МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине: «Разработка приложений баз данных для информационных систем»

на тему: «Разработка интерфейса приложения баз данных с использованием с использованием аутентификации и авторизации»

Выполнил: студент гр. ИТП-31
Король В. Н.
Принял: доцент
Асенчик О.Д.

Гомель 2023

Цель работы: получить навыки использования *ASP.NET MVC Core* для создания интерфейса типовых *web*-приложений для работы с информацией из реляционных баз данных.

Задание:

Используя разработанный ранее слой доступа к базе данным согласно своему варианту, спроектировать и создать интерфейс *Web*-приложения на основе *ASP.NET Core MVC Framework* и *Entity Framework Core*.

Web-приложение должно удовлетворять следующим требованиям:

1. Осуществлять ввод, редактирование, добавление и просмотр данных не менее чем из трех таблиц реляционной базы согласно варианту. Не менее, чем одна из таблиц должна находится на стороне отношения «многие» в схеме базы данных.
2. Иметь единое стилевое оформление, основанное на использовании мастер-страниц.
3. Иметь удобную систему навигации (строка меню, гиперссылки, кнопки), которая обеспечивает оптимальный путь перехода между двумя произвольно выбранными страницами в соответствии с логикой приложения.
4. Пользователь для работы с приложением должен пройти аутентификацию.
5. Должно поддерживать реализацию не менее двух ролевых политик.
6. Администратор должен иметь возможность управлять пользователями: просматривать, создавать, удалять и редактировать данные учетных записей.
7. Представления для просмотра данных из таблиц должны предусматривать разбиение данных на страницы, фильтрацию по одному или нескольким полям.
8. Осуществить кэширования данных для отображения с помощью встроенного инструмента кэширования – объекта *ImemoryCache*. Выводить кэшированные данные таблиц *MemoryCache* на соответствующие страницы на сайт, генерируемые с использованием представлений (*Views*). Данные в кэше хранить неизменными до проведения операций вставки, изменения или удаления данных. После проведения этих операций кэш должен формироваться заново.
9. Реализовать сохранение состояния (значений) элементов представлений, предназначенных для осуществления фильтрации, с использованием куки и (или) с объекта *Session*. Осуществить заполнение элементов представлений, предназначенных для осуществления фильтрации, при их загрузке данными, ранее сохранёнными в объекте куки и (или) *Session*.

Для проверки преподавателем следует разместить код разработанного проекта на *GitHub*.

Ход работы

В ходе выполнения лабораторной работы были разработаны классы модели, класс контекста для работы с этими моделями, класс инициализации

для создания и заполнения базы данных если ее нет и класс Middleware для подключения класса инициализации в проект. Также был разработан класс, предназначенный для кэширования данных. Кэширования производится при помощи интерфейса *IMemoryCache*. Листинг всех этих классов указан в приложении А. Пример скорости запроса без кэширования указан на рисунке 1.

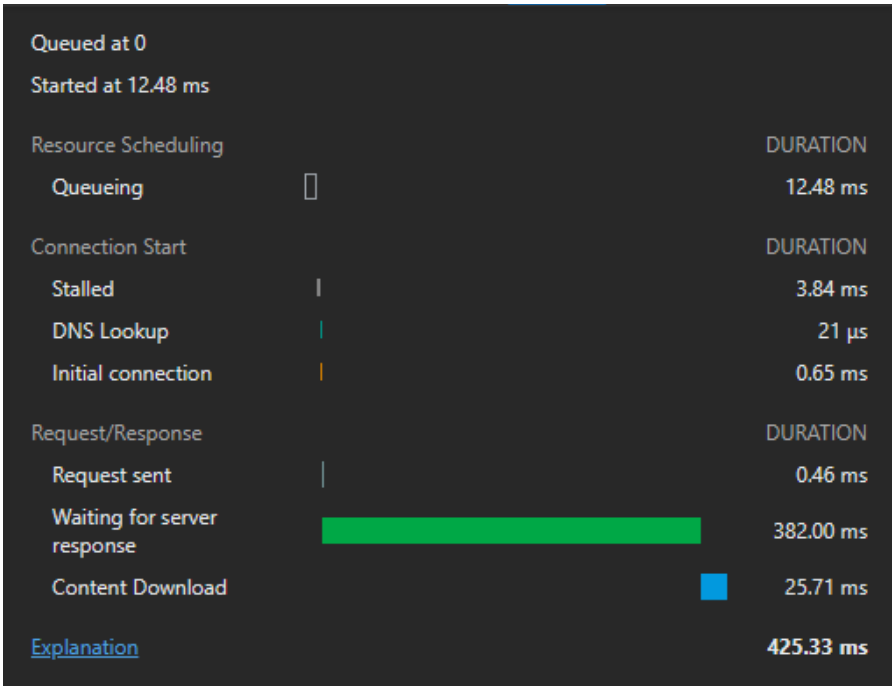

Рисунок 1 – Скорость запроса без кэширования

Далее была замерена скорость загрузки страницы данные в которую загружаются из кэша. Пример этих замеров указаны на рисунке 2.

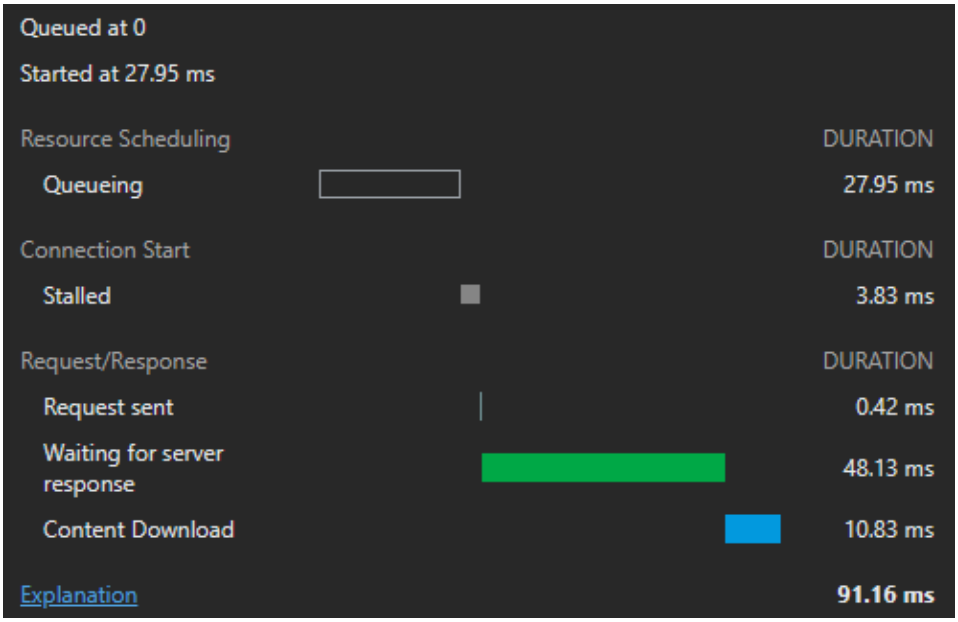

Рисунок 2 – Скорость запроса с кэшированием

Далее были разработаны контроллеры и представления для доступа к данным из базы данных. Все данные из таблицы были разделены на страницы для более удобного их просмотра. Были реализованы все *CRUD* операции такие как просмотр, удаление, обновления и создание. Был реализован механизм фильтрации с технологией куки которая помогает сохранять данные из предыдущей фильтрации. Листинг всех этих классов контролеров представлены а приложении А. Пример страницы с информацией о контрактах без фильтрации указан на рисунке 3.

