МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ П. О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

дисциплина «Разработка приложений баз данных для информационных систем»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Обработка *HTTP* запросов средствами *ASP.NET Core*. Сохранение состояния. Кэширование»

ВАРИАНТ №12

Брачное агентство

Выполнил:
студент группы ИТИ-31, Зеленский К.А.
Принял:
доцент Асенчик О.Д.

Гомель 2024

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

Ход работы и результаты.

На рисунке 1 представлен созданный *GitHub* репозиторий. *https://github.com/kevenmusic/RPBDISLabs/tree/main*.

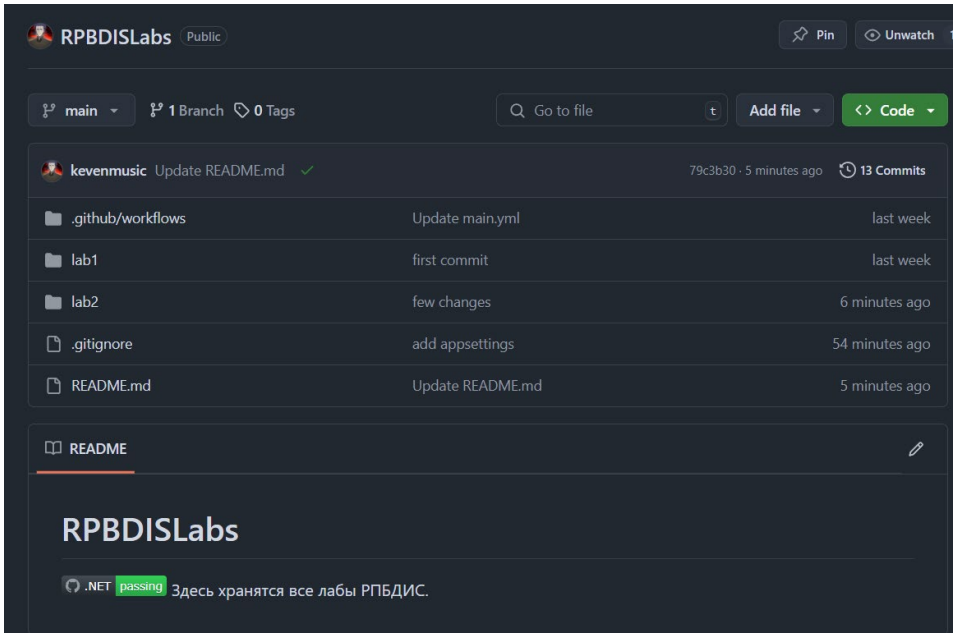

Рисунок 1 – *GitHub* Репозиторий

На рисунке 2 представлено кэширование 20 записей из таблицы «Клиенты» при помощи встроенного инструмента кэширования – объекта *IMemoryCache,* а также выводит в выходной поток для отображения браузером с использование метода *Response.WriteAsync* кэшированную информацию и выходит из конвейера обработки запроса. Данные в кэше хранятся неизменными в течение 262 секунд.

Список клиентов

Код Клиента	Имя	Фамилия	Отчество	Пол	Дата рождения	Знак зодиака	Национальность	Профессия
1	Максим	Петров	Александрович	Мужской	13.10.1963	Овен	Американец	Менеджер
2	Анна	Петрова	Ивановна	Женский	13.10.1986	Стрелец	Русский	Маркетолог
3	Дмитрий	Зайцев	Иванович	Мужской	13.10.1975	Рыбы	Русский	Электрик
4	Юлия	Морозова	Евгеньевна	Женский	13.10.1982	Весы	Бразилец	Программист
5	Максим	Попов	Евгеньевич	Мужской	13.10.1973	Телец	Испанец	Повар
6	Дарья	Петрова	Евгеньевна	Женский	13.10.1966	Скорпион	Бразилец	Инженер
7	Сергей	Семёнов	Сергеевич	Мужской	13.10.2000	Лев	Испанец	Учитель
8	Дарья	Фёдорова	Артёмовна	Женский	13.10.1957	Водолей	Русский	Полицейский
9	Сергей	Фёдоров	Сергеевич	Мужской	13.10.2003	Водолей	Японец	Слесарь
10	Екатерина	Васильева	Дмитриевна	Женский	13.10.1961	Овен	Русский	Повар
11	Роман	Иванов	Иванович	Мужской	13.10.1992	Рыбы	Бразилец	Экономист
12	Ольга	Иванова	Ивановна	Женский	13.10.2005	Близнецы	Итальянец	Повар
13	Евгений	Попов	Александрович	Мужской	13.10.1991	Стрелец	Испанец	Адвокат
14	Юлия	Кузнецова	Анатольевна	Женский	13.10.1958	Стрелец	Итальянец	Адвокат
15	Евгений	Сидоров	Павлович	Мужской	13.10.1998	Близнецы	Француз	Архитектор
16	Наталья	Иванова	Романовна	Женский	13.10.1993	Весы	Русский	Учитель
17	Максим	Кузнецов	Романович	Мужской	13.10.1984	Близнецы	Итальянец	Маркетолог
18	Наталья	Васильева	Анатольевна	Женский	13.10.1963	Скорпион	Немец	Психолог
19	Иван	Сидоров	Артёмович	Мужской	13.10.1957	Близнецы	Француз	Дизайнер
20	Мария	Кузнецова	Евгеньевна	Женский	13.10.1957	Скорпион	Индиец	Психолог

Главная

Рисунок 2 – Кэширование таблицы «Клиенты»

На рисунке 3 представлено кэширование 20 записей из таблицы «Сотрудники» при помощи встроенного инструмента кэширования – объекта *IMemoryCache,* а также выводит в выходной поток для отображения браузером с использование метода *Response.WriteAsync* кэшированную информацию и выходит из конвейера обработки запроса. Данные в кэше хранятся неизменными в течение 262 секунд.

Список сотрудников

Код Сотрудника	Имя	Фамилия	Должность
1	Сергей	Петров	Бухгалтер
2	Екатерина	Васильева	HR специалист
3	Артем	Фёдоров	Юрист
4	Екатерина	Фёдорова	Специалист по рекламе
5	Роман	Фёдоров	Юрист
6	Елена	Васильева	Офис-менеджер
7	Роман	Петров	Офис-менеджер
8	Юлия	Иванова	Специалист по рекламе
9	Анатолий	Иванов	Специалист по рекламе
10	Татьяна	Морозова	Консультант
11	Максим	Петров	HR специалист
12	Наталья	Иванова	Офис-менеджер
13	Максим	Сидоров	Юрист
14	Анна	Семёнова	IT специалист
15	Максим	Кузнецов	Консультант
16	Светлана	Сидорова	Офис-менеджер
17	Александр	Зайцев	Консультант
18	Дарья	Кузнецова	IT специалист
19	Павел	Петров	IT специалист
20	Дарья	Васильева	IT специалист

Главная

Рисунок 3 – Кэширование таблицы «Клиенты»

На рисунке 4 представлена собственная система маршрутизации входящих запросов, содержащая *URL* адрес входящего запроса *\info*.

