**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 2

Выполнила: студентка гр. ИТП-31

Гайкевич Т. Г.

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:

*dotnet ef dbcontext Scaffold "Server=<servername>,1433;Initial Catalog=<dbName>;Persist Security Info=False;User ID=<userID>;Password=<password>;MultipleActiveResultSets=False;Encrypt=*

*True;TrustServerCertificate=False;Connection Timeout=30;"Microsoft.EntityFrameworkCore.SqlServer*

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

Модели были переименованы и распределены по папкам.

Структура созданного приложения на рисунке 1.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Структура созданного приложения

Далее был создан компонент *Middleware.cs* для кеширования 20 записей двух таблиц: *Services, Clients*.

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).

Изображение выглядит как текст

Автоматически созданное описание

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).

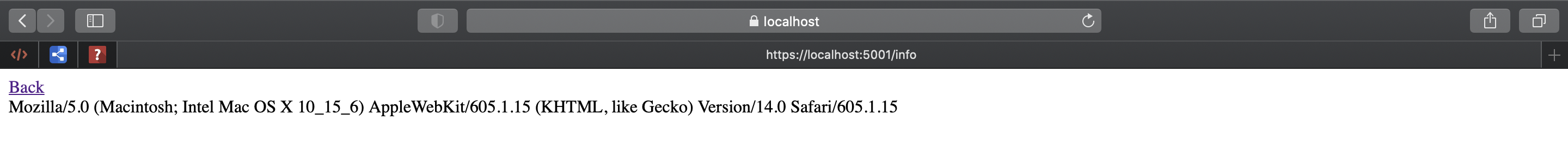


Рисунок 3 – Результат перехода на страницу *info*

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

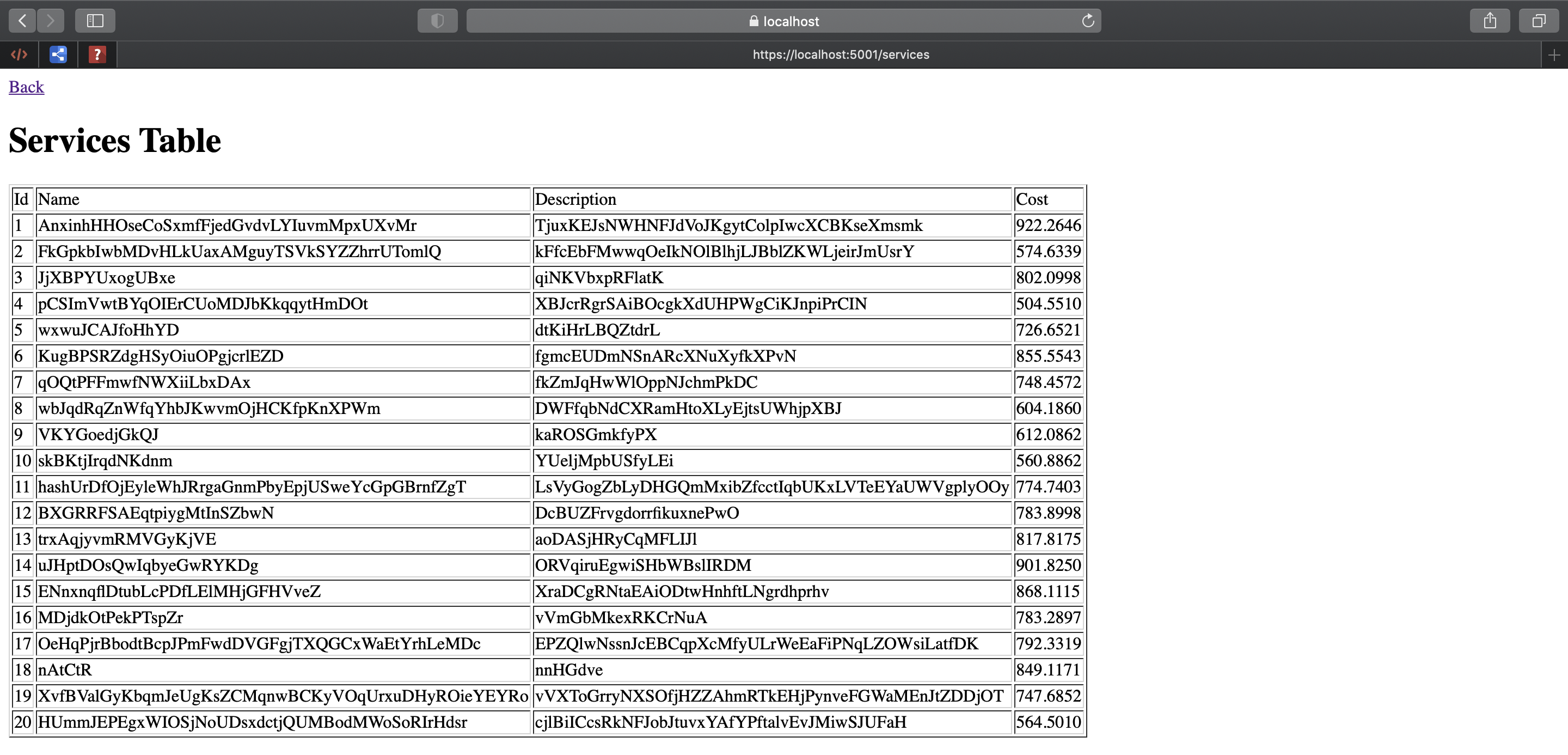


Рисунок 4 – Результат перехода на страницу *services*

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 *Services* находится поле, которое позволяет пользователю выбрать услугу, со стоимостью меньше какого-либо числа. Для таблицы *Clients* мы находим компоненты, в имени которых есть определённая последовательность символов.

