**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

по дисциплине «Разработка и проектирование 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

Воробьев В. А.

Принял: пр.

Ястребов А. А.

Гомель 2023

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

**Задание**:

Используя разработанный ранее слой доступа к базе данным согласно своего варианта, спроектировать и создать интерфейс *Web*-приложения на основе *ASP*.*NET* *Core MVC Framework* и *Entity Framework Core*. *Web*-приложение должно удовлетворять следующим требованиям:

– 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;

– иметь единое стилевое оформление, основанное на использовании мастер-страниц;

– иметь удобную систему навигации (строка меню, гиперссылки, кнопки), которая обеспечивает оптимальный путь перехода между двумя произвольно выбранными страницами в соответствии с логикой приложения;

– пользователь для работы с приложением должен пройти аутентификацию;

– должно поддерживать реализацию не менее двух ролевых политик.

– администратор должен иметь возможность управлять пользователями: просматривать, создавать, удалять и редактировать данные учетных записей;

– представления для просмотра данных из таблиц должны предусматривать разбиение данных на страницы, фильтрацию по одному или нескольким полям.

**Ход работы**

При запуске приложения открывается окно для работы с проектами. Внешний вид страницы для работы с проектами представлен на рисунке 1.

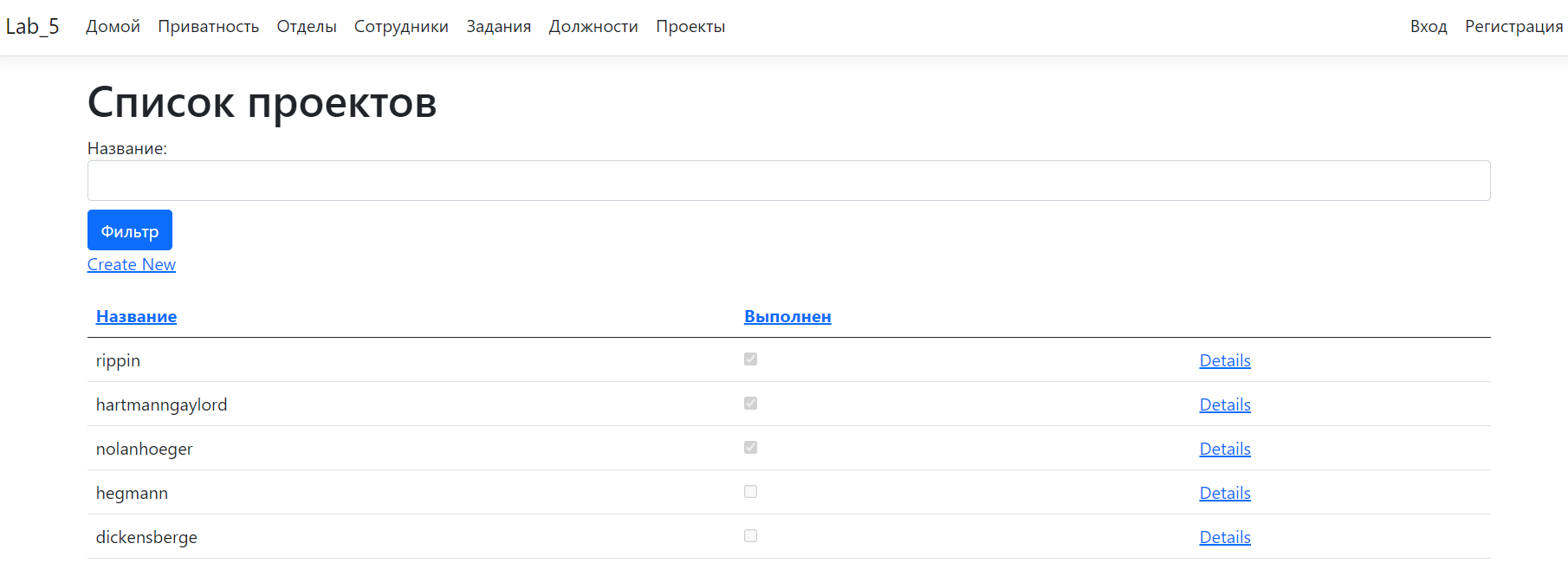


Рисунок 1 – Страница «Проекты»

Так как пользователь не авторизован, то он не может редактировать данные. Произведем вход в систему в качестве администратора и проверим данную страницу. Внешний вид страница входа представлен на рисунке 2.

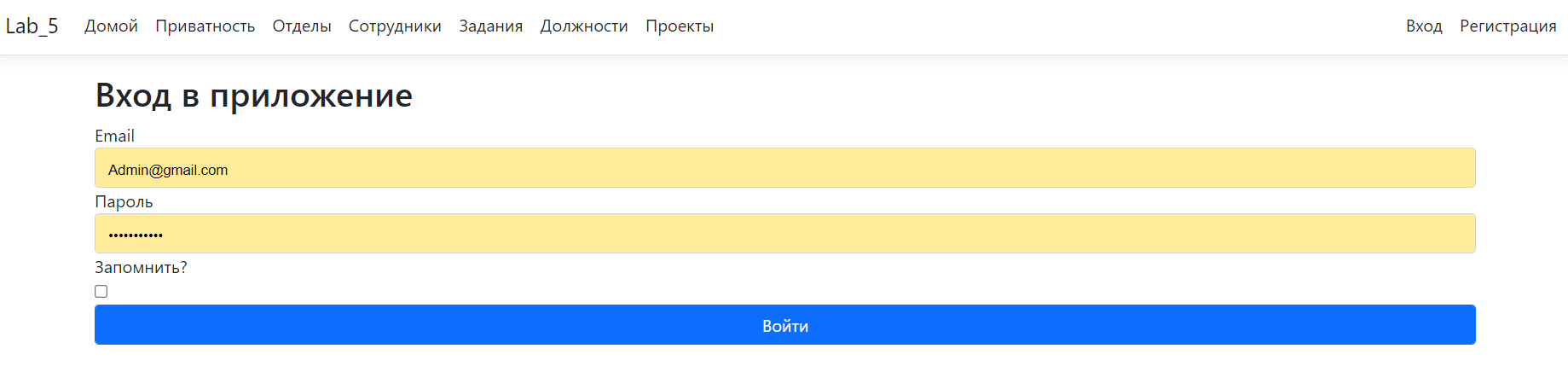


Рисунок 2 – Страница «Авторизация»

После авторизации перейдем на страницу «Проекты». Внешний вид страницы «Проекты» после авторизации представлен на рисунке 3.

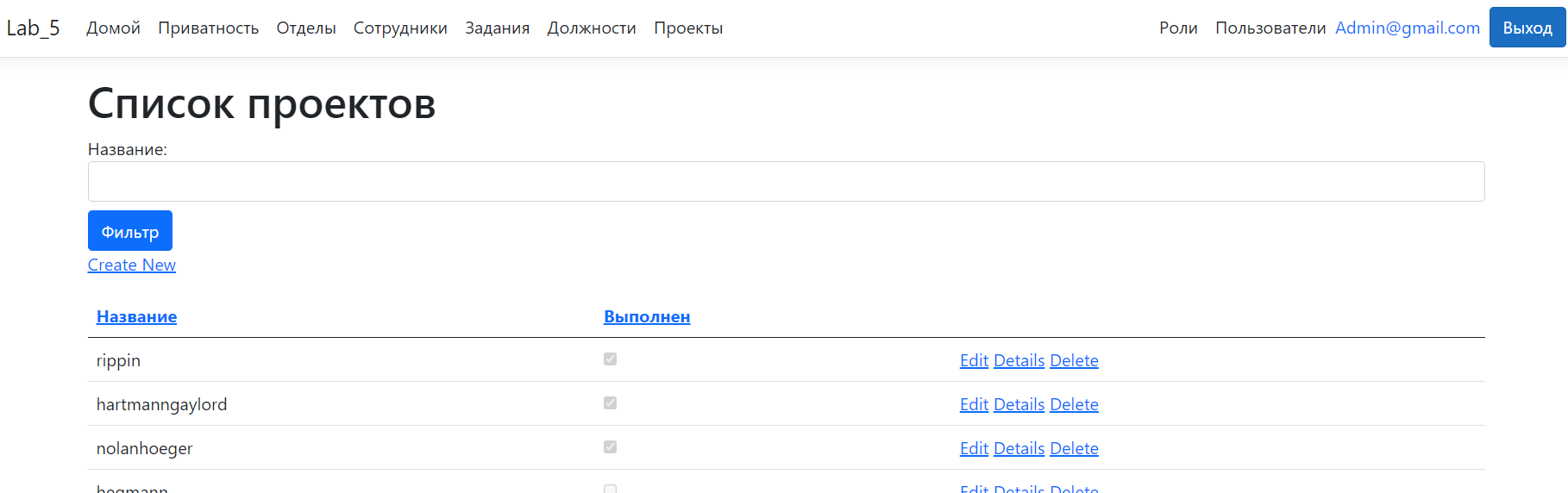


Рисунок 3 – Страница «Проекты»

Отредактируем первый проект, изменив ему название на «*Testproject*». Страница для редактирования проекта представлена на рисунке 4.

