**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 3**

по дисциплине «Разработка приложений баз данных для информационных систем»

на тему: «Обработка HTTP запросов средствами ASP.NET Core. Сохранение состояния. Кэширование»

Вариант 27

Выполнил: студент гр. ИТП-31

Коркуц С. И.

Принял: доцент

Асенчик О. Д.

Гомель 2020

**Цель работы:** Ознакомиться c методами обработкой HTTP средствами ASP.NET Core, методами сохранения состояния приложения и повышение производительности приложений путем использования разных видов кэширования.

**Задание:** Используя ранее разработанные объектную модель для доступа к данным в заданной предметной области разработать простое ASP.NET Core приложение.

1. С использование методов Run, Map и Use разработать:
   1. компоненты промежуточного уровня (middleware) и встроить их в конвейер обработки HTTP запроса с целью кэширования 20 записей из каждой таблицы базы данных заданной предметной области с помощью встроенного инструмента кэширования - объекта IMemoryCache. Данные в кэше хранить неизменными в течение 2\*N+240 секунд, где N- номер вашего варианта.
   2. собственную систему маршрутизации входящих запросов:

* если URL адрес входящего запроса содержит \info – выводить в выходной поток для отображения браузером информацию о клиенте и выходить из конвейера обработки запроса;
* если URL адрес входящего запроса содержит \table (где table – имя таблицы из базы данных) – выводить в выходной поток для отображения браузером с использование метода Response.WriteAsync кэшированную информацию из соответствующей таблицы базы данных и выходить из конвейера обработки запроса;
* если URL адрес входящего запроса содержит \searchform1 или \searchform2 – выводить в выходной поток для отображения браузером с использование метода Response.WriteAsync формы для поиска информации из базы данных и выходить из конвейера обработки запроса; 0

форма должна содержать, как минимум: одно поле, одного поле со списком, один список, одну кнопку;

* в противном случае (URL адрес входящего запроса не содержит перечисленных выше элементов) - продолжать обрабатывать другие компоненты конвейера обработки запросов и передавать управление системе маршрутизации MVC фреймворка;

1. Реализовать сохранение состояния элементов одной формы одной страницы с использованием куки.
2. Реализовать сохранение состояния элементов одной формы одной страницы в виде одного объекта специальной структуры с использованием объекта Session.
3. Осуществить заполнение элементов формы при их загрузке данными ранее сохранненными в объекте Session и куки.
4. С использованием средств разработчика браузера (Chrome, Firefox) продемонстрировать ускорение обработки запроса при наличии кэширования с использованием MemoryCache.

**Ход выполнения:**

Через *nuget* были установлены пакеты *entity framework core*, *entity framework tools*, *entity framework sqlserver*. Далее при помощи команды

*Scaffold-DbContext "Data Source=DESKTOP-V3UDE6A;Integrated Security=True;Initial Catalog=ComputerShop;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False" Microsoft.EntityFrameworkCore.SqlServer*

были созданы модели сущностей базы данных и класс для управления ими *ComputerShopContext*.

Модели были переименованы и распределены по папкам.

Структура созданного приложения на рисунке 1.

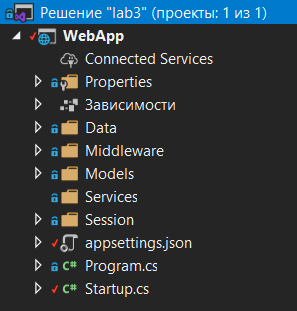


Рисунок 1 – Структура созданного приложения

Далее был создан компонент *CachingMiddleware.cs* для кеширования 20 записей трёх таблиц: *Services, Customers, Components*.

Он был добавлен в метод *Configure* класса *Startup* как расширения интерфейса *IApplicationBuilder*.

В начале обработки запроса он смотрит ключи в сессии, и если там нет ключей, обозначающих эти таблицы, то подтягивает их из базы данных.

После этого при помощи метода *Map* интерфейса *IApplicationBuilder* была настроена маршрутизация приложения.

Были созданы обработчики запросов к четырём страницам корневому каталогу, *info, services* и *searchform*.

При обращении к корневому каталогу в браузер выводится список из трёх страниц, при нажатии на элемент которого, пользователь переходит на эти страницы(рисунок 2).