Информация:

Сервер: localhost:5275
Путь: /info
Протокол: HTTP/1.1
Главная

Рисунок 4 – Страница *\info*

На рисунке 5 представлен вход на страницу с *URL \searchEmployees*, в браузере отображается форма для поиска информации из базы данных с использованием метода *Response.WriteAsync*, а также реализовано сохранение состояния элементов формы с помощью объекта *Session.*

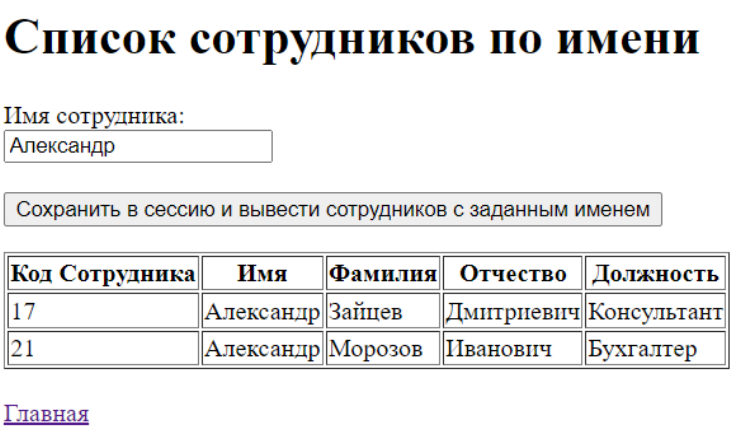

Рисунок 5 – Страница *\searchEmployees*

На рисунке 6 представлен вход на страницу с *URL \searchClients*, в браузере отображается форма для поиска информации из базы данных с использованием метода *Response.WriteAsync*, а также реализовано сохранение состояния элементов формы с использованием куки.

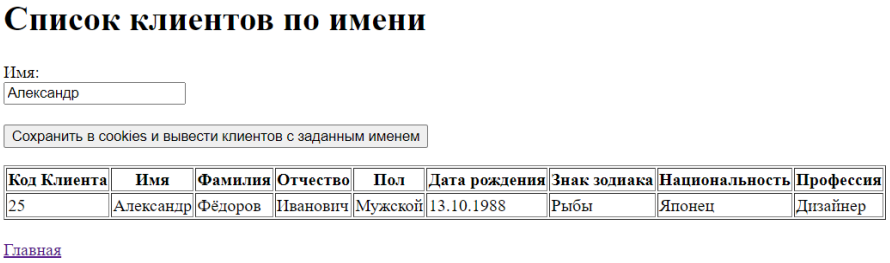

Рисунок 6 – Страница *\searchClients*

На рисунке 7 представлен рабочий процесс *GitHub Actions*, который осуществляет компиляцию проекта под две разные платформы при любом изменении в репозитории.

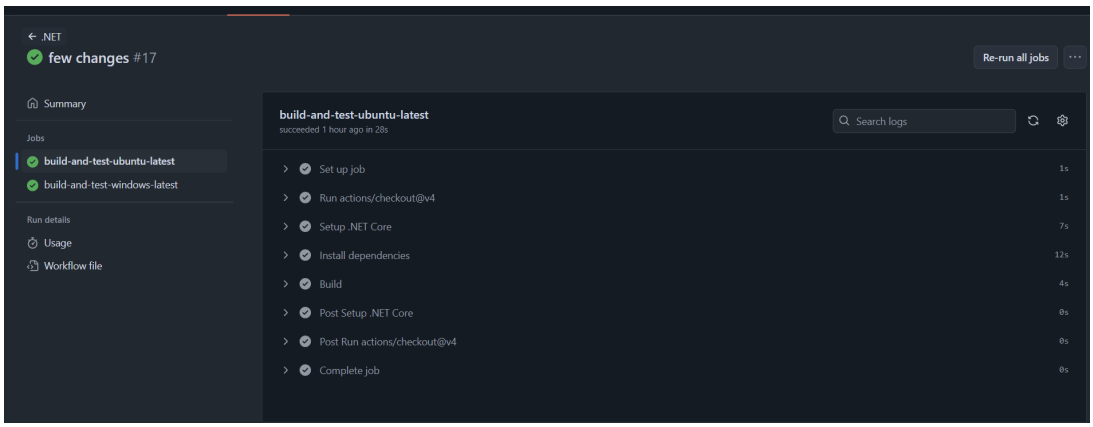

Рисунок 7 – Рабочий процесс *GitHub Actions*

На рисунке 8 представлен отредактированный *README.md* файл опубликованного проекта, вставив в него код для создания эмблемы состояния рабочего процесса (*status badge*), показывающей, чем в данный момент завершился рабочий процесс.

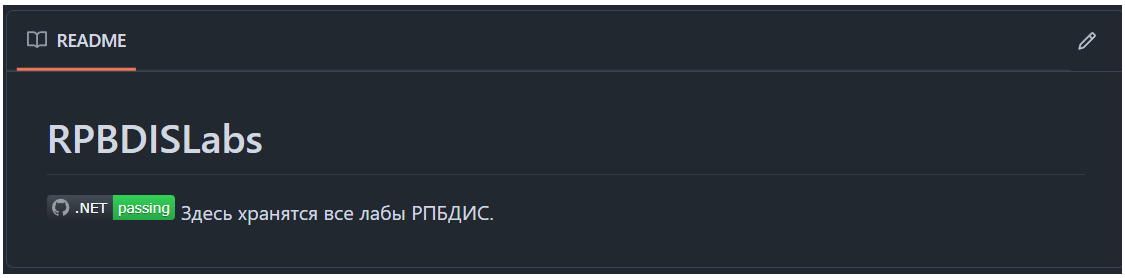

Рисунок 8 – Эмблема состояния

Выводы: был ознакомлен с методами обработки *HTTP* средствами *ASP.NET Core*, методами сохранения состояния приложения, а также повышения производительности приложений путем использования разных видов кэширования.

ПРИЛОЖЕНИЕ А
(обязательное)

