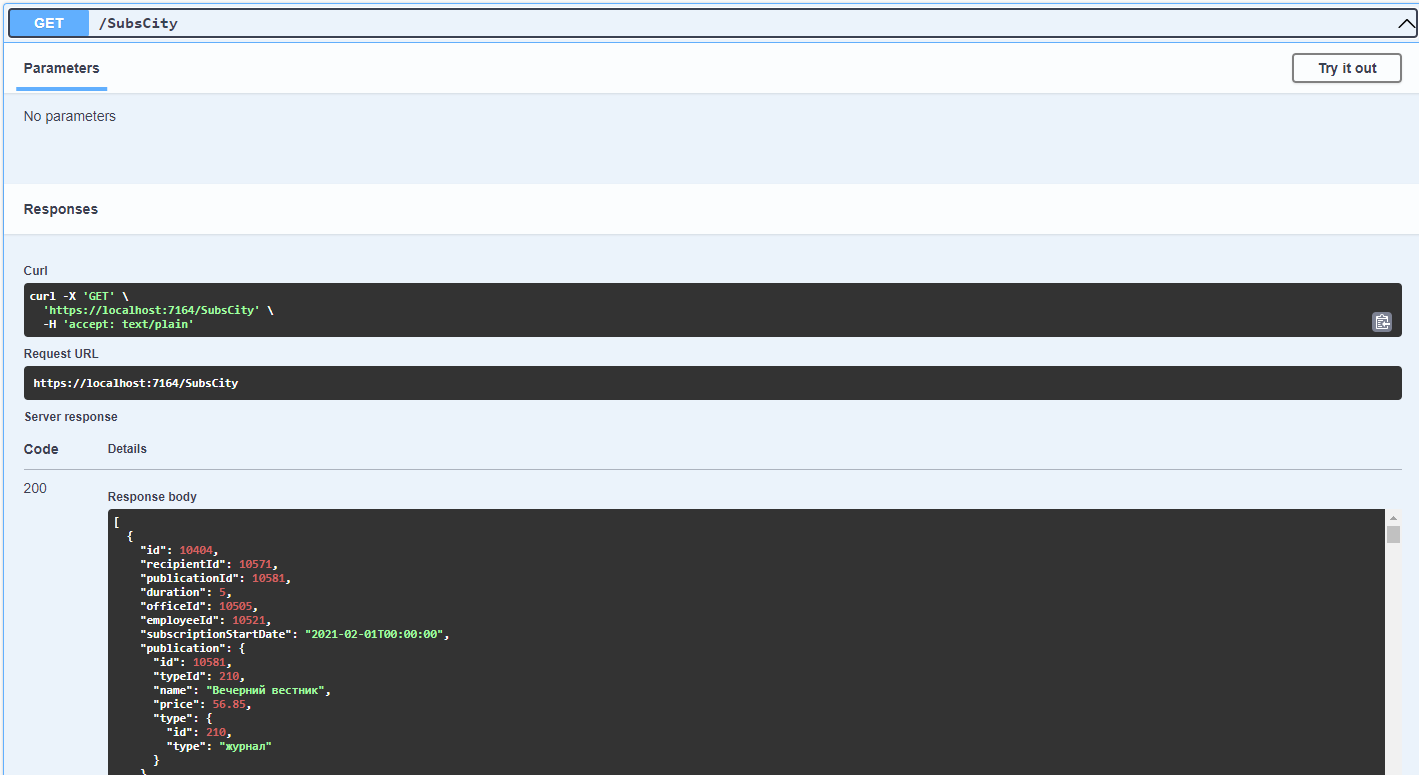


Рисунок 1 – Пример *GET* запроса в *Swagger*

Пример параметризированного *GET* запроса указан на рисунке 2.

