**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: **«РПБДИС»**

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

Выполнил: студент гр. ИТП-31

Сетко А.И.

Принял: старший преподаватель

Асенчик О.Д.

Гомель 2019

**Цель**:научиться использовать *ASP.NET MVC Сore* для создания 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.NETCoreMVCFramework* и *EntityFrameworkCore*.

*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 находиться 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 страницысайта, 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 сохраненными в объекте куки и (или) *Session*.

**Ход работы**

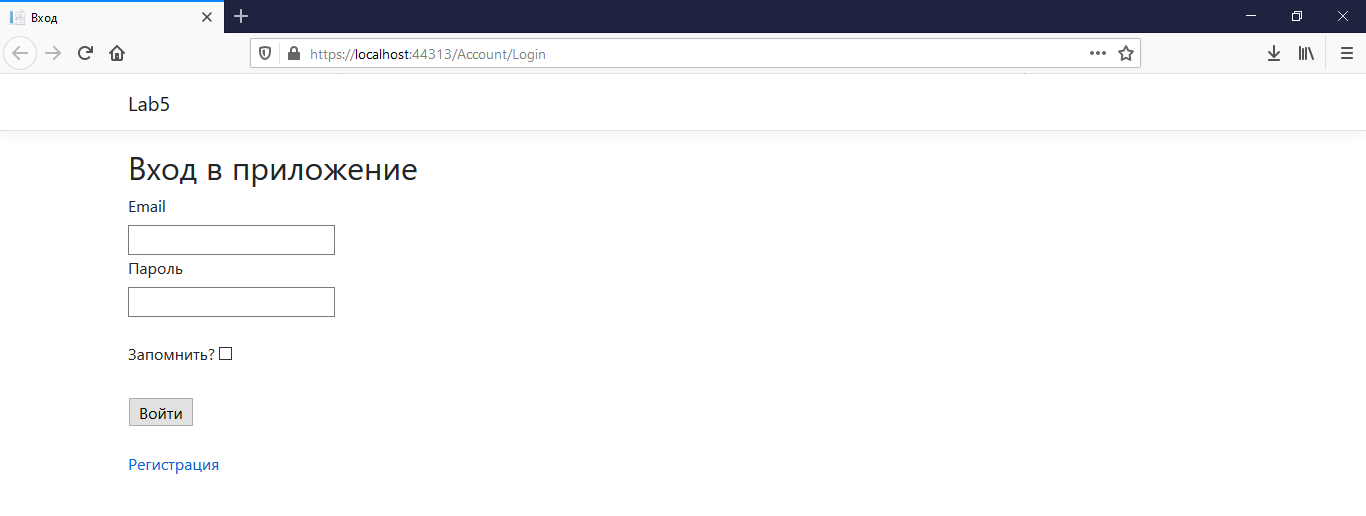
****

Рисунок 1 – Копия экрана браузера домашней страницы

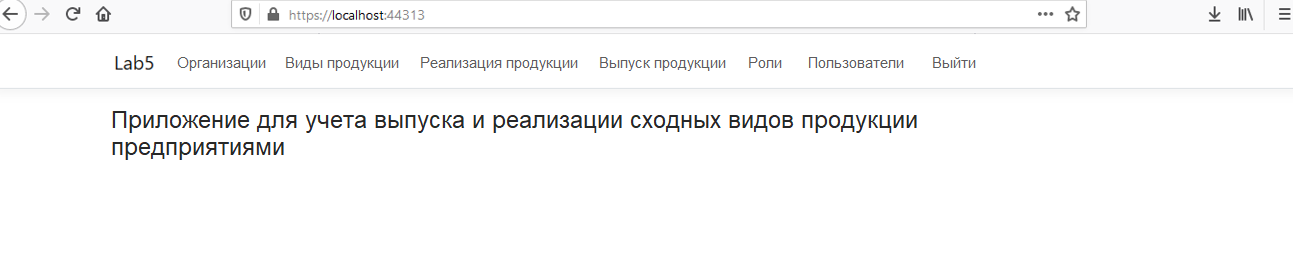


Рисунок 2 – Копия экрана браузера домашней страницы после авторизации администратором