Текст программы

Program.cs:

```
using DataLayer.Data;
using DataLayer.Models;
using MarriageAgency.Infrastructure;
using MarriageAgency.Middleware;
using MarriageAgency.Services.AdditionalServicesService;
using MarriageAgency.Services.ClientsService;
using MarriageAgency.Services.ContactsService;
using MarriageAgency.Services.EmployeesService;
using MarriageAgency.Services.NationalitiesService;
using MarriageAgency.Services.PhysicalAttributesService;
using MarriageAgency.Services.ServicesService;
using MarriageAgency.Services.ZodiacSignsService;
using Microsoft.EntityFrameworkCore;

namespace MarriageAgency
{
   public class Program
   {
      public static void Main(string[] args)
      {
         var builder = WebApplication.CreateBuilder(args);
         var services = builder.Services;
         string connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
         services.AddDbContext<MarriageAgencyContext>(options => options.UseSqlServer(connectionString));

         // добавление кэширования
         services.AddMemoryCache();

         // добавление поддержки сессии
         services.AddDistributedMemoryCache();
         services.AddSession();

         // внедрение зависимостей
         services.AddScoped<ICachedClientsService, CachedClientsService>();
         services.AddScoped<ICachedZodiacSignsService, CachedZodiacSignsService>();
         services.AddScoped<ICachedNationalitiesService, CachedNationalitiesService>();
         services.AddScoped<ICachedAdditionalServicesService, CachedAdditionalServicesService>();
         services.AddScoped<ICachedContactsService, CachedContactsService>();
         services.AddScoped<ICachedEmployeesService, CachedEmployeesService>();
         services.AddScoped<ICachedPhysicalAttributesService, CachedPhysicalAttributesService>();
         services.AddScoped<ICachedServicesService, CachedServicesService>();

         //Использование MVC - отключено
         //services.AddControllersWithViews();
         var app = builder.Build();

         // добавляем поддержку статических файлов
         app.UseStaticFiles();

         // добавляем middleware для работы с сессиями
         app.UseSession();

         // добавляем собственный компонент middleware по инициализации базы данных и производим ее ини-
циализацию
         app.UseDbInitializer();

         // Вывод информации о клиенте
         app.Map("/info", (appBuilder) =>
         {
```

```
            appBuilder.Run(async (context) =>
            {
                // Формирование строки для вывода
                string strResponse = "<HTML><HEAD><TITLE>Информация</TITLE></HEAD>" +
                "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
                "<BODY><H1>Информация:</H1>";
                strResponse += "<BR> Сервер: " + context.Request.Host;
                strResponse += "<BR> Путь: " + context.Request.PathBase;
                strResponse += "<BR> Протокол: " + context.Request.Protocol;
                strResponse += "<BR><A href='/'>Главная</A></BODY></HTML>";
                // Вывод данных
                await context.Response.WriteAsync(strResponse);
            });
        });

        // Вывод кэшированной информации из таблицы базы данных "Национальности"
        app.Map("/nationalities", (appBuilder) =>
        {
            appBuilder.Run(async (context) =>
            {
                //обращение к сервису
                ICachedNationalitiesService cachedNationalitiesService = con-
text.RequestServices.GetService<ICachedNationalitiesService>();
                IEnumerable<Nationality> nationalities = cachedNationalitiesService.GetNationalities(20);
                string HtmlString = "<HTML><HEAD><TITLE>Национальности</TITLE></HEAD>" +
                "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
                "<BODY><H1>Список национальностей</H1>" +
                "<TABLE BORDER=1>";
                HtmlString += "<TR>";
                HtmlString += "<TH>Код Национальности</TH>";
                HtmlString += "<TH>Название национальности</TH>";
                HtmlString += "<TH>Примечания</TH>";
                HtmlString += "</TR>";
                foreach (var nationality in nationalities)
                {
                    HtmlString += "<TR>";
                    HtmlString += "<TD>" + nationality.NationalityId + "</TD>";
                    HtmlString += "<TD>" + nationality.Name + "</TD>";
                    HtmlString += "<TD>" + nationality.Notes + "</TD>";
                    HtmlString += "</TR>";
                }
                HtmlString += "</TABLE>";
                HtmlString += "<BR><A href='/'>Главная</A></BR>";
                HtmlString += "</BODY></HTML>";

                // Вывод данных
                await context.Response.WriteAsync(HtmlString);
            });
        });

        // Вывод кэшированной информации из таблицы базы данных "Знаки зодиака"
        app.Map("/zodiacSigns", (appBuilder) =>
        {
            appBuilder.Run(async (context) =>
            {
                // обращение к сервису
                ICachedZodiacSignsService cachedZodiacSignsService = con-
text.RequestServices.GetService<ICachedZodiacSignsService>();
                IEnumerable<ZodiacSign> zodiacSigns = cachedZodiacSignsService.GetZodiacSigns(20);
                string HtmlString = "<HTML><HEAD><TITLE>Знаки зодиака</TITLE></HEAD>" +
                "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
                "<BODY><H1>Список знаков зодиака</H1>" +
                "<TABLE BORDER=1>";
                HtmlString += "<TR>";
                HtmlString += "<TH>Код Знака</TH>";
                HtmlString += "<TH>Название знака</TH>";
                HtmlString += "</TR>";
```

```
            foreach (var zodiacSign in zodiacSigns)
            {
                HtmlString += "<TR>";
                HtmlString += "<TD>" + zodiacSign.ZodiacSignId + "</TD>";
                HtmlString += "<TD>" + zodiacSign.Name + "</TD>";
                HtmlString += "</TR>";
            }
            HtmlString += "</TABLE>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "</BODY></HTML>";

            // Вывод данных
            await context.Response.WriteAsync(HtmlString);
        });
    });

    // Вывод кэшированной информации из таблицы базы данных "Физические атрибуты"
    app.Map("/physicalAttributes", (appBuilder) =>
    {
        appBuilder.Run(async (context) =>
        {
            // обращение к сервису
            ICachedPhysicalAttributesService cachedPhysicalAttributeService = con-
text.RequestServices.GetService<ICachedPhysicalAttributesService>();
            IEnumerable<PhysicalAttribute> physicalAttributes = cachedPhysicalAttributeServ-
ice.GetPhysicalAttributes(20);
            string HtmlString = "<HTML><HEAD><TITLE>Физические атрибуты</TITLE></HEAD>" +
            "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
            "<BODY><H1>Список физических атрибутов</H1>" +
            "<TABLE BORDER=1>";
            HtmlString += "<TR>";
            HtmlString += "<TH>Код Клиента</TH>";
            HtmlString += "<TH>Возраст</TH>";
            HtmlString += "<TH>Рост</TH>";
            HtmlString += "<TH>Вес</TH>";
            HtmlString += "<TH>Количество детей</TH>";
            HtmlString += "<TH>Семейное положение</TH>";
            HtmlString += "<TH>Вредные привычки</TH>";
            HtmlString += "<TH>Хобби</TH>";
            HtmlString += "</TR>";
            foreach (var attribute in physicalAttributes)
            {
                HtmlString += "<TR>";
                HtmlString += "<TD>" + attribute.ClientId + "</TD>";
                HtmlString += "<TD>" + attribute.Age + "</TD>";
                HtmlString += "<TD>" + attribute.Height + "</TD>";
                HtmlString += "<TD>" + attribute.Weight + "</TD>";
                HtmlString += "<TD>" + attribute.ChildrenCount + "</TD>";
                HtmlString += "<TD>" + attribute.MaritalStatus + "</TD>";
                HtmlString += "<TD>" + attribute.BadHabits + "</TD>";
                HtmlString += "<TD>" + attribute.Hobbies + "</TD>";
                HtmlString += "</TR>";
            }
            HtmlString += "</TABLE>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "</BODY></HTML>";

            // Вывод данных
            await context.Response.WriteAsync(HtmlString);
        });
    });

    // Вывод кэшированной информации из таблицы базы данных "Контакты"
    app.Map("/contacts", (appBuilder) =>
    {
        appBuilder.Run(async (context) =>
        {
```

```
            // обращение к сервису
            ICachedContactsService cachedContactService = con-
text.RequestServices.GetService<ICachedContactsService>();
            IEnumerable<Contact> contacts = cachedContactService.GetContacts(20);
            string HtmlString = "<HTML><HEAD><TITLE>Контакты</TITLE></HEAD>" +
            "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
            "<BODY><H1>Список контактов</H1>" +
            "<TABLE BORDER=1>";
            HtmlString += "<TR>";
            HtmlString += "<TH>Код Клиента</TH>";
            HtmlString += "<TH>Адрес</TH>";