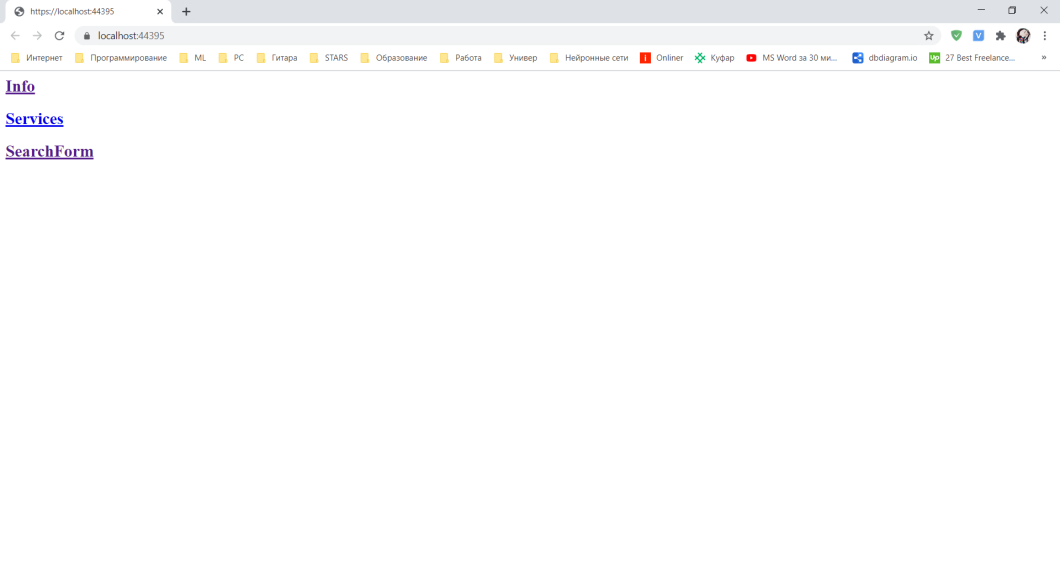


Рисунок 2 – Результат при обращении к корневому каталогу

При переходе на страницу *info* приложение из контекста получает данные о пользователе и выводит их на экран приложения(рисунок 3).