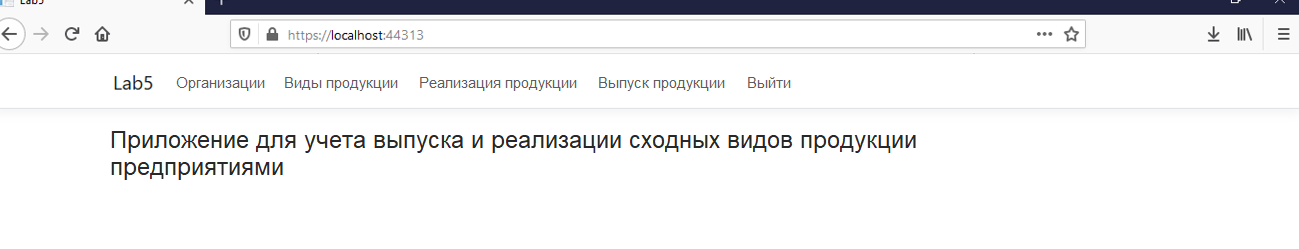


Рисунок 3 – Копия экрана браузера домашней страницы после авторизации обычным пользователем

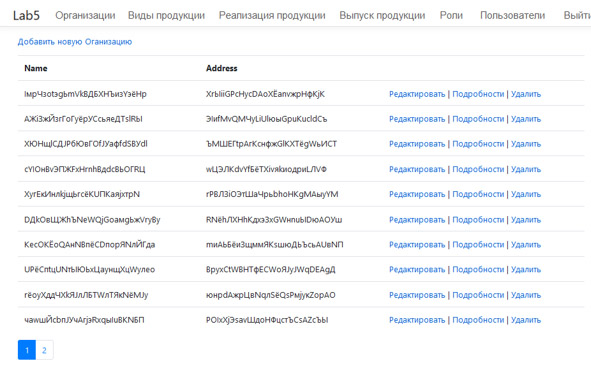


Рисунок 4– Копия экрана браузера со списком организаций

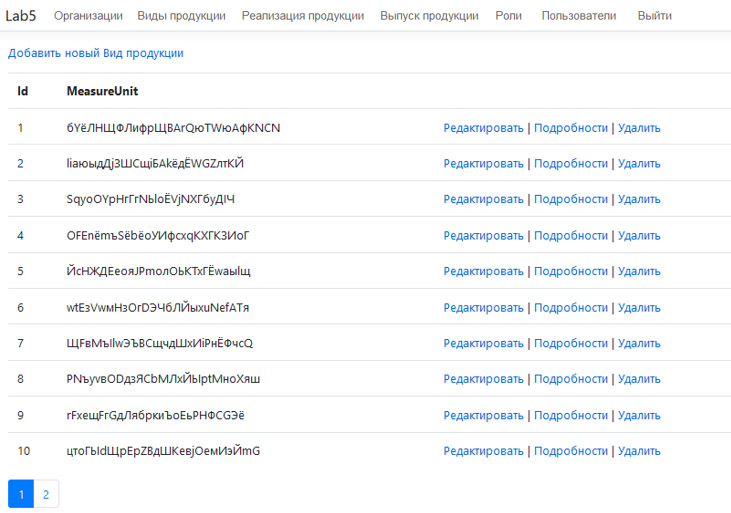


Рисунок 5 – Копия экрана браузера со списком видов продукции

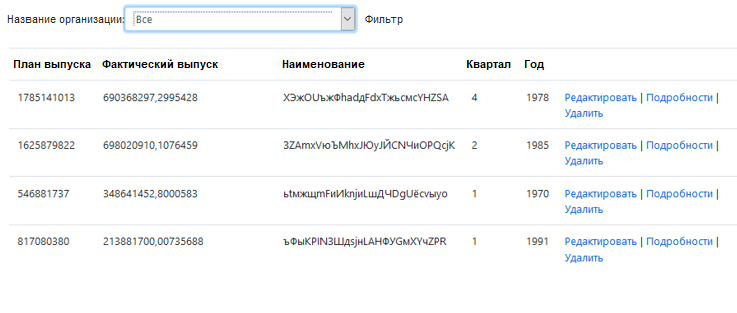


Рисунок 6 – Копия экрана браузера со списком планируемого и фактического выпуска продукции организациями