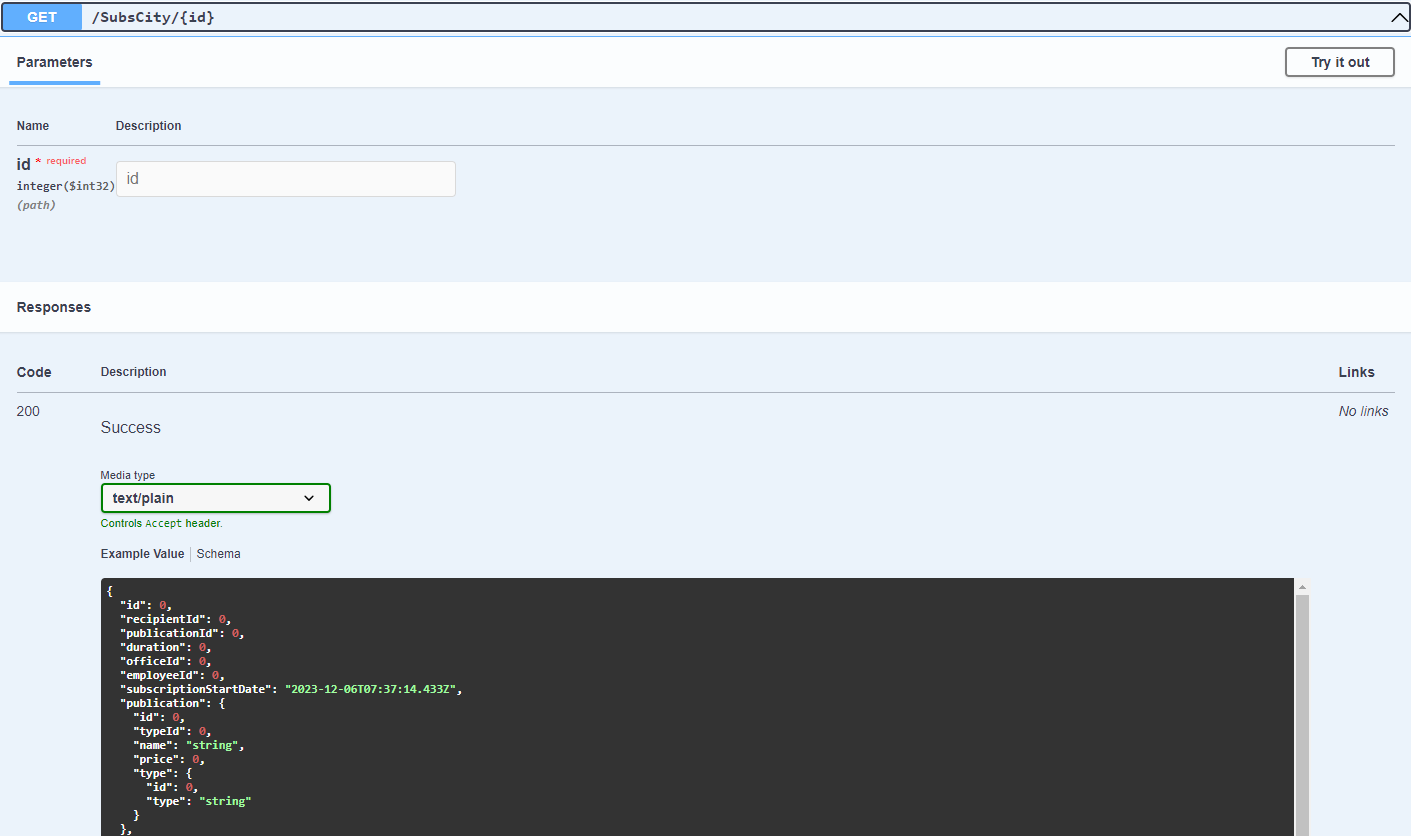
**

Рисунок 2 – Пример параметризированного *GET* запроса

Далее был протестирован *POST* запрос для операции добавления данных. Пример *POST* запроса в *Swagger* указан на рисунке 3.