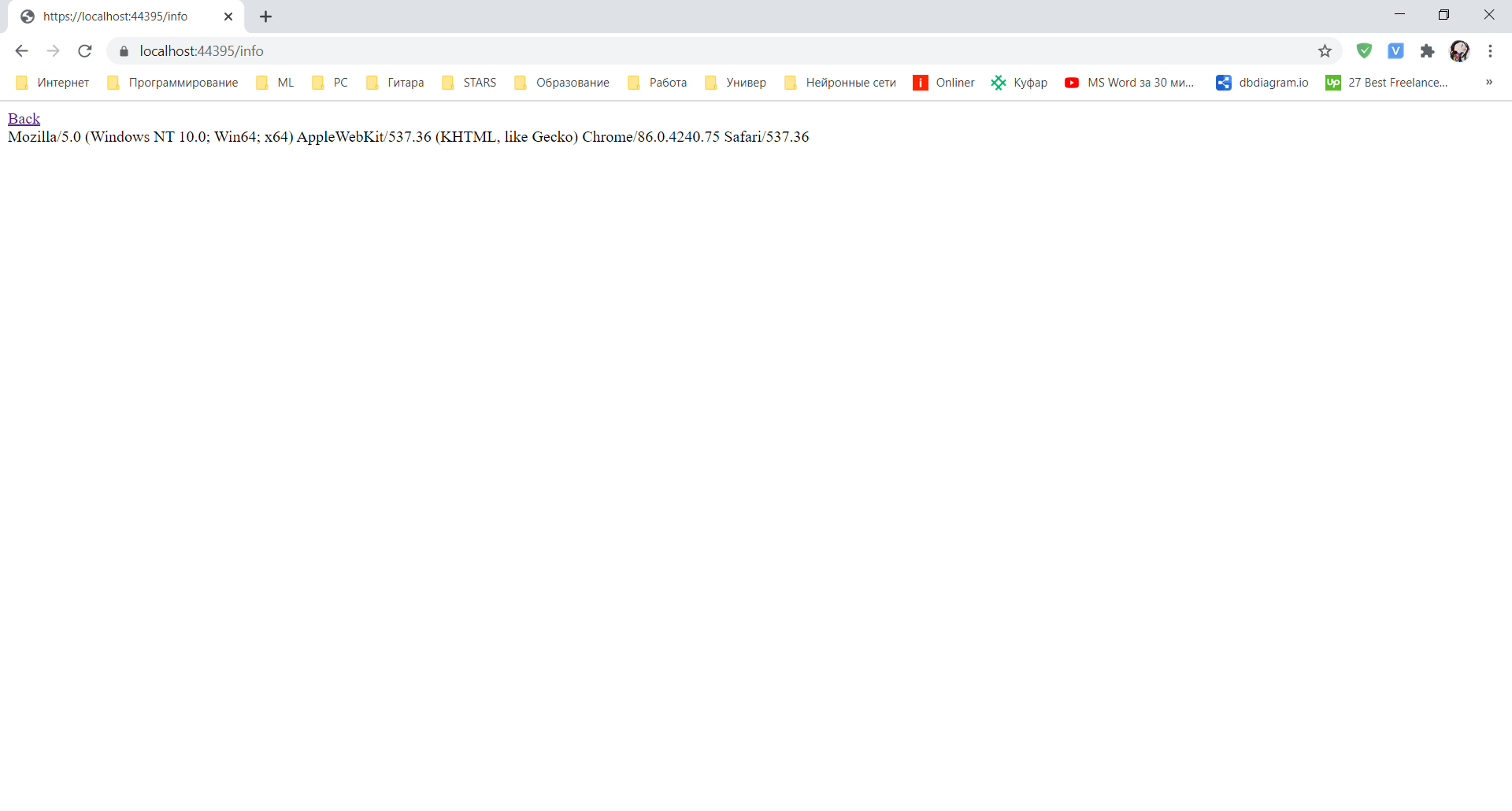


Рисунок 3 – Результат перехода на страницу *info*

При переходе на страницу *services* приложение берёт данные из кеша и выводит записи пользователю(рисунок 4).



Рисунок 4 – Результат перехода на страницу *services*

Далее было написано расширение для работы с сессией. При помощи него можно сохранять данные в сессии. Это расширение необходимо что бы не обращаться постоянно к базе данных, если пользователь уже выполнял определённый запрос.

Результат перехода на страницу *searchform* изображён на рисунке 5. После перехода на странице находится поле выбора таблицы, в которой будет совершаться поиск. Также есть два поля для ввода параметров для разных таблиц. Для таблицы *Customers* находится поле, которое позволяет пользователю выбрать заказчика, для которого скидка меньше какого-либо числа. Для таблицы *Components* мы находим компоненты, в имени типов которых есть определённая последовательность символов.