            HtmlString += "<TH>Телефон</TH>";
            HtmlString += "<TH>Паспортные данные</TH>";
            HtmlString += "</TR>";
            foreach (var contact in contacts)
            {
               HtmlString += "<TR>";
               HtmlString += "<TD>" + contact.ClientId + "</TD>";
               HtmlString += "<TD>" + contact.Address + "</TD>";
               HtmlString += "<TD>" + contact.Phone + "</TD>";
               HtmlString += "<TD>" + contact.PassportData + "</TD>";
               HtmlString += "</TR>";
            }
            HtmlString += "</TABLE>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "</BODY></HTML>";

            // Вывод данных
            await context.Response.WriteAsync(HtmlString);
         });
      });

      // Вывод кэшированной информации из таблицы базы данных "Дополнительные услуги"
      app.Map("/additionalServices", (appBuilder) =>
      {
         appBuilder.Run(async (context) =>
         {
            // обращение к сервису
            ICachedAdditionalServicesService cachedAdditionalServiceService = con-
text.RequestServices.GetService<ICachedAdditionalServicesService>();
            IEnumerable<AdditionalService> additionalServices = cachedAdditionalServiceServ-
ice.GetAdditionalServices(20);
            string HtmlString = "<HTML><HEAD><TITLE>Дополнительные услуги</TITLE></HEAD>" +
            "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
            "<BODY><H1>Список дополнительных услуг</H1>" +
            "<TABLE BORDER=1>";
            HtmlString += "<TR>";
            HtmlString += "<TH>Код Услуги</TH>";
            HtmlString += "<TH>Название</TH>";
            HtmlString += "<TH>Описание</TH>";
            HtmlString += "</TR>";
            foreach (var service in additionalServices)
            {
               HtmlString += "<TR>";
               HtmlString += "<TD>" + service.AdditionalServiceId + "</TD>";
               HtmlString += "<TD>" + service.Name + "</TD>";
               HtmlString += "<TD>" + service.Description + "</TD>";
               HtmlString += "</TR>";
            }
            HtmlString += "</TABLE>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "</BODY></HTML>";

            // Вывод данных
            await context.Response.WriteAsync(HtmlString);
         });
      });
```

```
        // Вывод кэшированной информации из таблицы базы данных "Сотрудники"
        app.Map("/employees", (appBuilder) =>
        {
            appBuilder.Run(async (context) =>
            {
                // обращение к сервису
                ICachedEmployeesService cachedEmployeeService = con-
text.RequestServices.GetService<ICachedEmployeesService>();
                IEnumerable<Employee> employees = cachedEmployeeService.GetEmployees(20);
                string HtmlString = "<HTML><HEAD><TITLE>Сотрудники</TITLE></HEAD>" +
                "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
                "<BODY><H1>Список сотрудников</H1>" +
                "<TABLE BORDER=1>";
                HtmlString += "<TR>";
                HtmlString += "<TH>Код Сотрудника</TH>";
                HtmlString += "<TH>Имя</TH>";
                HtmlString += "<TH>Фамилия</TH>";
                HtmlString += "<TH>Должность</TH>";
                HtmlString += "</TR>";
                foreach (var employee in employees)
                {
                    HtmlString += "<TR>";
                    HtmlString += "<TD>" + employee.EmployeeId + "</TD>";
                    HtmlString += "<TD>" + employee.FirstName + "</TD>";
                    HtmlString += "<TD>" + employee.LastName + "</TD>";
                    HtmlString += "<TD>" + employee.Position + "</TD>";
                    HtmlString += "</TR>";
                }
                HtmlString += "</TABLE>";
                HtmlString += "<BR><A href='/'>Главная</A></BR>";
                HtmlString += "</BODY></HTML>";

                // Вывод данных
                await context.Response.WriteAsync(HtmlString);
            });
        });

        app.Map("/searchEmployees", (appBuilder) =>
        {
            appBuilder.Run(async (context) =>
            {
                string employeeName = string.Empty;

                // Проверяем, передано ли значение employeeName в запросе
                if (!string.IsNullOrEmpty(context.Request.Query["employeeName"]))
                {
                    employeeName = context.Request.Query["employeeName"];
                    // Сохраняем employeeName в сессии
                    context.Session.SetString("employeeName", employeeName);
                }
                // Если employeeName уже сохранено в сессии, извлекаем его
                else if (context.Session.Keys.Contains("employeeName"))
                {
                    employeeName = context.Session.GetString("employeeName");
                }
                // Получение сервисов кэша для сотрудников, национальностей и других данных, если необходимо
                ICachedEmployeesService cachedEmployeesService = con-
text.RequestServices.GetService<ICachedEmployeesService>();
                IEnumerable<Employee> employees = cachedEmployeesService.GetEmployees(30);

                string HtmlString = "<HTML><HEAD><TITLE>Сотрудники</TITLE></HEAD>" +
                "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
                "<BODY><H1>Список сотрудников по имени</H1>" +
                "<BODY><FORM action ='/searchEmployees' method='get'>" +
                "Имя сотрудника:<BR><INPUT type = 'text' name = 'employeeName' value = '" + employeeName + "'>"
+
```

```
            "<BR><BR><INPUT type ='submit' value='Сохранить в сессию и вывести сотрудников с заданным
именем'></FORM>" +
            "<TABLE BORDER=1>";

            HtmlString += "<TR>";
            HtmlString += "<TH>Код Сотрудника</TH>";
            HtmlString += "<TH>Имя</TH>";
            HtmlString += "<TH>Фамилия</TH>";
            HtmlString += "<TH>Отчество</TH>";
            HtmlString += "<TH>Должность</TH>";
            HtmlString += "</TR>";
            // Фильтрация сотрудников по имени
            foreach (var employee in employees.Where(e => e.FirstName.Trim() == employeeName))
            {
               HtmlString += "<TR>";
               HtmlString += "<TD>" + employee.EmployeeId + "</TD>";
               HtmlString += "<TD>" + employee.FirstName + "</TD>";
               HtmlString += "<TD>" + employee.LastName + "</TD>";
               HtmlString += "<TD>" + employee.MiddleName + "</TD>";
               HtmlString += "<TD>" + employee.Position + "</TD>";
               HtmlString += "</TR>";
            }

            HtmlString += "</TABLE>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "</BODY></HTML>";

            await context.Response.WriteAsync(HtmlString);
         });
      });


      // Вывод кэшированной информации из таблицы базы данных "Услуги"
      app.Map("/services", (appBuilder) =>
      {
         appBuilder.Run(async (context) =>
         {
            // обращение к сервису
            ICachedAdditionalServicesService cachedAdditionalServiceService = con-
text.RequestServices.GetService<ICachedAdditionalServicesService>();
            cachedAdditionalServiceService.AddAdditionalServices("AdditionalServices20", 1000);
            ICachedClientsService cachedClientsService = con-
text.RequestServices.GetService<ICachedClientsService>();
            cachedClientsService.AddClients("Clients20", 1000);
            ICachedEmployeesService cachedEmployeeService = con-
text.RequestServices.GetService<ICachedEmployeesService>();
            cachedEmployeeService.AddEmployees("Employees20", 1000);
            ICachedServicesService cachedServicesService = con-
text.RequestServices.GetService<ICachedServicesService>();
            IEnumerable<Service> services = cachedServicesService.GetServices(20);
            string HtmlString = "<HTML><HEAD><TITLE>Услуги</TITLE></HEAD>" +
            "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
            "<BODY><H1>Список услуг</H1>" +
            "<TABLE BORDER=1>";
            HtmlString += "<TR>";
            HtmlString += "<TH>Код Услуги</TH>";
            HtmlString += "<TH>Дополнительная услуга</TH>";
            HtmlString += "<TH>Клиент</TH>";
            HtmlString += "<TH>Сотрудник</TH>";
            HtmlString += "<TH>Дата</TH>";
            HtmlString += "<TH>Стоимость</TH>";
            HtmlString += "</TR>";
            foreach (var service in services)
            {
               HtmlString += "<TR>";
               HtmlString += "<TD>" + service.ServiceId + "</TD>";
               HtmlString += "<TD>" + service.AdditionalService.Name + "</TD>";
```

```
                HtmlString += "<TD>" + service.Client.FirstName + "</TD>";
                HtmlString += "<TD>" + service.Employee.FirstName + "</TD>";
                HtmlString += "<TD>" + service.Date + "</TD>";
                HtmlString += "<TD>" + service.Cost + "</TD>";
                HtmlString += "</TR>";
            }
            HtmlString += "</TABLE>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "</BODY></HTML>";