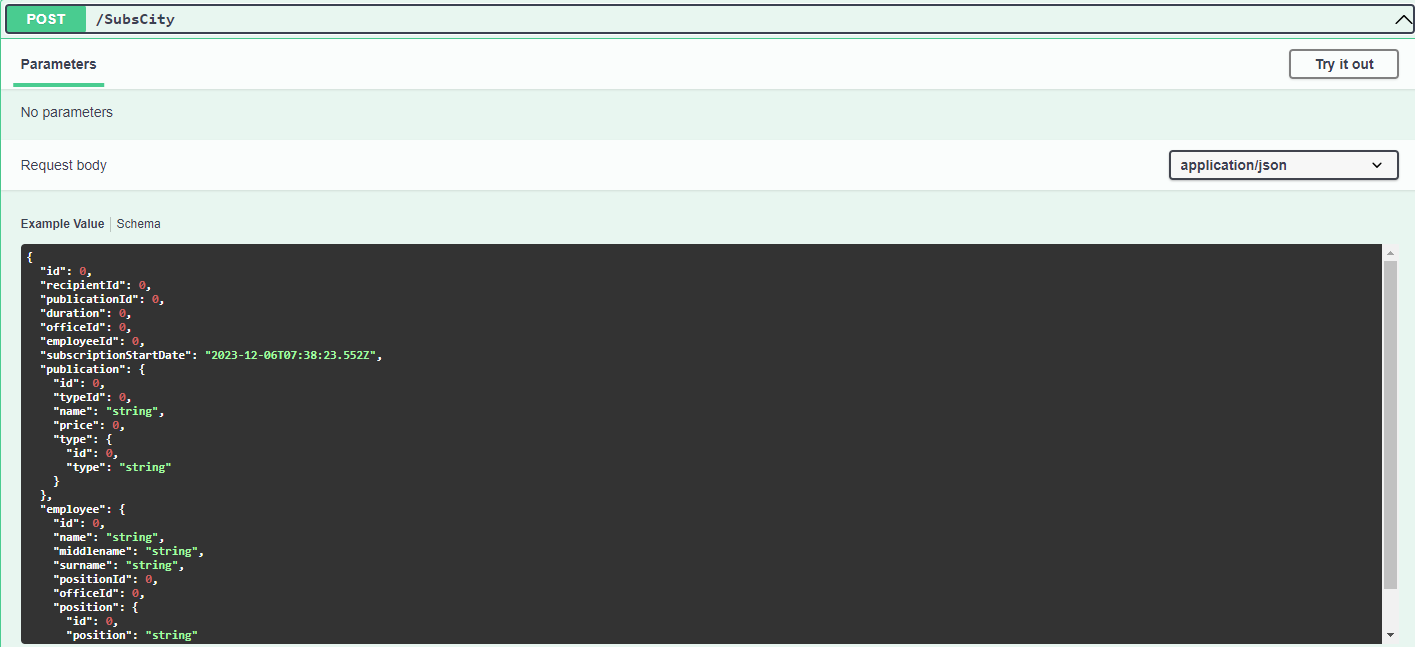


Рисунок 3 – Пример *POST* запроса в *Swagger*

Далее был протестирован *PUT* запрос для операции обновления данных. Пример *PUT* запроса в *Swagger* указан на рисунке 4.

