**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 4**

по дисциплине «Разработка приложений баз данных для информационных систем»

на тему: «Разработка моделей и контроллеров ASP.NET MVC приложения баз данных»

Вариант 27

Выполнил: студент гр. ИТП-31

Коркуц С. И.

Принял: доцент

Асенчик О. Д.

Гомель 2020

**Цель работы:** Ознакомиться с возможностями ASP.NET Core MVC и Entity Framework Core для разработки слоя доступа к данным, хранящимся в базе данных, и обработки запросов пользователя посредством контроллеров.

**Задание:** Cоздать с использованием ASP.NET Core MVC Web-приложение, содержащее набор классов, моделирующих предметную область, и осуществляющих генерацию и заполнение тестовыми наборами записей базу данных. Разработать один компонент middleware, контроллеры и представления для выборки и отображения информации из не менее чем 3- таблиц базы данных с использованием механизма внедрение зависимостей.

Для выполнения задания необходимо создать:

* Классы, моделирующие не менее чем три таблицы базы данных согласно вашему варианту. Перечень таблиц предварительно согласовывается с преподавателем. Одна из таблиц обязательно должна находиться на стороне отношения «многие» связи с другой таблицей в схеме базы данных.
* Класс контекста данных.
* Другие классы, например, классы View Model и т.п. (при необходимости).
* Компонент middleware, вызываемый в классе Startup, для инициализации базы данных путем заполнения ее таблиц тестовым набором записей.
* Классы контроллеров (по одному на каждую таблицу базы данных) для обработки обращений пользователя, выборки данных из таблиц и вызова соответствующих представлений для отображения выбранных данных.
* Разработать представления для отображения данных из таблиц, выбранных контроллерами. Представления, работающими с таблицами, стоящими на стороне отношения «многие» в схеме базы данных, должны выводить вместо кодов внешних ключей смысловые значения из связанных таблиц, стоящих на стороне отношения «один».
* Используя предварительно созданный и сконфигурированный в классе Startup профиль кэширования, подключить кэширование вывода для страниц с использованием атрибута ResponseCache для соответствующих методов контроллера. Данные в кэше хранить неизменными в течение 2\*N+240 секунд, где N- номер вашего варианта.
* С использованием средств разработчика браузера (Chrome, Firefox) продемонстрировать ускорение обработки запроса при наличии кэширования с использованием атрибута ResponseCache.
* Для проверки преподавателем следует разместить разработанный проект на GitHub.

**Ход выполнения:**

Сначала были созданы модели базы данных. Они были помещены в папку *Models*.

Далее был создан класс контекста базы данных. Он был помещён в папку *Data*.

После этого был написан класс, роль которого заключается в заполнении базы данных начальными записями. Он заполняет таблицу базы данных только в том случае, если в ней нет записей.

Потом был написан компонент промежуточного слоя, который создаёт класс для инициализации базы данных и инициализирует её. Он был добавлен в метод *Configure* код класса *Startup*.

Далее были созданы классы контроллеров. Они все были помещены в папку *Controllers*.

Для всех контроллеров были созданы папки в папке *Views*. В каждом контроллере есть метод *Index*. И поэтому в каждой папке с представлениями были созданы соответствующие представления для этих методов.

Эти представления просто выполняют вывод таблиц базы данных в *html* страницу.

Для каждого метода *Index* был прописан атрибут *ResponseCache* с длительностью кэширования 254 секунды и хранением в любой области кеша.

На рисунках 1 – 3 изображены все доступные *html* страницы.