            // Вывод данных
            await context.Response.WriteAsync(HtmlString);
        });
    });

    // Вывод кэшированной информации из таблицы базы данных "Клиенты"
    app.Map("/clients", (appBuilder) =>
    {
        appBuilder.Run(async (context) =>
        {
            //обращение к сервису
            ICachedNationalitiesService cachedNationalitiesService = con-
text.RequestServices.GetService<ICachedNationalitiesService>();
            cachedNationalitiesService.AddNationalities("Nationalities20", 1000);
            ICachedZodiacSignsService zodiacSignService = con-
text.RequestServices.GetService<ICachedZodiacSignsService>();
            zodiacSignService.AddZodiacSigns("ZodiacSigns20", 1000);
            ICachedClientsService cachedClientsService = con-
text.RequestServices.GetService<ICachedClientsService>();
            IEnumerable<Client> clients = cachedClientsService.GetClients(20);
            string HtmlString = "<HTML><HEAD><TITLE>Клиенты</TITLE></HEAD>" +
            "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
            "<BODY><H1>Список клиентов</H1>" +
            "<TABLE BORDER=1>";
            HtmlString += "<TR>";
            HtmlString += "<TH>Код Клиента</TH>";
            HtmlString += "<TH>Имя</TH>";
            HtmlString += "<TH>Фамилия</TH>";
            HtmlString += "<TH>Отчество</TH>";
            HtmlString += "<TH>Пол</TH>";
            HtmlString += "<TH>Дата рождения</TH>";
            HtmlString += "<TH>Знак зодиака</TH>";
            HtmlString += "<TH>Национальность</TH>";
            HtmlString += "<TH>Профессия</TH>";
            HtmlString += "</TR>";
            foreach (var client in clients)
            {
                HtmlString += "<TR>";
                HtmlString += "<TD>" + client.ClientId + "</TD>";
                HtmlString += "<TD>" + client.FirstName + "</TD>";
                HtmlString += "<TD>" + client.LastName + "</TD>";
                HtmlString += "<TD>" + client.MiddleName + "</TD>";
                HtmlString += "<TD>" + client.Gender + "</TD>";
                HtmlString += "<TD>" + client.BirthDate + "</TD>";
                HtmlString += "<TD>" + client.ZodiacSign?.Name + "</TD>";
                HtmlString += "<TD>" + client.Nationality?.Name + "</TD>";
                HtmlString += "<TD>" + client.Profession + "</TD>";
                HtmlString += "</TR>";
            }
            HtmlString += "</TABLE>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "</BODY></HTML>";

            // Вывод данных
            await context.Response.WriteAsync(HtmlString);
        });
    });
```

```
        app.Map("/searchClients", (appBuilder) =>
        {
            appBuilder.Run(async (context) =>
            {
                string clientName;
                context.Request.Cookies.TryGetValue("clientName", out clientName);
                ICachedNationalitiesService cachedNationalitiesService = con-
text.RequestServices.GetService<ICachedNationalitiesService>();
                cachedNationalitiesService.AddNationalities("Nationalities20", 1000);
                ICachedZodiacSignsService zodiacSignService = con-
text.RequestServices.GetService<ICachedZodiacSignsService>();
                zodiacSignService.AddZodiacSigns("ZodiacSigns20", 1000);
                ICachedClientsService cachedClientsService = con-
text.RequestServices.GetService<ICachedClientsService>();
                IEnumerable<Client> clients = cachedClientsService.GetClients(30);
                string HtmlString = "<HTML><HEAD><TITLE>Клиенты</TITLE></HEAD>" +
                "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
                "<BODY><H1>Список клиентов по имени</H1>" +
                "<BODY><FORM action ='/searchClients' method='GET'>" +
                "Имя:<BR><INPUT type = 'text' name = 'clientName' value = " + clientName + ">" +
                "<BR><BR><INPUT type ='submit' value='Сохранить в cookies и вывести клиентов с заданным име-
нем'></FORM>" +
                "<TABLE BORDER=1>";
                HtmlString += "<TR>";
                HtmlString += "<TH>Код Клиента</TH>";
                HtmlString += "<TH>Имя</TH>";
                HtmlString += "<TH>Фамилия</TH>";
                HtmlString += "<TH>Отчество</TH>";
                HtmlString += "<TH>Пол</TH>";
                HtmlString += "<TH>Дата рождения</TH>";
                HtmlString += "<TH>Знак зодиака</TH>";
                HtmlString += "<TH>Национальность</TH>";
                HtmlString += "<TH>Профессия</TH>";
                HtmlString += "</TR>";

                clientName = context.Request.Query["clientName"];
                if (!string.IsNullOrEmpty(clientName))
                {
                    context.Response.Cookies.Append("clientName", clientName);
                }
                foreach (var client in clients.Where(e => e.FirstName.Trim().ToLower() == clientName?.ToLower()))
                {
                    HtmlString += "<TR>";
                    HtmlString += "<TD>" + client.ClientId + "</TD>";
                    HtmlString += "<TD>" + client.FirstName + "</TD>";
                    HtmlString += "<TD>" + client.LastName + "</TD>";
                    HtmlString += "<TD>" + client.MiddleName + "</TD>";
                    HtmlString += "<TD>" + client.Gender + "</TD>";
                    HtmlString += "<TD>" + client.BirthDate + "</TD>";
                    HtmlString += "<TD>" + client.ZodiacSign?.Name + "</TD>";
                    HtmlString += "<TD>" + client.Nationality?.Name + "</TD>";
                    HtmlString += "<TD>" + client.Profession + "</TD>";
                    HtmlString += "</TR>";
                }
                HtmlString += "</TABLE>";
                HtmlString += "<BR><A href='/'>Главная</A></BR>";
                HtmlString += "</BODY></HTML>";

                await context.Response.WriteAsync(HtmlString);
            });
        });

        app.Run((context) =>
        {
            // Кэширование данных для всех таблиц
```

```
            ICachedNationalitiesService cachedNationalitiesService = con-
text.RequestServices.GetService<ICachedNationalitiesService>();
            cachedNationalitiesService.AddNationalities("Nationalities20", 20);

            ICachedClientsService cachedClientsService = con-
text.RequestServices.GetService<ICachedClientsService>();
            cachedClientsService.AddClients("Clients20", 1000);

            ICachedEmployeesService cachedEmployeesService = con-
text.RequestServices.GetService<ICachedEmployeesService>();
            cachedEmployeesService.AddEmployees("Employees20", 100);

            ICachedServicesService cachedServicesService = con-
text.RequestServices.GetService<ICachedServicesService>();
            cachedServicesService.AddServices("Services20", 50);

            ICachedAdditionalServicesService cachedAdditionalServicesService = con-
text.RequestServices.GetService<ICachedAdditionalServicesService>();
            cachedAdditionalServicesService.AddAdditionalServices("AdditionalServices20", 30);

            ICachedContactsService cachedContactsService = con-
text.RequestServices.GetService<ICachedContactsService>();
            cachedContactsService.AddContacts("Contacts20", 500);

            ICachedPhysicalAttributesService cachedPhysicalAttributesService = con-
text.RequestServices.GetService<ICachedPhysicalAttributesService>();
            cachedPhysicalAttributesService.AddPhysicalAttributes("PhysicalAttributes20", 200);

            ICachedZodiacSignsService cachedZodiacSignsService = con-
text.RequestServices.GetService<ICachedZodiacSignsService>();
            cachedZodiacSignsService.AddZodiacSigns("ZodiacSigns20", 12);

            string HtmlString = "<HTML><HEAD><TITLE>Главная</TITLE></HEAD>" +
            "<META http-equiv='Content-Type' content='text/html; charset=utf-8'/>" +
            "<BODY><H1>Главная</H1>";
            HtmlString += "<H2>Данные записаны в кэш сервера</H2>";
            HtmlString += "<BR><A href='/'>Главная</A></BR>";
            HtmlString += "<BR><A href='/clients'>Клиенты</A></BR>";
            HtmlString += "<BR><A href='/searchClients'>Поиск по клиентам</A></BR>";
            HtmlString += "<BR><A href='/employees'>Сотрудники</A></BR>";
            HtmlString += "<BR><A href='/searchEmployees'>Поиск по сотрудникам</A></BR>";
            HtmlString += "<BR><A href='/services'>Услуги</A></BR>";
            HtmlString += "<BR><A href='/additionalServices'>Дополонительные услуги</A></BR>";
            HtmlString += "<BR><A href='/contacts'>Контакты</A></BR>";
            HtmlString += "<BR><A href='/physicalAttributes'>Физические атрибуты</A></BR>";
            HtmlString += "<BR><A href='/zodiacSigns'>Знаки зодиака</A></BR>";
            HtmlString += "<BR><A href='/nationalities'>Национальности</A></BR>";
            HtmlString += "<BR><A href='/info'>Информация о клиенте</A></BR>";
            HtmlString += "</BODY></HTML>";

            return context.Response.WriteAsync(HtmlString);
        });

        app.Run();
    }
  }
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Текст программы

SessionExtensions.cs