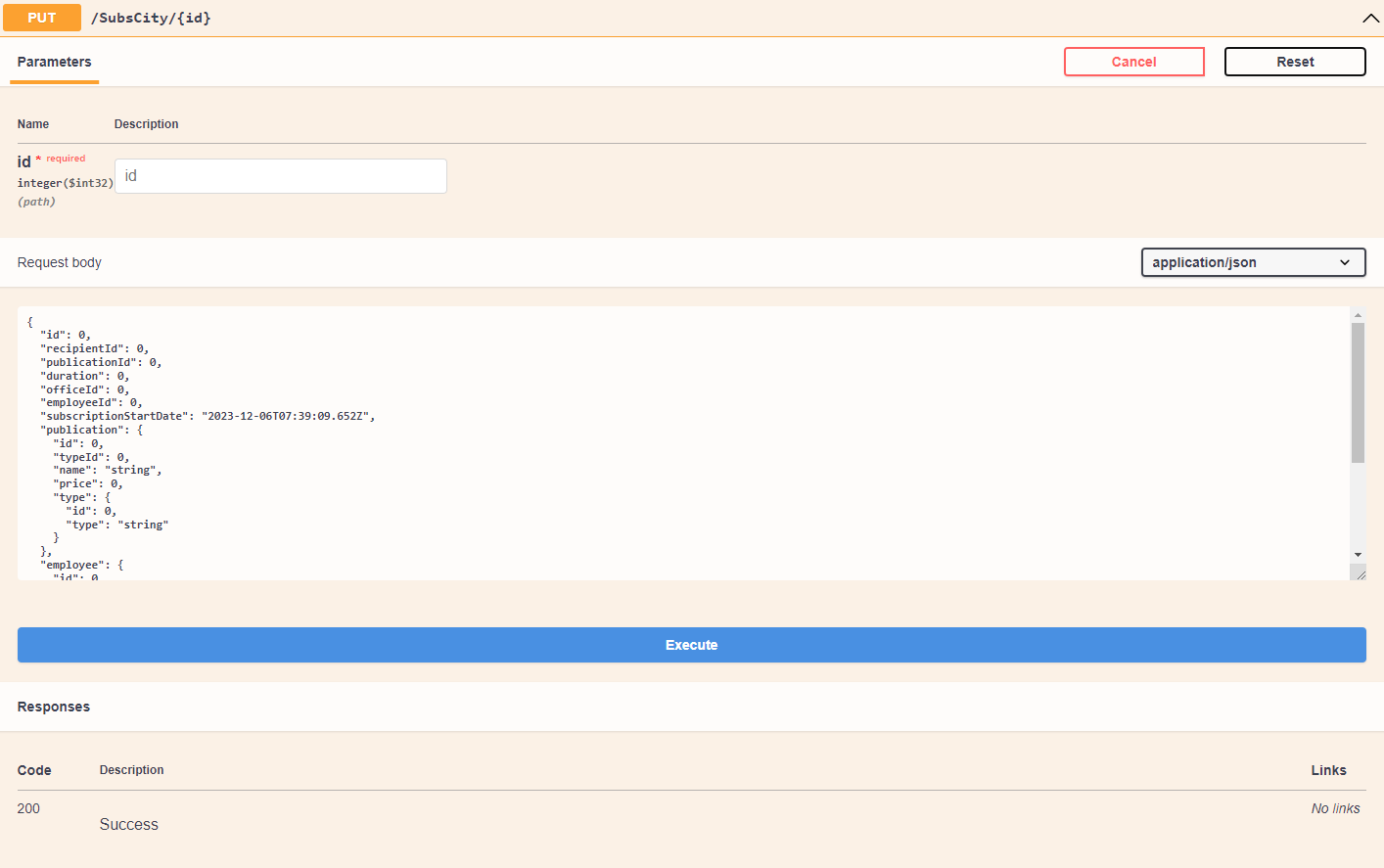


Рисунок 4 – Пример *PUT* запроса в *Swagger*

Далее был протестирован *DELETE* запрос для операции удаления конкретного элемента из базы данных. Пример этого *GET* запроса в *Swagger* указан на рисунке 5.