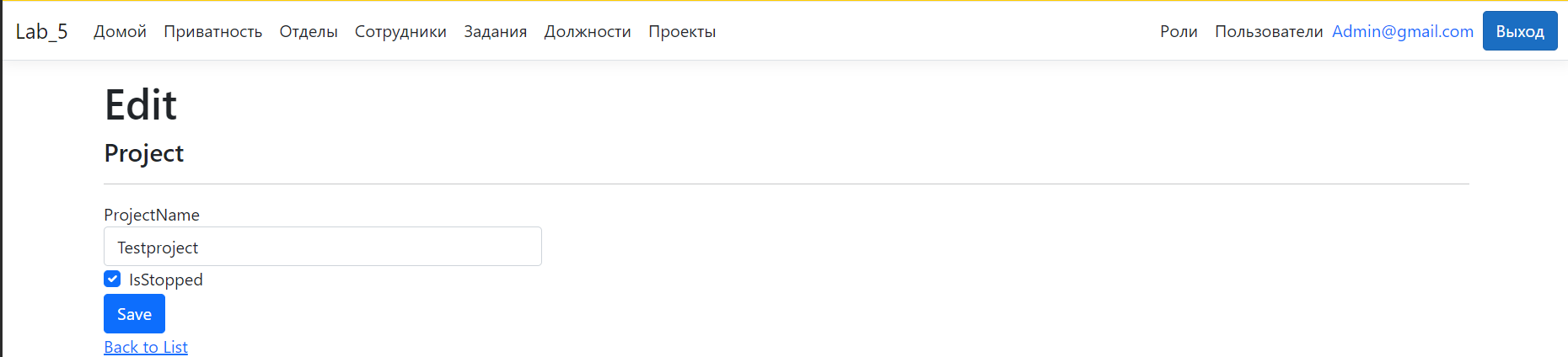


Рисунок 4 – Страница редактирования проекта

Проверим содержимое данной страницы после изменения проекта. Результат представлен на рисунке 5.

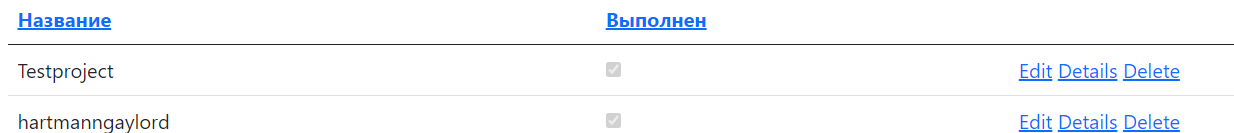


Рисунок 5 – Отредактированный проект

Проведем фильтрацию проектов по названию, найдем проект с именем, которое содержит в себе буквы *nol.* Результат фильтрации представлен на рисунке 6.

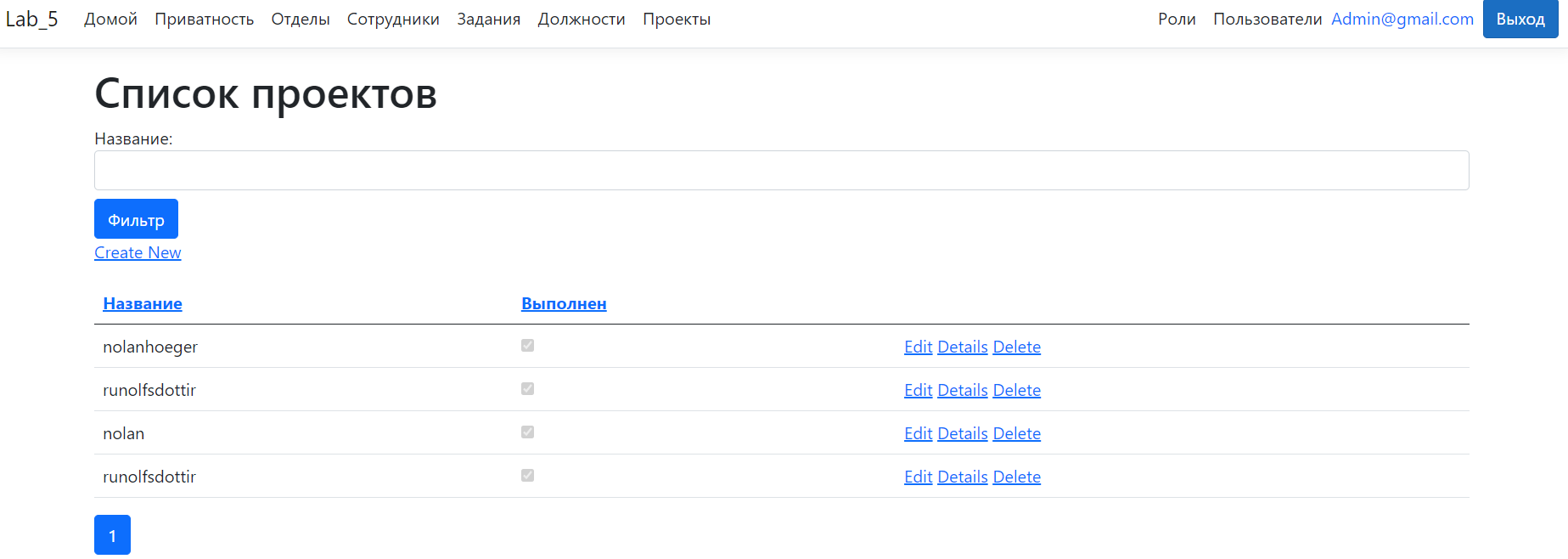


Рисунок 6 – Результат фильтрации

На странице «Должности» также можно производить фильтрации по полям «Название» и «Зарплата». Проведем фильтрацию по зарплате, отобрав записи, где зарплата больше пяти тысяч. Результат фильтрации по зарплате представлен на рисунке 7.

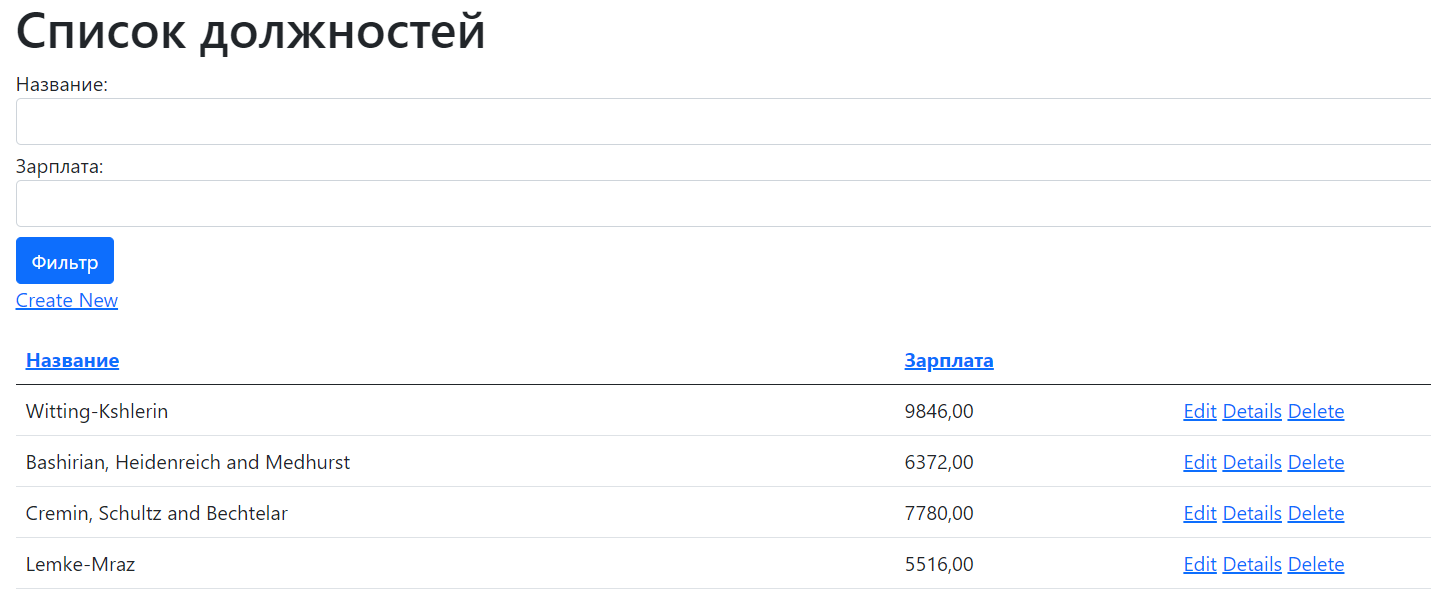


Рисунок 7 – Результат фильтрации

На странице «Задания» можно производить фильтрацию по полям «Сотрудник» и «Проект». Внешний вид страницы для работы с заданиями представлен на рисунке 8.

