Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра автоматизации обработки информации (АОИ)

Cайт технической поддержки пользователей

Отчет по лабораторной работе по дисциплине

«Сайт для технической поддержки пользователей»

Вариант 12

Студент гр.

Руководитель

Томск 2022

Оглавление

[1. Цели и задачи лабораторной работы 3](#_Toc1)

[2. Индивидуальное задание 4](#_Toc2)

[3. Перечень библиотек и основных классов, методов, использованных в программе 5](#_Toc3)

[4. Результаты работы программы 7](#_Toc4)

[5. Анализ полученных результатов 13](#_Toc5)

[5. Выводы о проделанной работе 14](#_Toc6)

[6. Листинг программы 15](#_Toc7)

# Цели и задачи лабораторной работы

Целью выполнения лабораторной работы является приобретение практических навыков создания простого интернет-приложения при помощи технологии asp.net core.

Для достижения поставленной цели необходимо:

ознакомиться c технологией разработки интернет-приложений при помощи asp.net core mvc;

ознакомиться с порядком работы на Visual Studio Code;

реализовать интернет-приложение “Технической поддержки пользователей”;

*дополнительно реализовать приложение на линукс системе*.

# Индивидуальное задание

Реализовать сайт технической поддержки пользователей с разделением функциональности на основе ролей. Предусмотреть обработку ввода невалидных данных в формы и при входе в систему.

# Перечень библиотек и основных классов, методов, использованных в программе

Для разработки интернет-приложений создано достаточно много разнообразных платформ и фреймворков. Для реализации задания по лабораторной работе был выбран язык программирования C# и современная мощная кроссплатформенная технология разработки интернет-приложений asp.net core mvc. Эта платформа весьма интересна с точки зрения ее миграции с Windows. Таким образов, выполняя данную лабораторную работу был внесен еще один пункт, реализовать приложение не только на C# dotnet 6, но и осуществлять процесс разработки на linux. На машине для разработки была установлена manjaro KDE, для дальнейшей работы необходимо установить средства разработки. Для linux хорошим бесплатным выбором станет редактор кода с расширениямиVisual Studio Code. Безусловно, Rider от JetBrains был бы хорошим выбором, но, к сожалению, он платный. Для работы с C# на линукс были установлены следующие расширения omnisharp, git, docker (для запуска контейнера с базой данных). Предварительно необходимо установить dotnet sdk и runtime. Описание процесса установки можно найти на официальном сайте Microsoft. В качестве базы данных была выбрана MS SQL Server 2019 Express, отчасти обусловлено тоже интересом использования и тестирования продукции упомянутого вендора на линукс. Для взаимодействия с базой данных были выбраны слейдующие фреймворки Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCoreDesign, Microsoft.EntityFrameworkCore.Tools, Microsoft.EntityFrameworkCoreSqlServer. Все перечисленые библиотеки и фреймворки позволяют использовать подход codeFirst при создании базы данных. За авторизацию, аутентификацию и идентификацию отвечает библиотека Microsoft.AspNetCore.Identity  это стандартная библиотека для управления пользователями, паролями, данными профиля и др от вендора, примеры использования расположены на официальной документации производителя. Для преобразования моделей базы данных в модель представления была использована библиотека AutoMapper. Для реализации пользовательского интерфейса была использована библиотека bootstrap.