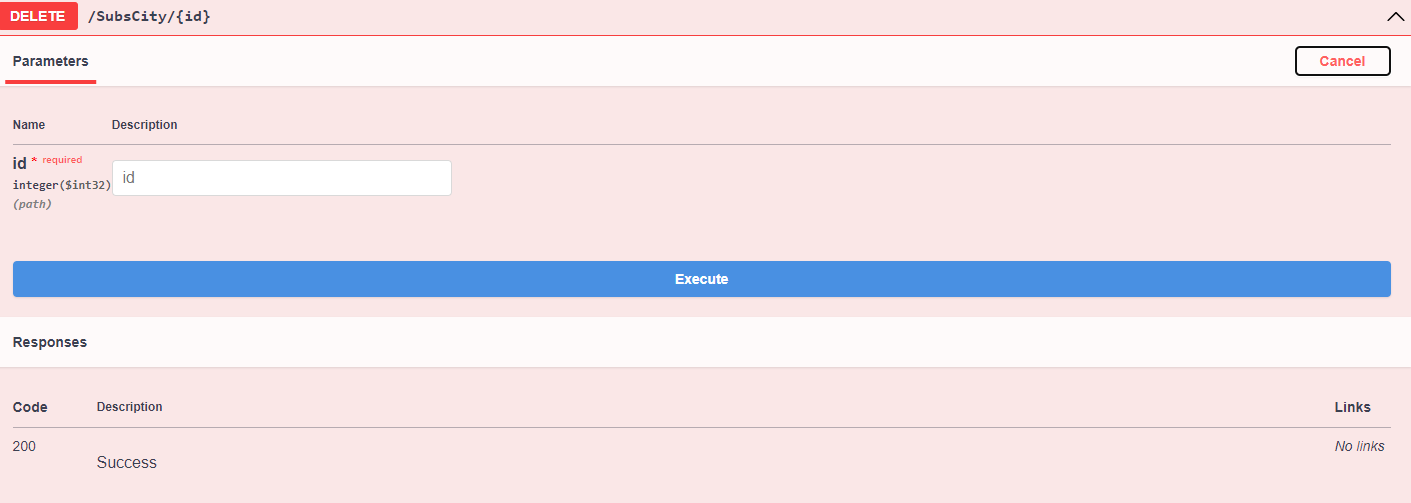


Рисунок 5 – Пример *GET* запроса конкретного элемента по *id* в *Swagger*

Затем была создана часть приложения, предназначенная для использования написанного *API* и получения доступа к данным. Клиентское приложение было разработано с использованием *HTML* и *JavaScript*. С помощью *JavaScript* клиентский код обеспечивает полный доступ к реализованному *API*. Пример кода клиентского приложении указан в приложении А. Пример главного меню приложения указан на рисунке 6.

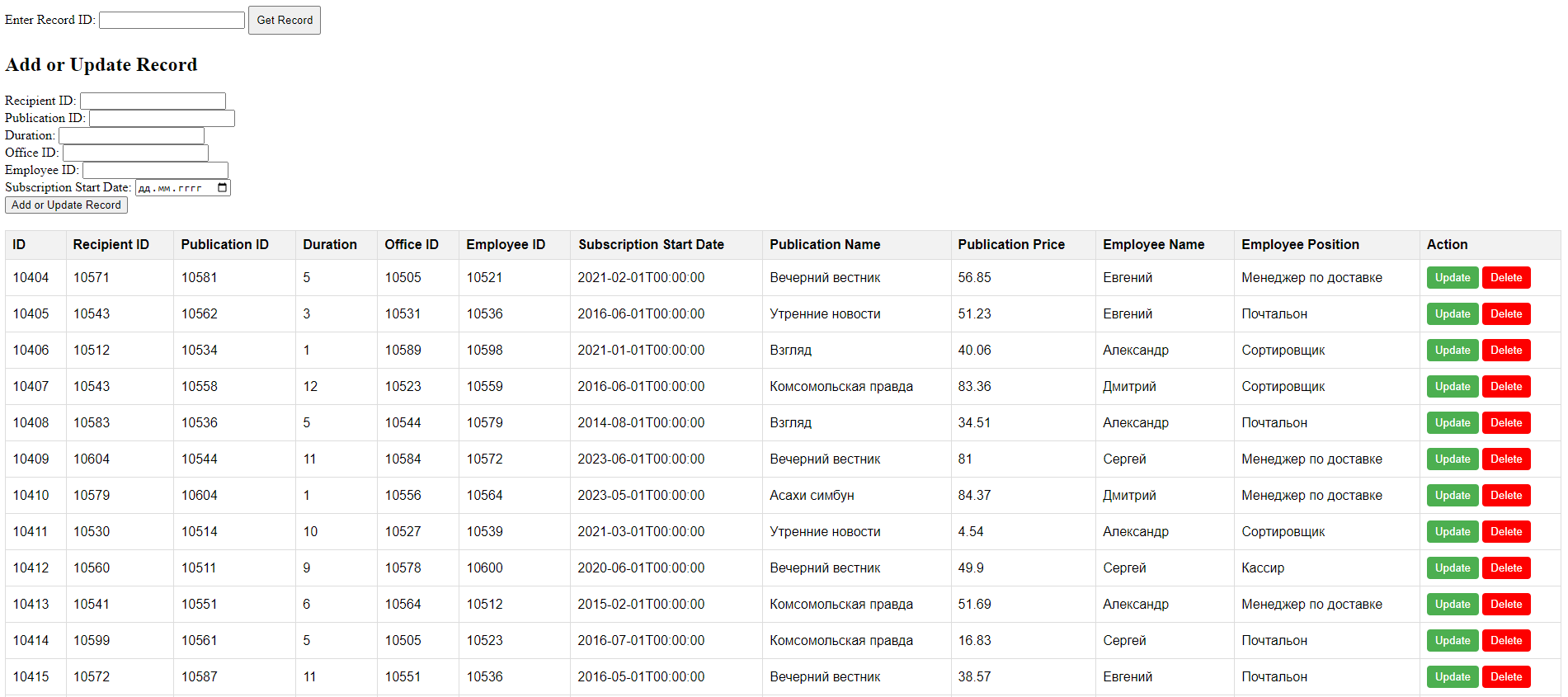


Рисунок 6 – Пример меню *web*-приложения

После выполнения лабораторной работы созданные проект был добавлен в локальный *git* репозиторий а потом перенесен в *GitHub* репозиторий своего аккаунта. Чтобы ознакомится с созданным проектом можно по ссылке [*Javaro3/lab5\_DB (github.com)*](https://github.com/Javaro3/lab5_DB)*.*

**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**

Листинг класса *Program*

using Laba4.Models;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddDbContext<SubsCityContext>(option => option.UseSqlServer(connectionString));

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowSpecificOrigin",

builder => builder

.AllowAnyOrigin() // В реальном применении уточните конкретные источники, которые разрешены.