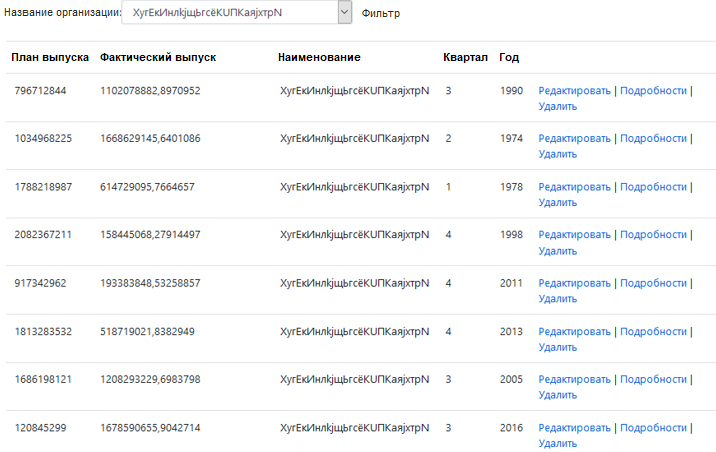


Рисунок 7 – Копия экрана браузера со фактического выпуска продукции

выбранной организацией

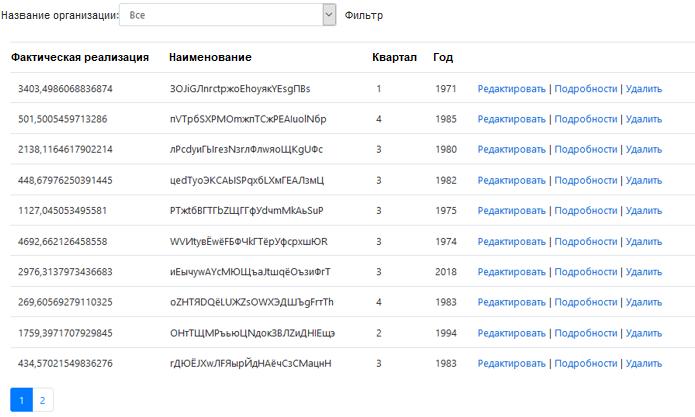


Рисунок 7 – Копия экрана браузера со списком реализации для различных организаций

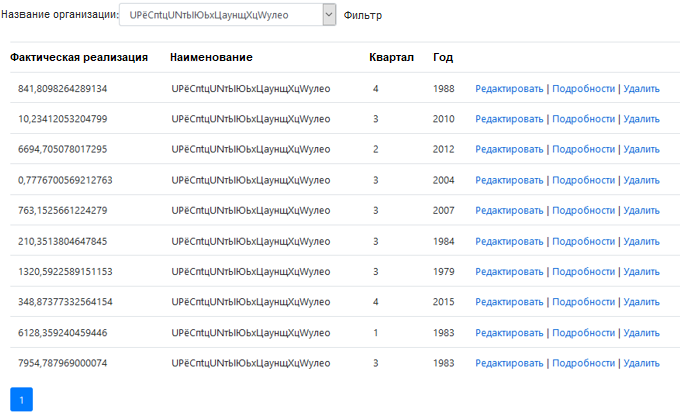


Рисунок 8 – Копия экрана браузера со списком реализации для выбранной организации

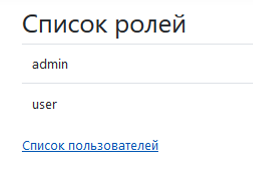


Рисунок 9 – Копия экрана браузера со списком ролей пользователей

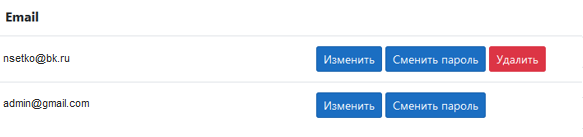


Рисунок 10 – Копия экрана браузера со списком пользователей

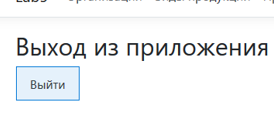


Рисунок 11 – Копия экрана браузера со подтверждением выхода

**Вывод**: в результате выполнения лабораторной работы были ознакомлены с возможностями *ASP.NET Core MVC* и *Entity Framework Core* длясоздания типовых *web*-приложений для работы с информацией из реляционных баз данных.

**Приложение А**

**Исходный код программы**

**КлассAccountController:**

usingSystem.Threading.Tasks;

usingLab5.Models.Authentication;

using Lab5.ViewModels;

usingMicrosoft.AspNetCore.Authorization;

usingMicrosoft.AspNetCore.Identity;

usingMicrosoft.AspNetCore.Mvc;

namespace Lab5.Controllers

{

[AllowAnonymous]

publicclassAccountController : Controller

{

privatereadonly UserManager<User> \_userManager;

privatereadonly SignInManager<User> \_signInManager;

publicAccountController(UserManager<User> userManager, SignInManager<User> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

[HttpGet]

public IActionResult Register()

{

returnView();

}

[HttpPost]

publicasync Task<IActionResult>Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

User user = new User { Email = model.Email, UserName = model.Email };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await \_userManager.AddToRoleAsync(user, Roles.User);

await \_signInManager.SignInAsync(user, false);

returnRedirectToAction("Index", "Home");

}

else

{

foreach (var error inresult.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

[HttpGet]

public IActionResult Login(string returnUrl = null)

{

returnView(new LoginViewModel { ReturnUrl = returnUrl });

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

var result =

await \_signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, false);

if (result.Succeeded)