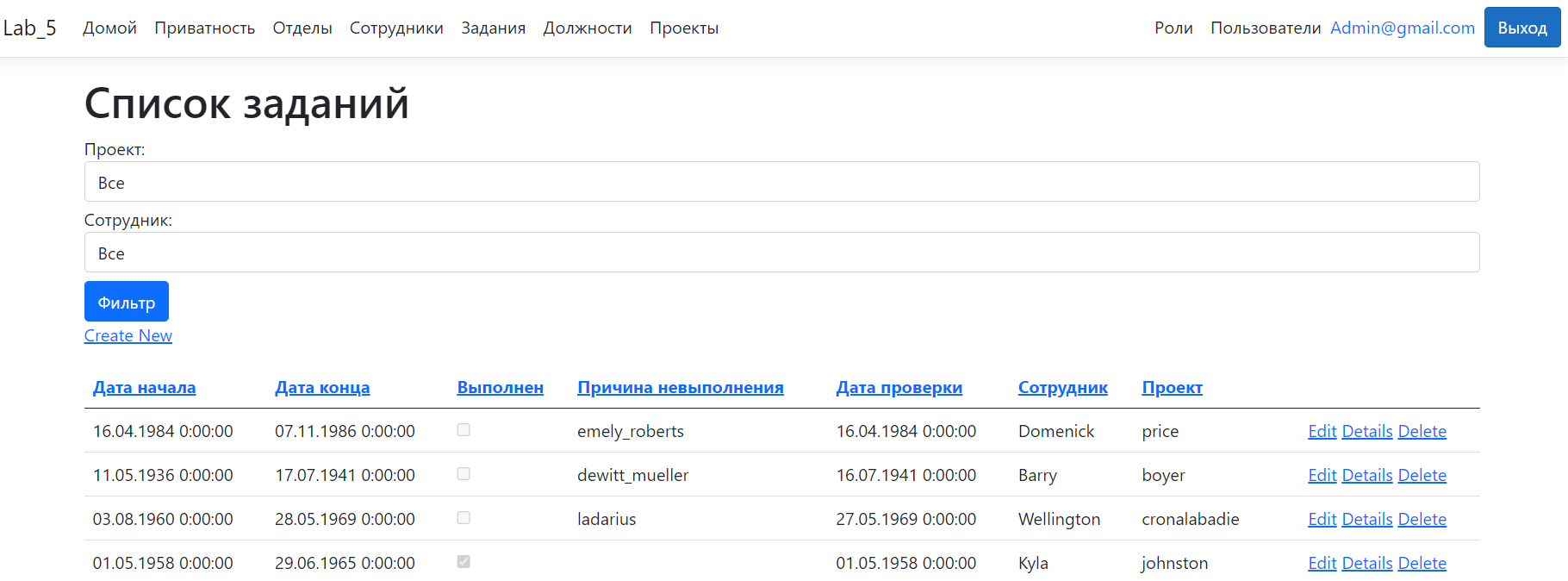


Рисунок 8 – Страница «Задания»

Проведем фильтрацию по сотруднику, результат фильтрации представлен на рисунке 9.

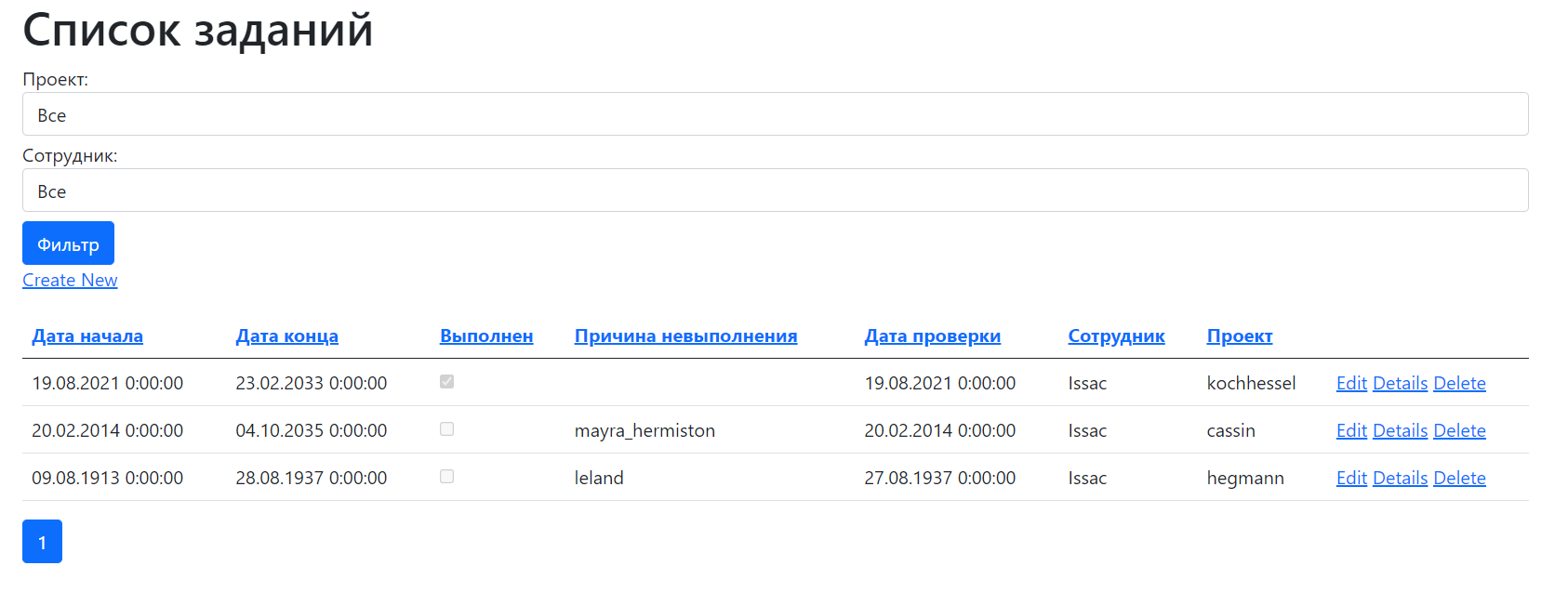


Рисунок 9 – Результат фильтрации

На странице «Сотрудники» производится фильтрация по должности и имени. Внешний вид страницы «Сотрудники» представлен на рисунке 10.