# Результаты работы программы

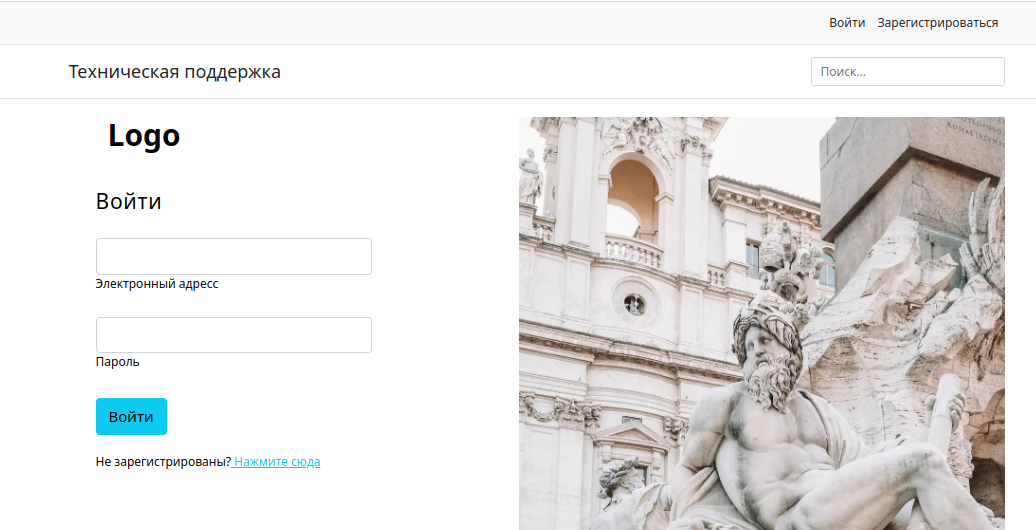


Рисунок . – Внешний вид страницы входа

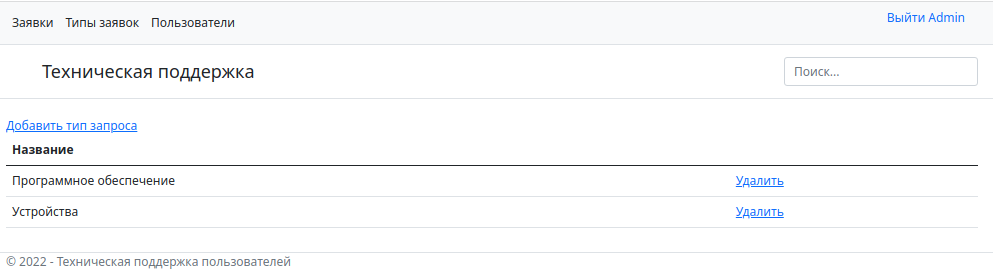


Рисунок . – Внешний вид страницы со списком типов заявок

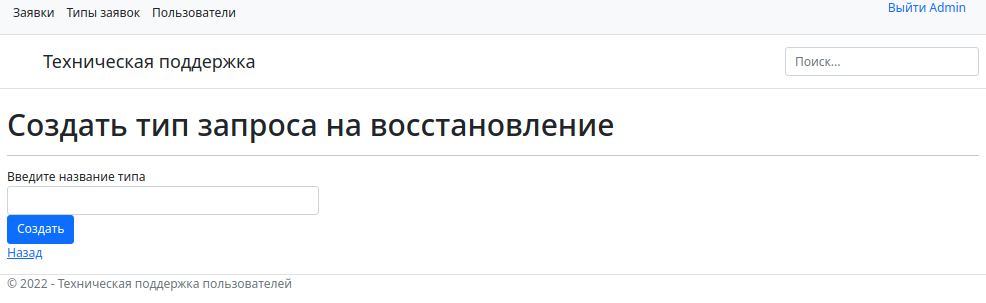


Рисунок . – Внешний вид страницы для добавления типов запроса на восстановление

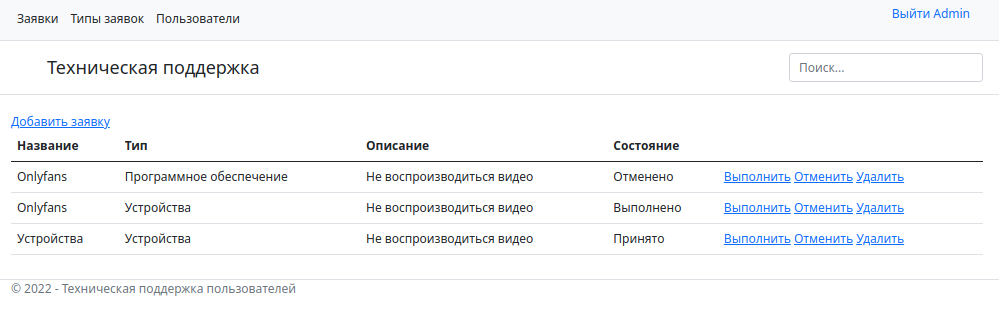


Рисунок 4.4 – Внешний вид списка заявок в техническую поддержку

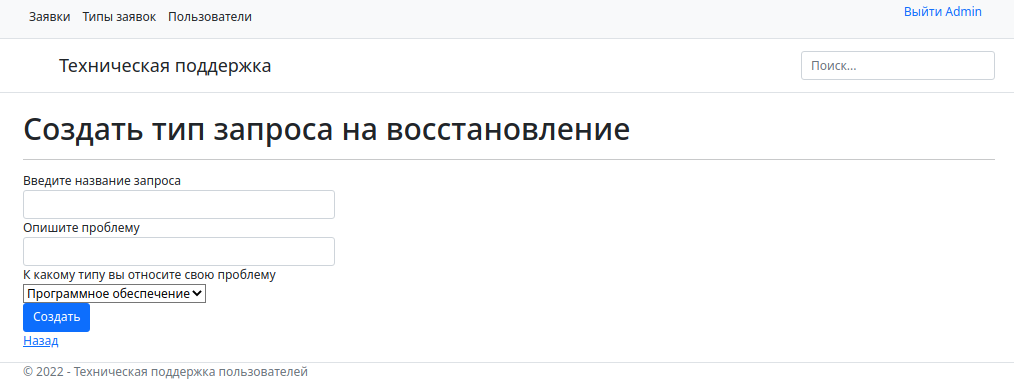


Рисунок 4.5 – Внешний вид страницы добавления заявки в техническую поддержку

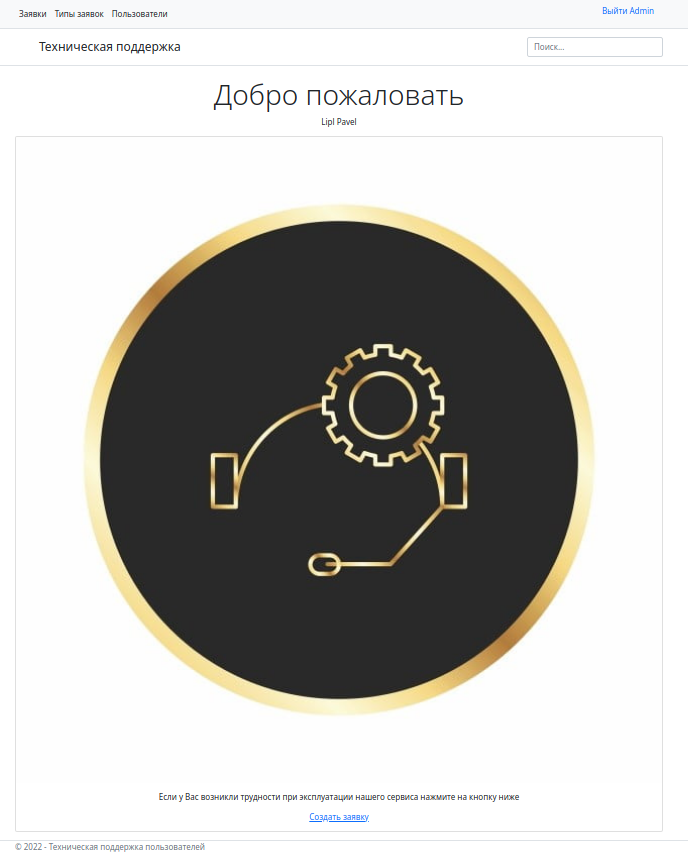


Рисунок 4.6 – Внешний вид главной страницы для администратора

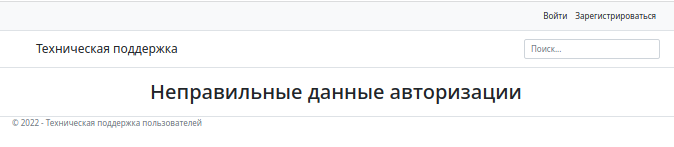


Рисунок 4.7 – Внешний вид страницы при ошибочных данных авторизации

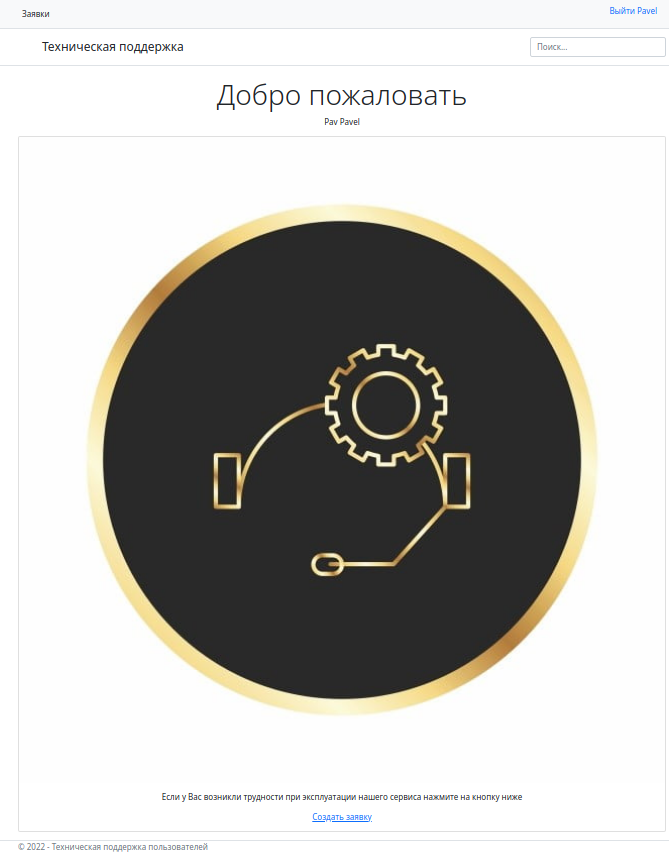


Рисунок 4.8 – Внешний вид главной страницы для обычного пользователя

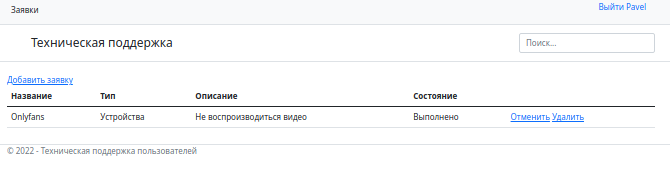


Рисунок 4.9 – Внешний вид страницы списка заявок пользователя