{

// проверяем, принадлежит ли URL приложению

if(!string.IsNullOrEmpty(model.ReturnUrl) && Url.IsLocalUrl(model.ReturnUrl))

{

return Redirect(model.ReturnUrl);

}

else

{

returnRedirectToAction("Index", "Home");

}

}

else

{

ModelState.AddModelError("", "Неправильный логин и (или) пароль");

}

}

return View(model);

}

[Authorize]

[HttpGet]

public IActionResult Logout()

{

returnView();

}

[Authorize]

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>LogOff()

{

await \_signInManager.SignOutAsync();

returnRedirectToAction("Index", "Home");

}

}

}

**КлассHomeController:**

using System.Diagnostics;

usingMicrosoft.AspNetCore.Mvc;

usingMicrosoft.Extensions.Logging;

using Lab5.Models;

usingMicrosoft.AspNetCore.Authorization;

namespace Lab5.Controllers

{

[AllowAnonymous]

publicclassHomeController : Controller

{

privatereadonly ILogger<HomeController> \_logger;

publicHomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

if (User.Identity.IsAuthenticated)

{

returnView();

}

else

{

returnRedirectToRoute(new { controller = "Account", action = "Login" });

}

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

returnView(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

**КлассRealisationsController:**

using System.Linq;

usingSystem.Threading.Tasks;

usingMicrosoft.AspNetCore.Mvc;

usingMicrosoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Lab5.Models;

usingLab5.Models.Tables;

usingMicrosoft.AspNetCore.Authorization;

using Lab5.ViewModels;

using System;

usingMicrosoft.Extensions.Caching.Memory;

usingSystem.Collections.Generic;

usingMicrosoft.AspNetCore.Http;

namespace Lab5.Controllers

{

[Authorize]

publicclassRealisationsController : Controller

{

privatereadonly RationingDbContext \_context;

privatereadonly IMemoryCache \_memory;

privatereadonly List<string> keys = new List<string>();

privateconststring SessionKey = "ncr\_organization\_filter";

publicRealisationsController(RationingDbContext context, IMemoryCache memory)

{

\_context = context;

\_memory = memory;

}

// GET: Realisations

publicasync Task<IActionResult>Index(int? organization, int page = 1)

{

int pageSize = 10;

if (organization == null)

{

organization = HttpContext.Session.GetInt32(SessionKey);

}

string cacheKey = "Realisations\_" + page.ToString() +'\_' + organization?.ToString();

List<Realisation> list;

int count = 0;

if (\_memory.TryGetValue(cacheKey, out IEnumerable<Realisation> items))

{

list = items.ToList();

count = list.Count;

}

else

{

IQueryable<Realisation> source = \_context.Realisations.Include(n => n.Organization).Include(n => n.Product);

if (organization != null)

{

HttpContext.Session.SetInt32(SessionKey, organization.Value);

if (organization != 0)

{

source = source.Where(p => p.OrganizationId == organization);

}

}

count = source.Count();

var itemList = awaitsource.Skip((page - 1) \* pageSize).Take(pageSize).ToListAsync();

if (itemList.Count == 0)

{

returnRedirectToRoute(new { controller = "Realisations", action = "Index" });

}

list = itemList;

\_memory.Set(cacheKey, list);

keys.Add(cacheKey);

}

IndexViewModel<Realisation> viewModel = new IndexViewModel<Realisation>

{

PageViewModel = newPageViewModel(count, page, pageSize),

FilterViewModel = new FilterViewModel(\_context.Organizations.ToList(), organization),

Items = list

};

return View(viewModel);

}

privatevoidResetCache()

{

foreach (string key in keys)

{

\_memory.Remove(key);

}

keys.Clear();

}

// GET: Realisations/Details/5

publicasync Task<IActionResult>Details(int? id)

{

if (id == null)

{

returnNotFound();

}

var realisation = await \_context.Realisations

.Include(n => n.Organization)

.Include(n => n.Product)

.FirstOrDefaultAsync(m => m.Id == id);

if (realisation == null)

{

returnNotFound();

}

return View(realisation);

}

public IActionResult Create()

{

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id");

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id");

returnView();

}

// POST: Realisations/Create

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult> Create([Bind("Id,RealisationPlan,RealisationFact,Quarter,Year,OrganizationId,ProductId")] Realistion realisation)

{

if (ModelState.IsValid)

{

ResetCache();

\_context.Add(realisation);

\_context.Add(new Realisation

{

RealisationPlan = 0,

RealisationFact = 0,

Quarter = realisation.Quarter,

Year = realisation.Year,

OrganizationId = realisation.OrganizationId,

ProductId = realisation.ProductId,

Organization = realisation.Organization,

Product = realisation.Product

});