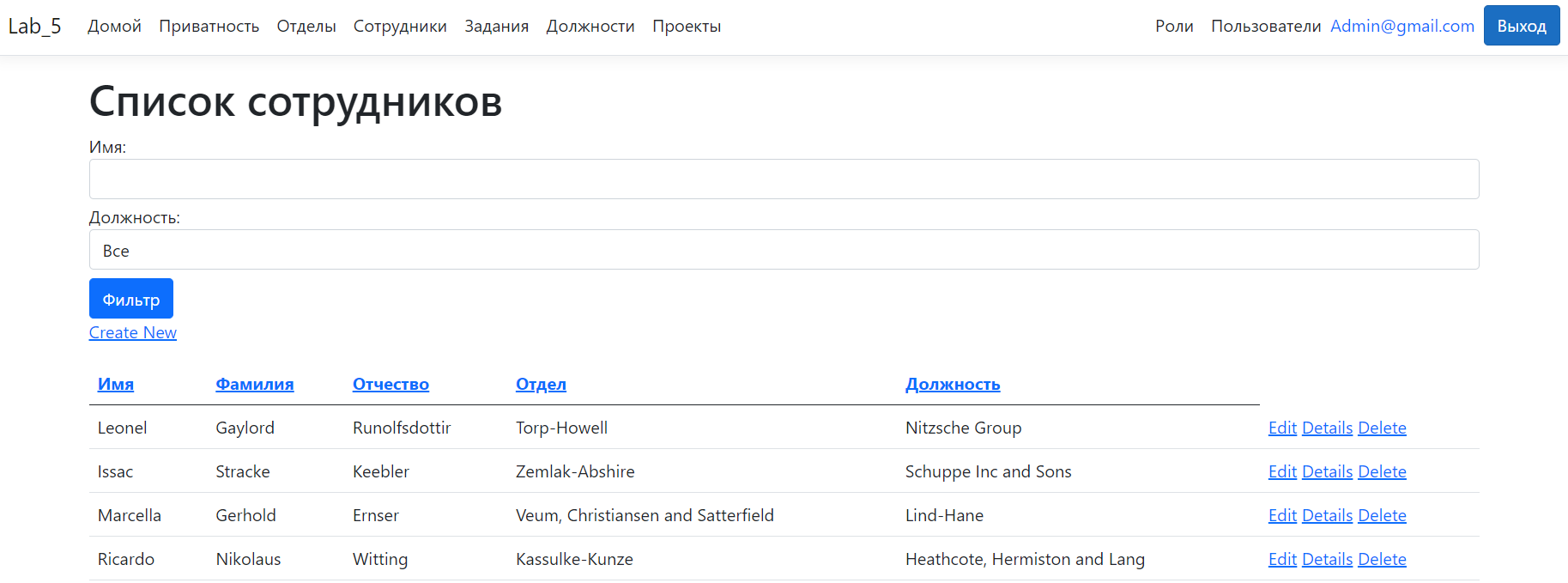


Рисунок 10 – Страница «Сотрудники»

Проведем фильтрацию по должности, результат фильтрации представлен на рисунке 11.



Рисунок 11 – Результат фильтрации

На странице «Отделы» проводится фильтрация по имени и номеру телефона отделения. Внешний вид страницы «Отделы» представлен на рисунке 12.

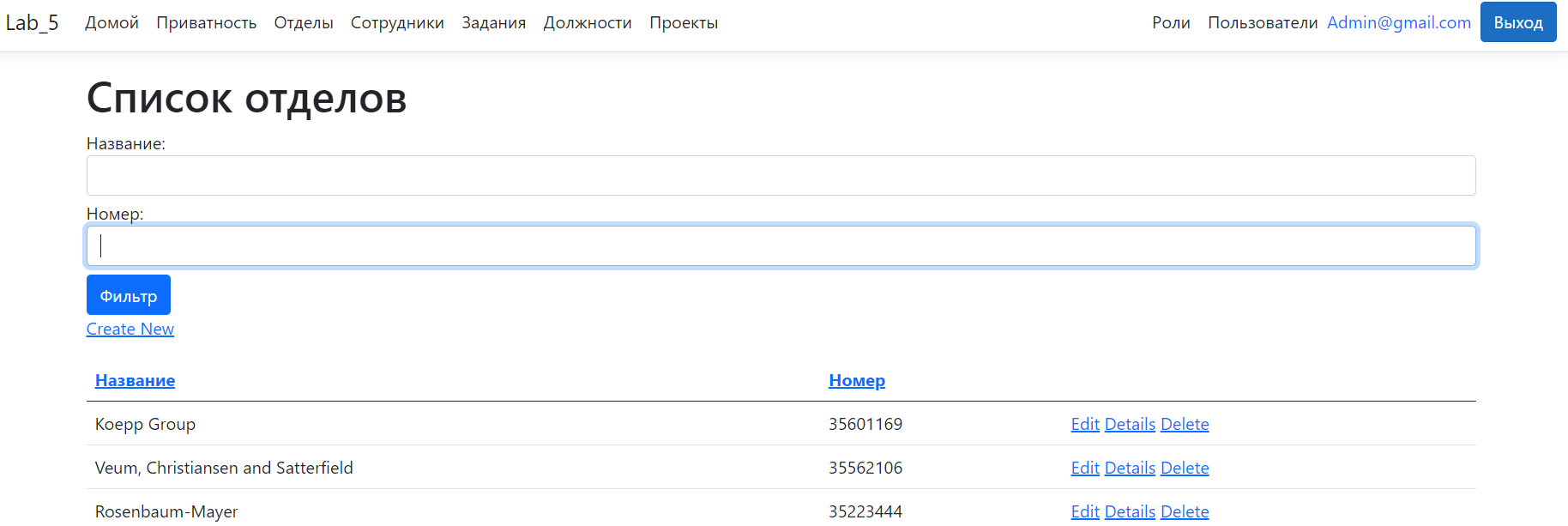


Рисунок 12 – Страница «Отделы»

Проведем фильтрацию по имени, найдем отделы, содержащие в себе слово *koe*, результат фильтрации представлен на рисунке 13.

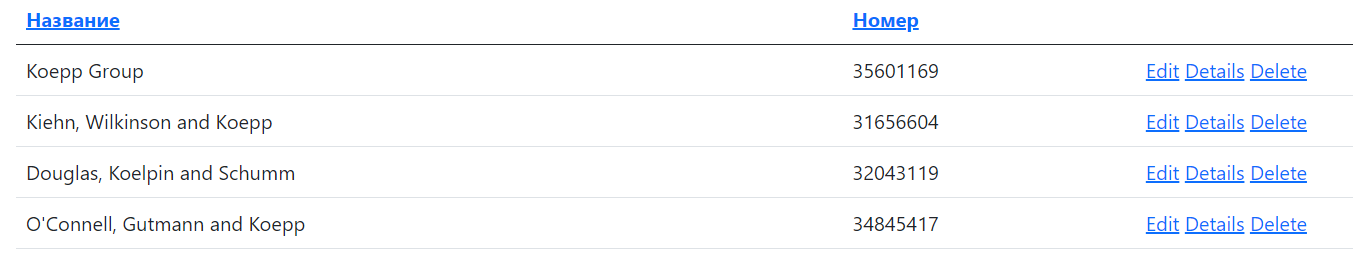


Рисунок 13 – Результат фильтрации

В приложении имеется три роли: главный администратор, который может работать с ролями и пользователями, администратор, который может редактировать данные таблиц, сотрудник, который может редактировать задания.

Попробуем зайти на страницу «Пользователи» не авторизовавшись, при попытке перехода произойдет переадресация на страницу «Авторизации». Результат попытки перехода на страницу «Пользователи» представлен на рисунке 14.

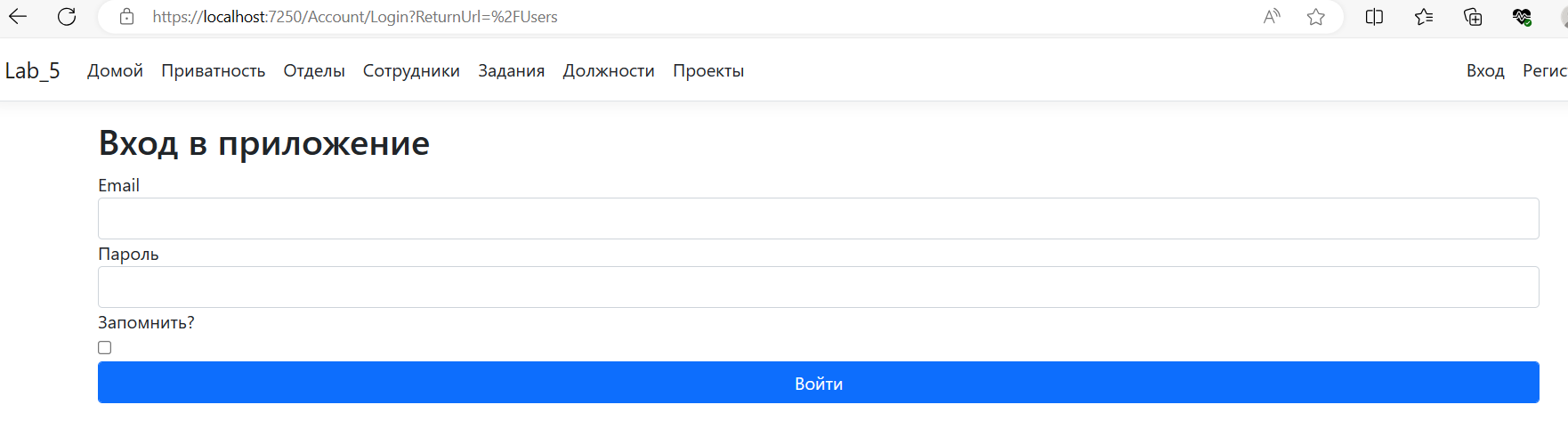


Рисунок 14 – Попытка входа

Войдем в качестве администратора и просмотрим список пользователей. Результат представлен на рисунке 15.