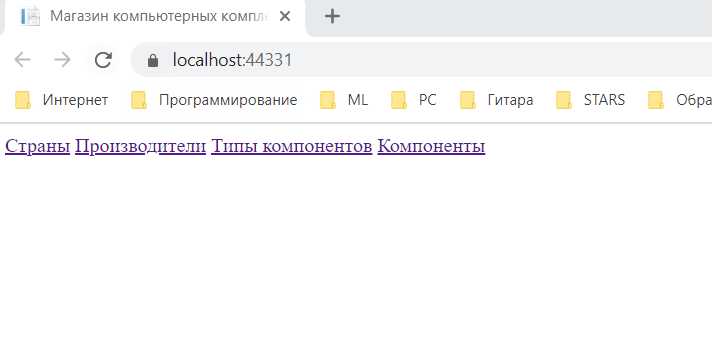


Рисунок 1 – Корневой каталог веб-приложения

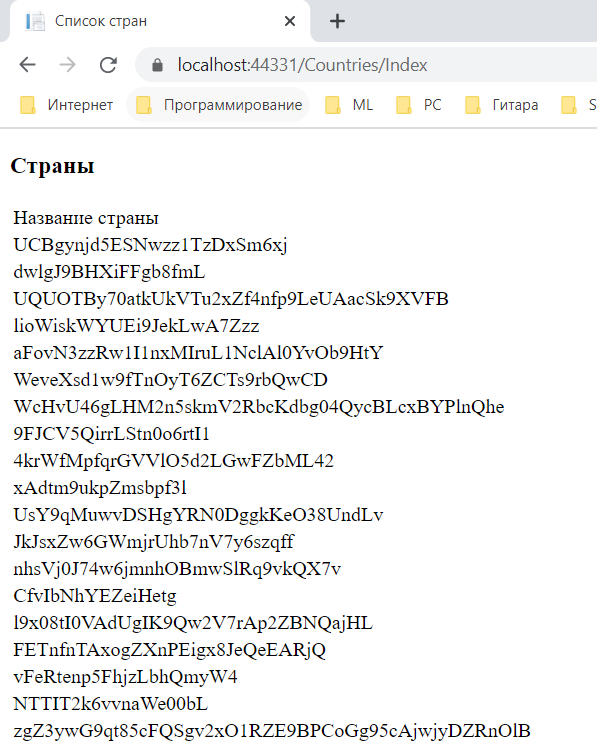


Рисунок 2 – Вывод приложения при нажатии на вкладку *Страны*

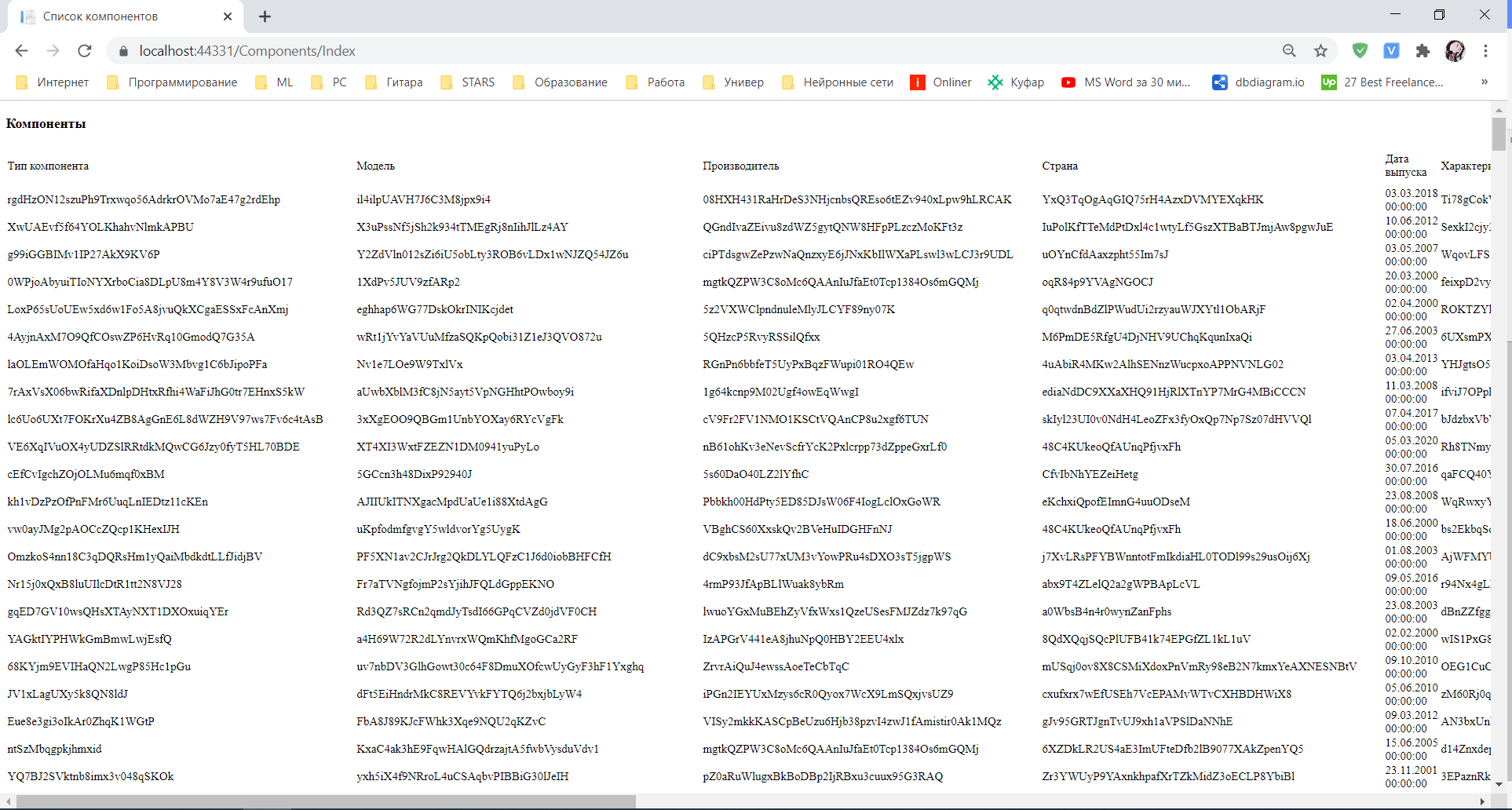


Рисунок 3 – Вывод страницы при нажатии на вкладку компоненты

Можно заметить что все идентификационные поля при выводе компонентов заменены на наименования этих полей.

Структура созданного приложения изображена на рисунке 4.

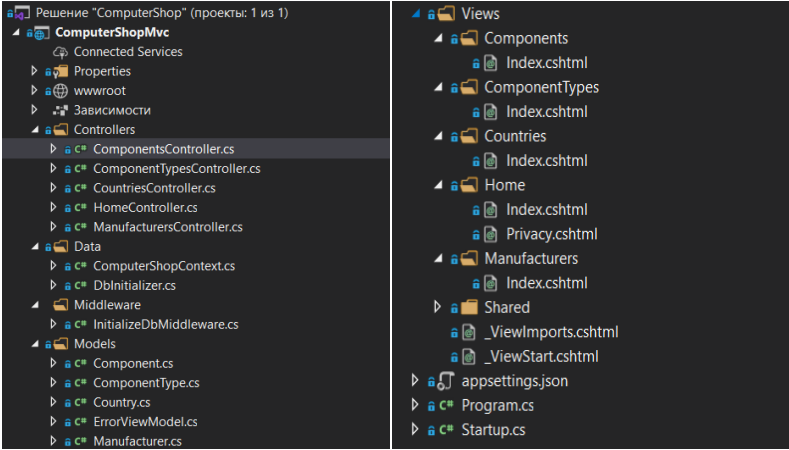


Рисунок 4 – Структура созданного приложения

На гитхабе был создан репозиторий для проекта. Ссылка <https://github.com/korkuts-stanislau/5semIS/tree/master/lab4.>

**Вывод:** В ходе выполнения был создан проект ASP.NET CORE MVC. Были созданы модели, контроллеры и представления. Был создан компонент промежуточного слоя для заполнения базы данных.

**ПРИЛОЖЕНИЕ А**

Листинг созданных классов

Листинг класса *Startup.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ComputerShopMvc.Middleware;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

namespace ComputerShopMvc

{

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)

{

string connectionString = Configuration.GetConnectionString("DefaultConnection");

services.AddDbContext<Data.ComputerShopContext>(options => options.UseSqlServer(connectionString));

services.AddControllersWithViews();

}

// 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();

}

else

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseDbInitializer();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}

Листинг класса *DbInitializer.cs*

using Microsoft.AspNetCore.Authentication;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ComputerShopMvc.Data

{

public class DbInitializer

{

int \_refferenceTableSize;

int \_operationalTableSize;

public DbInitializer(int refferenceTableSize, int operationalTableSize)

{

\_refferenceTableSize = refferenceTableSize;

\_operationalTableSize = operationalTableSize;

}

public void Initialize(ComputerShopContext dbContext)

{

Random rand = new Random();

if (!dbContext.Countries.Any())

{

for(int i = 0; i < \_refferenceTableSize; i++)

{

dbContext.Countries.Add(new Models.Country

{

CountryName = GetRandomString(50)

});

}

}

dbContext.SaveChanges();

if (!dbContext.Manufacturers.Any())

{

for (int i = 0; i < \_refferenceTableSize; i++)

{

dbContext.Manufacturers.Add(new Models.Manufacturer

{

ManufacturerName = GetRandomString(50)

});

}

}

dbContext.SaveChanges();

if (!dbContext.ComponentTypes.Any())

{

for (int i = 0; i < \_refferenceTableSize; i++)