```
using Microsoft.AspNetCore.Http;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.Json;
using System.Threading.Tasks;

namespace MarriageAgency.Infrastructure
{
   public static class SessionExtensions
   {
      public static void Set<T>(this ISession session, string key, T value)
      {
         session.SetString(key, JsonSerializer.Serialize<T>(value));
      }

      public static T Get<T>(this ISession session, string key)
      {
         var value = session.GetString(key);
         return value == null ? default(T) : JsonSerializer.Deserialize<T>(value);
      }
   }
}
```

ПРИЛОЖЕНИЕ В
(обязательное)

Текст программы

DbInitializerMiddleware.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DataLayer.Models;
using DataLayer.Data;

namespace MarriageAgency.Middleware
{
   public class DbInitializerMiddleware
   {
      private readonly RequestDelegate _next;

      public DbInitializerMiddleware(RequestDelegate next)
      {
         _next = next;
      }

      public Task Invoke(HttpContext context, IServiceProvider serviceProvider, MarriageAgencyContext db)
      {
         if (!(context.Session.Keys.Contains("starting")))
         {
            DbInitializer.Initialize(db);
            context.Session.SetString("starting", "Yes");
         }

         return _next.Invoke(context);
      }
   }

   public static class DbInitializerExtensions
   {
      public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)
      {
         return builder.UseMiddleware<DbInitializerMiddleware>();
      }

   }
}
```

ПРИЛОЖЕНИЕ Г
(обязательное)

Текст программы

ICachedAdditionalServicesService.cs
```
using DataLayer.Models;

namespace MarriageAgency.Services.AdditionalServicesService
{
   public interface ICachedAdditionalServicesService
   {
      public IEnumerable<AdditionalService> GetAdditionalServices(int rowNumber);
      public void AddAdditionalServices(string cacheKey, int rowNumber);
      public IEnumerable<AdditionalService> GetAdditionalServices(string cacheKey, int rowNumber);
   }
}
```

CachedAdditionalServicesService.cs
```
using Microsoft.Extensions.Caching.Memory;
using DataLayer.Data;
using DataLayer.Models;

namespace MarriageAgency.Services.AdditionalServicesService
{
   public class CachedAdditionalServicesService : ICachedAdditionalServicesService
   {
      private readonly IMemoryCache _cache;
      private readonly MarriageAgencyContext _context;

      public CachedAdditionalServicesService(IMemoryCache cache, MarriageAgencyContext context)
      {
         _cache = cache;
         _context = context;
      }

      public void AddAdditionalServices(string cacheKey, int rowNumber)
      {
         IEnumerable<AdditionalService> additionalServices =
_context.AdditionalServices.Take(rowNumber).ToList();
         if (additionalServices != null)
         {
            _cache.Set(cacheKey, additionalServices, new MemoryCacheEntryOptions
            {
               AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
            });
         }
      }

      public IEnumerable<AdditionalService> GetAdditionalServices(int rowNumber)
      {
         return _context.AdditionalServices.Take(rowNumber).ToList();
      }

      public IEnumerable<AdditionalService> GetAdditionalServices(string cacheKey, int rowNumber)
      {
         IEnumerable<AdditionalService> additionalServices;
         if (!_cache.TryGetValue(cacheKey, out additionalServices))
         {
            additionalServices = _context.AdditionalServices.Take(rowNumber).ToList();
            if (additionalServices != null)
            {
```

```
                _cache.Set(cacheKey, additionalServices, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return additionalServices;
    }
  }
}
```