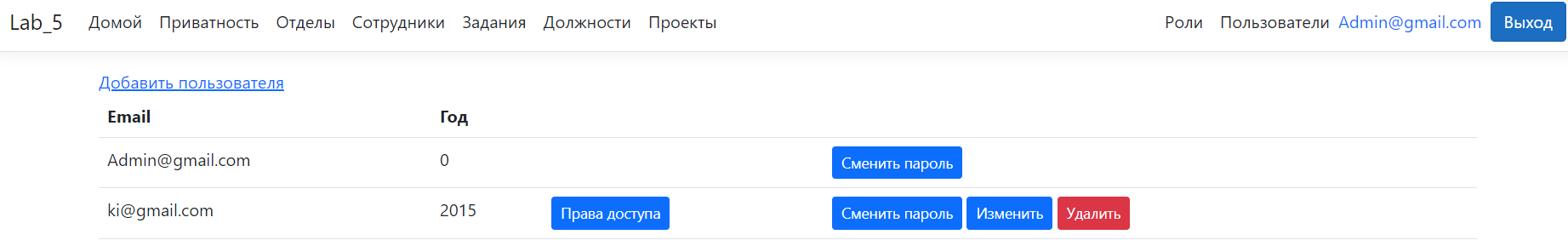


Рисунок 15 – Страница «Пользователи»

Администратор может поменять себе пароль, но удалить себя он не может, так как приложение не будет работать. Изменим пользователю права доступа на «Сотрудник», результат изменения представлен на рисунке 16.

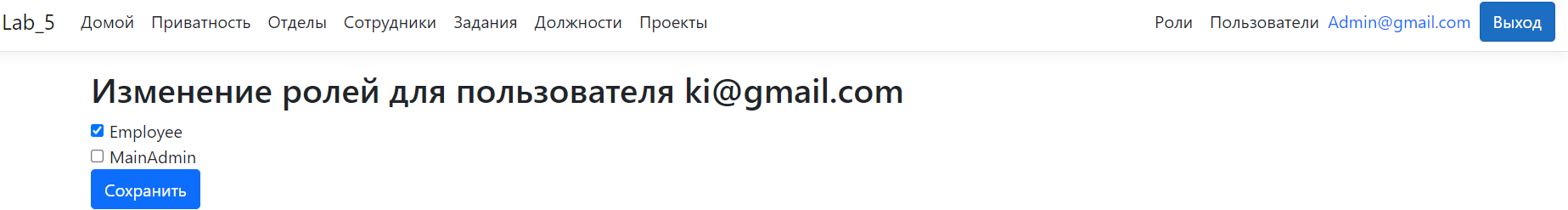


Рисунок 16 – Изменение роли

Произведем вход на данного пользователя и перейдем на страницу «Должности». Результат перехода представлен на рисунке 17.

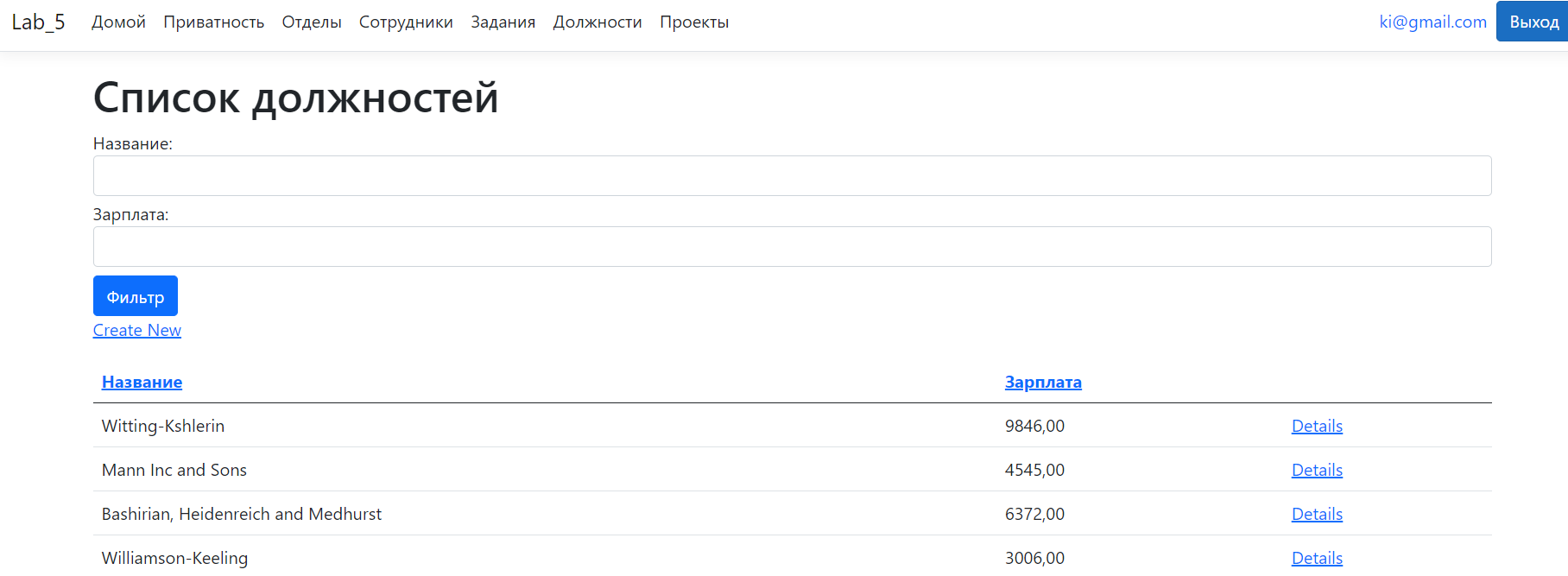


Рисунок 17 – Страница «Должности»

Сотрудник может работать только с заданиями, поэтому он не может редактировать другие записи. Перейдем на страницу «Задания». Результат переход представлен на рисунке 18.

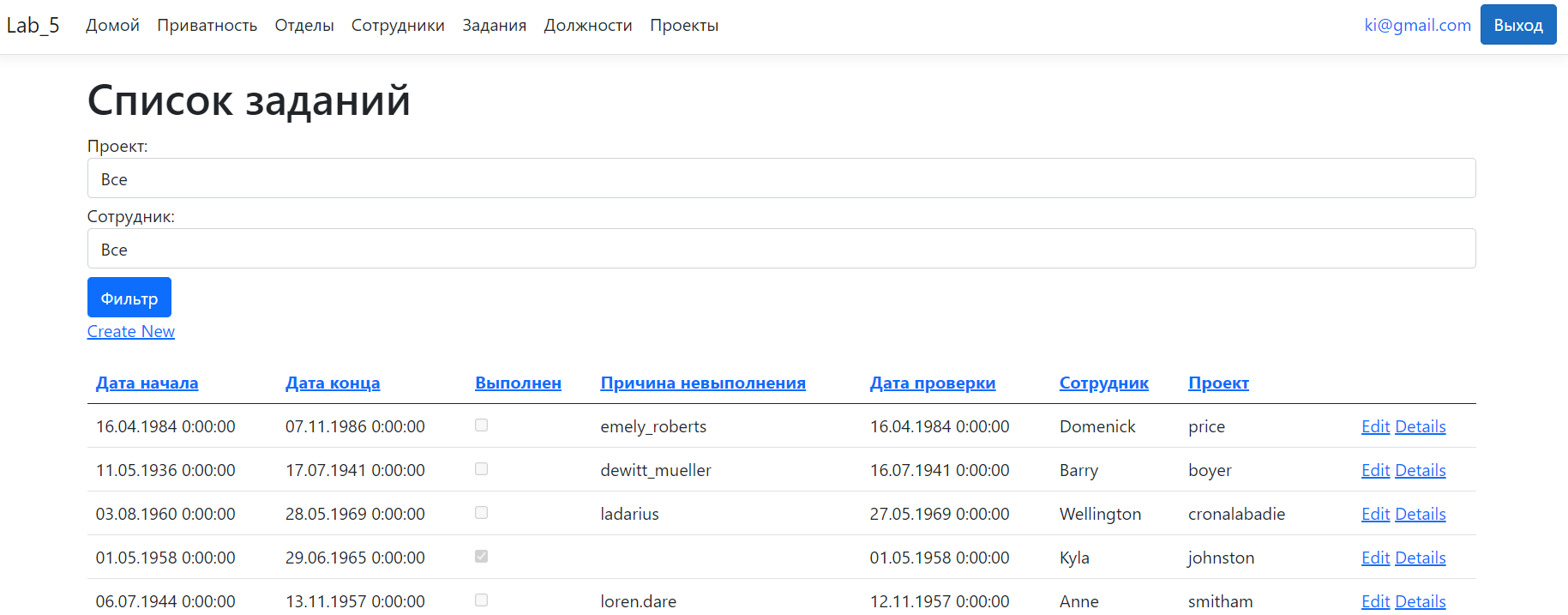


Рисунок 18 – Страница «Задания»

Отредактируем первое задание, изменив его причину невыполнения на *no*\_*reason.* Результат редактирования представлен на рисунке 19.