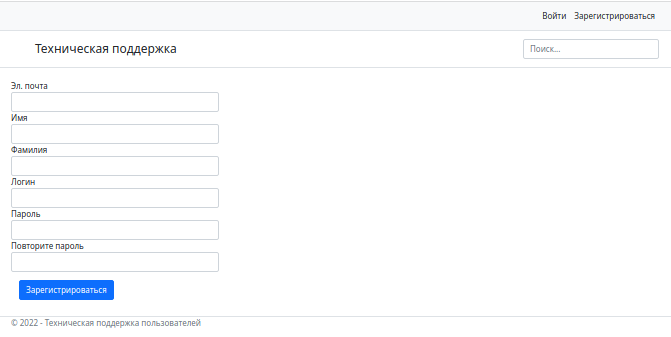


Рисунок 4.10 – Внешний вид страницы регистрации

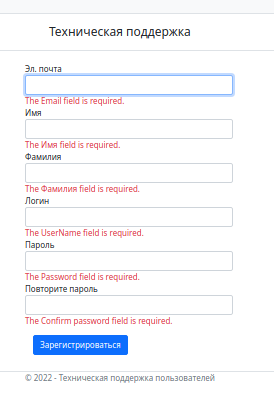


Рисунок 4.11 – Внешний вид страницы регистрации при не введенных данных

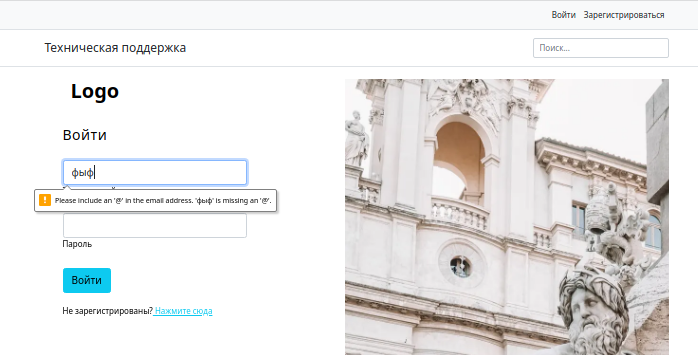


Рисунок . – Внешний вид страницы входа при не правильно введенных данных

# 5. Анализ полученных результатов

В результате выполнения лабораторной работы было разработано интернет-приложение, реализованное на языке программирования C# 10 с использованием технологии asp.net mvc (dotnet 6.0). Основными особенностями реализованного приложения можно выделить реализацию паттерна IRepository для организации доступа к данным из СУБД. Весь функционал как взаимодействия с базой данных, так и логики работы реализован в классе бизнес логики и введен на уровень контроллеров. Это позволит менять интерфейс взаимодействия с пользователями, например, в виде телеграмм бота. На уровне контроллеров была организована работа только с запросами клиентов.