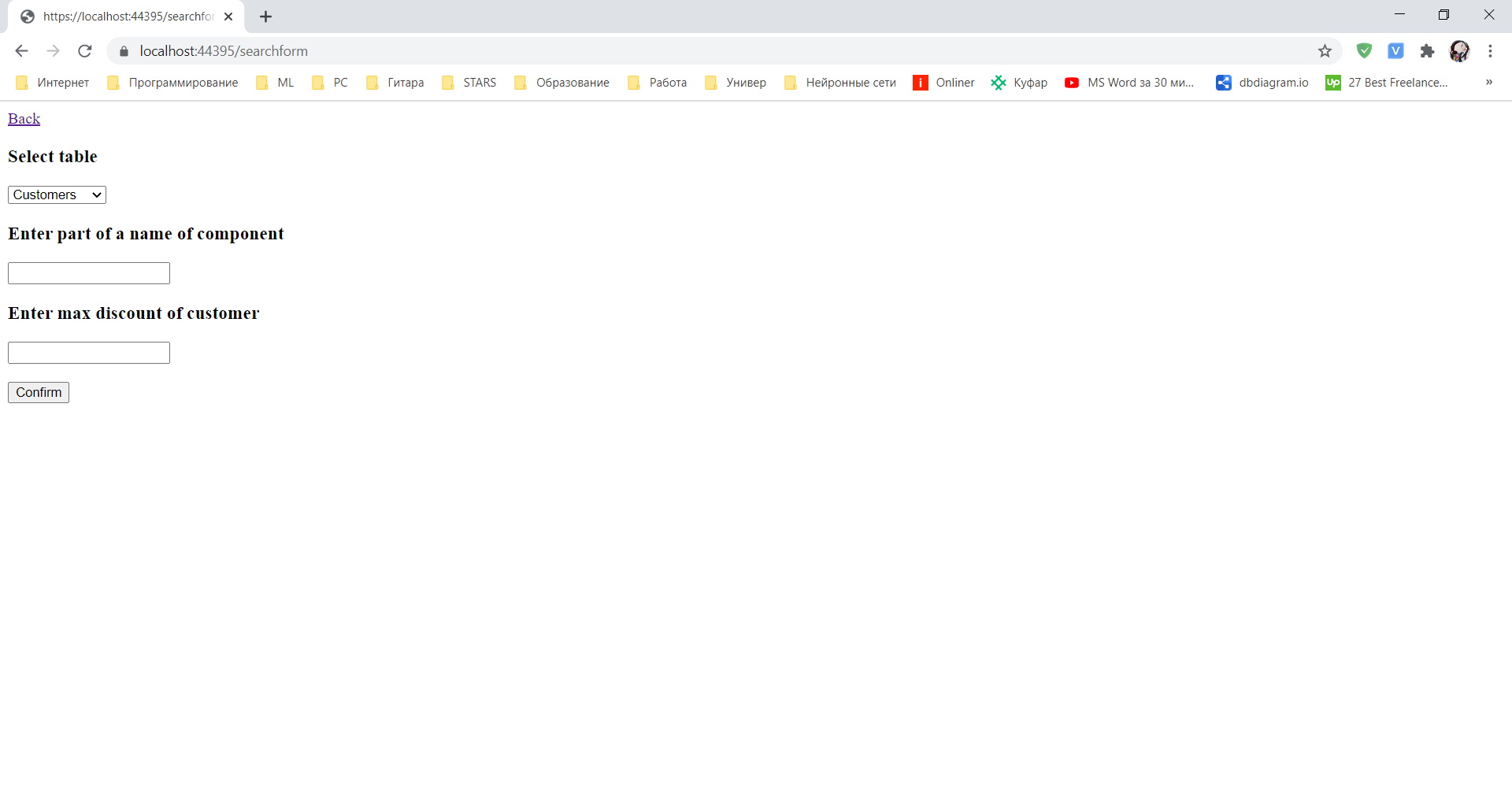


Рисунок 5 – Результат перехода на страницу *searchform*

Если в куки есть сохранённые данные, то вместо пустых полей выведутся последние введённые значения.

При выборе таблицы и после ввода значений в поля фильтры можно нажать на кнопку *confirm*. После нажатия приложение проверяет есть ли данные по этому запросу в сессии, если есть отображает их, если нет – берёт из базы данных (рисунок 6).