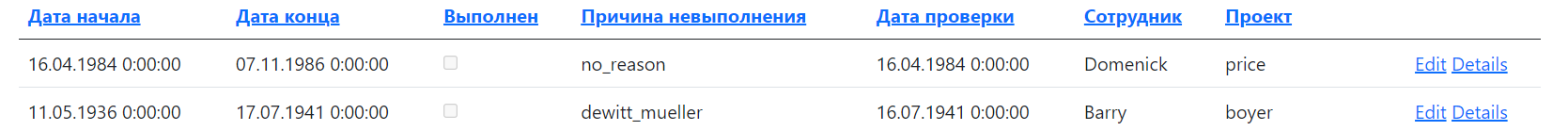


Рисунок 19 – Результат редактирования

Текст программы для создания контроллеров представлен в приложении А.

**Вывод:** в результате выполнения лабораторное работы получены навыки использования *ASP*.*NET MVC Сore* для создания интерфейса типовых *web*-приложений для работы с информацией из реляционных баз данных, изменено приложения, разработанное в предыдущей лабораторной работе. Добавлена возможность авторизации и функциональность по ролям.

**ПРИЛОЖЕНИЕ А**

**(Обязательное)**

**Текст программы для создания контроллеров и представлений**

using Lab\_5.Models.ViewModels.Identities;

using Lab\_5.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

namespace Lab\_5.Controllers

{

[AllowAnonymous]

public class AccountController : Controller

{

private readonly UserManager<User> \_userManager;

private readonly SignInManager<User> \_signInManager;

public AccountController(UserManager<User> userManager, SignInManager<User> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

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

// добавляем пользователя

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

if (result.Succeeded)

{

UserManager<User> userManager = HttpContext.RequestServices.GetRequiredService<UserManager<User>>();

RoleManager<IdentityRole> roleManager = HttpContext.RequestServices.GetRequiredService<RoleManager<IdentityRole>>();

userManager.AddToRoleAsync(user, "User");

await \_signInManager.SignInAsync(user, false);

return RedirectToAction("Index", "Home");

}

else

{

foreach (var error in result.Errors)

{

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

}

}

}

return View(model);

}

[HttpGet]

public IActionResult Login(string returnUrl = null)

{

return View(new LoginViewModel { ReturnUrl = returnUrl });

}

[HttpPost]

[ValidateAntiForgeryToken]

public async 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

{

return RedirectToAction("Index", "Home");

}

}

else

{

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

}

}

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

// удаляем аутентификационные куки

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Lab\_5;

using Lab\_5.Data;

using Lab\_5.Models;

using Lab\_5.Models.ViewModels;

using Lab\_5.Models.ViewModels.Departments;

using Microsoft.Data.SqlClient;

using Microsoft.AspNetCore.Authorization;

namespace Lab\_5.Controllers

{

public class DepartmentsController : Controller

{

private readonly ApplicationDbContext \_context;

public DepartmentsController(ApplicationDbContext context)

{

\_context = context;

}

[Authorize(Roles = "Admin,MainAdmin,Employee")]

[AllowAnonymous]

// GET: Departments

public IActionResult Index(int page = 1, SortState sortOrder = SortState.DepartmentNameAsc, string name = "", int phone = 0)

{

IQueryable<Department> departments = \_context.Departments;

int pageSize = 10;

if (name != null && name.Trim() != "")

{

departments = departments.Where(d => d.DepartmentName.Contains(name));

}

if (phone != null && phone != 0)

{

departments = departments.Where(d => d.DepartmentPhone.ToString().Contains(phone.ToString()));

}

var count = departments.Count();

var items = departments.Skip((page - 1) \* pageSize).Take(pageSize);

switch (sortOrder)

{

case SortState.DepartmentNameDesc:

items = items.OrderByDescending(item => item.DepartmentName);

break;

case SortState.DepartmentPhoneDesc:

items = items.OrderByDescending(item => item.DepartmentPhone);

break;

}

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

PaginationViewModel<Department, DepartmentFilterViewModel, DepartmentSortViewModel> viewModel = new(items, pageViewModel, new DepartmentFilterViewModel(name, phone), new DepartmentSortViewModel(sortOrder));

return items != null ?

View(viewModel) :

Problem("Entity set 'ApplicationDbContext.Departments' is null.");

}

// GET: Departments/Details/5

[Authorize(Roles = "Admin,MainAdmin, Employee")]

[AllowAnonymous]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Departments == null)

{

return NotFound();

}

var department = await \_context.Departments

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

if (department == null)

{

return NotFound();

}

return View(department);

}

// GET: Departments/Create

[Authorize(Roles = "Admin,MainAdmin")]

public IActionResult Create()

{

return View();

}

// POST: Departments/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Create([Bind("DepartmentId,DepartmentName,DepartmentPhone")] Department department)

{

if (ModelState.IsValid)

{

\_context.Add(department);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(department);

}

// GET: Departments/Edit/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Departments == null)

{

return NotFound();

}

var department = await \_context.Departments.FindAsync(id);

if (department == null)

{

return NotFound();

}

return View(department);

}

// POST: Departments/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int id, [Bind("DepartmentId,DepartmentName,DepartmentPhone")] Department department)

{

if (id != department.DepartmentId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(department);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DepartmentExists(department.DepartmentId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(department);

}

// GET: Departments/Delete/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Departments == null)

{

return NotFound();

}

var department = await \_context.Departments

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

if (department == null)

{

return NotFound();

}

return View(department);

}

// POST: Departments/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Departments == null)

{

return Problem("Entity set 'ApplicationDbContext.Departments' is null.");

}

var department = await \_context.Departments.FindAsync(id);

if (department != null)

{

\_context.Departments.Remove(department);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool DepartmentExists(int id)

{

return (\_context.Departments?.Any(e => e.DepartmentId == id)).GetValueOrDefault();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Lab\_5;

using Lab\_5.Data;

using Lab\_5.Models;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Lab\_5.Models.ViewModels;

using Lab\_5.Models.ViewModels.Employees;

using Microsoft.AspNetCore.Authorization;

namespace Lab\_5.Controllers

{

public class EmployeesController : Controller

{

private readonly ApplicationDbContext \_context;

public EmployeesController(ApplicationDbContext context)

{

\_context = context;

}

[Authorize(Roles = "Admin,MainAdmin,Employee")]

[AllowAnonymous]

// GET: Employees

public async Task<IActionResult> Index(int page = 1, int postId = 0, string name = "", SortState sortOrder = SortState.EmployeeNameAsc)

{

int pageSize = 10;

IQueryable<Employee> source = \_context.Employees.Include(e => e.Department).Include(e => e.Post);

if (postId != 0)

{

source = source.Where(emp => emp.PostId == postId);

}

if (name != null && name.Trim() != "")

{

source = source.Where(emp => emp.EmployeeName.Contains(name));

}

var count = source.Count();

var items = source.Skip((page - 1) \* pageSize).Take(pageSize);

switch (sortOrder)

{

case SortState.EmployeeNameDesc:

items = items.OrderByDescending(s => s.EmployeeName);

break;

case SortState.EmployeeMiddleNameDesc:

items = items.OrderByDescending(s => s.EmployeeMiddlename);

break;

case SortState.EmployeeSurnameDesc:

items = items.OrderByDescending(s => s.EmployeeSurname);

break;

case SortState.EmployeeDepartmentDesc:

items = items.OrderByDescending(s => s.Department.DepartmentName);

break;

case SortState.EmployeePostDesc:

items = items.OrderByDescending(s => s.Post.PostName);

break;

}

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

PaginationViewModel<Employee, EmployeesFilterViewModel, EmployeeSortViewModel> viewModel = new

(items, pageViewModel, new EmployeesFilterViewModel(\_context.Posts.ToList(), name, postId), new EmployeeSortViewModel(sortOrder));

return items != null ?