Примененная архитектура позволяет обеспечить масштабируемость уровня бизнес логики и переносимость на другие уровни взаимодействия с пользовательскими интерфейсами. С целью дальнейшего совершенствования целесообразно выделить уровень работы с базой данных и ядра системы.

# Выводы о проделанной работе

Результатом проделанной работы является интернет-приложения технической поддержки пользователей. Приложение разработано с упомянутыми фреймоврками и библиотеками, результат представлен наглядно в главе 4 и описан в главе 5. Важно отметить результаты дополнительных целей, а именно реализации интернет-приложения средствами компании Microsoft на базе операционной системы linux, дистрибутива manjaro. Опыт выполнения лабораторной работы показал, что разработка на линукс системах для платформы dotnet вполне возможна, хоть и менее комфортабельна (Visual Studio 2022 более удобна за счет большей функциональности и углубленной проработки интеллектуалного ввода, а также автоматического определения требуемых библиотек с возможностью автоматизированной установки).

Таким образом, считаю, что поставленные задачи выполнены и цель лабораторной работы достигнута.

# Листинг программы

using System.Diagnostics;

using Microsoft.AspNetCore.Mvc;

using RequestBoard.Models;

using RequestBoard.Models.DbModels;

using RequestBoard.Models.Interfaces.IRepository;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Authorization;

namespace RequestBoard.Controllers;

[Authorize]

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private readonly IBusinessLayer \_businnesLayer;

private readonly UserManager<ApplicationUser> \_userManager;

public HomeController(ILogger<HomeController> logger, IBusinessLayer businnesLayer, UserManager<ApplicationUser> userManager)

{

\_logger = logger;

\_userManager = userManager;

\_businnesLayer = businnesLayer;

}

public async Task<IActionResult> Index()

{

var user = await \_userManager.FindByNameAsync(User.Identity.Name);

return View(user);

}

public IActionResult RequestTypeList()

