**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: **«РПБДИС»**

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

Сетко А.И.

Принял: старший преподаватель

Асенчик О.Д.

Гомель 2019

**Цель**: 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. 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**

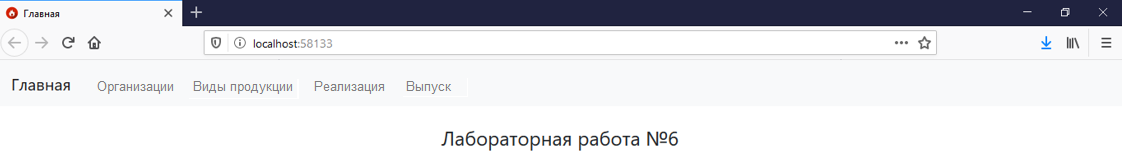


Рисунок 1 – Копия экрана браузера главной страницы сайта

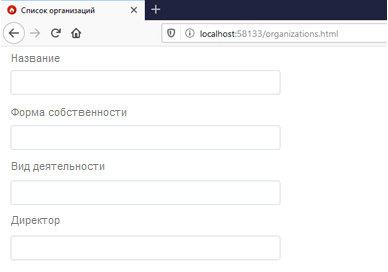


Рисунок 2 – Копия экрана браузера с формой на странице списка организаций

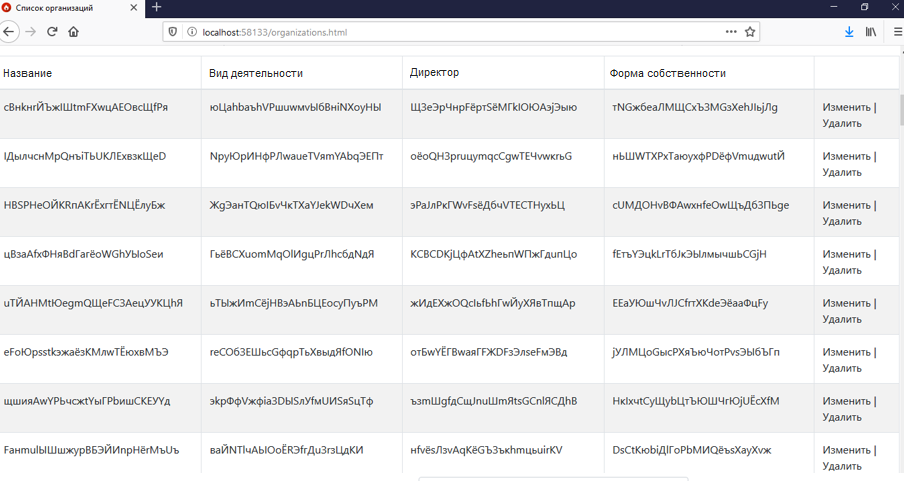