{

dbContext.ComponentTypes.Add(new Models.ComponentType

{

ComponentTypeName = GetRandomString(50),

ComponentTypeDescription = GetRandomString(150)

});

}

}

dbContext.SaveChanges();

if (!dbContext.Components.Any())

{

var componentTypes = dbContext.ComponentTypes.ToList();

var manufacturers = dbContext.Manufacturers.ToList();

var countries = dbContext.Countries.ToList();

for (int i = 0; i < \_operationalTableSize; i++)

{

var componentType = componentTypes.ElementAt(rand.Next(dbContext.ComponentTypes.Count() - 1));

var manufacturer = manufacturers.ElementAt(rand.Next(dbContext.Manufacturers.Count() - 1));

var country = countries.ElementAt(rand.Next(dbContext.Countries.Count() - 1));

dbContext.Components.Add(new Models.Component

{

ComponentModel = GetRandomString(50),

ComponentReleaseDate = GetRandomDate(new DateTime(2000, 1, 1), DateTime.Now),

ComponentCharacteristics = GetRandomString(150),

ComponentWarrantyPeriodInMonths = rand.Next(6, 36),

ComponentDescription = GetRandomString(150),

ComponentPrice = rand.Next(200, 3000),

ComponentType = componentType,

ComponentTypeId = componentType.ComponentTypeId,

Manufacturer = manufacturer,

ManufacturerId = manufacturer.ManufacturerId,

Country = country,

CountryId = country.CountryId

});

}

}

dbContext.SaveChanges();

}

public string GetRandomString(int maxLength)

{

Random rand = new Random();

int length = rand.Next(maxLength / 3, maxLength);

var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

var str = new char[length];

for (int i = 0; i < length; i++)

{

str[i] = chars[rand.Next(chars.Length)];

}

return new string(str);

}

public DateTime GetRandomDate(DateTime minDate, DateTime maxDate)

{

Random rand = new Random();

int range = (maxDate - minDate).Days;

return minDate.AddDays(rand.Next(range));

}

}

}

Листинг класса *ComponentController.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

namespace ComputerShopMvc.Controllers

{

public class ComponentsController : Controller

{

Data.ComputerShopContext \_dbContext;

public ComponentsController(Data.ComputerShopContext dbContext)

{

\_dbContext = dbContext;

}

[ResponseCache(Location = ResponseCacheLocation.Any, Duration = 254)]

public IActionResult Index()

{

return View(\_dbContext.Components.Include(component => component.ComponentType)

.Include(component => component.Country)

.Include(component => component.Manufacturer)

.Take(500)

.ToList());

}

}

}

Листинг класса *InitializeDbMiddleware*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ComputerShopMvc.Data;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

namespace ComputerShopMvc.Middleware

{

// You may need to install the Microsoft.AspNetCore.Http.Abstractions package into your project

public class InitializeDbMiddleware

{

private readonly RequestDelegate \_next;

public InitializeDbMiddleware(RequestDelegate next)

{

\_next = next;

}

public Task Invoke(HttpContext httpContext, ComputerShopContext dbContext)

{

DbInitializer initializer = new DbInitializer(500, 5000);

initializer.Initialize(dbContext);

return \_next(httpContext);

}

}

// Extension method used to add the middleware to the HTTP request pipeline.

public static class InitializeDbMiddlewareExtensions

{

public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)

{

return builder.UseMiddleware<InitializeDbMiddleware>();

}

}

}

Листинг класса *Index.cshtml* из папки *Views/Components*

@model IEnumerable<ComputerShopMvc.Models.Component>

@{

ViewData["title"] = "Список компонентов";

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<title>Список компонентов</title>

</head>

<body>

<h3>Компоненты</h3>

<table>

<tr>

<td>Тип компонента</td>

<td>Модель</td>

<td>Производитель</td>

<td>Страна</td>

<td>Дата выпуска</td>

<td>Характеристики</td>

<td>Гарантия в месяцах</td>

<td>Описание</td>

<td>Цена</td>

</tr>

@foreach (var component in Model)

{

<tr>

<td>@component.ComponentType.ComponentTypeName</td>

<td>@component.ComponentModel</td>

<td>@component.Manufacturer.ManufacturerName</td>

<td>@component.Country.CountryName</td>

<td>@component.ComponentReleaseDate</td>

<td>@component.ComponentCharacteristics</td>

<td>@component.ComponentWarrantyPeriodInMonths</td>

<td>@component.ComponentDescription</td>

<td>@component.ComponentPrice</td>

</tr>

}

</table>

</body>

</html>