Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

\_\_\_\_\_\_\_\_ Ілля АХАЛАДЗЕ

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 р.

**ВЕБ-ЗАСТОСУНОК СИСТЕМА ПОШУКУ ВИКОНАВЦІВ ТА ЗАМОВНИКІВ ПОСЛУГ**

**Текст програми**

КПІ.ІП-1310.045440.03.12

“ПОГОДЖЕНО”

Керівник роботи:

\_\_\_\_\_\_\_\_\_\_\_\_ Ілля АХАЛАДЗЕ

|  |
| --- |
| Виконавець: |
| \_\_\_\_\_\_\_\_\_\_\_ Олександр ДЕМ’ЯНЧУК |

Київ – 2024

**Файл RegisterCommand.cs**

using Dealoviy.Application.Authentication.Common;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Authentication.Commands.Register;

public record RegisterCommand(

string Username,

string? DisplayName,

string Password

) : IRequest<ErrorOr<AuthenticationResult>>;

**Файл RegisterCommandHandler.cs**

using Dealoviy.Application.Authentication.Common;

using Dealoviy.Application.Common.Interfaces.Authentication;

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Common.Interfaces.Security;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Users;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Authentication.Commands.Register;

public class RegisterCommandHandler

: IRequestHandler<RegisterCommand, ErrorOr<AuthenticationResult>>

{

private readonly IJwtTokenGenerator \_jwtTokenGenerator;

private readonly IUserRepository \_userRepository;

private readonly IPasswordHasher \_passwordHasher;

public RegisterCommandHandler(

IJwtTokenGenerator jwtTokenGenerator,

IUserRepository userRepository,

IPasswordHasher passwordHasher)

{

\_jwtTokenGenerator = jwtTokenGenerator;

\_userRepository = userRepository;

\_passwordHasher = passwordHasher;

}

public async Task<ErrorOr<AuthenticationResult>> Handle(RegisterCommand request, CancellationToken cancellationToken)

{

if (await \_userRepository.GetUserByUsernameAsync(request.Username) is not null)

{

return Errors.DuplicateUsername;

}

var hashedPassword = \_passwordHasher.HashPassword(request.Password);

var user = User.Create(

request.Username,

request.DisplayName,

hashedPassword);

await \_userRepository.AddAsync(user);

var token = \_jwtTokenGenerator.GenerateToken(user);

var contractorProfileId = user.ContractorProfileId is null

? null

: user.ContractorProfileId.ToString();

return new AuthenticationResult(

user.Id,

user.Username,

user.DisplayName,

contractorProfileId,

token);

}

}

**Файл RegisterCommandValidator.cs**

using FluentValidation;

namespace Dealoviy.Application.Authentication.Commands.Register;

public class RegisterCommandValidator : AbstractValidator<RegisterCommand>

{

public RegisterCommandValidator()

{

RuleFor(x => x.Username)

.NotEmpty()

.WithErrorCode("Validation.Username.Empty")

.WithMessage("Username cannot be empty");

RuleFor(x => x.Username)

.MaximumLength(20)

.WithErrorCode("Validation.FirstName.TooLong")

.WithMessage("First name cannot be longer than 20 characters");

RuleFor(x => x.DisplayName)

.MaximumLength(50)

.WithErrorCode("Validation.DisplayName.TooLong")

.WithMessage("Display name cannot be longer than 50 characters");

RuleFor(x => x.Password)

.NotEmpty()

.WithErrorCode("Validation.Password.Empty")

.WithMessage("Password cannot be empty");

RuleFor(x => x.Password)

.MinimumLength(8)

.WithErrorCode("Validation.Password.TooShort")

.WithMessage("Password cannot be shorter than 8 characters");

RuleFor(x => x.Password)

.MaximumLength(50)

.WithErrorCode("Validation.Password.TooLong")

.WithMessage("Password cannot be longer than 50 characters");

}

}

**Файл AuthenticationResult.cs**

namespace Dealoviy.Application.Authentication.Common;

public record AuthenticationResult(

Guid Id,

string Username,

string? DisplayName,

string? ContractorProfileId,

string Token);

**Файл LoginQuery.cs**

using Dealoviy.Application.Authentication.Common;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Authentication.Queries.Login;

public record LoginQuery(

string Username,

string Password) : IRequest<ErrorOr<AuthenticationResult>>;

**Файл LoginQueryHandler.cs**

using Dealoviy.Application.Authentication.Common;

using Dealoviy.Application.Common.Interfaces.Authentication;

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Common.Interfaces.Security;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Users;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Authentication.Queries.Login;

public class LoginQueryHandler : IRequestHandler<LoginQuery, ErrorOr<AuthenticationResult>>

{

private readonly IJwtTokenGenerator \_jwtTokenGenerator;

private readonly IUserRepository \_userRepository;

private readonly IPasswordHasher \_passwordHasher;

public LoginQueryHandler(

IJwtTokenGenerator jwtTokenGenerator,

IUserRepository userRepository,

IPasswordHasher passwordHasher)

{

\_jwtTokenGenerator = jwtTokenGenerator;

\_userRepository = userRepository;

\_passwordHasher = passwordHasher;

}

public async Task<ErrorOr<AuthenticationResult>> Handle(LoginQuery request, CancellationToken cancellationToken)

{

if (await \_userRepository.GetUserByUsernameAsync(request.Username) is not User user)

{

return Errors.InvalidCredentials;

}

if(!\_passwordHasher.VerifyPassword(user.PasswordHash, request.Password))

{

return Errors.InvalidCredentials;

}

var token = \_jwtTokenGenerator.GenerateToken(user);

var contractorProfileId = user.ContractorProfileId is null

? null

: user.ContractorProfileId.ToString();

return new AuthenticationResult(

user.Id,

user.Username,

user.DisplayName,

contractorProfileId,

token);

}

}

**Файл CityResult.cs**

namespace Dealoviy.Application.Cities.Common;

public record CityResult(

Guid Id,

string Name

);

**Файл GetCitiesInRegionQuery.cs**

using Dealoviy.Application.Cities.Common;

using MediatR;

namespace Dealoviy.Application.Cities.Queries.GetCitiesInRegion;

public record GetCitiesInRegionQuery(Guid RegionId)

: IRequest<IEnumerable<CityResult>>;

**Файл GetCitiesInRegionQueryHandler.cs**

using Dealoviy.Application.Cities.Common;

using Dealoviy.Application.Common.Interfaces.Persistence;

using MapsterMapper;

using MediatR;

namespace Dealoviy.Application.Cities.Queries.GetCitiesInRegion;

public class GetCitiesInRegionQueryHandler : IRequestHandler<GetCitiesInRegionQuery, IEnumerable<CityResult>>

{

private readonly ICityRepository \_cityRepository;

private readonly IMapper \_mapper;

public GetCitiesInRegionQueryHandler(

ICityRepository cityRepository,

IMapper mapper)

{

\_cityRepository = cityRepository;

\_mapper = mapper;

}

public async Task<IEnumerable<CityResult>> Handle(GetCitiesInRegionQuery request, CancellationToken cancellationToken)