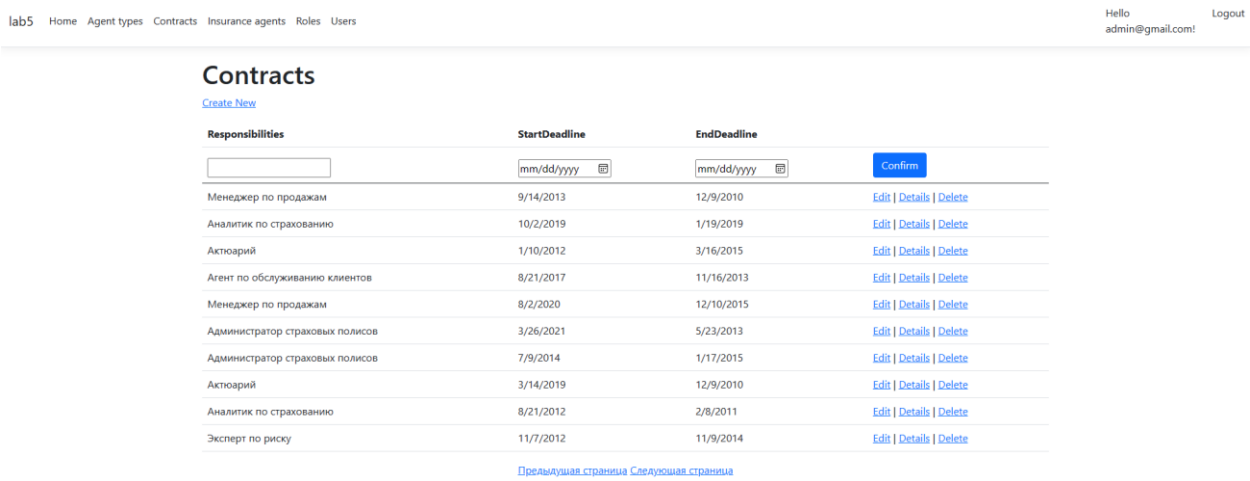

Рисунок 3 – Пример информации о контрактах без фильтрации

Далее в поля для фильтрации были внесены значения и нажата кнопка *Confirm* после этого на экран была выведена информация, которая соответствует фильтрам. И после пере захода на страницу вся эта информация будет сохранена. Пример запроса с фильтрацией указана на рисунке 4.

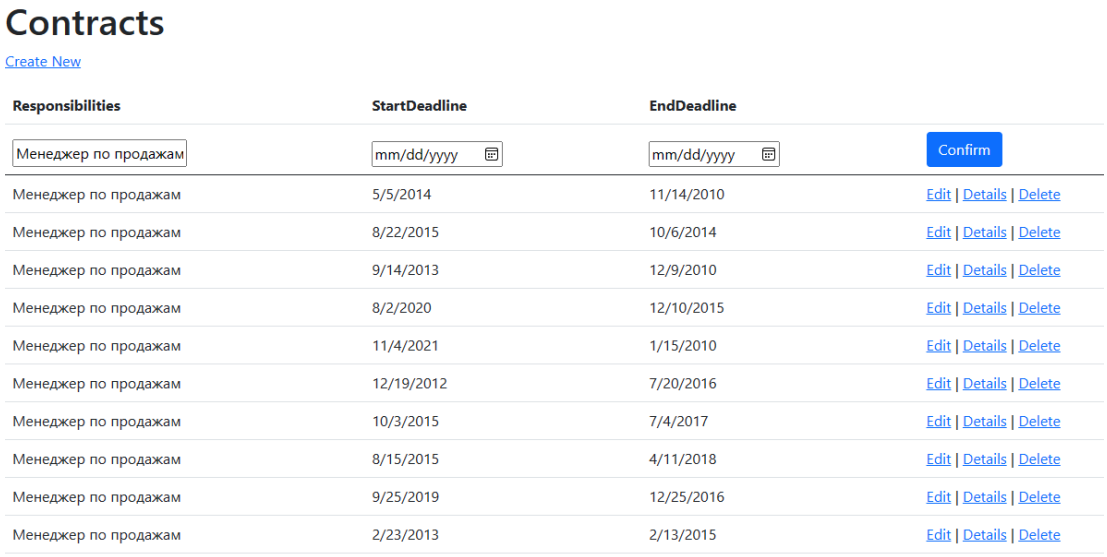

Рисунок 4 – Пример информации о контрактах с фильтрацией

Пример данного представления для таблицы типов агентов указана на рисунке 5.

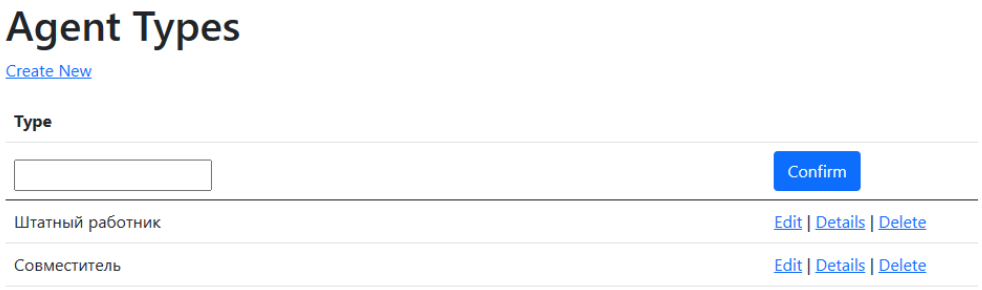

Рисунок 5 – Пример страницы с информацией о типах агентах

Пример данного представления для таблицы страховых агентов указана на рисунке 6.

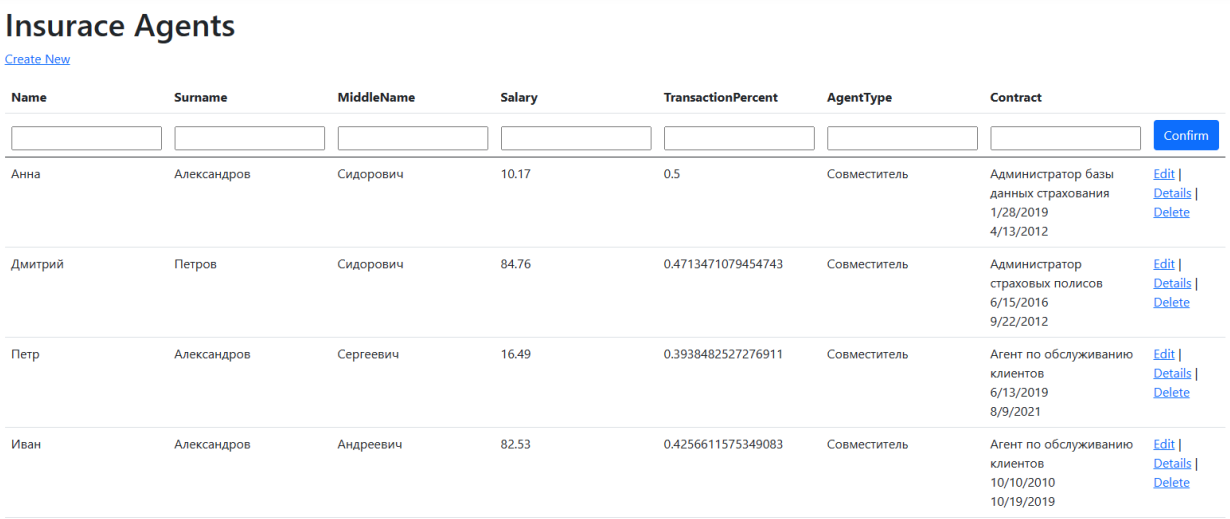

Рисунок 6 – Пример страницы с информацией о страховых агентах

Далее была реализована система аутентификации и авторизации пользователя. Для этого технология *ASP .NET Identity* реализует дополнительный таблицы в базе данных предназначения для хранения информации о пользователях. Были созданы контролеры и представления для регистрации и входа в систему пользователей с разными ролями. Пример окна для регистрации указан на рисунке 7.