.AllowAnyHeader()

.AllowAnyMethod());

});

var app = builder.Build();

app.UseCors("AllowSpecificOrigin");

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthorization();

app.UseDefaultFiles();

app.MapControllers();

app.UseStaticFiles();

app.Run();

Листинг класса *SubsCityController*

using Laba4.Models;

using Microsoft.AspNetCore.Cors;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Text.Json;

namespace Lab6.Controllers

{

[EnableCors("AllowSpecificOrigin")]

[ApiController]

[Route("[controller]")]

public class SubsCityController : ControllerBase

{

private readonly SubsCityContext \_context;

public SubsCityController(SubsCityContext subsCityContext)

{

\_context = subsCityContext;

}

[HttpGet]

public IEnumerable<Subscription> Get()

{

var data = \_context.Subscriptions.Include(e => e.Employee).ThenInclude(p => p.Position)

.Include(p => p.Publication).ThenInclude(t => t.Type).ToList();

return data;

}

[HttpGet("{id}")]

public Subscription Get(int id)

{

var data = \_context.Subscriptions.Include(e => e.Employee).ThenInclude(p => p.Position)

.Include(p => p.Publication).ThenInclude(t => t.Type).FirstOrDefault(s => s.Id == id);

return data;

}

[HttpPost]

public IActionResult Post([FromBody] Subscription subscription)

{

if (subscription == null)

{

return BadRequest();

}

\_context.Subscriptions.Add(subscription);

var result = \_context.SaveChanges();

return Ok(subscription);

}

[HttpPut("{id}")]

public IActionResult Put(int id, [FromBody] Subscription updatedSubscription)

{

if (updatedSubscription == null)

{

return BadRequest();

}

var existingSubscription = \_context.Subscriptions.FirstOrDefault(e => e.Id == id);

if (existingSubscription == null)

{

return NotFound();

}

existingSubscription.RecipientId = updatedSubscription.RecipientId;

existingSubscription.PublicationId = updatedSubscription.PublicationId;

existingSubscription.Duration = updatedSubscription.Duration;

existingSubscription.OfficeId = updatedSubscription.OfficeId;

existingSubscription.EmployeeId = updatedSubscription.EmployeeId;

existingSubscription.SubscriptionStartDate = updatedSubscription.SubscriptionStartDate;

\_context.SaveChanges();

return Ok(existingSubscription);

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

Subscription insuranceCase = \_context.Subscriptions.FirstOrDefault(e => e.Id == id);

if (insuranceCase == null)

{

return NotFound();

}

\_context.Subscriptions.Remove(insuranceCase);

\_context.SaveChanges();

return Ok(insuranceCase);

}

}

}

Листинг файла *client.html*

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Subscriptions Table</title>

<style>

table {

font-family: Arial, sans-serif;

border-collapse: collapse;

width: 100%;

margin-top: 20px;

}

th,

td {

border: 1px solid #dddddd;

text-align: left;

padding: 8px;

}

th {

background-color: #f2f2f2;

}

#idInput {

margin-bottom: 10px;

}

#getRecordButton,

#addRecordButton {

padding: 8px;

cursor: pointer;

}

#addRecordForm,

#updateRecordForm {

margin-bottom: 20px;

}

.updateButton {

background-color: #4CAF50;

color: white;

padding: 6px 10px;

border: none;

border-radius: 4px;

cursor: pointer;

}

.deleteButton {

background-color: #fd0000;

color: white;

padding: 6px 10px;

border: none;

border-radius: 4px;

cursor: pointer;