View(viewModel) :

Problem("Entity set 'ApplicationDbContext.Employees' is null.");

}

// GET: Employees/Details/5

[Authorize(Roles = "Admin,MainAdmin,Employee")]

[AllowAnonymous]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Employees == null)

{

return NotFound();

}

var employee = await \_context.Employees

.Include(e => e.Department)

.Include(e => e.Post)

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

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// GET: Employees/Create

[Authorize(Roles = "Admin,MainAdmin")]

public IActionResult Create()

{

ViewData["DepartmentId"] = new SelectList(\_context.Departments, "DepartmentId", "DepartmentName");

ViewData["PostId"] = new SelectList(\_context.Posts, "PostId", "PostName");

return View();

}

// POST: Employees/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Create([Bind("EmployeeId,EmployeeName,EmployeeSurname,EmployeeMiddlename,PostId,DepartmentId")] Employee employee)

{

if (ModelState.IsValid)

{

\_context.Add(employee);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["DepartmentId"] = new SelectList(\_context.Departments, "DepartmentId", "DepartmentName", employee.DepartmentId);

ViewData["PostId"] = new SelectList(\_context.Posts, "PostId", "PostName", employee.PostId);

return View(employee);

}

// GET: Employees/Edit/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Employees == null)

{

return NotFound();

}

var employee = await \_context.Employees.FindAsync(id);

if (employee == null)

{

return NotFound();

}

ViewData["DepartmentId"] = new SelectList(\_context.Departments, "DepartmentId", "DepartmentName", employee.DepartmentId);

ViewData["PostId"] = new SelectList(\_context.Posts, "PostId", "PostName", employee.PostId);

return View(employee);

}

// POST: Employees/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int id, [Bind("EmployeeId,EmployeeName,EmployeeSurname,EmployeeMiddlename,PostId,DepartmentId")] Employee employee)

{

if (id != employee.EmployeeId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(employee);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeeExists(employee.EmployeeId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["DepartmentId"] = new SelectList(\_context.Departments, "DepartmentId", "DepartmentName", employee.DepartmentId);

ViewData["PostId"] = new SelectList(\_context.Posts, "PostId", "PostName", employee.PostId);

return View(employee);

}

// GET: Employees/Delete/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Employees == null)

{

return NotFound();

}

var employee = await \_context.Employees

.Include(e => e.Department)

.Include(e => e.Post)

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

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// POST: Employees/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Employees == null)

{

return Problem("Entity set 'ApplicationDbContext.Employees' is null.");

}

var employee = await \_context.Employees.FindAsync(id);

if (employee != null)

{

\_context.Employees.Remove(employee);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EmployeeExists(int id)

{

return (\_context.Employees?.Any(e => e.EmployeeId == id)).GetValueOrDefault();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Lab\_5;

using Lab\_5.Data;

using Lab\_5.Models;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Lab\_5.Models.ViewModels;

using Lab\_5.Models.ViewModels.Posts;

using Microsoft.AspNetCore.Authorization;

namespace Lab\_5.Controllers

{

public class PostsController : Controller

{

private readonly ApplicationDbContext \_context;

public PostsController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Posts

[Authorize(Roles = "Admin,MainAdmin,Employee")]

[AllowAnonymous]

public async Task<IActionResult> Index(int page = 1, string postName = "", decimal salary = 0, SortState sortOrder = SortState.NameAsc)

{

IQueryable<Post> posts = \_context.Posts;

int pageSize = 10;

if (postName != null && postName.Trim() != "")

{

posts = posts.Where(p => p.PostName.Contains(postName));

}

if (salary != null && salary != 0)

{

posts = posts.Where(p => p.PostSalary >= salary);

}

var count = posts.Count();

var items = posts.Skip((page - 1) \* pageSize).Take(pageSize);

switch (sortOrder)

{

case SortState.SalaryDesc:

items = items.OrderByDescending(item => item.PostSalary);

break;

case SortState.NameDesc:

items = items.OrderByDescending(item => item.PostName);

break;

}

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

PaginationViewModel<Post, PostFilterViewModel, PostSortViewModel> viewModel = new(items, pageViewModel, new PostFilterViewModel(postName, salary), new PostSortViewModel(sortOrder));

return \_context.Posts != null ?

View(viewModel) :

Problem("Entity set 'ApplicationDbContext.Posts' is null.");

}

// GET: Posts/Details/5

[Authorize(Roles = "Admin,MainAdmin,Employee")]

[AllowAnonymous]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Posts == null)

{

return NotFound();

}

var post = await \_context.Posts

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

if (post == null)

{

return NotFound();

}

return View(post);

}

// GET: Posts/Create

[Authorize(Roles = "Admin,MainAdmin")]

public IActionResult Create()

{

return View();

}

// POST: Posts/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Create([Bind("PostId,PostName,PostSalary")] Post post)

{

if (ModelState.IsValid)

{

\_context.Add(post);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(post);

}

// GET: Posts/Edit/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Posts == null)

{

return NotFound();

}

var post = await \_context.Posts.FindAsync(id);

if (post == null)

{

return NotFound();

}

return View(post);

}

// POST: Posts/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int id, [Bind("PostId,PostName,PostSalary")] Post post)

{

if (id != post.PostId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(post);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!PostExists(post.PostId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(post);

}

// GET: Posts/Delete/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Posts == null)

{

return NotFound();

}

var post = await \_context.Posts

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

if (post == null)

{

return NotFound();

}

return View(post);

}

// POST: Posts/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Posts == null)

{

return Problem("Entity set 'ApplicationDbContext.Posts' is null.");

}

var post = await \_context.Posts.FindAsync(id);

if (post != null)

{

\_context.Posts.Remove(post);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool PostExists(int id)

{

return (\_context.Posts?.Any(e => e.PostId == id)).GetValueOrDefault();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Lab\_5;

using Lab\_5.Data;

using Lab\_5.Models;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Lab\_5.Models.ViewModels;

using Lab\_5.Models.ViewModels.Projects;

using Faker;

using Lab\_5.Models.ViewModels.Departments;

using SortState = Lab\_5.Models.ViewModels.Projects.SortState;

using Microsoft.AspNetCore.Authorization;

namespace Lab\_5.Controllers

{

public class ProjectsController : Controller

{

private readonly ApplicationDbContext \_context;

public ProjectsController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Projects

[Authorize(Roles = "Admin,MainAdmin,Employee")]

[AllowAnonymous]

public async Task<IActionResult> Index(int page = 1, string name = "", SortState sortOrder = SortState.ProjectNameAsc)

{

IQueryable<Project> projects = \_context.Projects;

int pageSize = 10;

if (name != null && name.Trim() != "")

{

projects = projects.Where(project => project.ProjectName.Contains(name));

}

var count = projects.Count();

var items = projects.Skip((page - 1) \* pageSize).Take(pageSize);

switch (sortOrder)

{

case SortState.ProjectNameDesc:

items = items.OrderByDescending(item => item.ProjectName);

break;

case SortState.ProjectIsStoppedDesc:

items = items.OrderByDescending(item => item.IsStopped);

break;

}

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

PaginationViewModel<Project, ProjectFilterViewModel, ProjectSortViewModel> viewModel = new(items, pageViewModel, new ProjectFilterViewModel(name), new ProjectSortViewModel(sortOrder));

return \_context.Projects != null ?

View(viewModel) :

Problem("Entity set 'ApplicationDbContext.Projects' is null.");

}

// GET: Projects/Details/5

[Authorize(Roles = "Admin,MainAdmin,Employee")]

[AllowAnonymous]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Projects == null)

{

return NotFound();

}

var project = await \_context.Projects

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

if (project == null)

{

return NotFound();

}

return View(project);

}

// GET: Projects/Create

[Authorize(Roles = "Admin,MainAdmin")]

public IActionResult Create()

{

return View();

}

// POST: Projects/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Create([Bind("ProjectId,ProjectName,IsStopped")] Project project)

{

if (ModelState.IsValid)

{

\_context.Add(project);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(project);

}

// GET: Projects/Edit/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Projects == null)

{

return NotFound();

}

var project = await \_context.Projects.FindAsync(id);

if (project == null)

{

return NotFound();

}

return View(project);

}

// POST: Projects/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Edit(int id, [Bind("ProjectId,ProjectName,IsStopped")] Project project)

{

if (id != project.ProjectId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(project);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ProjectExists(project.ProjectId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(project);

}

// GET: Projects/Delete/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Projects == null)

{

return NotFound();

}

var project = await \_context.Projects

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

if (project == null)

{

return NotFound();

}

return View(project);

}

// POST: Projects/Delete/5

[HttpPost, ActionName("Delete")]

[Authorize(Roles = "Admin,MainAdmin")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Projects == null)

{

return Problem("Entity set 'ApplicationDbContext.Projects' is null.");

}

var project = await \_context.Projects.FindAsync(id);

if (project != null)

{

\_context.Projects.Remove(project);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ProjectExists(int id)