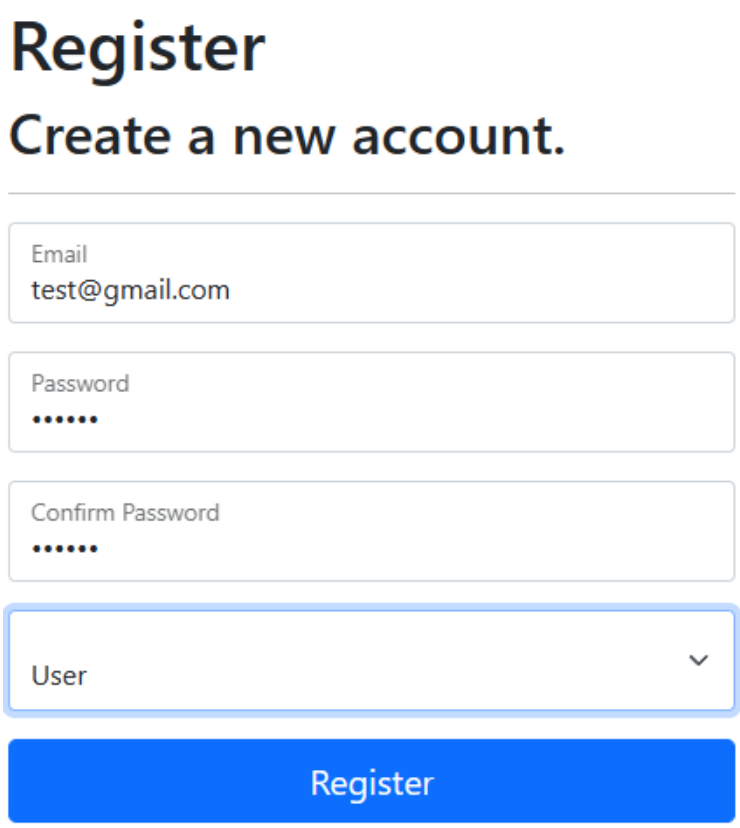

Рисунок 7 – Пример страницы для регистрации

Пример окна для входа в систему указана на рисунке 8.

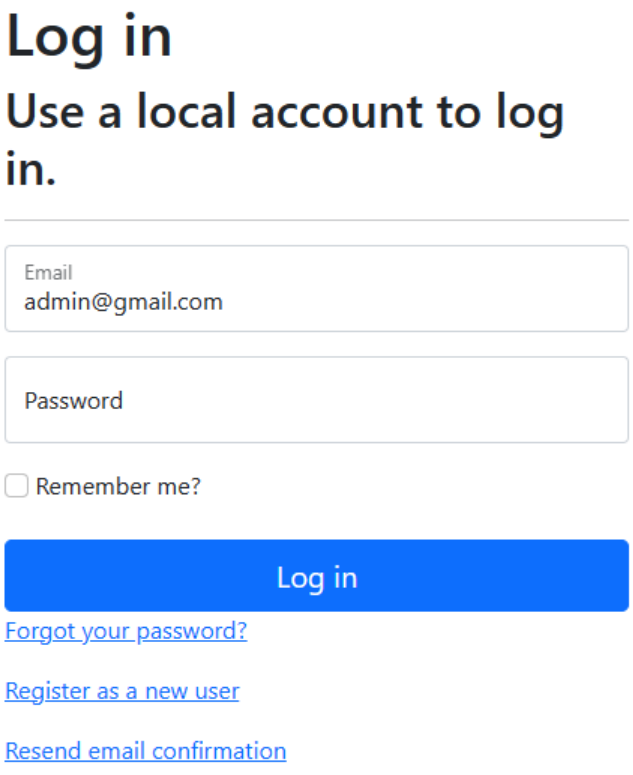

Рисунок 7 – Пример страницы для входа в систему

Для каждой страницы был реализован уровень доступа при помощи атрибута *Authorize*. Если на определенную страницу попытается получить доступ пользователь без необходимых прав будет выведено сообщение, указанное на рисунке 8.

Рисунок 8 – Пример сообщения о запрете доступа

Были реализованы две роли пользователь и администратор. Пользователь может просматривать всю информацию из таблиц бизнес-логики. Администратор может редактировать роли и пользователей. Пример станицы для редактирования ролей указана на рисунке 9.

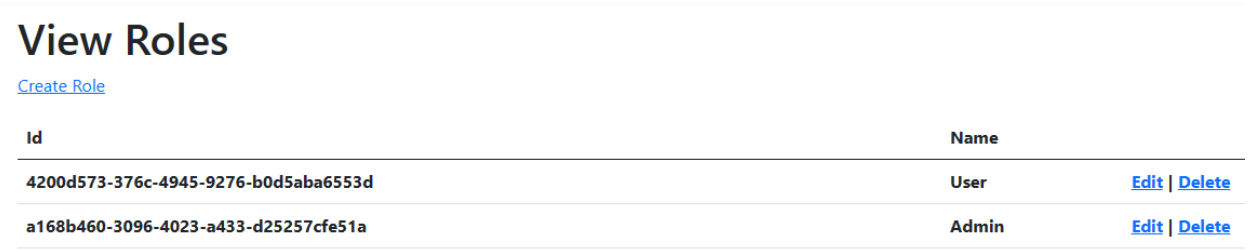

Рисунок 9 – Страница для редактирования ролей в системе

Пример станицы для редактирования пользователей указана на рисунке 10.

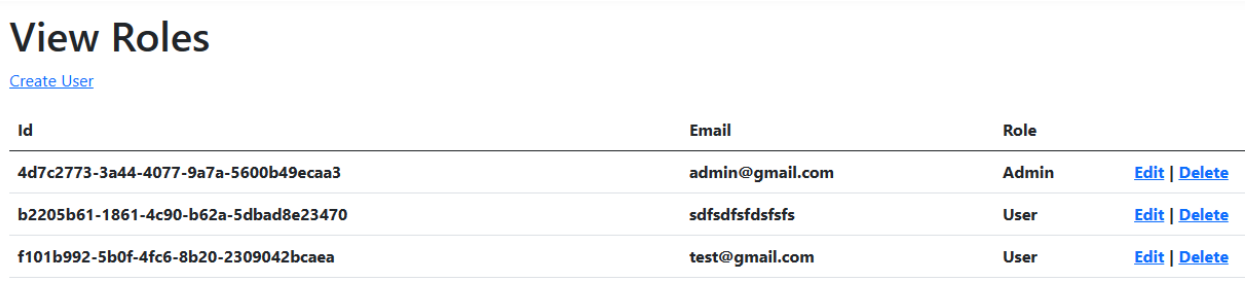

Рисунок 10 – Страница для редактирования пользователей в системе

После выполнения лабораторной работы созданные проект был добавлен в локальный *git* репозиторий а потом перенесен в *GitHub* репозиторий своего аккаунта. Чтобы ознакомится с созданным проектом можно по ссылке *Javaro3/lab5_DB (github.com)*.

Вывод: в ходе выполнения лабораторной работы была изучена такая технология *ASP .NET Core MVC*. Было разработаны классы моделей и контекста, предназначенные для работы с данными. Классы контроллера для связи моделей с представлениями. Классы представления, предназначенные для работы с данными из базы данных. Был разработан единообразный дизайн для всех страниц. При помощи технологии *ASP .NET Identity* был реализован вход в систему под разными ролями. Для более удобной работы с данными каждая выборка была разделена на страницы.