{

var regions = await \_cityRepository.GetCitiesByRegionIdAsync(request.RegionId);

return \_mapper.Map<IEnumerable<CityResult>>(regions);

}

}

**Файл IContractorProfileCommand.cs**

using Dealoviy.Application.Common.Models;

namespace Dealoviy.Application.ContractorProfiles.Commands.Common.Interfaces;

public interface IContractorProfileCommand

{

Guid UserId { get; }

string AdditionalInfo { get; }

List<ContactInfoModel> ContactInfos { get; }

}

**Файл ContractorProfileCommandBaseValidator.cs**

using Dealoviy.Application.Common.Validators;

using Dealoviy.Application.ContractorProfiles.Commands.Common.Interfaces;

using FluentValidation;

namespace Dealoviy.Application.ContractorProfiles.Commands.Common.Validators;

public abstract class ContractorProfileCommandBaseValidator<TContractorProfileCommand>

: AbstractValidator<TContractorProfileCommand>

where TContractorProfileCommand : IContractorProfileCommand

{

protected ContractorProfileCommandBaseValidator()

{

RuleFor(x => x.AdditionalInfo)

.NotEmpty()

.WithErrorCode("Validation.AdditionalInfo.Required")

.WithMessage("Additional info is required");

RuleFor(x => x.ContactInfos)

.NotEmpty()

.WithErrorCode("Validation.ContactInfos.Required")

.WithMessage("Contact infos are required");

RuleForEach(x => x.ContactInfos)

.SetValidator(new ContactInfoModelValidator());

}

}

**Файл CreateContractorProfileCommand.cs**

using Dealoviy.Application.Common.Models;

using Dealoviy.Application.ContractorProfiles.Commands.Common.Interfaces;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.ContractorProfiles.Commands.Create;

public record CreateContractorProfileCommand(

Guid UserId,

string AdditionalInfo,

List<ContactInfoModel> ContactInfos

) : IContractorProfileCommand,

IRequest<ErrorOr<Unit>>;

**Файл CreateContractorProfileCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Domain.Common.ContactInfo;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.ContractorProfiles;

using Dealoviy.Domain.Users;

using MediatR;

using ErrorOr;

using MapsterMapper;

namespace Dealoviy.Application.ContractorProfiles.Commands.Create;

public class CreateContractorProfileCommandHandler : IRequestHandler<CreateContractorProfileCommand, ErrorOr<Unit>>

{

private readonly IContractorProfileRepository \_contractorProfileRepository;

private readonly IUserRepository \_userRepository;

private readonly IMapper \_mapper;

public CreateContractorProfileCommandHandler(

IContractorProfileRepository contractorProfileRepository,

IUserRepository userRepository,

IMapper mapper)

{

\_contractorProfileRepository = contractorProfileRepository;

\_userRepository = userRepository;

\_mapper = mapper;

}

public async Task<ErrorOr<Unit>> Handle(CreateContractorProfileCommand request, CancellationToken cancellationToken)

{

if (await \_userRepository.GetUserByIdAsync(request.UserId)

is not User user)

{

return Errors.UserNotFound;

}

var contractorProfileResult = ContractorProfile.Create(

request.AdditionalInfo,

\_mapper.Map<List<ContactInfoCreateModel>>(request.ContactInfos)

);

if (contractorProfileResult.IsError)

{

return contractorProfileResult.Errors;

}

var contractorProfile = contractorProfileResult.Value;

await \_contractorProfileRepository.AddAsync(contractorProfile);

user.SetContractorProfileId(contractorProfile.Id);

await \_userRepository.UpdateAsync(user);

return Unit.Value;

}

}

**Файл CreateContractorProfileCommandValidator.cs**

using Dealoviy.Application.ContractorProfiles.Commands.Common.Validators;

namespace Dealoviy.Application.ContractorProfiles.Commands.Create;

public class CreateContractorProfileCommandValidator :

ContractorProfileCommandBaseValidator<CreateContractorProfileCommand>

{

public CreateContractorProfileCommandValidator() : base()

{

}

}

**Файл ContractorProfileResult.cs**

namespace Dealoviy.Application.ContractorProfiles.Queries.Common;

public record ContractorProfileResult(

Guid Id,

string Name,

string AdditionalInfo,

string[] ContactTypes

);

**Файл GetContractorProfileByIdQuery.cs**

using Dealoviy.Application.ContractorProfiles.Queries.Common;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.ContractorProfiles.Queries.GetById;

public record GetContractorProfileByIdQuery(Guid Id)

: IRequest<ErrorOr<ContractorProfileResult>>;

**Файл GetContractorProfileByIdQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.ContractorProfiles.Queries.Common;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.ContractorProfiles;

using Dealoviy.Domain.Users;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.ContractorProfiles.Queries.GetById;

public class GetContractorProfileByIdQueryHandler

: IRequestHandler<GetContractorProfileByIdQuery, ErrorOr<ContractorProfileResult>>

{

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

public GetContractorProfileByIdQueryHandler(IUserRepository userRepository, IContractorProfileRepository contractorProfileRepository)

{

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

}

public async Task<ErrorOr<ContractorProfileResult>> Handle(

GetContractorProfileByIdQuery request,

CancellationToken cancellationToken)

{

if (await \_contractorProfileRepository.GetByIdAsync(request.Id) is not ContractorProfile profile)

{

return Errors.ContractorProfileNotFound;

}

if (await \_userRepository.GetByContractorIdAsync(request.Id) is not User user)

{

return Errors.UserNotFound;

}

return new ContractorProfileResult(

profile.Id,

user.GetDisplayName(),

profile.AdditionalInfo,

profile.ContactInfos.

Select(ct => ct.Type.ToString()).ToArray());

}

}

**Файл FinishOrderCommand.cs**

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Orders.Commands.Finish;

public record FinishOrderCommand(Guid OrderId, Guid UserContractorId)

: IRequest<ErrorOr<Unit>>;

**Файл FinishOrderCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Domain.Orders;

using Dealoviy.Domain.Users;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Orders.Commands.Finish;

public class FinishOrderCommandHandler

: IRequestHandler<FinishOrderCommand, ErrorOr<Unit>>

{

private readonly IOrderRepository \_orderRepository;

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

public FinishOrderCommandHandler(

IOrderRepository orderRepository,

IUserRepository userRepository,

IContractorProfileRepository contractorProfileRepository)

{

\_orderRepository = orderRepository;

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

}

public async Task<ErrorOr<Unit>> Handle(FinishOrderCommand request, CancellationToken cancellationToken)

{

if (await \_orderRepository.GetByIdAsync(request.OrderId)

is not Order order)

{

return Error.NotFound("Order.NotFound", "Order was not found");

}

if (await \_userRepository.GetUserByIdAsync(request.UserContractorId)

is not User contractor)

{

return Error.NotFound("Contractor.NotFound", "Contractor was not found");

}

if (contractor.ContractorProfileId is null)

{

return Error.Conflict("User.NotContractor", "User is not a contractor");

}

if (await \_contractorProfileRepository.GetByIdAsync(contractor.ContractorProfileId.Value)

is null)