Рисунок 3 – Копия экрана браузера списка организаций

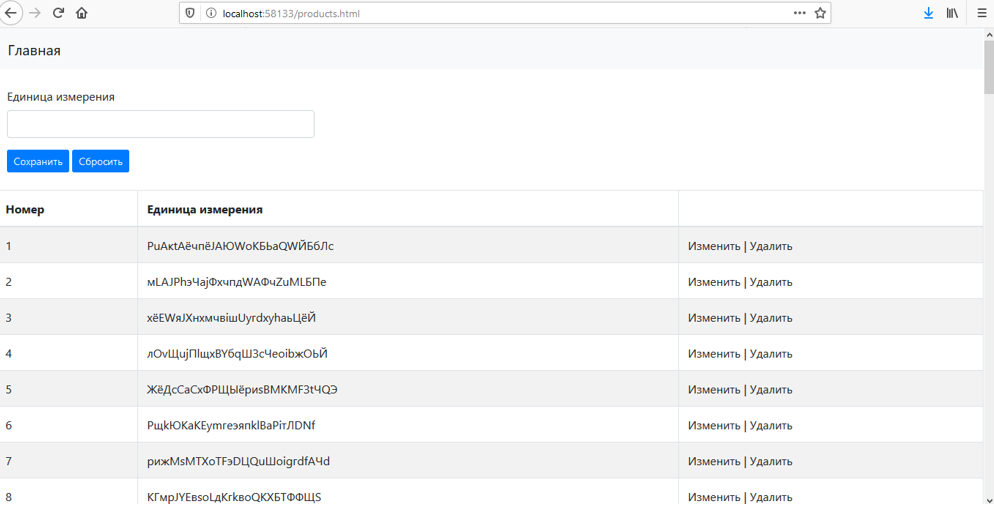


Рисунок 4 – Копия экрана браузера с формой и списком видов продукций

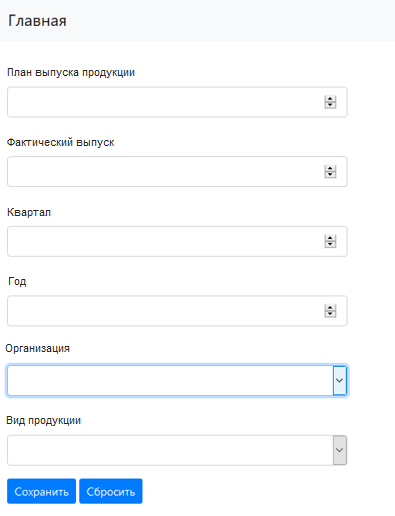


Рисунок 5 – Копия экрана браузера с формой на странице списка выпуска продукции организациями

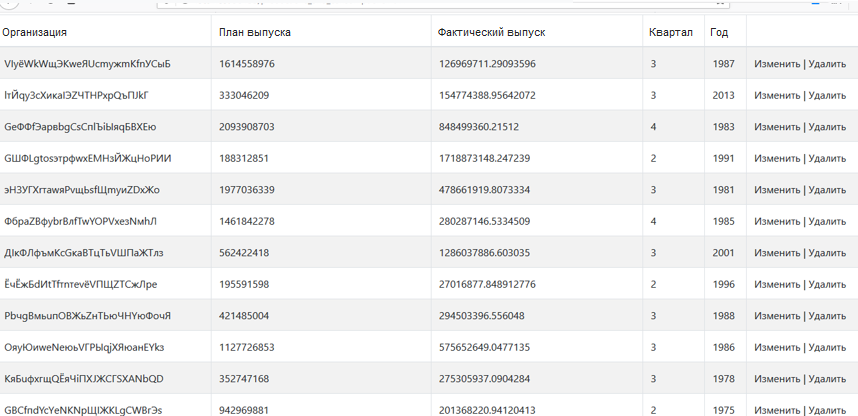


Рисунок 6 – Копия экрана браузера со списком выпуска продукции организациями

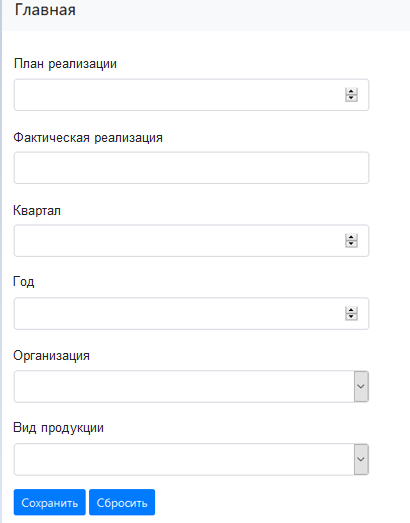


Рисунок 7 – Копия экрана браузера с формой на страницу реализации продукции организаций