ПРИЛОЖНИЕ А

Листинг класса *Program*

```
using lab5.Data;
using lab5.Middleware;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity;

internal partial class Program {
    private static void Main(string[] args) {
        var builder = WebApplication.CreateBuilder(args);

        string connectionString =
builder.Configuration.GetConnectionString("MSSQL");
        builder.Services.AddDbContext<InsuranceCompanyContext>(option =>
option.UseSqlServer(connectionString));
        builder.Services.AddDistributedMemoryCache();
        builder.Services.AddSession();
        builder.Services
            .AddDefaultIdentity<IdentityUser>()
            .AddDefaultTokenProviders()
            .AddRoles<IdentityRole>()
            .AddEntityFrameworkStores<InsuranceCompanyContext>();

        builder.Services.AddTransient<InsuranceCompanyCache>();
        builder.Services.AddMemoryCache();
        builder.Services.AddDistributedMemoryCache();

        builder.Services.AddControllersWithViews(options => {
            options.CacheProfiles.Add("ModelCache",
                new CacheProfile() {
                    Location = ResponseCacheLocation.Any,
                    Duration = 2 * 16 + 240
                });
        });


        var app = builder.Build();

        if (!app.Environment.IsDevelopment()) {
            app.UseExceptionHandler("/Home/Error");
        }

        app.UseStaticFiles();
        app.UseSession();
        app.UseDbInitializerMiddleware();
        app.UseRouting();
        app.UseAuthorization();
        app.MapRazorPages();

        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");

        app.Run();
    }
}
```

Листинг класса *AgentType*

```
namespace lab5.Models {
    public class AgentType {
        public int Id { get; set; }
        public string Type { get; set; } = null!;
```

```
        public virtual ICollection<InsuranceAgent> InsuranceAgents { get; set; } =
new List<InsuranceAgent>();
    }
}
```

Листинг класса *Contract*

```
namespace lab5.Models {
    public class Contract {
        public int Id { get; set; }
        public string Responsibilities { get; set; } = null!;
        public DateTime StartDeadline { get; set; }
        public DateTime EndDeadline { get; set; }
        public virtual ICollection<InsuranceAgent> InsuranceAgents { get; set; } =
new List<InsuranceAgent>();

    }
}
```

Листинг класса *InsuranceAgent*

```
namespace lab5.Models {
    public class InsuranceAgent {
        public int Id { get; set; }
        public string Name { get; set; } = null!;
        public string Surname { get; set; } = null!;
        public string MiddleName { get; set; } = null!;
        public int AgentTypeId { get; set; }
        public decimal Salary { get; set; }
        public int ContractId { get; set; }
        public double TransactionPercent { get; set; }
        public virtual AgentType AgentType { get; set; } = null!;
        public virtual Contract Contract { get; set; } = null!;
    }
}
```

Листинг класса *DbInitializerMiddleware*

```
using lab5.Data;


namespace lab5.Middleware {
    public class DbInitializerMiddleware {
        private readonly RequestDelegate _next;

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

        public Task Invoke(HttpContext httpContext, InsuranceCompanyContext db) {
            if (!(httpContext.Session.Keys.Contains("database"))) {
                DbInitializer.Initialize(db);
                httpContext.Session.SetString("database", "initial");
            }
            return _next.Invoke(httpContext);
        }
    }

    public static class DbInitializerMiddlewareExtensions {
        public static IApplicationBuilder UseDbInitializerMiddleware(this
IApplicationBuilder builder) {
            return builder.UseMiddleware<DbInitializerMiddleware>();
        }
    }
}
```

Листинг класса *DbInitializer*

```
using lab5.Models;

namespace lab5.Data {
    public static class DbInitializer {
        public static void Initialize(InsuranceCompanyContext db) {
            db.Database.EnsureCreated();

            if (!db.AgentTypes.Any()) {
                InitializeAgentTypes(db);
            }

            if (!db.Contracts.Any()) {
                InitializeContracts(db);
            }

            if (!db.InsuranceAgents.Any()) {
                InitializeInsuranceAgents(db);
            }
        }

        private static void InitializeAgentTypes(InsuranceCompanyContext db) {
            db.AgentTypes.AddRange(
                new AgentType() { Type = "Штатный работник" },
                new AgentType() { Type = "Совместитель" });
            db.SaveChanges();
        }

        private static void InitializeContracts(InsuranceCompanyContext db) {
            Random rand = new();
            var responsibilities = new List<string>() {
                "Страховой агент",
                "Актюарий",
                "Агент по обслуживанию клиентов",
                "Менеджер по продажам",
                "Администратор страховых полисов",
                "Эксперт по риску",
                "Управляющий отделом страхования",
                "Аналитик по страхованию",
                "Администратор базы данных страхования"
            };

            for (int i = 0; i < 100; i++) {
                db.Contracts.Add(
                    new Contract() {
                        Responsibilities = responsibilities[rand.Next(0,
responsibilities.Count)],
                        StartDeadline = rand.NextDate(),
                        EndDeadline = rand.NextDate(),
                    });
            }

            db.SaveChanges();
        }

        private static void InitializeInsuranceAgents(InsuranceCompanyContext db) {
            Random rand = new();
            var names = new List<string>() {
                "Иван",
                "Анна",
                "Петр",
                "Екатерина",
                "Александр",
                "София",
                "Михаил",
```

```
                "Елена",
                "Дмитрий",
                "Мария"
            };

            var surnames = new List<string>() {
                "Иванов",
                "Петров",
                "Сидоров",
                "Смирнов",
                "Козлов",
                "Михайлов",
                "Александров",
                "Егоров",
                "Васильев",
                "Кузнецов"
            };

            var middleNames = new List<string>() {
                "Иванович",
                "Петрович",
                "Сидорович",
                "Михайлович",
                "Александрович",
                "Дмитриевич",
                "Сергеевич",
                "Андреевич",
                "Николаевич",
                "Геннадьевич"
            };

            for (int i = 0; i < 100; i++) {
                db.InsuranceAgents.Add(
                    new InsuranceAgent() {
                        Name = names[rand.Next(0, names.Count)],
                        Surname = surnames[rand.Next(0, surnames.Count)],
                        MiddleName = middleNames[rand.Next(0, middleNames.Count)],
                        AgentTypeId = rand.Next(1, 3),
                        ContractId = rand.Next(1, 101),
                        Salary = (decimal)(100 * rand.NextDouble()),
                        TransactionPercent = rand.NextDouble()
                    });
            }

            db.SaveChanges();
        }

        private static DateTime NextDate(this Random rand) {
            var day = rand.Next(1, 29);
            var month = rand.Next(1, 13);
            var year = rand.Next(2010, 2022);
            return new DateTime(year, month, day);
        }
    }

}
```

Листинг класса *InsuranceCompanyCache*

```
using lab5.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Caching.Memory;

namespace lab5.Data {
    public class InsuranceCompanyCache {
        private IMemoryCache _cache;
```

```
        private InsuranceCompanyContext _db;


        public InsuranceCompanyCache(InsuranceCompanyContext db, IMemoryCache
memoryCache) {
            _db = db;
            _cache = memoryCache;
        }

        public IEnumerable<AgentType> GetAgentTypes() {
            _cache.TryGetValue("AgentTypes", out IEnumerable<AgentType>?
agentTypes);

            if (agentTypes is null) {
                agentTypes = SetAgentTypes();
            }
            return agentTypes;
        }

        public IEnumerable<AgentType> SetAgentTypes() {
            var agentTypes = _db.AgentTypes.ToList();
            _cache.Set("AgentTypes", agentTypes, new
MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(100000)));
            return agentTypes;
        }

        public IEnumerable<Contract> GetContracts() {
            _cache.TryGetValue("Contracts", out IEnumerable<Contract>? contracts);

            if (contracts is null) {
                contracts = SetContracts();
            }
            return contracts;
        }

        public IEnumerable<Contract> SetContracts() {
            var contracts = _db.Contracts.ToList();
            _cache.Set("Contracts", contracts, new
MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(100000)));
            return contracts;
        }

        public IEnumerable<InsuranceAgent> GetInsuranceAgents() {
            _cache.TryGetValue("InsuranceAgents", out IEnumerable<InsuranceAgent>?
insuranceAgents);

            if (insuranceAgents is null) {
                insuranceAgents = SetInsuranceAgents();
            }
            return insuranceAgents;
        }

        public IEnumerable<InsuranceAgent> SetInsuranceAgents() {
            var insuranceAgents = _db.InsuranceAgents.Include(e =>
e.AgentType).Include(e => e.Contract).ToList();
            _cache.Set("InsuranceAgents", insuranceAgents, new
MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(100000)));
            return insuranceAgents;
        }
    }
}
```

Листинг класса *InsuranceCompanyContext*

```
using lab5.Migrations;
using lab5.Models;
```