{

return Error.NotFound("ContractorProfile.NotFound", "Contractor profile was not found");

}

order.UpdateOrderStatus(OrderStatus.Finished);

await \_orderRepository.UpdateAsync(order);

return Unit.Value;

}

}

**Файл StartOrderCommand.cs**

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Orders.Commands.Start;

public record StartOrderCommand(Guid OrderId, Guid UserContractorId)

: IRequest<ErrorOr<Unit>>;

**Файл StartOrderCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Domain.Orders;

using Dealoviy.Domain.Users;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Orders.Commands.Start;

public class StartOrderCommandHandler

: IRequestHandler<StartOrderCommand, ErrorOr<Unit>>

{

private readonly IOrderRepository \_orderRepository;

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

public StartOrderCommandHandler(

IOrderRepository orderRepository,

IUserRepository userRepository,

IContractorProfileRepository contractorProfileRepository)

{

\_orderRepository = orderRepository;

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

}

public async Task<ErrorOr<Unit>> Handle(StartOrderCommand request, CancellationToken cancellationToken)

{

if (await \_orderRepository.GetByIdAsync(request.OrderId)

is not Order order)

{

return Error.NotFound("Order.NotFound", "Order was not found");

}

if (await \_userRepository.GetUserByIdAsync(request.UserContractorId)

is not User contractor)

{

return Error.NotFound("Contractor.NotFound", "Contractor was not found");

}

if (contractor.ContractorProfileId is null)

{

return Error.Conflict("User.NotContractor", "User is not a contractor");

}

if (await \_contractorProfileRepository.GetByIdAsync(contractor.ContractorProfileId.Value)

is null)

{

return Error.NotFound("ContractorProfile.NotFound", "Contractor profile was not found");

}

order.UpdateOrderStatus(OrderStatus.InProgress);

await \_orderRepository.UpdateAsync(order);

return Unit.Value;

}

}

**Файл GetOrdersForServiceQuery.cs**

using Dealoviy.Contracts.Orders;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Orders.Queries.GetOrdersForService;

public record GetOrdersForServiceQuery(Guid ServiceId)

: IRequest<ErrorOr<IEnumerable<ServiceOrderResponse>>>;

**Файл GetOrdersForServiceQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Contracts.Common;

using Dealoviy.Contracts.Orders;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Users;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Orders.Queries.GetOrdersForService;

public class GetOrdersForServiceQueryHandler

: IRequestHandler<GetOrdersForServiceQuery,

ErrorOr<IEnumerable<ServiceOrderResponse>>>

{

private readonly IServiceRepository \_serviceRepository;

private readonly IOrderRepository \_orderRepository;

private readonly IUserRepository \_userRepository;

public GetOrdersForServiceQueryHandler(

IServiceRepository serviceRepository,

IOrderRepository orderRepository,

IUserRepository userRepository)

{

\_serviceRepository = serviceRepository;

\_orderRepository = orderRepository;

\_userRepository = userRepository;

}

public async Task<ErrorOr<IEnumerable<ServiceOrderResponse>>>

Handle(GetOrdersForServiceQuery request,

CancellationToken cancellationToken)

{

if (await \_serviceRepository.GetByIdAsync(request.ServiceId)

is not { } service)

{

return Errors.ServiceNotFound;

}

var orders = await \_orderRepository.GetByServiceIdAsync(service.Id);

var customersTasks = orders

.Select(r => \_userRepository.GetUserByIdAsync(r.CustomerId));

var customers = new List<User>();

foreach (var task in customersTasks)

{

customers.Add(await task);

}

return orders

.Select((o, i) => new ServiceOrderResponse(

o.Id,

o.Description,

o.PaymentAmount,

o.OrderDate,

o.OrderStatus.ToString(),

customers[i].GetDisplayName(),

new ContactInfoResponse(

o.CustomerContactInfo.Type.ToString(),

o.CustomerContactInfo.Value)

))

.OrderByDescending(o => o.RequestDate)

.ToList();

}

}

**Файл GetOrdersForUserQuery.cs**

using Dealoviy.Contracts.Orders;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Orders.Queries.GetOrdersForUser;

public record GetOrdersForUserQuery(Guid UserId)

: IRequest<ErrorOr<IEnumerable<UserOrderResponse>>>;

**Файл GetOrdersForUserQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Contracts.Common;

using Dealoviy.Contracts.Orders;

using Dealoviy.Domain.ContractorProfiles;

using Dealoviy.Domain.Services;

using Dealoviy.Domain.Users;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Orders.Queries.GetOrdersForUser;

public class GetOrdersForUserQueryHandler

: IRequestHandler<GetOrdersForUserQuery, ErrorOr<IEnumerable<UserOrderResponse>>>

{

private readonly IServiceRepository \_serviceRepository;

private readonly IOrderRepository \_orderRepository;

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

public GetOrdersForUserQueryHandler(

IServiceRepository serviceRepository,

IOrderRepository orderRepository,

IUserRepository userRepository,

IContractorProfileRepository contractorProfileRepository)

{

\_serviceRepository = serviceRepository;

\_orderRepository = orderRepository;

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

}

public async Task<ErrorOr<IEnumerable<UserOrderResponse>>> Handle(

GetOrdersForUserQuery request,

CancellationToken cancellationToken)

{

var orders = await \_orderRepository.GetByCustomerIdAsync(request.UserId);

var servicesTasks = orders

.Select(r => \_serviceRepository.GetByIdAsync(r.ServiceId));

var services = new List<Service>();

foreach (var task in servicesTasks)

{

services.Add(await task);

}

var contractorTasks = services

.Select(s => \_contractorProfileRepository.GetByIdAsync(s.ContractorId));

var contractors = new List<ContractorProfile>();

foreach (var task in contractorTasks)

{

contractors.Add(await task);

}

var usersTasks = contractors

.Select(c => \_userRepository.GetByContractorIdAsync(c.Id));

var users = new List<User>();

foreach (var task in usersTasks)

{

users.Add(await task);

}

return orders

.Select((r, i) => new UserOrderResponse(

r.Id,

r.Description,

r.PaymentAmount,

r.OrderDate,

r.OrderStatus.ToString(),

users[i].GetDisplayName(),

services[i].Id,

services[i].Name,

new ContactInfoResponse(

r.ContractorContactInfo.Type.ToString(),

r.ContractorContactInfo.Value)))

.OrderByDescending(r => r.RequestDate)

.ToList();

}

}

**Файл RegionResult.cs**

namespace Dealoviy.Application.Regions.Common;

public record RegionResult(

Guid Id,

string Name

);

**Файл GetRegionQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Regions.Common;

using MapsterMapper;

using MediatR;

namespace Dealoviy.Application.Regions.Queries.GetRegion;

public class GetRegionQueryHandler : IRequestHandler<GetRegionsQuery, IEnumerable<RegionResult>>

{

private readonly IRegionRepository \_regionRepository;

private readonly IMapper \_mapper;

public GetRegionQueryHandler(

IRegionRepository regionRepository,

IMapper mapper)

{

\_regionRepository = regionRepository;

\_mapper = mapper;