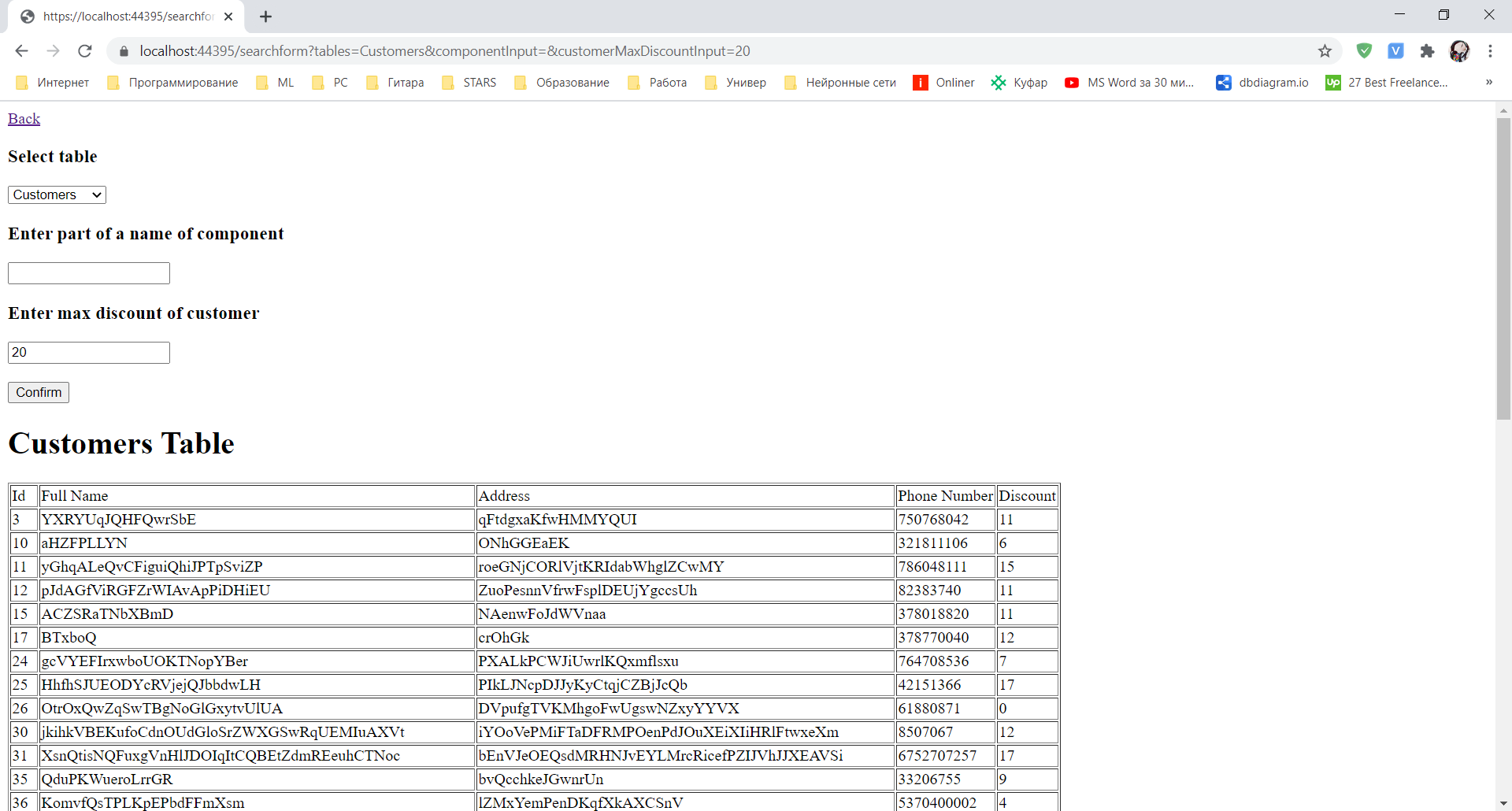


Рисунок 6 – Результат нажатия на кнопку *confirm*

При выходе из этой страницы и повторном заходе, данные в поле сохранятся (рисунки 7-8).