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace lab5.Data {
    public class InsuranceCompanyContext : IdentityDbContext {
        public InsuranceCompanyContext() {}

        public InsuranceCompanyContext(DbContextOptions<InsuranceCompanyContext>
options) : base(options) { }

        public DbSet<AgentType> AgentTypes { get; set; }
        public DbSet<Contract> Contracts { get; set; }
        public DbSet<InsuranceAgent> InsuranceAgents { get; set; }
    }
}
```

Листинг класса *AgentTypesController*

```
using lab5.Data;
using lab5.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace lab5.Controllers {
    [Authorize]
    public class AgentTypesController : Controller {
        private readonly InsuranceCompanyContext _context;

        public AgentTypesController(InsuranceCompanyContext context) {
            _context = context;
        }

        // GET: AgentTypes
        public async Task<IActionResult> Index() {
            var agentTypes = GetAgentTypesCookies();
            return View(agentTypes);
        }

        [HttpPost]
        public async Task<IActionResult> Index(AgentType agentType) {
            var agentTypes = SetAgentTypesCookies(agentType);
            return View(agentTypes);
        }

        private IEnumerable<AgentType> GetAgentTypesCookies() {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            if (HttpContext.Request.Cookies.ContainsKey("AgentType")) {
                var type = HttpContext.Request.Cookies["AgentType"];
                ViewData["Type"] = type;
                if (type != "")
                    return cache.GetAgentTypes().Where(e => e.Type == type);
            }
            return cache.GetAgentTypes();
        }

        private IEnumerable<AgentType> SetAgentTypesCookies(AgentType agentType) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            ViewData["Type"] = agentType.Type;
            HttpContext.Response.Cookies.Append("AgentType", agentType.Type == null
? "" : agentType.Type);
            if (agentType.Type != null)
                return cache.GetAgentTypes().Where(e => e.Type == agentType.Type);
            return cache.GetAgentTypes();
```

```
        }

        // GET: AgentTypes/Details/5
        public async Task<IActionResult> Details(int? id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id == null) {
                return NotFound();
            }

            var agentType = cache.GetAgentTypes()
                .FirstOrDefault(m => m.Id == id);
            if (agentType == null) {
                return NotFound();
            }

            return View(agentType);
        }


        // GET: AgentTypes/Create
        public IActionResult Create() {
            return View();
        }

        // POST: AgentTypes/Create
        // To protect from overposting attacks, enable the specific properties you
want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Id,Type")] AgentType
agentType) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (ModelState.IsValid) {
                _context.Add(agentType);
                await _context.SaveChangesAsync();
                cache.SetAgentTypes();
                return RedirectToAction(nameof(Index));
            }
            return View(agentType);
        }

        // GET: AgentTypes/Edit/5
        public async Task<IActionResult> Edit(int? id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id == null) {
                return NotFound();
            }

            var agentType = cache.GetAgentTypes().FirstOrDefault(e => e.Id == id);
            if (agentType == null) {
                return NotFound();
            }
            return View(agentType);
        }

        // POST: AgentTypes/Edit/5
        // To protect from overposting attacks, enable the specific properties you
want to bind to.
```

```
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,Type")] AgentType
agentType) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id != agentType.Id) {
                return NotFound();
            }

            if (ModelState.IsValid) {
                try {
                    _context.Update(agentType);
                    await _context.SaveChangesAsync();
                    cache.SetAgentTypes();
                }
                catch (DbUpdateConcurrencyException) {
                    if (!AgentTypeExists(agentType.Id)) {
                        return NotFound();
                    }
                    else {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(agentType);
        }

        // GET: AgentTypes/Delete/5
        public async Task<IActionResult> Delete(int? id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id == null) {
                return NotFound();
            }

            var agentType = cache.GetAgentTypes()
                .FirstOrDefault(m => m.Id == id);
            if (agentType == null) {
                return NotFound();
            }

            return View(agentType);
        }

        // POST: AgentTypes/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            var agentType = cache.GetAgentTypes().FirstOrDefault(e => e.Id == id);
            if (agentType != null) {
                _context.AgentTypes.Remove(agentType);
            }

            await _context.SaveChangesAsync();
            cache.SetAgentTypes();
            return RedirectToAction(nameof(Index));
        }
```

```
        private bool AgentTypeExists(int id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            return (cache.GetAgentTypes()?.Any(e => e.Id ==
id)).GetValueOrDefault();
        }
    }
}
```

Листинг класса *AppRolesController*

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace lab5.Controllers {
    [Authorize(Roles = "Admin")]
    public class AppRolesController : Controller {
        private readonly RoleManager<IdentityRole> _roleManager;

        public AppRolesController(RoleManager<IdentityRole> roleManager) {
            _roleManager = roleManager;
        }

        public IActionResult Index() {
            var roles = _roleManager.Roles;
            return View(roles);
        }

        [HttpGet]
        public IActionResult Create() {
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Create(IdentityRole model) {
            if (!_roleManager.RoleExistsAsync(model.Name).GetAwaiter().GetResult())
{
                _roleManager.CreateAsync(new
IdentityRole(model.Name)).GetAwaiter().GetResult();
            }
            return RedirectToAction(nameof(Index));
        }

        public async Task<IActionResult> Delete(string? id) {
            if (id == null || _roleManager.Roles == null) {
                return NotFound();
            }

            var role = await _roleManager.Roles
                .FirstOrDefaultAsync(m => m.Id == id);
            if (role == null) {
                return NotFound();
            }

            await _roleManager.DeleteAsync(role);

            return RedirectToAction(nameof(Index));
        }

        public async Task<IActionResult> Edit(string? id) {
            if (id == null || _roleManager.Roles == null) {
                return NotFound();
```

```
            }

            var role = await _roleManager.Roles.FirstOrDefaultAsync(e => e.Id ==
id);
            if (role == null) {
                return NotFound();
            }
            return View(role);
        }

        [HttpPost]
        public async Task<IActionResult> Edit(IdentityRole role) {
            var updateRole = _roleManager.Roles.FirstOrDefault(e => e.Id ==
role.Id);
            updateRole.Name = role.Name;
            _roleManager.UpdateAsync(updateRole).GetAwaiter().GetResult();
            return RedirectToAction(nameof(Index));
        }
    }
}
```

Листинг класса *AppUsersController*

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace lab5.Controllers {

    [Authorize(Roles = "Admin")]
    public class AppUsersController : Controller {
        private readonly UserManager<IdentityUser> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;

        public AppUsersController(UserManager<IdentityUser> userManager,
RoleManager<IdentityRole> roleManager) {
            _userManager = userManager;
            _roleManager = roleManager;
        }

        public IActionResult Index() {
            List<(IdentityUser User, string Role)> usersRole = new();
            var users = _userManager.Users.ToList();
            foreach (var user in users) {
                var role =
_userManager.GetRolesAsync(user).GetAwaiter().GetResult();
                usersRole.Add(new(user, role[0]));
            }
            return View(usersRole);
        }

        public async Task<IActionResult> Delete(string? id) {
            if (id == null || _userManager.Users == null) {
                return NotFound();
            }

            var user = await _userManager.Users
                .FirstOrDefaultAsync(m => m.Id == id);
            if (user == null) {
                return NotFound();
            }

            await _userManager.DeleteAsync(user);
```

```
            return RedirectToAction(nameof(Index));
        }

        public IActionResult Create() {
            ViewData["RoleList"] = _roleManager.Roles.Select(e => new
SelectListItem() {
                Text = e.Name,
                Value = e.Name
            });
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Create(string email, string password,
string role) {
            var user = new IdentityUser() { UserName = email, Email = email };

            var result = await _userManager.CreateAsync(user, password);
            if (result.Succeeded) {
                var currentUser = await _userManager.FindByNameAsync(user.UserName);

                var roleresult = await _userManager.AddToRoleAsync(currentUser,
role);
            }


            return RedirectToAction(nameof(Index));
        }

        public IActionResult Edit(string? id) {
            ViewData["RoleList"] = _roleManager.Roles.Select(e => new
SelectListItem() {
                Text = e.Name,
                Value = e.Name
            });
            var user = _userManager.Users.FirstOrDefault(e => e.Id == id);
            var role = _userManager.GetRolesAsync(user).GetAwaiter().GetResult()[0];
            return View((User: user, Role: role));
        }