}

public async Task<IEnumerable<RegionResult>> Handle(GetRegionsQuery request, CancellationToken cancellationToken)

{

var regions = await \_regionRepository.GetAllRegionsAsync();

return \_mapper.Map<IEnumerable<RegionResult>>(regions);

}

}

**Файл GetRegionsQuery.cs**

using Dealoviy.Application.Regions.Common;

using MediatR;

namespace Dealoviy.Application.Regions.Queries.GetRegion;

public record GetRegionsQuery() : IRequest<IEnumerable<RegionResult>>;

**Файл AcceptRequestCommand.cs**

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Requests.Commands.AcceptRequest;

public record AcceptRequestCommand(Guid RequestId, Guid UserContractorId)

: IRequest<ErrorOr<Unit>>;

**Файл AcceptRequestCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Common.Interfaces.Services;

using Dealoviy.Domain.Orders;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Requests.Commands.AcceptRequest;

public class AcceptRequestCommandHandler

: IRequestHandler<AcceptRequestCommand, ErrorOr<Unit>>

{

private readonly IRequestRepository \_requestRepository;

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

private readonly IOrderRepository \_orderRepository;

private readonly IDateTimeProvider \_dateTimeProvider;

public AcceptRequestCommandHandler(

IRequestRepository requestRepository,

IUserRepository userRepository,

IContractorProfileRepository contractorProfileRepository,

IOrderRepository orderRepository,

IDateTimeProvider dateTimeProvider)

{

\_requestRepository = requestRepository;

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

\_orderRepository = orderRepository;

\_dateTimeProvider = dateTimeProvider;

}

public async Task<ErrorOr<Unit>> Handle(AcceptRequestCommand request, CancellationToken cancellationToken)

{

var requestEntity = await \_requestRepository.GetByIdAsync(request.RequestId);

if (requestEntity is null)

return Error.NotFound("Request not found");

var contractor = await \_userRepository.GetUserByIdAsync(request.UserContractorId);

if(contractor.ContractorProfileId is null)

return Error.NotFound("Contractor not found");

var contractorProfile = await \_contractorProfileRepository.GetByIdAsync(contractor.ContractorProfileId.Value);

if (contractorProfile is null)

return Error.NotFound("Contractor profile not found");

var order = Order.Create(requestEntity,

\_dateTimeProvider.UtcNow);

await \_orderRepository.AddAsync(order);

await \_requestRepository.DeleteAsync(requestEntity);

return Unit.Value;

}

}

**Файл DeclineRequestCommand.cs**

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Requests.Commands.DeclineRequest;

public record DeclineRequestCommand(Guid RequestId, Guid UserContractorId)

: IRequest<ErrorOr<Unit>>;

**Файл DeclineRequestCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Domain.Requests;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Requests.Commands.DeclineRequest;

public class DeclineRequestCommandHandler

: IRequestHandler<DeclineRequestCommand, ErrorOr<Unit>>

{

private readonly IRequestRepository \_requestRepository;

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

public DeclineRequestCommandHandler(

IRequestRepository requestRepository,

IUserRepository userRepository,

IContractorProfileRepository contractorProfileRepository)

{

\_requestRepository = requestRepository;

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

}

public async Task<ErrorOr<Unit>> Handle(DeclineRequestCommand request, CancellationToken cancellationToken)

{

var requestEntity = await \_requestRepository.GetByIdAsync(request.RequestId);

if (requestEntity is null)

return Error.NotFound("Request not found");

var contractor = await \_userRepository.GetUserByIdAsync(request.UserContractorId);

if(contractor.ContractorProfileId is null)

return Error.NotFound("Contractor not found");

var contractorProfile = await \_contractorProfileRepository.GetByIdAsync(contractor.ContractorProfileId.Value);

if (contractorProfile is null)

return Error.NotFound("Contractor profile not found");

requestEntity.UpdateRequestStatus(RequestStatus.Declined);

await \_requestRepository.UpdateAsync(requestEntity);

return Unit.Value;

}

}

**Файл CreateRequestCommand.cs**

using Dealoviy.Application.Common.Models;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Requests.Commands.Create;

public record CreateRequestCommand(

Guid ServiceId,

Guid CustomerId,

string Description,

ContactInfoModel CustomerContactInfo,

int PaymentAmount) : IRequest<ErrorOr<Unit>>;

**Файл CreateRequestCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Common.Interfaces.Services;

using Dealoviy.Domain.Common.ContactInfo;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Requests;

using Dealoviy.Domain.Services;

using MediatR;

using ErrorOr;

using Mapster;

namespace Dealoviy.Application.Requests.Commands.Create;

public class CreateRequestCommandHandler

: IRequestHandler<CreateRequestCommand, ErrorOr<Unit>>

{

private readonly IRequestRepository \_requestRepository;

private readonly IServiceRepository \_serviceRepository;

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

private readonly IDateTimeProvider \_dateTimeProvider;

public CreateRequestCommandHandler(

IRequestRepository requestRepository,

IServiceRepository serviceRepository,

IUserRepository userRepository,

IContractorProfileRepository contractorProfileRepository,

IDateTimeProvider dateTimeProvider)

{

\_requestRepository = requestRepository;

\_serviceRepository = serviceRepository;

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

\_dateTimeProvider = dateTimeProvider;

}

public async Task<ErrorOr<Unit>> Handle(CreateRequestCommand request, CancellationToken cancellationToken)

{

if (await \_serviceRepository.GetByIdAsync(request.ServiceId)

is not Service service)

{

return Errors.ServiceNotFound;

}

if (await \_userRepository.GetUserByIdAsync(request.CustomerId) is null)

{

return Errors.UserNotFound;

}

var contractor = await \_contractorProfileRepository.GetByIdAsync(service.ContractorId);

var contractorContactInfo = contractor.ContactInfos

.FirstOrDefault(ci => ci.Type.ToString() == request.CustomerContactInfo.Type);

var requestResult = Request.Create(

request.CustomerId,

request.ServiceId,

request.Description,

request.PaymentAmount,

\_dateTimeProvider.UtcNow,

RequestStatus.Pending,

request.CustomerContactInfo.Adapt<ContactInfoCreateModel>(),

contractorContactInfo

);

if (requestResult.IsError)

{

return requestResult.Errors;

}

await \_requestRepository.AddAsync(requestResult.Value);

return Unit.Value;

}

}

**Файл DeleteRequestIfDeclinedCommand.cs**

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Requests.Commands.DeleteIfDeclined;

public record DeleteRequestIfDeclinedCommand(Guid RequestId, Guid UserCustomerId)

: IRequest<ErrorOr<Unit>>;

**Файл DeleteRequestIfDeclinedCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Domain.Requests;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Requests.Commands.DeleteIfDeclined;

public class DeleteRequestIfDeclinedCommandHandler

: IRequestHandler<DeleteRequestIfDeclinedCommand,

ErrorOr<Unit>>

{

private readonly IRequestRepository \_requestRepository;

private readonly IUserRepository \_userRepository;

public DeleteRequestIfDeclinedCommandHandler(

IRequestRepository requestRepository,

IUserRepository userRepository)

{

\_requestRepository = requestRepository;