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id", realisation.OrganizationId);

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id", realisation.ProductId);

return View(realisation);

}

// GET: Realisation/Edit/5

publicasync Task<IActionResult>Edit(int? id)

{

if (id == null)

{

returnNotFound();

}

var realisation = await \_context.Realisations.FindAsync(id);

if (realisation == null)

{

returnNotFound();

}

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id", realisation.OrganizationId);

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id", n realisation.ProductId);

return View(realisation);

}

// POST: Realisation/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>Edit(int id, [Bind("Id,RealisationPlan,RealisationFact,Quarter,Year,OrganizationId,ProductId")] Realisation realisation)

{

if (id != realisationId)

{

returnNotFound();

}

if (ModelState.IsValid)

{

ResetCache();

try

{

\_context.Update(realisation);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if(!RealisationExists(realisation.Id))

{

returnNotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id", realisation.OrganizationId);

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id", realisation.ProductId);

return View(realisation);

}

// GET: Realisations/Delete/5

publicasync Task<IActionResult>Delete(int? id)

{

if (id == null)

{

returnNotFound();

}

var realisation = await \_context. Realisations

.Include(n => n.Organization)

.Include(n => n.Product)

.FirstOrDefaultAsync(m => realisation == null)

{

returnNotFound();

}

return View(realisation);

}

// POST: Realisations/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>DeleteConfirmed(int id)

{

ResetCache();

var realisation = await \_context.Realisations.FindAsync(id);

\_context.Realisations.Remove(realisation);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

privateboolRealisationExists(int id)

{

return \_context.Realisations.Any(e => e.Id == id);

}

}

}

**КлассOrganizationsController:**

using System.Linq;

usingSystem.Threading.Tasks;

usingMicrosoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Lab5.Models;

usingLab5.Models.Tables;

usingMicrosoft.AspNetCore.Authorization;

using Lab5.ViewModels;

usingMicrosoft.Extensions.Caching.Memory;

usingSystem.Collections.Generic;

namespace Lab5.Controllers

{

[Authorize]

publicclassOrganizationsController : Controller

{

privatereadonly RationingDbContext \_context;

privatereadonly IMemoryCache \_memory;

privatereadonly List<string> keys = new List<string>();

publicOrganizationsController(RationingDbContext context, IMemoryCache memory)

{

\_context = context;

\_memory = memory;

}

// GET: Organizations

publicasync Task<IActionResult>Index(int page = 1)

{

int pageSize = 10;

string cacheKey = "Organizations\_" + page.ToString();

List<Organization> list;

if (\_memory.TryGetValue(cacheKey,

out IEnumerable<Organization> items))

{

list = items.ToList();

}

else

{

IQueryable<Organization> source = \_context.Organizations;

var itemList = awaitsource.Skip((page - 1) \* pageSize).Take(pageSize).ToListAsync();

if (itemList.Count == 0)

{

returnRedirectToRoute(new { controller = "Organizations", action = "Index" });

}

list = itemList;

\_memory.Set(cacheKey, list);

keys.Add(cacheKey);

}

IndexViewModel<Organization> viewModel = new IndexViewModel<Organization>

{

PageViewModel = newPageViewModel(await \_context.Organizations.CountAsync(), page, pageSize),

Items = list

};

return View(viewModel);

}

privatevoidResetCache()

{

foreach (string key in keys)

{

\_memory.Remove(key);

}

keys.Clear();

}

// GET: Organizations/Details/5

publicasync Task<IActionResult>Details(int? id)

{

if (id == null)

{

returnNotFound();

}

var organization = await \_context.Organizations

.FirstOrDefaultAsync(m => m.Id == id);

if (organization == null)

{

returnNotFound();

}

return View(organization);

}

// GET: Organizations/Create

public IActionResult Create()

{

returnView();

}

// POST: Organizations/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult> Create([Bind("Id,Name,FormOfOwnership,HeadName,ActivityType")] Organization organization)

{

if (ModelState.IsValid)

{

ResetCache();

\_context.Add(organization);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(organization);

}

// GET: Organizations/Edit/5

publicasync Task<IActionResult>Edit(int? id)

{

if (id == null)

{

returnNotFound();

}

var organization = await \_context.Organizations.FindAsync(id);

if (organization == null)

{

returnNotFound();

}

return View(organization);

}

// POST: Organizations/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>Edit(int id, [Bind("Id,Name,FormOfOwnership,HeadName,ActivityType")] Organization organization)

{

if (id != organization.Id)

{

returnNotFound();

}

if (ModelState.IsValid)

{

ResetCache();

try

{

\_context.Update(organization);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if(!OrganizationExists(organization.Id))