{

return (\_context.Projects?.Any(e => e.ProjectId == id)).GetValueOrDefault();

}

}

}

using Lab\_5.Models.ViewModels.Users;

using Lab\_5.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

namespace Lab\_5.Controllers

{

public class RolesController : Controller

{

RoleManager<IdentityRole> \_roleManager;

UserManager<User> \_userManager;

public RolesController(RoleManager<IdentityRole> roleManager, UserManager<User> userManager)

{

\_roleManager = roleManager;

\_userManager = userManager;

}

[Authorize(Roles = "Admin, MainAdmin")]

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

[Authorize(Roles = "MainAdmin")]

public IActionResult Create() => View();

[HttpPost]

[Authorize(Roles = "MainAdmin")]

public async Task<IActionResult> Create(string name)

{

if (!string.IsNullOrEmpty(name))

{

IdentityResult result = await \_roleManager.CreateAsync(new IdentityRole(name));

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

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

}

}

}

return View(name);

}

[HttpPost]

[Authorize(Roles = "MainAdmin")]

public async Task<IActionResult> Delete(string id)

{

IdentityRole role = await \_roleManager.FindByIdAsync(id);

if (role != null)

{

IdentityResult result = await \_roleManager.DeleteAsync(role);

}

return RedirectToAction("Index");

}

[Authorize(Roles = "MainAdmin")]

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

[Authorize(Roles = "MainAdmin")]

public async Task<IActionResult> Edit(string userId)

{

// получаем пользователя

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);

}

return NotFound();

}

[HttpPost]

[Authorize(Roles = "MainAdmin")]

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

{

// получаем пользователя

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

if (user != null)

{

// получем список ролей пользователя

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

// получаем все роли

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

// получаем список ролей, которые были добавлены

var addedRoles = roles.Except(userRoles);

// получаем роли, которые были удалены

var removedRoles = userRoles.Except(roles);

await \_userManager.AddToRolesAsync(user, addedRoles);

await \_userManager.RemoveFromRolesAsync(user, removedRoles);

return RedirectToAction("UserList");

}

return NotFound();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Lab\_5;

using Lab\_5.Data;

using Lab\_5.Models;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Lab\_5.Models.ViewModels;

using Lab\_5.Models.ViewModels.Tasks;

using Microsoft.AspNetCore.Authorization;

namespace Lab\_5.Controllers

{

public class TasksController : Controller

{

private readonly ApplicationDbContext \_context;

public TasksController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Tasks

[Authorize(Roles = "Admin,MainAdmin, Employee")]

[AllowAnonymous]

public async Task<IActionResult> Index(int page = 1, int projectId = 0, int employeeId = 0, SortState sortOrder = SortState.ProjectAsc)

{

IQueryable<Task> applicationDbContext = \_context.Tasks.Include(t => t.Employee).Include(t => t.Project);

int pageSize = 10;

IQueryable<Task> source = applicationDbContext;

if (projectId != 0)

{

source = source.Where(task => task.ProjectId == projectId);

}

if (employeeId != 0)

{

source = source.Where(task => task.Employee.EmployeeId == employeeId);

}

var count = source.Count();

var items = source.Skip((page - 1) \* pageSize).Take(pageSize);

switch (sortOrder)

{

case SortState.BeginDateDesc:

items = items.OrderByDescending(item => item.BeginDate);

break;

case SortState.CheckDateDesc:

items = items.OrderByDescending(item => item.CheckDate);

break;

case SortState.EndDateDesc:

items = items.OrderByDescending(item => item.EndDate);

break;

case SortState.EmployeeDesc:

items = items.OrderByDescending(item => item.Employee.EmployeeName);

break;

case SortState.ProjectDesc:

items = items.OrderByDescending(item => item.Project.ProjectName);

break;

case SortState.IsCompletedDesc:

items = items.OrderByDescending(item => item.IsCompleted);

break;

case SortState.FailedReasonDesc:

items = items.OrderByDescending(item => item.FailedReason);

break;

}

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

PaginationViewModel<Task, TaskFilterViewModel, TaskSortViewModel> viewModel = new(items, pageViewModel, new TaskFilterViewModel(\_context.Projects.ToList(), \_context.Employees.ToList(), projectId, employeeId), new TaskSortViewModel(sortOrder));

return View(viewModel);

}

// GET: Tasks/Details/5

[Authorize(Roles = "Admin,MainAdmin, Employee")]

[AllowAnonymous]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Tasks == null)

{

return NotFound();

}

var task = await \_context.Tasks

.Include(t => t.Employee)

.Include(t => t.Project)

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

if (task == null)

{

return NotFound();

}

return View(task);

}

// GET: Tasks/Create

[Authorize(Roles = "Admin,MainAdmin")]

public IActionResult Create()

{

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "EmployeeId", "EmployeeName");

ViewData["ProjectId"] = new SelectList(\_context.Projects, "ProjectId", "ProjectName");

return View();

}

// POST: Tasks/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]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Create([Bind("TaskId,BeginDate,EndDate,IsCompleted,FailedReason,CheckDate,ProjectId,EmployeeId")] Task task)

{

if (ModelState.IsValid)

{

\_context.Add(task);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "EmployeeId", "EmployeeName", task.EmployeeId);

ViewData["ProjectId"] = new SelectList(\_context.Projects, "ProjectId", "ProjectName", task.ProjectId);

return View(task);

}

// GET: Tasks/Edit/5

[Authorize(Roles = "Admin,MainAdmin,Employee")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Tasks == null)

{

return NotFound();

}

var task = await \_context.Tasks.FindAsync(id);

if (task == null)

{

return NotFound();

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "EmployeeId", "EmployeeName", task.EmployeeId);

ViewData["ProjectId"] = new SelectList(\_context.Projects, "ProjectId", "ProjectName", task.ProjectId);

return View(task);

}

// POST: Tasks/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]

[Authorize(Roles = "Admin,MainAdmin,Employee")]

public async Task<IActionResult> Edit(int id, [Bind("TaskId,BeginDate,EndDate,IsCompleted,FailedReason,CheckDate,ProjectId,EmployeeId")] Task task)

{

if (id != task.TaskId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(task);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!TaskExists(task.TaskId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "EmployeeId", "EmployeeName", task.EmployeeId);

ViewData["ProjectId"] = new SelectList(\_context.Projects, "ProjectId", "ProjectName", task.ProjectId);

return View(task);

}

// GET: Tasks/Delete/5

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Tasks == null)

{

return NotFound();

}

var task = await \_context.Tasks

.Include(t => t.Employee)

.Include(t => t.Project)

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

if (task == null)

{

return NotFound();

}

return View(task);

}

// POST: Tasks/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin,MainAdmin")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Tasks == null)

{

return Problem("Entity set 'ApplicationDbContext.Tasks' is null.");

}

var task = await \_context.Tasks.FindAsync(id);

if (task != null)

{

\_context.Tasks.Remove(task);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool TaskExists(int id)

{

return (\_context.Tasks?.Any(e => e.TaskId == id)).GetValueOrDefault();

}

}

}

using Lab\_5.Models;

using Lab\_5.Models.ViewModels.Users;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

namespace Lab\_5.Controllers

{

public class UsersController : Controller

{

UserManager<User> \_userManager;

public UsersController(UserManager<User> userManager)

{

\_userManager = userManager;

}

[Authorize(Roles = "Admin,MainAdmin")]

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

[Authorize(Roles = "MainAdmin")]

public IActionResult Create() => View();

[HttpPost]

[Authorize(Roles = "MainAdmin")]

public async Task<IActionResult> Create(CreateUserViewModel model)

{

if (ModelState.IsValid)

{

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

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

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

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

}

}

}

return View(model);

}

[Authorize(Roles = "MainAdmin")]

public async Task<IActionResult> Edit(string id)

{

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

if (user == null)

{

return NotFound();

}

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

return View(model);

}

[HttpPost]

[Authorize(Roles = "MainAdmin")]

public async 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;

user.Year = model.Year;

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

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

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

}

}

}

}

return View(model);

}

[HttpPost]

[Authorize(Roles = "MainAdmin")]

public async Task<ActionResult> Delete(string id)

{

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

if (user != null)

{

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

}

return RedirectToAction("Index");

}

[Authorize(Roles = "MainAdmin")]

public async Task<IActionResult> ChangePassword(string id)

{

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

if (user == null)

{

return NotFound();

}

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

return View(model);

}

[Authorize(Roles = "MainAdmin")]

[HttpPost]

public async 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 in result.Errors)

{

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

}

}

}

else

{

ModelState.AddModelError(string.Empty, "Пользователь не найден");

}

}

return View(model);

}

}

}