\_userRepository = userRepository;

}

public async Task<ErrorOr<Unit>> Handle(DeleteRequestIfDeclinedCommand request, CancellationToken cancellationToken)

{

var requestEntity = await \_requestRepository.GetByIdAsync(request.RequestId);

if(requestEntity is null)

return Error.NotFound("Request not found");

var customer = await \_userRepository.GetUserByIdAsync(request.UserCustomerId);

if(customer is null)

return Error.NotFound("Customer not found");

if(requestEntity.RequestStatus == RequestStatus.Declined)

await \_requestRepository.DeleteAsync(requestEntity);

return Unit.Value;

}

}

**Файл GetRequestsForServiceQuery.cs**

using Dealoviy.Contracts.Requests;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Requests.Queries.GetRequestsForService;

public record GetRequestsForServiceQuery(Guid ServiceId)

: IRequest<ErrorOr<IEnumerable<ServiceRequestResponse>>>;

**Файл GetRequestsForServiceQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Contracts.Common;

using Dealoviy.Contracts.Requests;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Requests;

using Dealoviy.Domain.Users;

using MediatR;

using ErrorOr;

using MapsterMapper;

namespace Dealoviy.Application.Requests.Queries.GetRequestsForService;

public class GetRequestsForServiceQueryHandler

: IRequestHandler<GetRequestsForServiceQuery,

ErrorOr<IEnumerable<ServiceRequestResponse>>>

{

private readonly IServiceRepository \_serviceRepository;

private readonly IRequestRepository \_requestRepository;

private readonly IMapper \_mapper;

private readonly IUserRepository \_userRepository;

public GetRequestsForServiceQueryHandler(

IServiceRepository serviceRepository,

IRequestRepository requestRepository,

IMapper mapper,

IUserRepository userRepository)

{

\_serviceRepository = serviceRepository;

\_requestRepository = requestRepository;

\_mapper = mapper;

\_userRepository = userRepository;

}

public async Task<ErrorOr<IEnumerable<ServiceRequestResponse>>>

Handle(GetRequestsForServiceQuery request,

CancellationToken cancellationToken)

{

if (await \_serviceRepository.GetByIdAsync(request.ServiceId)

is not { } service)

{

return Errors.ServiceNotFound;

}

var requests = await \_requestRepository.GetByServiceIdAsync(service.Id);

requests = requests.Where(r => r.RequestStatus == RequestStatus.Pending);

var customersTasks = requests

.Select(r => \_userRepository.GetUserByIdAsync(r.CustomerId));

var customers = new List<User>();

foreach (var task in customersTasks)

{

customers.Add(await task);

}

return requests

.Select((r, i) => new ServiceRequestResponse(

r.Id,

r.Description,

r.PaymentAmount,

r.RequestDate,

r.RequestStatus.ToString(),

customers[i].GetDisplayName(),

new ContactInfoResponse(

r.CustomerContactInfo.Type.ToString(),

r.CustomerContactInfo.Value)

))

.OrderByDescending(r => r.RequestDate)

.ToList();

}

}

**Файл GetRequestsForUserQuery.cs**

using Dealoviy.Contracts.Requests;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Requests.Queries.GetRequestsForUser;

public record GetRequestsForUserQuery(Guid UserId)

: IRequest<ErrorOr<IEnumerable<UserRequestResponse>>>;

**Файл GetRequestsForUserQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Common.Interfaces.Services;

using Dealoviy.Contracts.Common;

using Dealoviy.Contracts.Requests;

using Dealoviy.Domain.ContractorProfiles;

using Dealoviy.Domain.Services;

using Dealoviy.Domain.Users;

using MediatR;

using ErrorOr;

using MapsterMapper;

namespace Dealoviy.Application.Requests.Queries.GetRequestsForUser;

public class GetRequestsForUserQueryHandler

: IRequestHandler<GetRequestsForUserQuery, ErrorOr<IEnumerable<UserRequestResponse>>>

{

private readonly IServiceRepository \_serviceRepository;

private readonly IRequestRepository \_requestRepository;

private readonly IMapper \_mapper;

private readonly IDateTimeProvider \_dateTimeProvider;

private readonly IUserRepository \_userRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

public GetRequestsForUserQueryHandler(

IServiceRepository serviceRepository,

IRequestRepository requestRepository,

IMapper mapper,

IDateTimeProvider dateTimeProvider,

IUserRepository userRepository,

IContractorProfileRepository contractorProfileRepository)

{

\_serviceRepository = serviceRepository;

\_requestRepository = requestRepository;

\_mapper = mapper;

\_dateTimeProvider = dateTimeProvider;

\_userRepository = userRepository;

\_contractorProfileRepository = contractorProfileRepository;

}

public async Task<ErrorOr<IEnumerable<UserRequestResponse>>> Handle(GetRequestsForUserQuery request, CancellationToken cancellationToken)

{

var requests = await \_requestRepository.GetByCustomerIdAsync(request.UserId);

var servicesTasks = requests

.Select(r => \_serviceRepository.GetByIdAsync(r.ServiceId));

var services = new List<Service>();

foreach (var task in servicesTasks)

{

services.Add(await task);

}

var contractorTasks = services

.Select(s => \_contractorProfileRepository.GetByIdAsync(s.ContractorId));

var contractors = new List<ContractorProfile>();

foreach (var task in contractorTasks)

{

contractors.Add(await task);

}

var usersTasks = contractors

.Select(c => \_userRepository.GetByContractorIdAsync(c.Id));

var users = new List<User>();

foreach (var task in usersTasks)

{

users.Add(await task);

}

return requests

.Select((r, i) => new UserRequestResponse(

r.Id,

r.Description,

r.PaymentAmount,

r.RequestDate,

r.RequestStatus.ToString(),

users[i].GetDisplayName(),

services[i].Id,

services[i].Name,

new ContactInfoResponse(

r.ContractorContactInfo.Type.ToString(),

r.ContractorContactInfo.Value)))

.OrderByDescending(r => r.RequestDate)

.ToList();

}

}

**Файл AddReviewOnServiceCommand.cs**

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Reviews.Commands.AddReviewOnService;

public record AddReviewOnServiceCommand(

Guid ServiceId,

Guid UserId,

int Rating,

string Text) : IRequest<ErrorOr<Unit>>;

**Файл AddReviewOnServiceCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Common.Interfaces.Services;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Reviews;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Reviews.Commands.AddReviewOnService;

public class AddReviewOnServiceCommandHandler

: IRequestHandler<AddReviewOnServiceCommand, ErrorOr<Unit>>

{

private readonly IReviewRepository \_reviewRepository;

private readonly IUserRepository \_userRepository;

private readonly IServiceRepository \_serviceRepository;

private readonly IDateTimeProvider \_dateTimeProvider;

public AddReviewOnServiceCommandHandler(

IReviewRepository reviewRepository,

IUserRepository userRepository,

IServiceRepository serviceRepository,

IDateTimeProvider dateTimeProvider)

{

\_reviewRepository = reviewRepository;

\_userRepository = userRepository;

\_serviceRepository = serviceRepository;