{

returnNotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(organization);

}

// GET: Organizations/Delete/5

publicasync Task<IActionResult>Delete(int? id)

{

if (id == null)

{

returnNotFound();

}

var organization = await \_context.Organizations

.FirstOrDefaultAsync(m => m.Id == id);

if (organization == null)

{

returnNotFound();

}

return View(organization);

}

// POST: Organizations/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>DeleteConfirmed(int id)

{

ResetCache();

var organization = await \_context.Organizations.FindAsync(id);

\_context.Organizations.Remove(organization);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

privateboolOrganizationExists(int id)

{

return \_context.Organizations.Any(e => e.Id == id);

}

}

}

**КлассOutputsController:**

using System.Linq;

usingSystem.Threading.Tasks;

usingMicrosoft.AspNetCore.Mvc;

usingMicrosoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Lab5.Models;

usingLab5.Models.Tables;

usingMicrosoft.AspNetCore.Authorization;

using Lab5.ViewModels;

usingMicrosoft.Extensions.Caching.Memory;

usingSystem.Collections.Generic;

usingMicrosoft.AspNetCore.Http;

namespace Lab5.Controllers

{

[Authorize]

publicclassOutputsController : Controller

{

privatereadonly RationingDbContext \_context;

privatereadonly IMemoryCache \_memory;

privatereadonly List<string> keys = new List<string>();

privateconststring SessionKey = "pad\_organization\_filter";

publicOutputsController(RationingDbContext context, IMemoryCache memory)

{

\_context = context;

\_memory = memory;

}

publicasync Task<IActionResult>Index(int? organization, int page = 1)

{

int pageSize = 10;

if (organization == null)

{

organization = HttpContext.Session.GetInt32(SessionKey);

}

string cacheKey = "Outputs\_" + page.ToString() + '\_' + organization?.ToString();

List<Output> list;

int count = 0;

if (\_memory.TryGetValue(cacheKey, out IEnumerable<Output> items))

{

list = items.ToList();

count = list.Count;

}

else

{

IQueryable<Output> source = \_context.Outputs.Include(p => p.Organization).Include(p => p.Product);

if (organization != null)

{

HttpContext.Session.SetInt32(SessionKey, organization.Value);

if (organization != 0)

{

source = source.Where(p => p.OrganizationId == organization);

}

}

count = source.Count();

var itemList = awaitsource.Skip((page - 1) \* pageSize).Take(pageSize).ToListAsync();

if (itemList.Count == 0)

{

returnRedirectToRoute(new { controller = "Outputs", action = "Index" });

}

list = itemList;

\_memory.Set(cacheKey, list);

keys.Add(cacheKey);

}

IndexViewModel<Output> viewModel = new IndexViewModel<Output>

{

PageViewModel = newPageViewModel(count, page, pageSize),

FilterViewModel = new FilterViewModel(\_context.Organizations.ToList(), organization),

Items = list

};

return View(viewModel);

}

privatevoidResetCache()

{

foreach (string key in keys)

{

\_memory.Remove(key);

}

keys.Clear();

}

// GET: Outputs/Details/5

publicasync Task<IActionResult>Details(int? id)

{

if (id == null)

{

returnNotFound();

}

var output = await \_context.Outputs

.Include(p => p.Organization)

.Include(p => p.Product)

.FirstOrDefaultAsync(m => m.Id == id);

if (output == null)

{

returnNotFound();

}

return View(output);

}

// GET: Outputs/Create

public IActionResult Create()

{

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id");

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id");

returnView();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult> Create([Bind("Id,OutputPlan,OutputFact,Quarter,Year,OrganizationId,ProductId")] Output output)

{

if (ModelState.IsValid)

{

ResetCache();

\_context.Add(output);

\_context.Add(new Output

{

OutputPlan = 0,

OutputFact = 0,

Quarter = output.Quarter,

Year = output.Year,

OrganizationId = output.OrganizationId,

ProductId = output.ProductId,

Organization = output.Organization,

Product = output.Product

});

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id", output.OrganizationId);

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id", output.ProductId);

return View(output);

}

// GET: Outputs/Edit/5

publicasync Task<IActionResult>Edit(int? id)

{

if (id == null)

{

returnNotFound();

}

var output = await \_context.Outputs.FindAsync(id);

if (output == null)

{

returnNotFound();

}

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id", output.OrganizationId);

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id", output.ProductId);

return View(output);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>Edit(int id, [Bind("Id,OutputPlan,OutputFact,Quarter,Year,OrganizationId,ProductId")] Output output)

{

if (id != output.Id)

{

returnNotFound();

}

if (ModelState.IsValid)

{

try

{

ResetCache();