{

try

{

var models = \_businnesLayer.GetAllRequestType();

return View(models);

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

public IActionResult RequestList()

{

try

{

if (User.IsInRole("Admin"))

{

var models = \_businnesLayer.GetAllRequestToRestore();

return View(models);

}

else

{

var userId = \_userManager.GetUserId(User);

var models = \_businnesLayer.GetRequestsByUserId(userId);

return View(models);

}

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

[HttpGet]

public IActionResult CreateRequest()

{

try

{

var requestList = \_businnesLayer.GetAllRequestType();

ViewBag.RequestTypes = new SelectList(requestList, "Id", "Name");

return View();

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

public IActionResult DeleteRequest(Guid id)

{

try

{

var request = \_businnesLayer.RemoveRequestToRestore(id);

return request is null ? NotFound() : RedirectToAction("RequestList");

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

[HttpPost]

public IActionResult CreateRequest(RequestToRestore model)

{

if (model is null)

return BadRequest();

try

{

model.UserId = \_userManager.GetUserId(User);

model.Stage = Stages.Принято;

var request = \_businnesLayer.AddRequestToRestore(model);

if (request is null)

return NotFound();

return RedirectToAction("RequestList");

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

[HttpGet]

public IActionResult CreateRequestType()

{

return View();

}

[HttpPost]

public IActionResult CreateRequestType(RequestType model)

{

try

{

var item = \_businnesLayer.AddRequestType(model);

return item is null ? BadRequest() : RedirectToAction("RequestTypeList");

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

public IActionResult RemoveRequestType(Guid id)

{

try

{

var item = \_businnesLayer.RemoveRequestType(id);

return item is null ? BadRequest() : RedirectToAction("RequestTypeList");

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

public IActionResult RejectRequest(Guid id)

{

try

{

\_businnesLayer.RejectRequest(id);

return RedirectToAction("RequestList");

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return NotFound();

}

}

public IActionResult FullfillRequest(Guid id)

{

try

{

\_businnesLayer.FullfillRequest(id);

return RedirectToAction("RequestList");

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return BadRequest();

}

}

public IActionResult Find(string searchString)

{

try

{

var models = \_businnesLayer.FindRequestByName(searchString);

return View("RequestList", model:models);

}

catch (Exception ex)

{

\_logger.LogError(ex.Message);

return RedirectToAction("Index");

}

}

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

public IActionResult Error()

{

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

}

}

using RequestBoard.Models.DbModels;

using RequestBoard.Models.Interfaces.IRepository;

using AutoMapper;

using RequestBoard.Models.DtoModels;

namespace RequestBoard.BusinessLayer

{

public class BusinessLayerClass : IBusinessLayer

{

private readonly IRequestToRestoreRepository \_requestToRestoreRepository;

private readonly IRequestTypeRepository \_requestTypeRepository;

private readonly IMapper \_mapper;

public BusinessLayerClass(IRequestToRestoreRepository requestToRestoreRepository, IRequestTypeRepository requestTypeRepository, IMapper mapper)

{

\_requestToRestoreRepository = requestToRestoreRepository;

\_requestTypeRepository = requestTypeRepository;

\_mapper = mapper;

}

public RequestDto AddRequestToRestore(RequestToRestore model)

{

\_requestToRestoreRepository.AddEntity(model);

return \_mapper.Map<RequestDto>(model);

}

public RequestType AddRequestType(RequestType model)

{

\_requestTypeRepository.AddEntity(model);

return model;

}

public List<RequestDto> FindRequestByName(string name)

{

var items = \_requestToRestoreRepository.FindRequestByPartOfName(name);

return items is null ? throw new ArgumentException(nameof(items)) : \_mapper.Map<List<RequestDto>>(items);

}

public RequestDto FullfillRequest(Guid requestId)