\_dateTimeProvider = dateTimeProvider;

}

public async Task<ErrorOr<Unit>> Handle(AddReviewOnServiceCommand request, CancellationToken cancellationToken)

{

if (await \_serviceRepository.GetByIdAsync(request.ServiceId)

is not { } service)

{

return Errors.ServiceNotFound;

}

if (await \_userRepository.GetUserByIdAsync(request.UserId)

is null)

{

return Errors.UserNotFound;

}

var review = Review.Create(

request.ServiceId,

request.UserId,

request.Rating,

request.Text,

\_dateTimeProvider.UtcNow);

await \_reviewRepository.AddReview(review);

service.AverageRating.AddRating(review.Rating);

await \_serviceRepository.UpdateAsync(service);

return Unit.Value;

}

}

**Файл AddReviewOnServiceCommandValidator.cs**

using FluentValidation;

namespace Dealoviy.Application.Reviews.Commands.AddReviewOnService;

public class AddReviewOnServiceCommandValidator

: AbstractValidator<AddReviewOnServiceCommand>

{

public AddReviewOnServiceCommandValidator()

{

RuleFor(c => c.ServiceId)

.NotEmpty()

.WithErrorCode("Review.ServiceId.Required")

.WithMessage("Service id is required.");

RuleFor(c => c.UserId)

.NotEmpty()

.WithErrorCode("Review.UserId.Required")

.WithMessage("User id is required.");

RuleFor(c => c.Rating)

.InclusiveBetween(1, 5)

.WithErrorCode("Review.Rating.Invalid")

.WithMessage("Rating must be between 1 and 5.");

RuleFor(c => c.Text)

.NotEmpty()

.WithErrorCode("Review.Text.Required")

.WithMessage("Text is required.");

RuleFor(c => c.Text)

.MaximumLength(255)

.WithErrorCode("Review.Text.TooLong")

.WithMessage("Text must be less than 255 characters.");

}

}

**Файл GetReviewsForServiceQuery.cs**

using Dealoviy.Contracts.Reviews;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Reviews.Queries.GetReviewsForService;

public record GetReviewsForServiceQuery(Guid ServiceId)

: IRequest<ErrorOr<IEnumerable<ReviewResponse>>>;

**Файл GetReviewsForServiceQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Contracts.Reviews;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Users;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Reviews.Queries.GetReviewsForService;

public class GetReviewsForServiceQueryHandler

: IRequestHandler<GetReviewsForServiceQuery,

ErrorOr<IEnumerable<ReviewResponse>>>

{

private readonly IReviewRepository \_reviewRepository;

private readonly IServiceRepository \_serviceRepository;

private readonly IUserRepository \_userRepository;

public GetReviewsForServiceQueryHandler(

IReviewRepository reviewRepository,

IServiceRepository serviceRepository,

IUserRepository userRepository)

{

\_reviewRepository = reviewRepository;

\_serviceRepository = serviceRepository;

\_userRepository = userRepository;

}

public async Task<ErrorOr<IEnumerable<ReviewResponse>>> Handle(

GetReviewsForServiceQuery request,

CancellationToken cancellationToken)

{

if (await \_serviceRepository.GetByIdAsync(request.ServiceId)

is null)

{

return Errors.ServiceNotFound;

}

var reviews = await \_reviewRepository.GetReviewsByServiceId(request.ServiceId);

var customerTasks = reviews

.Select(r => \_userRepository.GetUserByIdAsync(r.UserId));

var customers = new List<User>();

foreach (var task in customerTasks)

{

customers.Add(await task);

}

var reviewResponses = reviews

.Zip(customers, (review, customer) => new ReviewResponse(

review.Id,

customer.GetDisplayName(),

review.Text,

review.CreatedAt,

review.Rating))

.OrderByDescending(r => r.ReviewDate)

.ToList();

return reviewResponses;

}

}

**Файл CreateServiceCommand.cs**

using Dealoviy.Application.Services.Common.Interfaces;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Services.Commands.Create;

public record CreateServiceCommand(

Guid UserId,

Guid CityId,

string Name,

string Description,

int LowerPriceBound,

int UpperPriceBound)

: IServiceCommand,

IRequest<ErrorOr<Unit>>;

**Файл CreateServiceCommandHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.ContractorProfiles;

using Dealoviy.Domain.Services;

using Dealoviy.Domain.Users;

using ErrorOr;

using MediatR;

namespace Dealoviy.Application.Services.Commands.Create;

public class CreateServiceCommandHandler

: IRequestHandler<CreateServiceCommand, ErrorOr<Unit>>

{

private readonly IUserRepository \_userRepository;

private readonly IServiceRepository \_serviceRepository;

private readonly IContractorProfileRepository \_contractorProfileRepository;

private readonly ICityRepository \_cityRepository;

public CreateServiceCommandHandler(

IUserRepository userRepository,

IServiceRepository serviceRepository,

IContractorProfileRepository contractorProfileRepository,

ICityRepository cityRepository)

{

\_userRepository = userRepository;

\_serviceRepository = serviceRepository;

\_contractorProfileRepository = contractorProfileRepository;

\_cityRepository = cityRepository;

}

public async Task<ErrorOr<Unit>> Handle(CreateServiceCommand request, CancellationToken cancellationToken)

{

if (await \_userRepository.GetUserByIdAsync(request.UserId) is not User user)

{

return Errors.UserNotFound;

}

if (user.ContractorProfileId is null)

{

return Errors.UserIsNotAContractor;

}

if (await \_contractorProfileRepository.GetByIdAsync(user.ContractorProfileId.Value)

is not ContractorProfile profile)

{

return Errors.ContractorProfileNotFound;

}

if (await \_cityRepository.GetCityByIdAsync(request.CityId)

is null)

{

return Errors.CityNotFound;

}

var serviceResult = Service.Create(

profile.Id,

request.CityId,

request.Name,

request.Description,

request.LowerPriceBound,

request.UpperPriceBound);

if (serviceResult.IsError)

{

return serviceResult.Errors;

}

await \_serviceRepository.AddAsync(serviceResult.Value);

return Unit.Value;

}

}

**Файл CreateServiceCommandValidator.cs**

using Dealoviy.Application.Services.Common.Validators;

namespace Dealoviy.Application.Services.Commands.Create;

public class CreateServiceCommandValidator

: ServiceCommandBaseValidator<CreateServiceCommand>

{

public CreateServiceCommandValidator() : base()

{

}

}

**Файл IServiceCommand.cs**

namespace Dealoviy.Application.Services.Common.Interfaces;

public interface IServiceCommand

{

Guid CityId { get; }

string Name { get; }

string Description { get; }

int LowerPriceBound { get; }

int UpperPriceBound { get; }

}

**Файл ServiceCommandBaseValidator.cs**

using Dealoviy.Application.Services.Common.Interfaces;

using FluentValidation;

namespace Dealoviy.Application.Services.Common.Validators;

public abstract class ServiceCommandBaseValidator<TServiceCommand>

: AbstractValidator<TServiceCommand>

where TServiceCommand : IServiceCommand

{

protected ServiceCommandBaseValidator()