ICachedClientsService.cs

```
using DataLayer.Models;

namespace MarriageAgency.Services.ClientsService
{
  public interface ICachedClientsService
  {
    public IEnumerable<Client> GetClients(int rowNumber);
    public void AddClients(string cacheKey, int rowNumber);
    public IEnumerable<Client> GetClients(string cacheKey, int rowNumber);
  }
}
```

CachedClientsService.cs

```
using Microsoft.Extensions.Caching.Memory;
using DataLayer.Data;
using DataLayer.Models;

namespace MarriageAgency.Services.ClientsService
{
  public class CachedClientsService : ICachedClientsService
  {
    private readonly IMemoryCache _cache;
    private readonly MarriageAgencyContext _context;

    public CachedClientsService(IMemoryCache cache, MarriageAgencyContext context)
    {
      _cache = cache;
      _context = context;
    }

    public void AddClients(string cacheKey, int rowNumber)
    {
      IEnumerable<Client> clients = _context.Clients.Take(rowNumber).ToList();
      if (clients != null)
      {
        _cache.Set(cacheKey, clients, new MemoryCacheEntryOptions
        {
          AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
        });
      }
    }

    public IEnumerable<Client> GetClients(int rowNumber)
    {
      return _context.Clients.Take(rowNumber).ToList();
    }

    public IEnumerable<Client> GetClients(string cacheKey, int rowNumber)
    {
      IEnumerable<Client> clients;
      if (!_cache.TryGetValue(cacheKey, out clients))
      {
        clients = _context.Clients.Take(rowNumber).ToList();
        if (clients != null)
        {
```

```
                _cache.Set(cacheKey, clients, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return clients;
    }
  }
}
```

ICachedContactsService.cs

```
using DataLayer.Models;

namespace MarriageAgency.Services.ContactsService
{
  public interface ICachedContactsService
  {
    public IEnumerable<Contact> GetContacts(int rowNumber);
    public void AddContacts(string cacheKey, int rowNumber);
    public IEnumerable<Contact> GetContacts(string cacheKey, int rowNumber);
  }
}
```

CachedContactsService.cs

```
using DataLayer.Data;
using DataLayer.Models;
using Microsoft.Extensions.Caching.Memory;

namespace MarriageAgency.Services.ContactsService
{
  public class CachedContactsService : ICachedContactsService
  {
    private readonly IMemoryCache _cache;
    private readonly MarriageAgencyContext _context;

    public CachedContactsService(IMemoryCache cache, MarriageAgencyContext context)
    {
      _cache = cache;
      _context = context;
    }

    public void AddContacts(string cacheKey, int rowNumber)
    {
      IEnumerable<Contact> contacts = _context.Contacts.Take(rowNumber).ToList();
      if (contacts != null)
      {
        _cache.Set(cacheKey, contacts, new MemoryCacheEntryOptions
        {
          AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
        });
      }
    }

    public IEnumerable<Contact> GetContacts(int rowNumber)
    {
      return _context.Contacts.Take(rowNumber).ToList();
    }

    public IEnumerable<Contact> GetContacts(string cacheKey, int rowNumber)
    {
      IEnumerable<Contact> contacts;
      if (!_cache.TryGetValue(cacheKey, out contacts))
      {
        contacts = _context.Contacts.Take(rowNumber).ToList();
        if (contacts != null)
        {
```

```
                _cache.Set(cacheKey, contacts, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return contacts;
    }
  }
}
```

ICachedEmployeesService.cs

```
using DataLayer.Models;

namespace MarriageAgency.Services.EmployeesService
{
  public interface ICachedEmployeesService
  {
    public IEnumerable<Employee> GetEmployees(int rowNumber);
    public void AddEmployees(string cacheKey, int rowNumber);
    public IEnumerable<Employee> GetEmployees(string cacheKey, int rowNumber);
  }
}
```

CachedEmployeesService.cs

```
using DataLayer.Data;
using DataLayer.Models;
using Microsoft.Extensions.Caching.Memory;

namespace MarriageAgency.Services.EmployeesService
{
  public class CachedEmployeesService : ICachedEmployeesService
  {
    private readonly IMemoryCache _cache;
    private readonly MarriageAgencyContext _context;

    public CachedEmployeesService(IMemoryCache cache, MarriageAgencyContext context)
    {
      _cache = cache;
      _context = context;
    }

    public void AddEmployees(string cacheKey, int rowNumber)
    {
      IEnumerable<Employee> employees = _context.Employees.Take(rowNumber).ToList();
      if (employees != null)
      {
        _cache.Set(cacheKey, employees, new MemoryCacheEntryOptions
        {
          AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
        });
      }
    }

    public IEnumerable<Employee> GetEmployees(int rowNumber)
    {
      return _context.Employees.Take(rowNumber).ToList();
    }

    public IEnumerable<Employee> GetEmployees(string cacheKey, int rowNumber)
    {
      IEnumerable<Employee> employees;
      if (!_cache.TryGetValue(cacheKey, out employees))
      {
        employees = _context.Employees.Take(rowNumber).ToList();
        if (employees != null)
        {
```

```
                _cache.Set(cacheKey, employees, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return employees;
    }
  }
}
```

ICachedNationalitiesService.cs

```
using DataLayer.Models;

namespace MarriageAgency.Services.NationalitiesService
{
  public interface ICachedNationalitiesService
  {
    public IEnumerable<Nationality> GetNationalities(int rowNumber);
    public void AddNationalities(string cacheKey, int rowNumber);
    public IEnumerable<Nationality> GetNationalities(string cacheKey, int rowNumber);
  }
}
```

CachedNationalitiesService.cs

```
using DataLayer.Data;
using DataLayer.Models;
using Microsoft.Extensions.Caching.Memory;

namespace MarriageAgency.Services.NationalitiesService
{
  public class CachedNationalitiesService : ICachedNationalitiesService
  {
    private readonly IMemoryCache _cache;
    private readonly MarriageAgencyContext _context;

    public CachedNationalitiesService(IMemoryCache cache, MarriageAgencyContext context)
    {
      _cache = cache;
      _context = context;
    }

    public void AddNationalities(string cacheKey, int rowNumber)
    {
      IEnumerable<Nationality> nationalities = _context.Nationalities.Take(rowNumber).ToList();
      if (nationalities != null)
      {
        _cache.Set(cacheKey, nationalities, new MemoryCacheEntryOptions
        {
          AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
        });
      }
    }

    public IEnumerable<Nationality> GetNationalities(int rowNumber)
    {
      return _context.Nationalities.Take(rowNumber).ToList();
    }

    public IEnumerable<Nationality> GetNationalities(string cacheKey, int rowNumber)
    {
      IEnumerable<Nationality> nationalities;
      if (!_cache.TryGetValue(cacheKey, out nationalities))
      {
        nationalities = _context.Nationalities.Take(rowNumber).ToList();
        if (nationalities != null)
        {
```

```
                _cache.Set(cacheKey, nationalities, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return nationalities;
    }
  }
}
```