        [HttpPost]
        public async Task<IActionResult> Edit(string id, string email, string role)
{
            var user = _userManager.Users.FirstOrDefault(e => e.Id == id);
            user.UserName = email;
            user.Email = email;
            _userManager.UpdateAsync(user).GetAwaiter().GetResult();
            _userManager.AddToRoleAsync(user, role).GetAwaiter().GetResult();
            var roles = _roleManager.Roles.ToList();
            foreach (var i in roles) {
                if (i.Name != role)
                    _userManager.RemoveFromRoleAsync(user,
i.Name).GetAwaiter().GetResult();
            }
            return RedirectToAction(nameof(Index));
        }
    }
}
```

Листинг класса *ContractsController*

```
using lab5.Data;
using lab5.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
```

```
namespace lab5.Controllers {
    [Authorize]
    public class ContractsController : Controller {
        private readonly InsuranceCompanyContext _context;

        public ContractsController(InsuranceCompanyContext context) {
            _context = context;
        }

        // GET: Contracts
        public async Task<IActionResult> Index(int page = 1, int pageSize = 10) {
            var agentTypes = GetContractCookies();
            ViewData["ItemsCount"] = agentTypes.Count();
            agentTypes = agentTypes.Skip((page - 1) *
pageSize).Take(pageSize).ToList();
            ViewData["Page"] = page;
            ViewData["PageSize"] = pageSize;
            return View(agentTypes);
        }

        [HttpPost]
        public async Task<IActionResult> Index(Contract contract, int page = 1, int
pageSize = 10) {
            var agentTypes = SetContractsCookies(contract);
            ViewData["ItemsCount"] = agentTypes.Count();
            agentTypes = agentTypes.Skip((page - 1) *
pageSize).Take(pageSize).ToList();
            ViewData["Page"] = page;
            ViewData["PageSize"] = pageSize;
            return View(agentTypes);
        }

        private IEnumerable<Contract> GetContractCookies() {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            IEnumerable<Contract> contracts = cache.GetContracts();

            if (HttpContext.Request.Cookies.ContainsKey("ContractResponsibilities"))
{
                var responsibilities =
HttpContext.Request.Cookies["ContractResponsibilities"];
                ViewData["ContractResponsibilities"] = responsibilities;
                contracts = contracts.Where(e => responsibilities == "" ||
e.Responsibilities == responsibilities);
            }

            if (HttpContext.Request.Cookies.ContainsKey("ContractStartDeadline")) {
                var startDeadline =
HttpContext.Request.Cookies["ContractStartDeadline"];
                ViewData["ContractStartDeadline"] = startDeadline;
                if (startDeadline != "") {
                    var startDeadlineDate = DateTime.Parse(startDeadline);
                    contracts = contracts.Where(e => e.StartDeadline >=
startDeadlineDate);
                }
            }

            if (HttpContext.Request.Cookies.ContainsKey("ContractEndDeadline")) {
                var endDeadline =
HttpContext.Request.Cookies["ContractEndDeadline"];
                ViewData["ContractEndDeadline"] = endDeadline;
                if (endDeadline != "") {
                    var endDeadlineDate = DateTime.Parse(endDeadline);
```

```
                    contracts = contracts.Where(e => e.EndDeadline <=
endDeadlineDate);
                }
            }

            return contracts;
        }

        private IEnumerable<Contract> SetContractsCookies(Contract contract) {
            ViewData["ContractResponsibilities"] = contract.Responsibilities;
            ViewData["ContractStartDeadline"] = contract.StartDeadline == default ?
null : contract.StartDeadline;
            ViewData["ContractEndDeadline"] = contract.EndDeadline == default ? null
: contract.EndDeadline;
            HttpContext.Response.Cookies.Append("ContractResponsibilities",
contract.Responsibilities == null ? "" : contract.Responsibilities);
            HttpContext.Response.Cookies.Append("ContractStartDeadline",
contract.StartDeadline == default ? "" : contract.StartDeadline.ToString());
            HttpContext.Response.Cookies.Append("ContractEndDeadline",
contract.EndDeadline == default ? "" : contract.EndDeadline.ToString());

            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            IEnumerable<Contract> contracts = cache.GetContracts();
            if (contract.Responsibilities != null)
                contracts = contracts.Where(e => e.Responsibilities ==
contract.Responsibilities);

            if (contract.StartDeadline != default)
                contracts = contracts.Where(e => e.StartDeadline >=
contract.StartDeadline);

            if (contract.EndDeadline != default)
                contracts = contracts.Where(e => e.EndDeadline <=
contract.EndDeadline);

            return contracts;
        }

        // GET: Contracts/Details/5
        public async Task<IActionResult> Details(int? id) {
            if (id == null) {
                return NotFound();
            }

            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            var contract = cache.GetContracts()
                .FirstOrDefault(m => m.Id == id);
            if (contract == null) {
                return NotFound();
            }

            return View(contract);
        }

        // GET: Contracts/Create
        public IActionResult Create() {
            return View();
        }

        // POST: Contracts/Create
        // To protect from overposting attacks, enable the specific properties you
want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Id,Responsibilities,StartDeadline,EndDeadline")] Contract contract) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (ModelState.IsValid) {
                _context.Add(contract);
                await _context.SaveChangesAsync();
                cache.SetContracts();
                return RedirectToAction(nameof(Index));
            }
            return View(contract);
        }

        // GET: Contracts/Edit/5
        public async Task<IActionResult> Edit(int? id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id == null) {
                return NotFound();
            }

            var contract = cache.GetContracts().FirstOrDefault(e => e.Id == id);
            if (contract == null) {
                return NotFound();
            }
            return View(contract);
        }

        // POST: Contracts/Edit/5
        // To protect from overposting attacks, enable the specific properties you
want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,Responsibilities,StartDeadline,EndDeadline")] Contract contract) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            if (id != contract.Id) {
                return NotFound();
            }

            if (ModelState.IsValid) {
                try {
                    _context.Update(contract);
                    cache.SetContracts();
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException) {
                    if (!ContractExists(contract.Id)) {
                        return NotFound();
                    }
                    else {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(contract);
        }
```

```
        // GET: Contracts/Delete/5
        public async Task<IActionResult> Delete(int? id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id == null) {
                return NotFound();
            }

            var contract = cache.GetContracts().FirstOrDefault(m => m.Id == id);
            if (contract == null) {
                return NotFound();
            }

            return View(contract);
        }

        // POST: Contracts/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            var contract = cache.GetContracts().FirstOrDefault(e => e.Id == id);
            if (contract != null) {
                _context.Contracts.Remove(contract);
            }

            await _context.SaveChangesAsync();
            cache.SetContracts();
            return RedirectToAction(nameof(Index));
        }

        private bool ContractExists(int id) {
            var cache =
HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            return (cache.GetContracts()?.Any(e => e.Id == id)).GetValueOrDefault();
        }
    }
}
```

Листинг класса *AgentType*

```
using lab5.Data;
using lab5.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace lab5.Controllers {
    [Authorize]
    public class InsuranceAgentsController : Controller {
        private readonly InsuranceCompanyContext _context;

        public InsuranceAgentsController(InsuranceCompanyContext context) {
            _context = context;
        }

        // GET: InsuranceAgents
        public async Task<IActionResult> Index(int page = 1, int pageSize = 10) {
            var insuranceAgents = GetInsuranceAgentsCookies();
            ViewData["ItemsCount"] = insuranceAgents.Count();
            insuranceAgents = insuranceAgents.Skip((page - 1) *
pageSize).Take(pageSize).ToList();
            ViewData["Page"] = page;
```