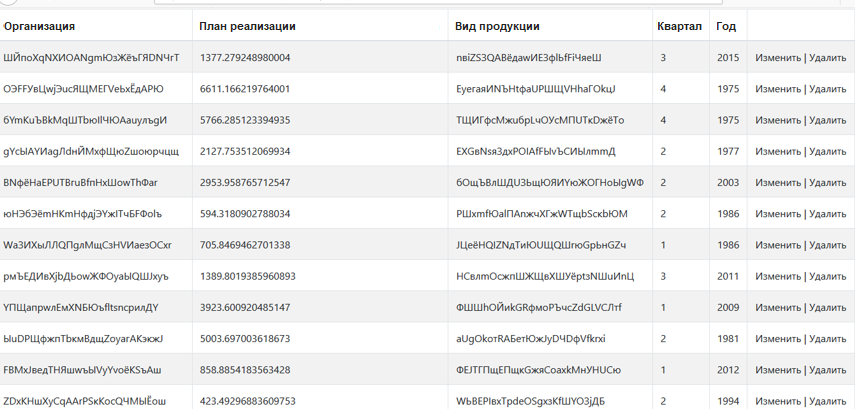


Рисунок 8 – Копия экрана браузера со списком реализации продукции организаций

**Вывод**: в результате выполнения лабораторной работы были ознакомлены с возможностями *ASP.NET Core* по получению, отображению и изменение данных при помощи технологии *WEB API*.

**Приложение А**

**Исходный код программы**

**Organization.cs:**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace Lab6.Models.Tables

{

public partial class Organization

{

public Organization()

{

Realisation = new HashSet<Realisation>();

Output = new HashSet<Output>();

}

public int Id { get; set; }

[Required(ErrorMessage = "Укажите название организации")]

[StringLength(40, MinimumLength = 5, ErrorMessage = "Минимальная длина названия организации должна быть 5 символов")]

public string Name { get; set; }

[Required(ErrorMessage = "Укажите форму собственности")]

[StringLength(30, MinimumLength = 3, ErrorMessage = "Минимальная длина формы собственности должна быть 3 символа")]

public string FormOfOwnership { get; set; }

[Required(ErrorMessage = "Укажите вид деятельности организации")]

[StringLength(30, MinimumLength = 5, ErrorMessage = "Минимальная длина вида деятельности должна быть 5 символов")]

public string ActivityType { get; set; }

[Required(ErrorMessage = "Укажите ФИО директора")]

[StringLength(50, MinimumLength = 5, ErrorMessage = "Минимальная длина ФИО директора должна быть 5 символов")]

public string HeadName { get; set; }

public virtual ICollection<Realisation> Realisation { get; set; }

public virtual ICollection<Output> Output { get; set; }

}

}

**Product.cs:**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace Lab6.Models.Tables

{

public partial class Product

{

public Product()

{

Realisation = new HashSet<Realisation>();

Output = new HashSet<Output>();

}

public int Id { get; set; }

[Required(ErrorMessage = "Укажите единицу измерения")]

[StringLength(50, ErrorMessage = "Максимальная длина единицы измерения должна быть 50 символов")]

public string MeasureUnit { get; set; }

public virtual ICollection<Realisation> Realisation { get; set; }

public virtual ICollection<Output> Output { get; set; }

}

}

**Output.cs:**

using System.ComponentModel.DataAnnotations;

namespace Lab6.Models.Tables

{

public partial class Output

{

public int Id { get; set; }

[Required(ErrorMessage = "Укажите план выпуска")]

public int OutputPlan { get; set; }

[Required(ErrorMessage = "Укажите факт выпуска")]

public double OutputFact { get; set; }

[Required(ErrorMessage = "Укажите квартал")]

[Range(1, 4, ErrorMessage = "Квартал должен быть равен 1, 2, 3 или 4")]

public short Quarter { get; set; }

[Required(ErrorMessage = "Укажите год")]

[Range(1900, 9999, ErrorMessage = "Год должен быть равен не менее 1900 и не более 9999")]

public short Year { get; set; }

public int OrganizationId { get; set; }

public int ProductId { get; set; }

public virtual Organization Organization { get; set; }

public virtual Product Product { get; set; }

}

}

**Realisation.cs:**

using System.ComponentModel.DataAnnotations;

namespace Lab6.Models.Tables

{

public partial class Realisation

{

public int Id { get; set; }

[Required(ErrorMessage = "Укажите план реализации")]

public double RealisationPlan { get; set; }

[Required(ErrorMessage = "Укажите факт реализации")]

public string RealisationFact { get; set; }

[Required(ErrorMessage = "Укажите квартал")]

[Range(1, 4, ErrorMessage = "Квартал должен быть равен 1, 2, 3 или 4")]

public short Quarter { get; set; }

[Required(ErrorMessage = "Укажите год")]