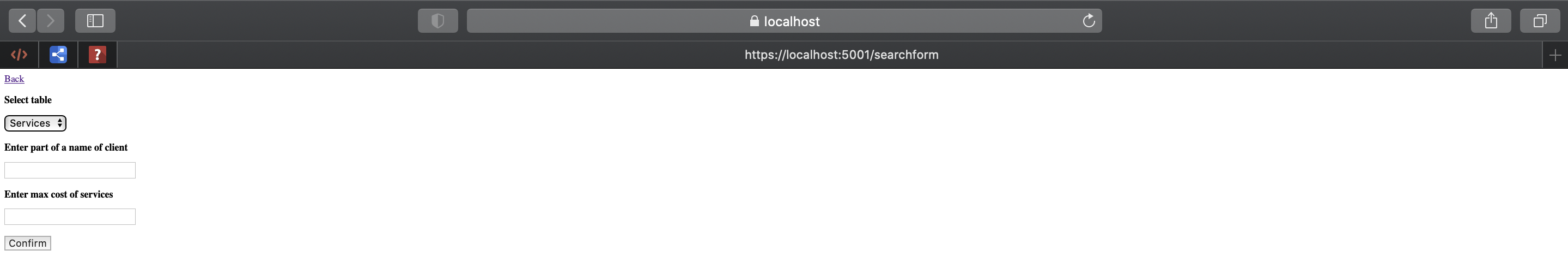


Рисунок 5 – Результат перехода на страницу *searchform*

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).