```
            ViewData["PageSize"] = pageSize;
            return View(insuranceAgents);
        }

        [HttpPost]
        public async Task<IActionResult> Index(InsuranceAgent insuranceAgent, string
type, string responsibilities, int page = 1, int pageSize = 10) {
            page = 1;
            var insuranceAgents = SetInsuranceAgentsCookies(insuranceAgent, type,
responsibilities);
            ViewData["ItemsCount"] = insuranceAgents.Count();
            insuranceAgents = insuranceAgents.Skip((page - 1) *
pageSize).Take(pageSize).ToList();
            ViewData["Page"] = page;
            ViewData["PageSize"] = pageSize;
            return View(insuranceAgents);
        }

        private IEnumerable<InsuranceAgent> GetInsuranceAgentsCookies() {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            IEnumerable<InsuranceAgent> insuranceAgents = cache.GetInsuranceAgents();

            if (HttpContext.Request.Cookies.ContainsKey("InsuranceAgentName")) {
                var name = HttpContext.Request.Cookies["InsuranceAgentName"];
                ViewData["InsuranceAgentName"] = name;
                insuranceAgents = insuranceAgents.Where(e => name == "" || e.Name ==
name);
            }

            if (HttpContext.Request.Cookies.ContainsKey("InsuranceAgentSurname")) {
                var surname = HttpContext.Request.Cookies["InsuranceAgentSurname"];
                ViewData["InsuranceAgentSurname"] = surname;
                insuranceAgents = insuranceAgents.Where(e => surname == "" || e.Surname
== surname);
            }

            if (HttpContext.Request.Cookies.ContainsKey("InsuranceAgentMiddleName")) {
                var middleName =
HttpContext.Request.Cookies["InsuranceAgentMiddleName"];
                ViewData["InsuranceAgentMiddleName"] = middleName;
                insuranceAgents = insuranceAgents.Where(e => middleName == "" ||
e.MiddleName == middleName);
            }

            if (HttpContext.Request.Cookies.ContainsKey("InsuranceAgentSalary")) {
                var salary = HttpContext.Request.Cookies["InsuranceAgentSalary"];
                ViewData["InsuranceAgentSalary"] = salary;
                if (salary != "") {
                    var salaryDecimal = Decimal.Parse(salary);
                    insuranceAgents = insuranceAgents.Where(e => e.Salary >=
salaryDecimal);
                }
            }

            if
(HttpContext.Request.Cookies.ContainsKey("InsuranceAgentTransactionPercent")) {
                var transactionPercent =
HttpContext.Request.Cookies["InsuranceAgentTransactionPercent"];
                ViewData["InsuranceAgentTransactionPercent"] = transactionPercent;
                if (transactionPercent != "") {
                    var transactionPercentDouble = Double.Parse(transactionPercent);
                    insuranceAgents = insuranceAgents.Where(e => e.TransactionPercent >=
transactionPercentDouble);
                }
            }

            if (HttpContext.Request.Cookies.ContainsKey("InsuranceAgentType")) {
                var type = HttpContext.Request.Cookies["InsuranceAgentType"];
                ViewData["InsuranceAgentType"] = type;
```

```
                insuranceAgents = insuranceAgents.Where(e => type == "" ||
e.AgentType.Type == type);
            }

            if
(HttpContext.Request.Cookies.ContainsKey("InsuranceAgentResponsibilities")) {
                var responsibilities =
HttpContext.Request.Cookies["InsuranceAgentResponsibilities"];
                ViewData["InsuranceAgentResponsibilities"] = responsibilities;
                insuranceAgents = insuranceAgents.Where(e => responsibilities == "" ||
e.Contract.Responsibilities == responsibilities);
            }

            return insuranceAgents;
        }

        private IEnumerable<InsuranceAgent> SetInsuranceAgentsCookies(InsuranceAgent
insuranceAgent, string type, string responsibilities) {
            ViewData["InsuranceAgentName"] = insuranceAgent.Name;
            ViewData["InsuranceAgentSurname"] = insuranceAgent.Surname;
            ViewData["InsuranceAgentMiddleName"] = insuranceAgent.MiddleName;
            ViewData["InsuranceAgentSalary"] = insuranceAgent.Salary;
            ViewData["InsuranceAgentTransactionPercent"] =
insuranceAgent.TransactionPercent;
            ViewData["InsuranceAgentType"] = type;
            ViewData["InsuranceAgentResponsibilities"] = responsibilities;


            HttpContext.Response.Cookies.Append("InsuranceAgentName",
insuranceAgent.Name == null ? "" : insuranceAgent.Name);
            HttpContext.Response.Cookies.Append("InsuranceAgentSurname",
insuranceAgent.Surname == null ? "" : insuranceAgent.Surname);
            HttpContext.Response.Cookies.Append("InsuranceAgentMiddleName",
insuranceAgent.MiddleName == null ? "" : insuranceAgent.MiddleName);
            HttpContext.Response.Cookies.Append("InsuranceAgentType", type == null ? ""
: type);
            HttpContext.Response.Cookies.Append("InsuranceAgentResponsibilities",
responsibilities == null ? "" : responsibilities);
            HttpContext.Response.Cookies.Append("InsuranceAgentSalary",
insuranceAgent.Salary == default ? "" : insuranceAgent.Salary.ToString());
            HttpContext.Response.Cookies.Append("InsuranceAgentTransactionPercent",
insuranceAgent.TransactionPercent == default ? "" :
insuranceAgent.TransactionPercent.ToString());

            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            IEnumerable<InsuranceAgent> insuranceAgents = cache.GetInsuranceAgents();

            if (insuranceAgent.Name != null)
                insuranceAgents = insuranceAgents.Where(e => e.Name ==
insuranceAgent.Name);

            if (insuranceAgent.Surname != null)
                insuranceAgents = insuranceAgents.Where(e => e.Surname ==
insuranceAgent.Surname);

            if (insuranceAgent.MiddleName != null)
                insuranceAgents = insuranceAgents.Where(e => e.MiddleName ==
insuranceAgent.MiddleName);

            if (type != null)
                insuranceAgents = insuranceAgents.Where(e => e.AgentType.Type == type);

            if (responsibilities != null)
                insuranceAgents = insuranceAgents.Where(e => e.Contract.Responsibilities
== responsibilities);

            if (insuranceAgent.Salary != default)
                insuranceAgents = insuranceAgents.Where(e => e.Salary >=
insuranceAgent.Salary);
```

```
            if (insuranceAgent.TransactionPercent != default)
                insuranceAgents = insuranceAgents.Where(e => e.TransactionPercent >=
insuranceAgent.TransactionPercent);

            return insuranceAgents;
        }

        // GET: InsuranceAgents/Details/5
        public async Task<IActionResult> Details(int? id) {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            if (id == null) {
                return NotFound();
            }

            var insuranceAgent = cache.GetInsuranceAgents().FirstOrDefault(m => m.Id ==
id);
            if (insuranceAgent == null) {
                return NotFound();
            }

            return View(insuranceAgent);
        }

        // GET: InsuranceAgents/Create
        public IActionResult Create() {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            ViewData["AgentTypeId"] = new SelectList(cache.GetAgentTypes(), "Id",
"Type");
            ViewData["ContractId"] = new SelectList(cache.GetContracts(), "Id",
"Responsibilities");
            return View();
        }

        // POST: InsuranceAgents/Create
        // To protect from overposting attacks, enable the specific properties you want
to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Id,Name,Surname,MiddleName,AgentTypeId,Salary,ContractId,TransactionPercen
t")] InsuranceAgent insuranceAgent) {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (ModelState.ErrorCount <= 2) {
                _context.Add(insuranceAgent);
                await _context.SaveChangesAsync();
                cache.SetInsuranceAgents();
                return RedirectToAction(nameof(Index));
            }
            ViewData["AgentTypeId"] = new SelectList(cache.GetAgentTypes(), "Id",
"Type", insuranceAgent.AgentTypeId);
            ViewData["ContractId"] = new SelectList(cache.GetContracts(), "Id",
"Responsibilities", insuranceAgent.ContractId);
            return View(insuranceAgent);
        }