{

RuleFor(x => x.Name)

.NotEmpty()

.WithErrorCode("Validation.Service.Name.Required")

.WithMessage("Service name is required");

RuleFor(x => x.Name)

.MaximumLength(100)

.WithErrorCode("Validation.Service.Name.MaxLength")

.WithMessage("Service name cannot be longer than 100 characters");

RuleFor(x => x.Description)

.NotEmpty()

.WithErrorCode("Validation.Service.Description.Required")

.WithMessage("Service description is required");

RuleFor(x => x.Description)

.MaximumLength(255)

.WithErrorCode("Validation.Service.Description.MaxLength")

.WithMessage("Service description cannot be longer than 255 characters");

}

}

**Файл ServiceResult.cs**

namespace Dealoviy.Application.Services.Queries.Common;

public record ServiceResult(

Guid ServiceId,

Guid ContractorId,

string Name,

string CityName,

string Description,

int LowerPriceBound,

int UpperPriceBound,

double AverageRating,

int ReviewsCount);

**Файл GetServiceByIdQuery.cs**

using Dealoviy.Application.Services.Queries.Common;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Services.Queries.GetById;

public record GetServiceByIdQuery(Guid Id)

: IRequest<ErrorOr<ServiceResult>>;

**Файл GetServiceByIdQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Services.Queries.Common;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Common.Location;

using Dealoviy.Domain.Services;

using MediatR;

using ErrorOr;

using MapsterMapper;

namespace Dealoviy.Application.Services.Queries.GetById;

public class GetServiceByIdQueryHandler

: IRequestHandler<GetServiceByIdQuery, ErrorOr<ServiceResult>>

{

private readonly IServiceRepository \_serviceRepository;

private readonly ICityRepository \_cityRepository;

private readonly IMapper \_mapper;

public GetServiceByIdQueryHandler(

IServiceRepository serviceRepository,

ICityRepository cityRepository,

IMapper mapper)

{

\_serviceRepository = serviceRepository;

\_cityRepository = cityRepository;

\_mapper = mapper;

}

public async Task<ErrorOr<ServiceResult>> Handle(GetServiceByIdQuery request, CancellationToken cancellationToken)

{

if (await \_serviceRepository.GetByIdAsync(request.Id)

is not Service service)

{

return Errors.ServiceNotFound;

}

if (await \_cityRepository.GetCityByIdAsync(service.CityId)

is not City city)

{

return Errors.CityNotFound;

}

return \_mapper.Map<ServiceResult>((service, city.Name));

}

}

**Файл GetByKeywordAndCityQuery.cs**

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Services.Queries.GetByKeywordAndCity;

public record GetByKeywordAndCityQuery(

string Keyword,

Guid CityId) : IRequest<ErrorOr<ServiceSearchResult>>;

**Файл GetByKeywordAndCityQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Application.Services.Queries.Common;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Common.Location;

using MediatR;

using ErrorOr;

using MapsterMapper;

namespace Dealoviy.Application.Services.Queries.GetByKeywordAndCity;

public class GetByKeywordAndCityQueryHandler

: IRequestHandler<GetByKeywordAndCityQuery, ErrorOr<ServiceSearchResult>>

{

private readonly IServiceRepository \_serviceRepository;

private readonly ICityRepository \_cityRepository;

private readonly IMapper \_mapper;

public GetByKeywordAndCityQueryHandler(

IServiceRepository serviceRepository,

ICityRepository cityRepository,

IMapper mapper)

{

\_serviceRepository = serviceRepository;

\_cityRepository = cityRepository;

\_mapper = mapper;

}

public async Task<ErrorOr<ServiceSearchResult>> Handle(

GetByKeywordAndCityQuery request,

CancellationToken cancellationToken)

{

if(await \_cityRepository.GetCityByIdAsync(request.CityId) is not City city)

{

return Errors.CityNotFound;

}

var services = await \_serviceRepository

.GetByKeywordAndCityAsync(request.Keyword, request.CityId);

var serviceResults = services.Select(service =>

\_mapper.Map<ServiceResult>((service, city.Name)))

.OrderByDescending(s => s.AverageRating)

.ToList();

return new ServiceSearchResult(

serviceResults,

serviceResults.Count,

request.Keyword,

city.Name);

}

}

**Файл ServiceSearchResult.cs**

using Dealoviy.Application.Services.Queries.Common;

namespace Dealoviy.Application.Services.Queries.GetByKeywordAndCity;

public record ServiceSearchResult(

IEnumerable<ServiceResult> Services,

int TotalCount,

string Keyword,

string CityName);

**Файл GetOwnServicesQuery.cs**

using Dealoviy.Contracts.Services;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Services.Queries.GetOwnServices;

public record GetOwnServicesQuery(Guid UserId) : IRequest<ErrorOr<IEnumerable<ServiceResponse>>>;

**Файл GetOwnServicesQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Contracts.Services;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Common.Location;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Services.Queries.GetOwnServices;

public class GetOwnServicesQueryHandler

: IRequestHandler<GetOwnServicesQuery, ErrorOr<IEnumerable<ServiceResponse>>>

{

private readonly IUserRepository \_userRepository;

private readonly IServiceRepository \_serviceRepository;

private readonly ICityRepository \_cityRepository;

public GetOwnServicesQueryHandler(

IUserRepository userRepository,

IServiceRepository serviceRepository,

ICityRepository cityRepository)

{

\_userRepository = userRepository;

\_serviceRepository = serviceRepository;

\_cityRepository = cityRepository;

}

public async Task<ErrorOr<IEnumerable<ServiceResponse>>> Handle(GetOwnServicesQuery request, CancellationToken cancellationToken)

{

if (await \_userRepository.GetUserByIdAsync(request.UserId)

is not { } user)

{

return Errors.UserNotFound;

}

if (user.ContractorProfileId is null)

{

return Errors.UserIsNotAContractor;

}

var services = await \_serviceRepository.GetByContractorIdAsync(user.ContractorProfileId.Value);

var citiesTasks = services.Select(service => \_cityRepository.GetCityByIdAsync(service.CityId));

var cities = new List<City>();

foreach (var task in citiesTasks)

{

cities.Add(await task);

}

var result = services.Zip(cities, (service, city) => new ServiceResponse(

service.Id,

service.ContractorId,

service.Name,

city.Name,

service.Description,

service.PriceRange.Lower,

service.PriceRange.Upper,

service.AverageRating.Value,

service.AverageRating.Count))

.ToList();

return result;

}

}

**Файл GetServicesWithStatsQuery.cs**

using Dealoviy.Contracts.Services;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Services.Queries.GetServicesWithStats;

public record GetServicesWithStatsQuery(Guid UserId)

: IRequest<ErrorOr<IEnumerable<ServiceTaskStatResponse>>>;

**Файл GetServicesWithStatsQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Contracts.Services;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Orders;

using Dealoviy.Domain.Requests;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Services.Queries.GetServicesWithStats;

public class GetServicesWithStatsQueryHandler

: IRequestHandler<GetServicesWithStatsQuery, ErrorOr<IEnumerable<ServiceTaskStatResponse>>>