}

</style>

</head>

<body>

<div>

<label for="idInput">Enter Record ID: </label>

<input type="text" id="idInput">

<button id="getRecordButton" onclick="getRecord()">Get Record</button>

</div>

<div id="addRecordForm">

<h2>Add or Update Record</h2>

<form id="subscriptionForm">

<label for="recipientId">Recipient ID: </label>

<input type="text" id="recipientId" name="recipientId" required><br>

<!-- Добавьте другие поля формы сюда, если есть -->

<button type="button" id="addOrUpdateRecordButton" onclick="addOrUpdateRecord()">Add or Update Record</button>

</form>

</div>

<table id="subscriptionsTable">

<thead>

<tr>

<th>ID</th>

<th>Recipient ID</th>

<th>Publication ID</th>

<th>Duration</th>

<th>Office ID</th>

<th>Employee ID</th>

<th>Subscription Start Date</th>

<th>Publication Name</th>

<th>Publication Price</th>

<th>Employee Name</th>

<th>Employee Position</th>

<th>Action</th>

</tr>

</thead>

<tbody>

<!-- Table rows will be populated dynamically using JavaScript -->

</tbody>

</table>

<script>

function populateTable(data) {

const tableBody = document.querySelector('#subscriptionsTable tbody');

tableBody.innerHTML = '';

data.forEach(subscription => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${subscription.id}</td>

<td>${subscription.recipientId}</td>

<td>${subscription.publicationId}</td>

<td>${subscription.duration}</td>

<td>${subscription.officeId}</td>

<td>${subscription.employeeId}</td>

<td>${subscription.subscriptionStartDate}</td>

<td>${subscription.publication.name}</td>

<td>${subscription.publication.price}</td>

<td>${subscription.employee.name}</td>

<td>${subscription.employee.position.position}</td>

<td>

<button class="updateButton" onclick="updateRecord(${subscription.id})">Update</button>

<button class="deleteButton" onclick="deleteRecord(${subscription.id})">Delete</button>

</td>

`;

tableBody.appendChild(row);

});

}

window.onload = function () {

fetch('https://localhost:7164/SubsCity')

.then(response => response.json())

.then(data => {

populateTable(data);

})

.catch(error => console.error('Error fetching data:', error));

};

function getRecord() {

const idInput = document.getElementById('idInput').value;

if (!isNaN(idInput)) {

fetch(`https://localhost:7164/SubsCity/${idInput}`)

.then(response => response.json())

.then(data => {

document.getElementById('recipientId').value = data.recipientId;

// Set values for other fields as well

document.getElementById('addRecordForm').style.display = 'block';

})

.catch(error => console.error('Error fetching data:', error));

} else {

fetch('https://localhost:7164/SubsCity')

.then(response => response.json())

.then(data => {

populateTable(data);

})

.catch(error => console.error('Error fetching data:', error));

}

}

function updateRecord(id) {

document.getElementById('idInput').value = id;

getRecord();

}

function addOrUpdateRecord() {

const recipientId = document.getElementById('recipientId').value;

const publicationId = document.getElementById('publicationId').value;

const duration = document.getElementById('duration').value;

const officeId = document.getElementById('officeId').value;

const employeeId = document.getElementById('employeeId').value;

const subscriptionStartDate = document.getElementById('subscriptionStartDate').value;

const data = {

recipientId: parseInt(recipientId),

publicationId: parseInt(publicationId),

duration: parseInt(duration),

officeId: parseInt(officeId),

employeeId: parseInt(employeeId),

subscriptionStartDate: subscriptionStartDate

};

const idInput = document.getElementById('idInput').value;

const isUpdate = !isNaN(idInput);

const url = isUpdate ? `https://localhost:7164/SubsCity/${idInput}` : 'https://localhost:7164/SubsCity';

fetch(url, {

method: isUpdate ? 'PUT' : 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(data),

})

.then(response => response.json())

.then(result => {

console.log(`Subscription ${isUpdate ? 'updated' : 'added'} successfully:`, result);

location.reload();

})

.catch(error => {

console.error(`Error ${isUpdate ? 'updating' : 'adding'} subscription:`, error);

});

}

function deleteRecord(id) {

if (confirm("Are you sure you want to delete this record?")) {

fetch(`https://localhost:7164/SubsCity/${id}`, {

method: 'DELETE',

})

.then(response => response.json())

.then(result => {

console.log('Subscription deleted successfully:', result);

location.reload();

})

.catch(error => {

console.error('Error deleting subscription:', error);

});

}

}

</script>

</body>

</html>