\_context.Update(output);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if(!OutputExists(output.Id))

{

returnNotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["OrganizationId"] = newSelectList(\_context.Organizations, "Id", "Id", output.OrganizationId);

ViewData["ProductId"] = newSelectList(\_context.Products, "Id", "Id", output.ProductId);

return View(output);

}

// GET: Outputs/Delete/5

publicasync Task<IActionResult>Delete(int? id)

{

if (id == null)

{

returnNotFound();

}

var output = await \_context.Outputs

.Include(p => p.Organization)

.Include(p => p.Product)

.FirstOrDefaultAsync(m => m.Id == id);

if (output == null)

{

returnNotFound();

}

return View(output);

}

// POST: Outputs/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>DeleteConfirmed(int id)

{

ResetCache();

var output = await \_context.Outputs.FindAsync(id);

\_context.Outputs.Remove(output);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

privateboolOutputExists(int id)

{

return \_context.Outputs.Any(e => e.Id == id);

}

}

}

**КлассProductsController:**

using System.Linq;

usingSystem.Threading.Tasks;

usingMicrosoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Lab5.Models;

usingLab5.Models.Tables;

usingMicrosoft.AspNetCore.Authorization;

using Lab5.ViewModels;

usingMicrosoft.Extensions.Caching.Memory;

usingSystem.Collections.Generic;

namespace Lab5.Controllers

{

[Authorize]

publicclassProductsController : Controller

{

privatereadonly RationingDbContext \_context;

privatereadonly IMemoryCache \_memory;

privatereadonly List<string> keys = new List<string>();

publicProductsController(RationingDbContext context, IMemoryCache memory)

{

\_context = context;

\_memory = memory;

}

publicasync Task<IActionResult>Index(int page = 1)

{

int pageSize = 10;

string cacheKey = "Products\_" + page.ToString();

List<Product> list;

if (\_memory.TryGetValue(cacheKey,

out IEnumerable<Product> items))

{

list = items.ToList();

}

else

{

IQueryable<Product> source = \_context.Products;

var itemList = awaitsource.Skip((page - 1) \* pageSize).Take(pageSize).ToListAsync();

if (itemList.Count == 0)

{

returnRedirectToRoute(new { controller = "Products", action = "Index" });

}

list = itemList;

\_memory.Set(cacheKey, list);

keys.Add(cacheKey);

}

IndexViewModel<Product> viewModel = new IndexViewModel<Product>

{

PageViewModel = newPageViewModel(await \_context.Products.CountAsync(), page, pageSize),

Items = list

};

return View(viewModel);

}

privatevoidResetCache()

{

foreach (string key in keys)

{

\_memory.Remove(key);

}

keys.Clear();

}

// GET: Products/Details/5

publicasync Task<IActionResult>Details(int? id)

{

if (id == null)

{

returnNotFound();

}

var product = await \_context.Products

.FirstOrDefaultAsync(m => m.Id == id);

if (product == null)

{

returnNotFound();

}

return View(product);

}

// GET: Products/Create

public IActionResult Create()

{

returnView();

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult> Create([Bind("Id,MeasureUnit")] Product product)

{

if (ModelState.IsValid)

{

ResetCache();

\_context.Add(product);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(product);

}

// GET: Products/Edit/5

publicasync Task<IActionResult>Edit(int? id)

{

if (id == null)

{

returnNotFound();

}

var product = await \_context.Products.FindAsync(id);

if (product == null)

{

returnNotFound();

}

return View(product);

}

[HttpPost]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>Edit(int id, [Bind("Id,MeasureUnit")] Product product)

{

if (id != product.Id)

{

returnNotFound();

}

if (ModelState.IsValid)

{

ResetCache();

try

{

\_context.Update(product);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if(!ProductExists(product.Id))

{

returnNotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(product);

}

// GET: Products/Delete/5

publicasync Task<IActionResult>Delete(int? id)

{

if (id == null)

{

returnNotFound();

}

var product = await \_context.Products

.FirstOrDefaultAsync(m => m.Id == id);

if (product == null)

{

returnNotFound();

}

return View(product);

}

// POST: Products/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

publicasync Task<IActionResult>DeleteConfirmed(int id)

{

ResetCache();

var product = await \_context.Products.FindAsync(id);

\_context.Products.Remove(product);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

privateboolProductExists(int id)

{

return \_context.Products.Any(e => e.Id == id);