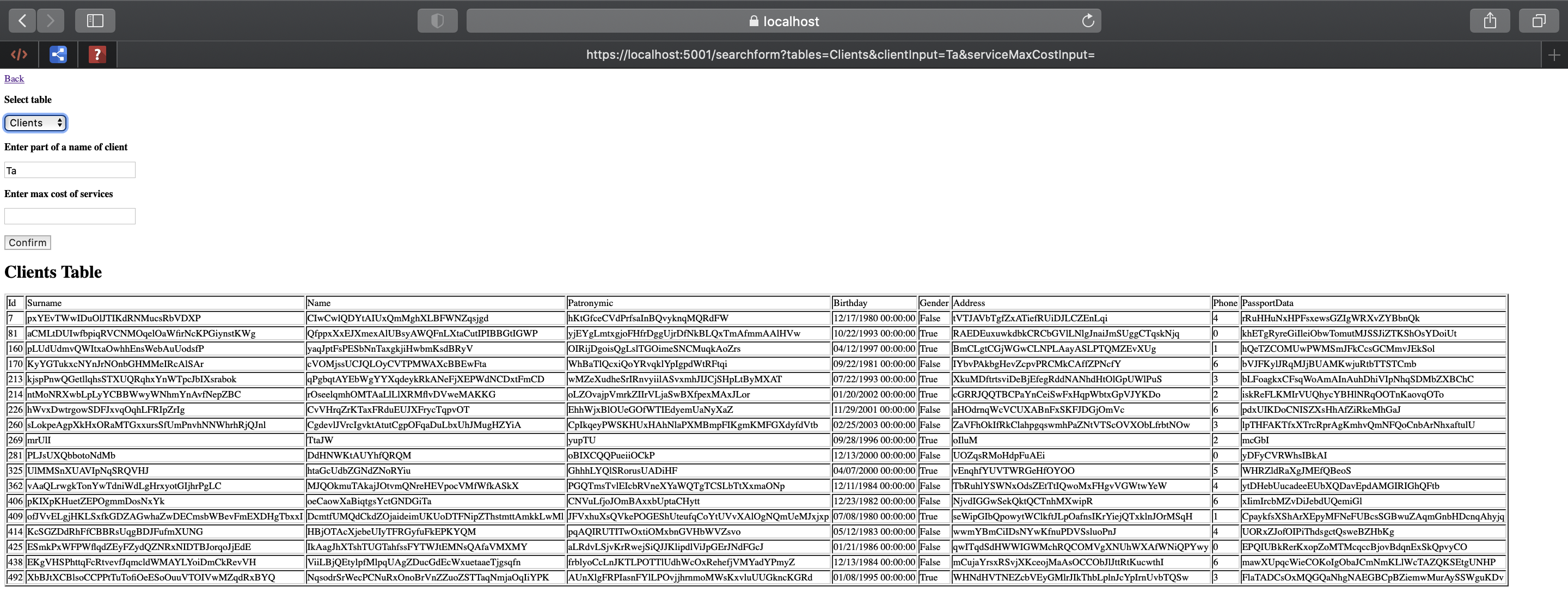


Рисунок 6 – Результат нажатия на кнопку *confirm*

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

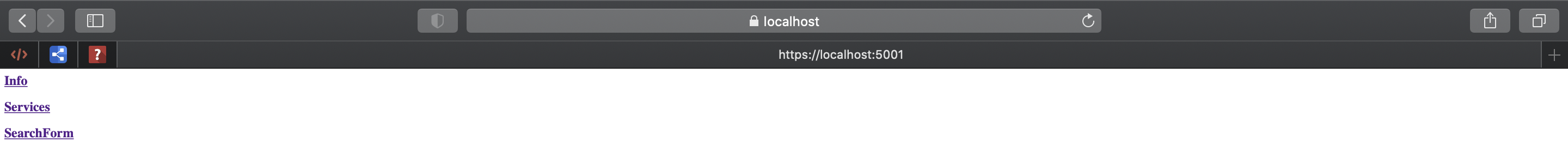


Рисунок 7 – Результат возвращения на корневой сайт

Изображение выглядит как текст

Автоматически созданное описание

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

*Middleware.cs*

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Caching.Memory;

namespace TouristAgencyApp

{

// 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, TouristAgencyContext dbContext)

{

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

if (services != null)

{

cache.Set("services", services,

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

}

var clients = dbContext.Clients.Take(20).ToList();

if (clients != null)

{

cache.Set("customers", clients,

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

using Microsoft.AspNetCore.Http;

using Newtonsoft.Json;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace TouristAgencyApp.Session

{

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;

using TouristAgencyApp.Session;

namespace TouristAgencyApp

{

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 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<TouristAgencyContext>(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, TouristAgencyContext dbContext)

{

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(MakeServicesHtmlTableFromEnumerable(services));

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

});

}

public void SearchForm(IApplicationBuilder app)

{

app.Run(async context =>

{

string client, seviceMaxCost;

GetFieldValuesFromCookies(context,

out client,

out seviceMaxCost);

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>Services</option>" +

"<option>Clients</option>" +

"</select>" +

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

$"<input name=\"clientInput\" value=\"{client}\">" +

"<h3>Enter max cost of services</h3>" +

$"<input name=\"serviceMaxCostInput\" value=\"{seviceMaxCost}\">" +

"<br><br>" +

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

"</form>");

if (context.Request.Query["clientInput"] != "" && context.Request.Query["tables"] == "Clients")

{

var clients = context.Session.Get<List<Client>>("clients" + context.Request.Query["clientInput"]) ??

context.RequestServices.GetService<TouristAgencyContext>().Clients

.Where(client => client.Name.Contains(context.Request.Query["clientInput"]))

.Take(50)

.ToList();

htmlBuilder.Append(MakeClientsTableFromEnumerable(clients));

context.Session.Set("clients" + context.Request.Query["clientInput"], clients);

}

else if (context.Request.Query["serviceMaxCostInput"] != "" && context.Request.Query["tables"] == "Services")

{

var services = context.Session.Get<List<Service>>("services" + context.Request.Query["serviceMaxCostInput"]) ??

context.RequestServices.GetService<TouristAgencyContext>().Services

.Where(service => service.Cost < decimal.Parse(context.Request.Query["serviceMaxCostInput"]))

.Take(50)

.ToList();

htmlBuilder.Append(MakeServicesHtmlTableFromEnumerable(services));

context.Session.Set("services" + context.Request.Query["serviceMaxCostInput"], services);

}

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

});

}

private static void GetFieldValuesFromCookies(HttpContext context,

out string client,

out string serviceMaxCost)

{

client = context.Request.Query["clientInput"];

serviceMaxCost = context.Request.Query["serviceMaxCostInput"];

if (client != null)

{

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

}

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

{

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

}

if (serviceMaxCost != null)

{

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

}

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

{

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

}

}

private string MakeServicesHtmlTableFromEnumerable(IEnumerable<Service> services)

{

StringBuilder builder = new StringBuilder();

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

"<TABLE BORDER=1>");

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

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

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

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

foreach (var service in services)

{

builder.Append("<TR>");

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

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

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

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

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

}

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

return builder.ToString();

}

private string MakeClientsTableFromEnumerable(IEnumerable<Client> clients)

{

StringBuilder builder = new StringBuilder();

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

"<TABLE BORDER=1>");

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

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

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

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

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

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

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

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

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

foreach (var client in clients)

{

builder.Append("<TR>");

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

builder.Append("<TD>" + client.Surname + "</TD>");

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

builder.Append("<TD>" + client.Patronymic + "</TD>");

builder.Append("<TD>" + client.Birthday + "</TD>");

builder.Append("<TD>" + client.Gender + "</TD>");

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

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

builder.Append("<TD>" + client.PassportData + "</TD>");

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

}

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

return builder.ToString();

}

}

}