{

if(requestId == Guid.Empty)

throw new ArgumentException(nameof(requestId));

var item = \_requestToRestoreRepository.GetEntityById(requestId);

item.Stage = Models.Stages.Выполнено;

\_requestToRestoreRepository.UpdateEntity(item);

return \_mapper.Map<RequestDto>(item);

}

public List<RequestDto> GetAllRequestToRestore()

{

var dbmodels = \_requestToRestoreRepository.GetAllEntities();

return dbmodels is null ? throw new ArgumentException(nameof(dbmodels)) : \_mapper.Map<List<RequestDto>>(dbmodels);

}

public List<RequestType> GetAllRequestType()

{

return (List<RequestType>)\_requestTypeRepository.GetAllEntities();

}

public RequestDto GetByIdRequestToRestore(Guid requestId)

{

var item = \_requestToRestoreRepository.GetEntityById(requestId);

return item is null ? throw new ArgumentException(nameof(item)) : \_mapper.Map<RequestDto>(item);

}

public RequestType GetByIdRequestType(Guid requestTypeId)

{

return \_requestTypeRepository.GetEntityById(requestTypeId);

}

public List<RequestDto> GetRequestsByUserId(string userId)

{

var items = \_requestToRestoreRepository.GetWithInclude(p => p.UserId == userId);

return items is null ? throw new ArgumentException(nameof(items)): \_mapper.Map<List<RequestDto>>(items);

}

public RequestDto RejectRequest(Guid requestId)

{

if(requestId == Guid.Empty)

throw new ArgumentException(nameof(requestId));

var item = \_requestToRestoreRepository.GetEntityById(requestId);

item.Stage = Models.Stages.Отменено;

\_requestToRestoreRepository.UpdateEntity(item);

return \_mapper.Map<RequestDto>(item);

}

public RequestDto RemoveRequestToRestore(Guid requestId)

{

var item = \_requestToRestoreRepository.GetEntityById(requestId);

\_requestToRestoreRepository.RemoveEntity(item);

return \_mapper.Map<RequestDto>(item);

}

public RequestType RemoveRequestType(Guid requestTypeId)

{

var item = \_requestTypeRepository.GetEntityById(requestTypeId);

\_requestTypeRepository.RemoveEntity(item);

return item;

}

public RequestDto UpdateRequestToRestore(Guid requestId, RequestToRestore model)

{

\_requestToRestoreRepository.UpdateEntity(model);

var item = \_requestToRestoreRepository.GetEntityById(requestId);

return \_mapper.Map<RequestDto>(item);

}

public RequestType UpdateRequestType(Guid requestTypeId, RequestType model)

{

\_requestTypeRepository.UpdateEntity(model);

return \_requestTypeRepository.GetEntityById(requestTypeId);

}

}

}

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using RequestBoard.Models.DbModels;

namespace RequestBoard.DataAccess

{

public class AppDbContext : IdentityDbContext<ApplicationUser, IdentityRole, string>

{

public AppDbContext(DbContextOptions<AppDbContext> options)

: base(options)

{

Database.EnsureCreated();

}

public DbSet<RequestType> RequestTypes { get; set; }

public DbSet<RequestToRestore> RequestToRestores{ get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Seed();

}

}

}

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using RequestBoard.Models.Interfaces;

namespace RequestBoard.Models.DbModels

{

public class RequestToRestore : IEntity<Guid>

{

public Guid Id { get; set; }

[Required(ErrorMessage ="Обязательное поле")]

public string Name { get; set; }

[Required(ErrorMessage ="Обязательное поле")]

public string Description { get; set; }

public string UserId { get; set; }

public Stages Stage {get;set;}

[ForeignKey("RequestType")]

public Guid RequestTypeId { get; set; }

public RequestType RequestType { get; set; }

}

}

using RequestBoard.Models.Interfaces;

namespace RequestBoard.Models.DbModels

{

public class RequestType : IEntity<Guid>

{

public Guid Id { get; set; }

public string Name { get; set; }

}

}