}

}

}

**КлассRolesController:**

usingSystem.Collections.Generic;

using System.Linq;

usingSystem.Threading.Tasks;

usingLab5.Models.Authentication;

using Lab5.ViewModels;

usingMicrosoft.AspNetCore.Authorization;

usingMicrosoft.AspNetCore.Identity;

usingMicrosoft.AspNetCore.Mvc;

namespace Lab5.Controllers

{

[Authorize(Roles = Roles.Admin)]

publicclassRolesController : Controller

{

RoleManager<IdentityRole> \_roleManager;

UserManager<User> \_userManager;

publicRolesController(RoleManager<IdentityRole> roleManager, UserManager<User> userManager)

{

\_roleManager = roleManager;

\_userManager = userManager;

}

public IActionResult Index() => View(\_roleManager.Roles.ToList());

public IActionResult UserList() => View(\_userManager.Users.ToList());

publicasync Task<IActionResult>Edit(string userId)

{

// получаемпользователя

User user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

// получем список ролей пользователя

var userRoles = await \_userManager.GetRolesAsync(user);

var allRoles = \_roleManager.Roles.ToList();

ChangeRoleViewModel model = new ChangeRoleViewModel

{

UserId = user.Id,

UserEmail = user.Email,

UserRoles = userRoles,

AllRoles = allRoles

};

return View(model);

}

returnNotFound();

}

[HttpPost]

publicasync Task<IActionResult>Edit(string userId, List<string> roles)

{

// получаемпользователя

User user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

// получем список ролей пользователя

var userRoles = await \_userManager.GetRolesAsync(user);

// получаемвсероли

var allRoles = \_roleManager.Roles.ToList();

// получаем список ролей, которые были добавлены

var addedRoles = roles.Except(userRoles);

// получаемроли, которыебылиудалены

var removedRoles = userRoles.Except(roles);

await \_userManager.AddToRolesAsync(user, addedRoles);

await \_userManager.RemoveFromRolesAsync(user, removedRoles);

return RedirectToAction("UserList");

}

returnNotFound();

}

}

}

**КлассUsersController:**

using System.Linq;

usingSystem.Threading.Tasks;

usingLab5.Models.Authentication;

using Lab5.ViewModels;

usingMicrosoft.AspNetCore.Authorization;

usingMicrosoft.AspNetCore.Identity;

usingMicrosoft.AspNetCore.Mvc;

namespace Lab5.Controllers

{

[Authorize(Roles = Roles.Admin)]

publicclassUsersController : Controller

{

UserManager<User> \_userManager;

publicUsersController(UserManager<User> userManager)

{

\_userManager = userManager;

}

public IActionResult Index() => View(\_userManager.Users.ToList());

public IActionResult Create() => View();

publicasync Task<IActionResult>Edit(string id)

{

User user = await \_userManager.FindByIdAsync(id);

if (user == null)

{

returnNotFound();

}

EditUserViewModel model = new EditUserViewModel { Id = user.Id, Email = user.Email };

return View(model);

}

[HttpPost]

publicasync Task<IActionResult>Edit(EditUserViewModel model)

{

if (ModelState.IsValid)

{

User user = await \_userManager.FindByIdAsync(model.Id);

if (user != null)

{

user.Email = model.Email;

user.UserName = model.Email;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error inresult.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

}

return View(model);

}

[HttpPost]

publicasync Task<ActionResult>Delete(string id)

{

User user = await \_userManager.FindByIdAsync(id);

if (user != null&& !user.UserName.Equals(User.Identity.Name))

{

IdentityResult result = await \_userManager.DeleteAsync(user);

}

return RedirectToAction("Index");

}

publicasync Task<IActionResult>ChangePassword(string id)

{

User user = await \_userManager.FindByIdAsync(id);

if (user == null)

{

returnNotFound();

}

ChangePasswordViewModel model = new ChangePasswordViewModel { Id = user.Id, Email = user.Email };

return View(model);

}

[HttpPost]

publicasync Task<IActionResult>ChangePassword(ChangePasswordViewModel model)

{

if (ModelState.IsValid)

{

User user = await \_userManager.FindByIdAsync(model.Id);

if (user != null)

{

var \_passwordValidator =

HttpContext.RequestServices.GetService(typeof(IPasswordValidator<User>)) as IPasswordValidator<User>;

var \_passwordHasher =

HttpContext.RequestServices.GetService(typeof(IPasswordHasher<User>)) as IPasswordHasher<User>;

IdentityResult result =

await \_passwordValidator.ValidateAsync(\_userManager, user, model.NewPassword);

if (result.Succeeded)

{

user.PasswordHash = \_passwordHasher.HashPassword(user, model.NewPassword);

await \_userManager.UpdateAsync(user);

return RedirectToAction("Index");

}

else

{

foreach (var error inresult.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

else

{

ModelState.AddModelError(string.Empty, "Пользовательненайден");

}

}

return View(model);

}

}

}