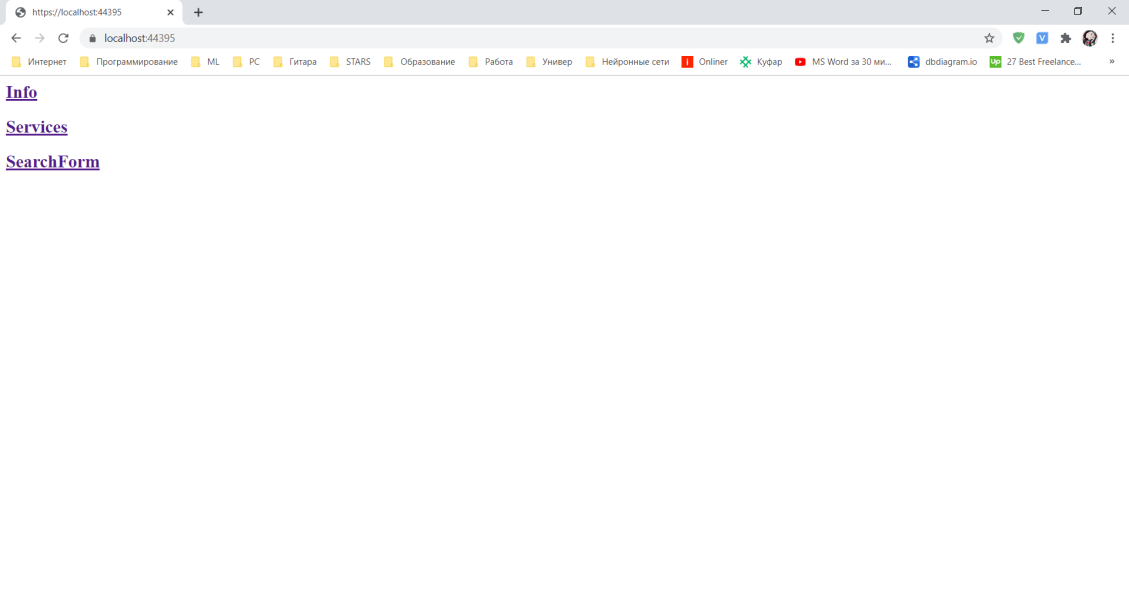


Рисунок 7 – Результат возвращения на корневой сайт

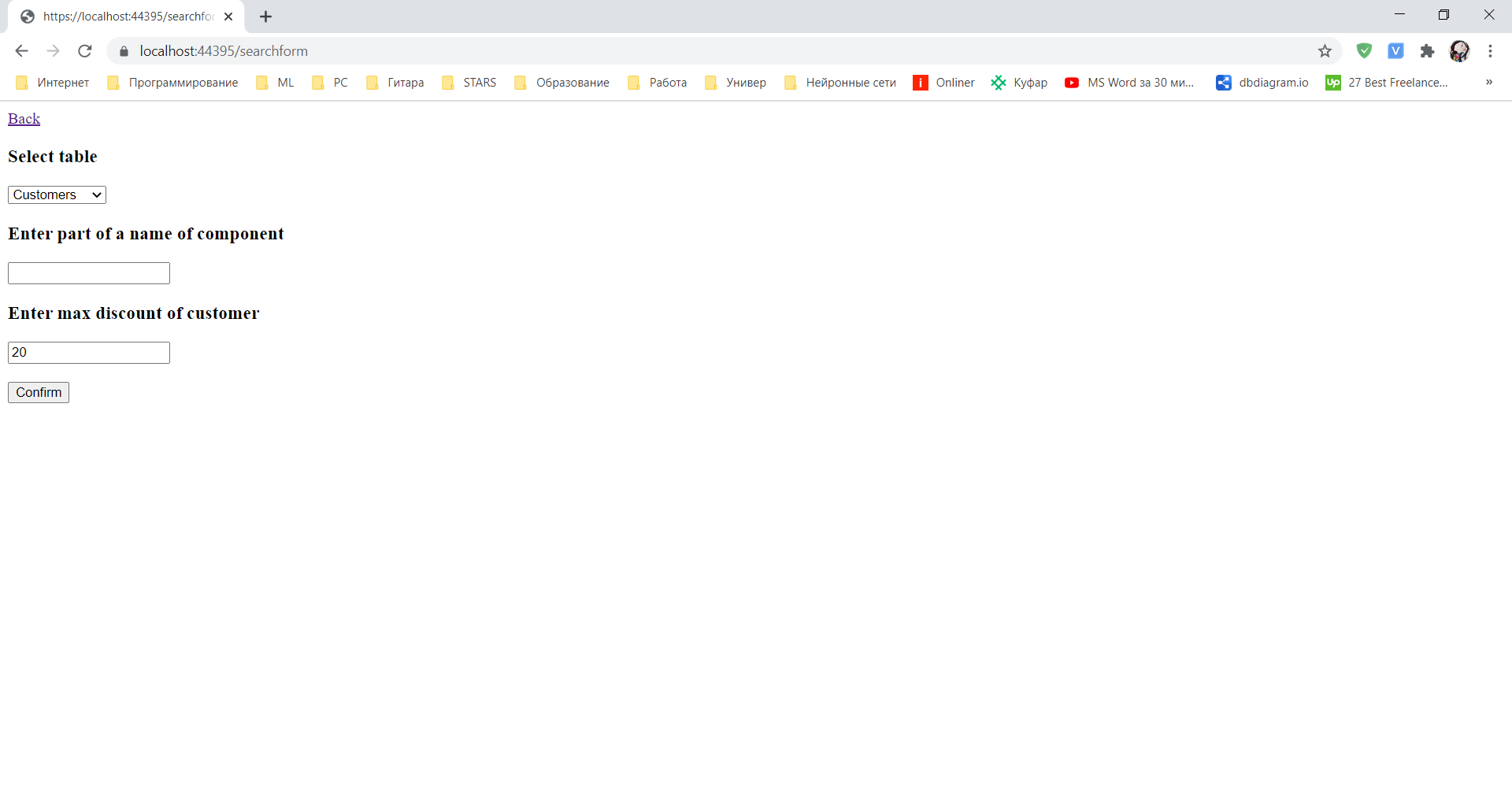


Рисунок 8 – Результат повторного перехода на форму поиска

**Вывод:**

В ходе работы было разработано веб-приложение для работы с базой данных. Были изучены механизмы кеширования, сохранения данных в сессии и в куки.

**ПРИЛОЖЕНИЕ А**

Листинг созданных классов

*CachingMiddleware.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Caching.Memory;

namespace WebApp

{

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

public class CachingMiddleware

{

private readonly RequestDelegate \_next;

public CachingMiddleware(RequestDelegate next)

{

\_next = next;

}

public Task Invoke(HttpContext httpContext, IMemoryCache cache, ComputerShopContext dbContext)

{

var services = dbContext.Services.Take(20).ToList();

if (services != null)

{

cache.Set("services", services,

new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(2 \* 7 + 240)));

}

var customers = dbContext.Customers.Take(20).ToList();

if (customers != null)

{

cache.Set("customers", customers,

new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(2 \* 7 + 240)));

}

var components = dbContext.Components.Take(20).ToList();

if (components != null)

{

cache.Set("components", components,

new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(2 \* 7 + 240)));

}

return \_next(httpContext);

}

}

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

public static class CachingMiddlewareExtensions

{

public static IApplicationBuilder UseCachingMiddleware(this IApplicationBuilder builder)

{

return builder.UseMiddleware<CachingMiddleware>();

}

}

}

*SessionExtension.cs*

using Microsoft.AspNetCore.Http;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace WebApp

{

public static class SessionExtension

{

public static void Set<T>(this ISession session, string key, T value)

{

session.SetString(key, JsonConvert.SerializeObject(value));

}

public static T Get<T>(this ISession session, string key)

{

var value = session.GetString(key);

return value == null ? default(T) : JsonConvert.DeserializeObject<T>(value);

}

}

}

*Startup.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.Extensions.Configuration;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using System.IO;

using Microsoft.EntityFrameworkCore;

using System.Text;

using Microsoft.Extensions.Caching.Memory;

using System.Net;

namespace WebApp

{

public class Startup

{

// This method gets called by the runtime. Use this method to add services to the container.