{

private readonly IUserRepository \_userRepository;

private readonly IServiceRepository \_serviceRepository;

private readonly IRequestRepository \_requestRepository;

private readonly IOrderRepository \_orderRepository;

public GetServicesWithStatsQueryHandler(

IUserRepository userRepository,

IServiceRepository serviceRepository,

IRequestRepository requestRepository,

IOrderRepository orderRepository)

{

\_userRepository = userRepository;

\_serviceRepository = serviceRepository;

\_requestRepository = requestRepository;

\_orderRepository = orderRepository;

}

public async Task<ErrorOr<IEnumerable<ServiceTaskStatResponse>>> Handle(GetServicesWithStatsQuery request, CancellationToken cancellationToken)

{

if (await \_userRepository.GetUserByIdAsync(request.UserId) is not { } user)

{

return Errors.UserNotFound;

}

if(user.ContractorProfileId is null)

{

return Errors.UserIsNotAContractor;

}

var services = await \_serviceRepository.GetByContractorIdAsync(user.ContractorProfileId.Value);

var serviceTaskStats = new List<ServiceTaskStatResponse>();

foreach (var service in services)

{

var requests = await \_requestRepository.GetByServiceIdAsync(service.Id);

var pendingRequestsCount = requests

.Count(r => r.RequestStatus == RequestStatus.Pending);

var orders = await \_orderRepository.GetByServiceIdAsync(service.Id);

var notFinishedOrdersCount = orders

.Count(o => o.OrderStatus != OrderStatus.Finished);

var serviceTaskStat = new ServiceTaskStatResponse(

service.Id,

service.Name,

pendingRequestsCount,

notFinishedOrdersCount);

serviceTaskStats.Add(serviceTaskStat);

}

return serviceTaskStats;

}

}

**Файл GetUserByIdQuery.cs**

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Users.Queries.GetById;

public record GetUserByIdQuery(Guid UserId)

: IRequest<ErrorOr<UserResult>>;

**Файл GetUserByIdQueryHandler.cs**

using Dealoviy.Application.Common.Interfaces.Persistence;

using Dealoviy.Domain.Common.Errors;

using Dealoviy.Domain.Users;

using MediatR;

using ErrorOr;

namespace Dealoviy.Application.Users.Queries.GetById;

public class GetUserByIdQueryHandler

: IRequestHandler<GetUserByIdQuery, ErrorOr<UserResult>>

{

private readonly IUserRepository \_userRepository;

public GetUserByIdQueryHandler(IUserRepository userRepository)

{

\_userRepository = userRepository;

}

public async Task<ErrorOr<UserResult>> Handle(GetUserByIdQuery request, CancellationToken cancellationToken)

{

if (await \_userRepository.GetUserByIdAsync(request.UserId)

is not User user)

{

return Errors.UserNotFound;

}

return new UserResult(

user.Id,

user.Username,

user.DisplayName,

user.ContractorProfileId);

}

}

**Файл UserResult.cs**

namespace Dealoviy.Application.Users.Queries.GetById;

public record UserResult(Guid UserId, string Username, string? DisplayName, Guid? ContractorProfileId);

using MediatR;

using ErrorOr;

using FluentValidation;

**Файл ValidationBehavior.cs**

namespace Dealoviy.Application.Common.Behaviors;

public class ValidationBehavior<TRequest, TResponse> :

IPipelineBehavior<TRequest, TResponse>

where TRequest : IRequest<TResponse>

where TResponse : IErrorOr

{

private readonly IValidator<TRequest>? \_validator;

public ValidationBehavior(IValidator<TRequest>? validator = null)

{

\_validator = validator;

}

public async Task<TResponse> Handle(

TRequest request,

RequestHandlerDelegate<TResponse> next,

CancellationToken cancellationToken)

{

if (\_validator is null)

{

return await next();

}

var validationResult = await \_validator.ValidateAsync(request, cancellationToken);

if (validationResult.IsValid)

{

return await next();

}

var errors = validationResult.Errors

.Select(e => Error.Validation(

code: e.ErrorCode,

description: e.ErrorMessage))

.ToList();

return (dynamic)errors;

}

}

**Файл IJwtTokenGenerator.cs**

using Dealoviy.Domain.Users;

namespace Dealoviy.Application.Common.Interfaces.Authentication;

public interface IJwtTokenGenerator

{

string GenerateToken(User user);

}

**Файл ICityRepository.cs**

using Dealoviy.Domain.Common.Location;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface ICityRepository

{

Task<IEnumerable<City>> GetCitiesByRegionIdAsync(Guid regionId);

Task<City?> GetCityByIdAsync(Guid id);

}

**Файл IContractorProfileRepository.cs**

using Dealoviy.Domain.ContractorProfiles;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface IContractorProfileRepository

{

Task<ContractorProfile?> GetByIdAsync(Guid id);

Task AddAsync(ContractorProfile profile);

Task UpdateAsync(ContractorProfile profile);

}

**Файл IOrderRepository.cs**

using Dealoviy.Domain.Orders;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface IOrderRepository

{

Task AddAsync(Order request);

Task<IEnumerable<Order>> GetByServiceIdAsync(Guid serviceId);

Task<IEnumerable<Order>> GetByCustomerIdAsync(Guid customerId);

Task<Order?> GetByIdAsync(Guid requestId);

Task UpdateAsync(Order request);

}

**Файл IRegionRepository.cs**

using Dealoviy.Domain.Common.Location;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface IRegionRepository

{

Task<IEnumerable<Region>> GetAllRegionsAsync();

Task<Region?> GetRegionByIdAsync(Guid id);

}

**Файл IRequestRepository.cs**

using Dealoviy.Domain.Requests;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface IRequestRepository

{

Task AddAsync(Request request);

Task<IEnumerable<Request>> GetByServiceIdAsync(Guid serviceId);

Task<IEnumerable<Request>> GetByCustomerIdAsync(Guid customerId);

Task<Request?> GetByIdAsync(Guid requestId);

Task UpdateAsync(Request request);

Task DeleteAsync(Request request);

}

**Файл IReviewRepository.cs**

using Dealoviy.Domain.Reviews;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface IReviewRepository

{

Task AddReview(Review review);

Task<IEnumerable<Review>> GetReviewsByServiceId(Guid serviceId);

}

**Файл IServiceRepository.cs**

using Dealoviy.Domain.Services;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface IServiceRepository

{

Task<Service?> GetByIdAsync(Guid id);

Task AddAsync(Service service);

Task UpdateAsync(Service service);

Task<IEnumerable<Service>> GetByKeywordAndCityAsync(string keyword, Guid cityId);

Task<IEnumerable<Service>> GetByContractorIdAsync(Guid contractorId);

}

**Файл IUserRepository.cs**

using Dealoviy.Domain.Users;

namespace Dealoviy.Application.Common.Interfaces.Persistence;

public interface IUserRepository

{

Task<User?> GetUserByIdAsync(Guid id);

Task<User?> GetUserByUsernameAsync(string username);

Task<User?> GetByContractorIdAsync(Guid contractorId);

Task AddAsync(User user);

Task UpdateAsync(User user);

}