[Range(1900, 9999, ErrorMessage = "Год должен быть равен не менее 1900 и не более 9999")]

public short Year { get; set; }

public int OrganizationId { get; set; }

public int ProductId { get; set; }

public virtual Organization Organization { get; set; }

public virtual Product Product { get; set; }

}

}

**RationingDbContext.cs:**

using Lab6.Models.Tables;

using Microsoft.EntityFrameworkCore;

namespace Lab6.Models

{

public partial class RationingDbContext : DbContext

{

public RationingDbContext()

{

}

public RationingDbContext(DbContextOptions<RationingDbContext> options)

: base(options)

{

}

public virtual DbSet<Realisation> Realisations { get; set; }

public virtual DbSet<Organization> Organizations { get; set; }

public virtual DbSet<Output> Outputs { get; set; }

public virtual DbSet<Product> Products { get; set; }

}

}

**OrganizationsController.cs:**

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Lab6.Models;

using Lab6.Models.Tables;

namespace Lab6.Controllers

{

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

[ApiController]

public class OrganizationsController : ControllerBase

{

private readonly RationingDbContext \_context;

public OrganizationsController(RationingDbContext context)

{

\_context = context;

}

// GET: api/Organizations

[HttpGet]

public async Task<ActionResult<IEnumerable<Organization>>> GetOrganizations()

{

return await \_context.Organizations.ToListAsync();

}

// GET: api/Organizations/5

[HttpGet("{id}")]

public async Task<ActionResult<Organization>> GetOrganization(int id)

{

var organization = await \_context.Organizations.FindAsync(id);

if (organization == null)

{

return NotFound();

}

return organization;

}

// PUT: api/Organizations/5

[HttpPut("{id}")]

public async Task<ActionResult<Organization>> PutOrganization(int id, Organization organization)

{

if (id != organization.Id)

{

return BadRequest();

}

\_context.Entry(organization).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!OrganizationExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return organization;

}

// POST: api/Organizations

[HttpPost]

public async Task<ActionResult<Organization>> PostOrganization(Organization organization)

{

\_context.Organizations.Add(organization);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetOrganization", new { id = organization.Id }, organization);

}

// DELETE: api/Organizations/5

[HttpDelete("{id}")]

public async Task<ActionResult<Organization>> DeleteOrganization(int id)

{

var organization = await \_context.Organizations.FindAsync(id);

if (organization == null)

{

return NotFound();

}

\_context.Organizations.Remove(organization);

await \_context.SaveChangesAsync();

return organization;

}

private bool OrganizationExists(int id)

{

return \_context.Organizations.Any(e => e.Id == id);

}

}

}

**ProductsController.cs:**

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Lab6.Models;

using Lab6.Models.Tables;

namespace Lab6.Controllers

{

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

[ApiController]

public class ProductsController : ControllerBase

{

private readonly RationingDbContext \_context;

public ProductsController(RationingDbContext context)

{

\_context = context;

}

// GET: api/Products

[HttpGet]

public async Task<ActionResult<IEnumerable<Product>>> GetProducts()

{

return await \_context.Products.ToListAsync();

}

// GET: api/Products/5

[HttpGet("{id}")]

public async Task<ActionResult<Product>> GetProduct(int id)

{

var product = await \_context.Products.FindAsync(id);

if (product == null)

{

return NotFound();

}

return product;

}

// PUT: api/Products/5

[HttpPut("{id}")]

public async Task<ActionResult<Product>> PutProduct(int id, Product product)

{

if (id != product.Id)

{

return BadRequest();

}

\_context.Entry(product).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ProductExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return product;

}

// POST: api/Products

[HttpPost]

public async Task<ActionResult<Product>> PostProduct(Product product)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetProduct", new { id = product.Id }, product);

}

// DELETE: api/Products/5

[HttpDelete("{id}")]

public async Task<ActionResult<Product>> DeleteProduct(int id)

{

var product = await \_context.Products.FindAsync(id);

if (product == null)

{

return NotFound();

}

\_context.Products.Remove(product);

await \_context.SaveChangesAsync();

return product;

}

private bool ProductExists(int id)