// For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940

public Startup()

{

}

public void ConfigureServices(IServiceCollection services)

{

//Подключаем базу данных к сервисам

var builder = new ConfigurationBuilder();

builder.SetBasePath(Directory.GetCurrentDirectory());

builder.AddJsonFile("appsettings.json");

var config = builder.Build();

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

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

//Добавление кеширования

services.AddMemoryCache();

services.AddDistributedMemoryCache();

//Добавление сессий

services.AddSession();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ComputerShopContext db)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseStaticFiles();

app.UseSession();

app.UseCachingMiddleware();

app.Map("/info", Info);

app.Map("/services", Services);

app.Map("/searchform", SearchForm);

app.UseEndpoints(endpoints =>

{

endpoints.MapGet("/", async context =>

{

StringBuilder htmlBuilder = new StringBuilder();

htmlBuilder.Append("<a href=\"/info\"><h2>Info</h2></a>");

htmlBuilder.Append("<a href=\"/services\"><h2>Services</h2></a>");

htmlBuilder.Append("<a href=\"/searchform\"><h2>SearchForm</h2></a>");

await context.Response.WriteAsync(htmlBuilder.ToString());

});

});

}

public void Info(IApplicationBuilder app)

{

app.Run(async context =>

{

StringBuilder htmlBuilder = new StringBuilder();

htmlBuilder.Append("<a href=\"/\"<h2>Back</h2></a></br>");

htmlBuilder.Append(context.Request.Headers["User-Agent"]);

await context.Response.WriteAsync(htmlBuilder.ToString());

});

}

public void Services(IApplicationBuilder app)

{

app.Run(async context =>

{

StringBuilder htmlBuilder = new StringBuilder();

htmlBuilder.Append("<a href=\"/\"<h2>Back</h2></a></br>");

IEnumerable<Service> services = (IEnumerable<Service>)context.RequestServices.GetService<IMemoryCache>().Get("services");

htmlBuilder.Append(GetServicesHtmlTableFromEnumerable(services));

await context.Response.WriteAsync(htmlBuilder.ToString());

});

}

public void SearchForm(IApplicationBuilder app)

{

app.Run(async context =>

{

string component, customerMaxDiscount;

GetFieldValuesFromCookies(context, out component, out customerMaxDiscount);

StringBuilder htmlBuilder = new StringBuilder();

htmlBuilder.Append("<a href=\"/\"<h2>Back</h2></a></br>");

htmlBuilder.Append("<form action=\"/searchform\">" +

"<h3>Select table</h3>" +

"<select name=\"tables\">" +

"<option>Customers</option>" +

"<option>Components</option>" +

"</select>" +

"<h3>Enter part of a name of component</h3>" +

$"<input name=\"componentInput\" value=\"{component}\">" +

"<h3>Enter max discount of customer</h3>" +

$"<input name=\"customerMaxDiscountInput\" value=\"{customerMaxDiscount}\">" +

"<br><br>" +

"<button type='submit'>Confirm</button>" +

"</form>");

if (context.Request.Query["componentInput"] != "" && context.Request.Query["tables"] == "Components")

{

var components = context.Session.Get<List<Component>>("components" + context.Request.Query["componentInput"]) ??

context.RequestServices.GetService<ComputerShopContext>().Components

.Where(component => component.ComponentType.ComponentTypeName.Contains(context.Request.Query["componentInput"]))

.Take(50)

.ToList();

htmlBuilder.Append(MakeComponentsTableFromEnumerable(components));

context.Session.Set("components" + context.Request.Query["componentInput"], components);

}

else if (context.Request.Query["customerMaxDiscountInput"] != "" && context.Request.Query["tables"] == "Customers")

{

var customers = context.Session.Get<List<Customer>>("customers" + context.Request.Query["customerMaxDiscountInput"]) ??

context.RequestServices.GetService<ComputerShopContext>().Customers

.Where(customer => customer.CustomerDiscount < int.Parse(context.Request.Query["customerMaxDiscountInput"]))