ICachedPhysicalAttributesService.cs

```
using DataLayer.Models;

namespace MarriageAgency.Services.PhysicalAttributesService
{
  public interface ICachedPhysicalAttributesService
  {
    public IEnumerable<PhysicalAttribute> GetPhysicalAttributes(int rowNumber);
    public void AddPhysicalAttributes(string cacheKey, int rowNumber);
    public IEnumerable<PhysicalAttribute> GetPhysicalAttributes(string cacheKey, int rowNumber);
  }
}
```

CachedPhysicalAttributesService.cs

```
using DataLayer.Data;
using DataLayer.Models;
using Microsoft.Extensions.Caching.Memory;

namespace MarriageAgency.Services.PhysicalAttributesService
{
  public class CachedPhysicalAttributesService : ICachedPhysicalAttributesService
  {
    private readonly IMemoryCache _cache;
    private readonly MarriageAgencyContext _context;

    public CachedPhysicalAttributesService(IMemoryCache cache, MarriageAgencyContext context)
    {
      _cache = cache;
      _context = context;
    }

    public void AddPhysicalAttributes(string cacheKey, int rowNumber)
    {
      IEnumerable<PhysicalAttribute> physicalAttributes = _context.PhysicalAttributes.Take(rowNumber).ToList();
      if (physicalAttributes != null)
      {
        _cache.Set(cacheKey, physicalAttributes, new MemoryCacheEntryOptions
        {
          AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
        });
      }
    }

    public IEnumerable<PhysicalAttribute> GetPhysicalAttributes(int rowNumber)
    {
      return _context.PhysicalAttributes.Take(rowNumber).ToList();
    }

    public IEnumerable<PhysicalAttribute> GetPhysicalAttributes(string cacheKey, int rowNumber)
    {
      IEnumerable<PhysicalAttribute> physicalAttributes;
      if (!_cache.TryGetValue(cacheKey, out physicalAttributes))
      {
        physicalAttributes = _context.PhysicalAttributes.Take(rowNumber).ToList();
        if (physicalAttributes != null)
        {
```

```
                _cache.Set(cacheKey, physicalAttributes, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return physicalAttributes;
    }
  }
}
```

ICachedServicesService.cs

```
using DataLayer.Models;

namespace MarriageAgency.Services.ServicesService
{
  public interface ICachedServicesService
  {
    public IEnumerable<Service> GetServices(int rowNumber);
    public void AddServices(string cacheKey, int rowNumber);
    public IEnumerable<Service> GetServices(string cacheKey, int rowNumber);
  }
}
```

CachedServicesService.cs

```
using Microsoft.Extensions.Caching.Memory;
using DataLayer.Data;
using DataLayer.Models;

namespace MarriageAgency.Services.ServicesService
{
  public class CachedServicesService : ICachedServicesService
  {
    private readonly IMemoryCache _cache;
    private readonly MarriageAgencyContext _context;

    public CachedServicesService(IMemoryCache cache, MarriageAgencyContext context)
    {
      _cache = cache;
      _context = context;
    }

    public void AddServices(string cacheKey, int rowNumber)
    {
      IEnumerable<Service> services = _context.Services.Take(rowNumber).ToList();
      if (services != null)
      {
        _cache.Set(cacheKey, services, new MemoryCacheEntryOptions
        {
          AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
        });
      }
    }

    public IEnumerable<Service> GetServices(int rowNumber)
    {
      return _context.Services.Take(rowNumber).ToList();
    }

    public IEnumerable<Service> GetServices(string cacheKey, int rowNumber)
    {
      IEnumerable<Service> services;
      if (!_cache.TryGetValue(cacheKey, out services))
      {
        services = _context.Services.Take(rowNumber).ToList();
        if (services != null)
        {
```

```
                _cache.Set(cacheKey, services, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return services;
    }
  }
}
```

ICachedZodiacSignsService.cs

```
using DataLayer.Models;

namespace MarriageAgency.Services.ZodiacSignsService
{
  public interface ICachedZodiacSignsService
  {
    public IEnumerable<ZodiacSign> GetZodiacSigns(int rowNumber);
    public void AddZodiacSigns(string cacheKey, int rowNumber);
    public IEnumerable<ZodiacSign> GetZodiacSigns(string cacheKey, int rowNumber);
  }
}
```

CachedZodiacSignsService.cs

```
using DataLayer.Data;
using DataLayer.Models;
using Microsoft.Extensions.Caching.Memory;

namespace MarriageAgency.Services.ZodiacSignsService
{
  public class CachedZodiacSignsService : ICachedZodiacSignsService
  {
    private readonly IMemoryCache _cache;
    private readonly MarriageAgencyContext _context;

    public CachedZodiacSignsService(IMemoryCache cache, MarriageAgencyContext context)
    {
      _cache = cache;
      _context = context;
    }

    public void AddZodiacSigns(string cacheKey, int rowNumber)
    {
      IEnumerable<ZodiacSign> zodiacSigns = _context.ZodiacSigns.Take(rowNumber).ToList();
      if (zodiacSigns != null)
      {
        _cache.Set(cacheKey, zodiacSigns, new MemoryCacheEntryOptions
        {
          AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(264)
        });
      }
    }

    public IEnumerable<ZodiacSign> GetZodiacSigns(int rowNumber)
    {
      return _context.ZodiacSigns.Take(rowNumber).ToList();
    }

    public IEnumerable<ZodiacSign> GetZodiacSigns(string cacheKey, int rowNumber)
    {
      IEnumerable<ZodiacSign> zodiacSigns;
      if (!_cache.TryGetValue(cacheKey, out zodiacSigns))
      {
        zodiacSigns = _context.ZodiacSigns.Take(rowNumber).ToList();
        if (zodiacSigns != null)
        {
```

```
                _cache.Set(cacheKey, zodiacSigns, new MemoryCacheEntryOp-
tions().SetAbsoluteExpiration(TimeSpan.FromSeconds(264)));
            }
        }
        return zodiacSigns;
    }
  }
}
```

ПРИЛОЖЕНИЕ Д
(обязательное)

Текст программы

Main.yml

```
name: .NET

on:
  push:
  pull_request:
    branches: [ main ]
    paths:
    - '**.cs'
    - '**.csproj'

env:
  DOTNET_VERSION: '8.0' # The .NET SDK version to use

jobs:
  build-and-test:
    name: build-and-test-${{matrix.os}}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]

    steps:
    - uses: actions/checkout@v4

    - name: Setup .NET Core
      uses: actions/setup-dotnet@v4
      with:
        dotnet-version: ${{ env.DOTNET_VERSION }}

    - name: Install dependencies
      run: dotnet restore lab2/EFCoreLINQ/EFCoreLINQ/EFCoreLINQ.csproj

    - name: Build
      run: dotnet build lab2/EFCoreLINQ/EFCoreLINQ/EFCoreLINQ.csproj --configuration Release --no-restore
```