{

return \_context.Products.Any(e => e.Id == id);

}

}

}

**OutputsController.cs:**

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Lab6.Models;

using Lab6.Models.Tables;

namespace Lab6.Controllers

{

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

[ApiController]

public class OutputsController : ControllerBase

{

private readonly RationingDbContext \_context;

public OutputsController(RationingDbContext context)

{

\_context = context;

}

// GET: api/Outputs

[HttpGet]

public async Task<ActionResult<IEnumerable<Output>>> GetOutputs()

{

return await \_context.Outputs.ToListAsync();

}

// GET: api/Outputs/5

[HttpGet("{id}")]

public async Task<ActionResult<Output>> GetOutput(int id)

{

var output = await \_context.Outputs.FindAsync(id);

if (output == null)

{

return NotFound();

}

return output;

}

// GET api/Outputs/organizations

[HttpGet("organizations")]

public IEnumerable<Organization> GetOrganizations()

{

return \_context.Organizations.ToList();

}

// GET api/Outputs/products

[HttpGet("products")]

public IEnumerable<Product> GetProducts()

{

return \_context.Products.ToList();

}

// PUT: api/Outputs/5

[HttpPut("{id}")]

public async Task<ActionResult<Output>> PutOutput(int id, Output poutput)

{

if (id != output.Id)

{

return BadRequest();

}

\_context.Entry(output).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!OutputExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return output;

}

// POST: api/Outputs

[HttpPost]

public async Task<ActionResult<Output>> PostOutput(Output output)

{

\_context.Outputs.Add(output);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetOutput", new { id = output.Id }, output);

}

// DELETE: api/Outputs/5

[HttpDelete("{id}")]

public async Task<ActionResult<Output>> DeleteOutput(int id)

{

var output = await \_context.Outputs.FindAsync(id);

if (output == null)

{

return NotFound();

}

\_context.Outputs.Remove(output);

await \_context.SaveChangesAsync();

return output;

}

private bool OutputExists(int id)

{

return \_context.Outputs.Any(e => e.Id == id);

}

}

}

**RealisationsController.cs:**

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Lab6.Models;

using Lab6.Models.Tables;

namespace Lab6.Controllers

{

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

[ApiController]

public class RealisationsController : ControllerBase

{

private readonly RationingDbContext \_context;

public RealisationsController(RationingDbContext context)

{

\_context = context;

}

// GET: api/Realisations

[HttpGet]

public async Task<ActionResult<IEnumerable<Realisation>>> GetRealisations()

{

return await \_context.Realisations.ToListAsync();

}

// GET: api/Realisations/5

[HttpGet("{id}")]

public async Task<ActionResult<Realisation>> GetRealisation(int id)

{

var realization = await \_context.Realisations.FindAsync(id);

if (realisation == null)

{

return NotFound();

}

return realisation;

}

// GET api/Realisations/organizations

[HttpGet("organizations")]

public IEnumerable<Organization> GetOrganizations()

{

return \_context.Organizations.ToList();

}

// GET api/Realisations/products

[HttpGet("products")]

public IEnumerable<Product> GetProducts()

{

return \_context.Products.ToList();

}

// PUT: api/Realisations/5

[HttpPut("{id}")]

public async Task<ActionResult<Realisation>> PutRealisation(int id, Realisation realisation)

{

if (id != realisation.Id)

{

return BadRequest();

}

\_context.Entry(realisation).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!RealisationExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return realisation;

}

// POST: api/Realisations

[HttpPost]

public async Task<ActionResult<Realisation>> Realisation(Realisation realisation)

{

\_context.Realisations.Add(realisation);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetRealisation", new { id = realisation.Id }, realisation);

}

// DELETE: api/Realisations/5

[HttpDelete("{id}")]

public async Task<ActionResult<Realisation>> DeleteRealisation(int id)

{

var realisation = await \_context.Realisations.FindAsync(id);

if (realisation == null)

{

return NotFound();

}

\_context.Realisations.Remove(realisation);

await \_context.SaveChangesAsync();

return realisation;

}

private bool RealisationExists(int id)

{

return \_context.Realisations.Any(e => e.Id == id);

}

}

}