.Take(50)

.ToList();

htmlBuilder.Append(MakeCustomersHtmlTableFromEnumerable(customers));

context.Session.Set("customers" + context.Request.Query["customerMaxDiscountInput"], customers);

}

await context.Response.WriteAsync(htmlBuilder.ToString());

});

}

private static void GetFieldValuesFromCookies(HttpContext context, out string component, out string customerMaxDiscount)

{

component = context.Request.Query["componentInput"];

customerMaxDiscount = context.Request.Query["customerMaxDiscountInput"];

if (component != null)

{

context.Response.Cookies.Append("componentType", component);

}

else if (context.Request.Cookies["componentType"] != null)

{

component = context.Request.Cookies["componentType"];

}

if (customerMaxDiscount != null)

{

context.Response.Cookies.Append("customerMaxDiscount", customerMaxDiscount);

}

else if (context.Request.Cookies["customerMaxDiscount"] != null)

{

customerMaxDiscount = context.Request.Cookies["customerMaxDiscount"];

}

}

private string GetServicesHtmlTableFromEnumerable(IEnumerable<Service> services)

{

StringBuilder builder = new StringBuilder();

builder.Append("<H1>Services Table</H1>" +

"<TABLE BORDER=1>");

builder.Append("<TD>Id</TD>");

builder.Append("<TD>Service name</TD>");

builder.Append("<TD>Service description</TD>");

builder.Append("<TD>Service Price</TD>");

foreach (var service in services)

{

builder.Append("<TR>");

builder.Append("<TD>" + service.ServiceId + "</TD>");

builder.Append("<TD>" + service.ServiceName + "</TD>");

builder.Append("<TD>" + service.ServiceDescription + "</TD>");

builder.Append("<TD>" + service.ServicePrice + "</TD>");

builder.Append("</TR>");

}

builder.Append("</TABLE>");

return builder.ToString();

}

private string MakeCustomersHtmlTableFromEnumerable(IEnumerable<Customer> customers)

{

StringBuilder builder = new StringBuilder();

builder.Append("<H1>Customers Table</H1>" +

"<TABLE BORDER=1>");

builder.Append("<TD>Id</TD>");

builder.Append("<TD>Full Name</TD>");

builder.Append("<TD>Address</TD>");

builder.Append("<TD>Phone Number</TD>");

builder.Append("<TD>Discount</TD>");

foreach (var customer in customers)

{

builder.Append("<TR>");

builder.Append("<TD>" + customer.CustomerId + "</TD>");

builder.Append("<TD>" + customer.CustomerFullName + "</TD>");

builder.Append("<TD>" + customer.CustomerAddress + "</TD>");

builder.Append("<TD>" + customer.CustomerPhoneNumber + "</TD>");

builder.Append("<TD>" + customer.CustomerDiscount + "</TD>");

builder.Append("</TR>");

}

builder.Append("</TABLE>");

return builder.ToString();

}

private string MakeComponentsTableFromEnumerable(IEnumerable<Component> components)

{

StringBuilder builder = new StringBuilder();

builder.Append("<H1>Components Table</H1>" +

"<TABLE BORDER=1>");

builder.Append("<TD>Id</TD>");

builder.Append("<TD>ComponentTypeId</TD>");

builder.Append("<TD>Model</TD>");

builder.Append("<TD>ManufacturerId</TD>");

builder.Append("<TD>CountryId</TD>");

builder.Append("<TD>Release Date</TD>");

builder.Append("<TD>Characteristics</TD>");

builder.Append("<TD>WarrantyPeriodInMonths</TD>");

builder.Append("<TD>Description</TD>");

builder.Append("<TD>Price</TD>");

foreach (var component in components)

{

builder.Append("<TR>");

builder.Append("<TD>" + component.ComponentId + "</TD>");

builder.Append("<TD>" + component.ComponentTypeId + "</TD>");

builder.Append("<TD>" + component.ComponentModel + "</TD>");

builder.Append("<TD>" + component.ComponentManufacturerId + "</TD>");

builder.Append("<TD>" + component.ComponentCountryId + "</TD>");

builder.Append("<TD>" + component.ComponentReleaseDate + "</TD>");

builder.Append("<TD>" + component.ComponentCharacteristics + "</TD>");

builder.Append("<TD>" + component.ComponentWarrantyInMonths + "</TD>");

builder.Append("<TD>" + component.ComponentDescription + "</TD>");

builder.Append("<TD>" + component.ComponentPrice + "</TD>");

builder.Append("</TR>");

}

builder.Append("</TABLE>");

return builder.ToString();

}

}

}