        // GET: InsuranceAgents/Edit/5
        public async Task<IActionResult> Edit(int? id) {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id == null) {
                return NotFound();
            }

            var insuranceAgent = cache.GetInsuranceAgents().FirstOrDefault(e => e.Id ==
id);
            if (insuranceAgent == null) {
```

```
                return NotFound();
            }
            ViewData["AgentTypeId"] = new SelectList(cache.GetAgentTypes(), "Id",
"Type", insuranceAgent.AgentTypeId);
            ViewData["ContractId"] = new SelectList(cache.GetContracts(), "Id",
"Responsibilities", insuranceAgent.ContractId);
            return View(insuranceAgent);
        }

        // POST: InsuranceAgents/Edit/5
        // To protect from overposting attacks, enable the specific properties you want
to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,Surname,MiddleName,AgentTypeId,Salary,ContractId,TransactionPercent")]
InsuranceAgent insuranceAgent) {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id != insuranceAgent.Id) {
                return NotFound();
            }

            if (ModelState.ErrorCount <= 2) {
                try {
                    _context.Update(insuranceAgent);
                    await _context.SaveChangesAsync();
                    cache.SetInsuranceAgents();
                }
                catch (DbUpdateConcurrencyException) {
                    if (!InsuranceAgentExists(insuranceAgent.Id)) {
                        return NotFound();
                    }
                    else {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["AgentTypeId"] = new SelectList(cache.GetAgentTypes(), "Id",
"Type", insuranceAgent.AgentTypeId);
            ViewData["ContractId"] = new SelectList(cache.GetContracts(), "Id",
"Responsibilities", insuranceAgent.ContractId);
            return View(insuranceAgent);
        }

        // GET: InsuranceAgents/Delete/5
        public async Task<IActionResult> Delete(int? id) {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();

            if (id == null) {
                return NotFound();
            }

            var insuranceAgent = cache.GetInsuranceAgents().FirstOrDefault(m => m.Id ==
id);
            if (insuranceAgent == null) {
                return NotFound();
            }

            return View(insuranceAgent);
        }

        // POST: InsuranceAgents/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id) {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
```

```
            var insuranceAgent = cache.GetInsuranceAgents().FirstOrDefault(e => e.Id ==
id);
            if (insuranceAgent != null) {
                _context.InsuranceAgents.Remove(insuranceAgent);
            }

            await _context.SaveChangesAsync();
            cache.SetInsuranceAgents();
            return RedirectToAction(nameof(Index));
        }

        private bool InsuranceAgentExists(int id) {
            var cache = HttpContext.RequestServices.GetService<InsuranceCompanyCache>();
            return (cache.GetInsuranceAgents()?.Any(e => e.Id ==
id)).GetValueOrDefault();
        }
    }
}
```

Листинг класса *RegisterModel*

```
// Licensed to the .NET Foundation under one or more agreements.
// The .NET Foundation licenses this file to you under the MIT license.
#nullable disable

using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.WebUtilities;
using System.ComponentModel.DataAnnotations;
using System.Text;
using System.Text.Encodings.Web;

namespace lab5.Areas.Identity.Pages.Account {
    public class RegisterModel : PageModel {
        private readonly SignInManager<IdentityUser> _signInManager;
        private readonly UserManager<IdentityUser> _userManager;
        private readonly IUserStore<IdentityUser> _userStore;
        private readonly IUserEmailStore<IdentityUser> _emailStore;
        private readonly ILogger<RegisterModel> _logger;
        private readonly IEmailSender _emailSender;
        private readonly RoleManager<IdentityRole> _roleManager;
        public RegisterModel(
            UserManager<IdentityUser> userManager,
            IUserStore<IdentityUser> userStore,
            SignInManager<IdentityUser> signInManager,
            ILogger<RegisterModel> logger,
            IEmailSender emailSender,
            RoleManager<IdentityRole> roleManager) {
            _userManager = userManager;
            _userStore = userStore;
            _emailStore = GetEmailStore();
            _signInManager = signInManager;
            _logger = logger;
            _emailSender = emailSender;
            _roleManager = roleManager;
        }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
```

```
        ///     directly from your code. This API may change or be removed in future
releases.
        /// </summary>
        [BindProperty]
        public InputModel Input { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future
releases.
        /// </summary>
        public string ReturnUrl { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future
releases.
        /// </summary>
        public IList<AuthenticationScheme> ExternalLogins { get; set; }

        /// <summary>
        ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future
releases.
        /// </summary>
        public class InputModel {
            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Required]
            [EmailAddress]
            [Display(Name = "Email")]
            public string Email { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [Required]
            [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at
max {1} characters long.", MinimumLength = 6)]
            [DataType(DataType.Password)]
            [Display(Name = "Password")]
            public string Password { get; set; }

            /// <summary>
            ///     This API supports the ASP.NET Core Identity default UI
infrastructure and is not intended to be used
            ///     directly from your code. This API may change or be removed in
future releases.
            /// </summary>
            [DataType(DataType.Password)]
            [Display(Name = "Confirm password")]
            [Compare("Password", ErrorMessage = "The password and confirmation
password do not match.")]
            public string ConfirmPassword { get; set; }
```

```
            [Required]
            public string? Role { get; set; }

            [ValidateNever]
            public IEnumerable<SelectListItem> RoleList { get; set; }
        }


        public async Task OnGetAsync(string returnUrl = null) {
            ReturnUrl = returnUrl;
            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            Input = new InputModel() {
                RoleList = _roleManager.Roles.Select(x => x.Name).Select(e => new
SelectListItem() {
                    Text = e,
                    Value = e
                })
            };
        }

        public async Task<IActionResult> OnPostAsync(string returnUrl = null) {
            returnUrl ??= Url.Content("~/");
            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid) {
                var user = CreateUser();

                await _userStore.SetUserNameAsync(user, Input.Email,
CancellationToken.None);
                await _emailStore.SetEmailAsync(user, Input.Email,
CancellationToken.None);
                var result = await _userManager.CreateAsync(user, Input.Password);

                if (result.Succeeded) {
                    _logger.LogInformation("User created a new account with
password.");

                    await _userManager.AddToRoleAsync(user, Input.Role);

                    var userId = await _userManager.GetUserIdAsync(user);
                    var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
                    code =
WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
                    var callbackUrl = Url.Page(
                        "/Account/ConfirmEmail",
                        pageHandler: null,
                        values: new { area = "Identity", userId = userId, code =
code, returnUrl = returnUrl },
                        protocol: Request.Scheme);

                    await _emailSender.SendEmailAsync(Input.Email, "Confirm your
email",
                        $"Please confirm your account by <a
href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

                    if (_userManager.Options.SignIn.RequireConfirmedAccount) {
                        return RedirectToPage("RegisterConfirmation", new { email =
Input.Email, returnUrl = returnUrl });
                    }
                    else {
                        await _signInManager.SignInAsync(user, isPersistent: false);
                        return LocalRedirect(returnUrl);
```

```
                    }
                }
                foreach (var error in result.Errors) {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
            }

            // If we got this far, something failed, redisplay form
            return Page();
        }

        private IdentityUser CreateUser() {
            try {
                return Activator.CreateInstance<IdentityUser>();
            }
            catch {
                throw new InvalidOperationException($"Can't create an instance of
'{nameof(IdentityUser)}'. " +
                    $"Ensure that '{nameof(IdentityUser)}' is not an abstract class
and has a parameterless constructor, or alternatively " +
                    $"override the register page in
/Areas/Identity/Pages/Account/Register.cshtml");
            }
        }

        private IUserEmailStore<IdentityUser> GetEmailStore() {
            if (!_userManager.SupportsUserEmail) {
                throw new NotSupportedException("The default UI requires a user
store with email support.");
            }
            return (IUserEmailStore<IdentityUser>)_userStore;
        }
    }
}
```