Приложение А

Листинг исходного кода

CalendarEvent.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace DAL.Entities
{
   public class CalendarEvent : IComparable<CalendarEvent>, IComparable
   {
      [Key]
      [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
      public string Id { get; set; }

      public string OwnerId { get; set; }

      public DateTime EventDate { get; set; }

      public TimeSpan Duration { get; set; }

      public string Comment { get; set; }

      public ICollection<Tag> Tags { get; set; }

      public int CompareTo(object obj)
      {
         if (ReferenceEquals(null, obj)) return 1;
         if (ReferenceEquals(this, obj)) return 0;
         return obj is CalendarEvent other
            ? CompareTo(other)
            : throw new ArgumentException($"Object must be of type {nameof(CalendarEvent)}");
      }

      public int CompareTo(CalendarEvent other)
      {
         if (ReferenceEquals(this, other)) return 0;
         if (ReferenceEquals(null, other)) return 1;
         return string.Compare(Id, other.Id, StringComparison.Ordinal);
      }
   }
}
```

Group.cs

```
using System;
using System.Collections.Generic;

namespace DAL.Entities
{
   public class Group : IComparable<Group>, IComparable
   {
```

```
        public int Id { get; set; }

        public string CommandOwner { get; set; }

        public string CommandName { get; set; }

        public ICollection<User> GroupParticipants { get; set; }

        public int CompareTo(object obj)
        {
            if (ReferenceEquals(null, obj)) return 1;
            if (ReferenceEquals(this, obj)) return 0;
            return obj is Group other
                ? CompareTo(other)
                : throw new ArgumentException($"Object must be of type {nameof(Group)}");
        }

        public int CompareTo(Group other)
        {
            if (ReferenceEquals(this, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            return Id.CompareTo(other.Id);
        }
    }
}
```

Message.cs

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace DAL.Entities
{
    public class Message : IComparable<Message>, IComparable
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public string Id { get; set; }

        public string Sender { get; set; }

        public string Recipient { get; set; }

        [DataType(DataType.Text)] public DateTime Sended { get; set; }

        public string MessageBody { get; set; }

        public int CompareTo(object obj)
        {
            if (ReferenceEquals(null, obj)) return 1;
            if (ReferenceEquals(this, obj)) return 0;
            return obj is Message other
                ? CompareTo(other)
                : throw new ArgumentException($"Object must be of type {nameof(Message)}");
        }

        public int CompareTo(Message other)
```

```
        {
            if (ReferenceEquals(this, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            return string.Compare(Id, other.Id, StringComparison.Ordinal);
        }
    }
}
```

Project.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace DAL.Entities
{
    public class Project : IComparable<Project>, IComparable
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        public ICollection<User> Participants { get; set; }

        public ICollection<ProjectTask> Tasks { get; set; }

        public string ProjectOwner { get; set; }

        public string Name { get; set; }

        public DateTime ProjectStart { get; set; }

        public DateTime ProjectEnd { get; set; }

        public int CompareTo(object obj)
        {
            if (ReferenceEquals(null, obj)) return 1;
            if (ReferenceEquals(this, obj)) return 0;
            return obj is Project other
                ? CompareTo(other)
                : throw new ArgumentException($"Object must be of type {nameof(Project)}");
        }

        public int CompareTo(Project other)
        {
            if (ReferenceEquals(this, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            return Id.CompareTo(other.Id);
        }
    }
}
```

ProjectTask.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
```

```
namespace DAL.Entities
{
   public class ProjectTask : IComparable<ProjectTask>, IComparable
   {
      [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
      public int Id { get; set; }

      public int ProjectId { get; set; }

      public ICollection<User> Participants { get; set; }

      public string TaskName { get; set; }

      public ICollection<Tag> Tags { get; set; }

      public TaskPriority Priority { get; set; }

      public DateTime TaskStart { get; set; }

      public DateTime TaskEnd { get; set; }

      public int CompareTo(object obj)
      {
         if (ReferenceEquals(null, obj)) return 1;
         if (ReferenceEquals(this, obj)) return 0;
         return obj is ProjectTask other
            ? CompareTo(other)
            : throw new ArgumentException($"Object must be of type {nameof(ProjectTask)}");
      }

      public int CompareTo(ProjectTask other)
      {
         if (ReferenceEquals(this, other)) return 0;
         if (ReferenceEquals(null, other)) return 1;
         return Id.CompareTo(other.Id);
      }
   }
}
```

Tag.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Drawing;

namespace DAL.Entities
{
   public class Tag : IComparable<Tag>, IComparable
   {
      [Key] public int Id { get; set; }

      public string Name { get; set; }

      public string OwnerId { get; set; }

      public KnownColor TagColor { get; set; } = KnownColor.Silver;
```

```
        public ICollection<CalendarEvent> CalendarEvents { get; set; }

        public ICollection<ProjectTask> ProjectTasks { get; set; }

        public int CompareTo(object obj)
        {
            if (ReferenceEquals(null, obj)) return 1;
            if (ReferenceEquals(this, obj)) return 0;
            return obj is Tag other
                ? CompareTo(other)
                : throw new ArgumentException($"Object must be of type {nameof(Tag)}");
        }

        public int CompareTo(Tag other)
        {
            if (ReferenceEquals(this, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            return Id.CompareTo(other.Id);
        }
    }
}
```

TaskPriority.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DAL.Entities
{
    public enum TaskPriority
    {
        Lowest,
        Low,
        Medium,
        High,
        Highest
    }
}
```

User.cs

```
#nullable enable
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace DAL.Entities
{
    public class User : IComparable<User>, IComparable
    {
        [Key] public string Id { get; set; }

        public string FullName { get; set; }
```

```
        [DataType(DataType.Text)] public DateTime Birthday { get; set; }

        public ICollection<Group> Groups { get; set; }

        public ICollection<Project> InProjects { get; set; }

        public ICollection<Project> UserProjects { get; set; }

        public ICollection<ProjectTask> Tasks { get; set; }

        public int CompareTo(object? obj)
        {
            if (ReferenceEquals(null, obj)) return 1;
            if (ReferenceEquals(this, obj)) return 0;
            return obj is User other
                ? CompareTo(other)
                : throw new ArgumentException($"Object must be of type {nameof(User)}");
        }

        public int CompareTo(User? other)
        {
            if (ReferenceEquals(this, other)) return 0;
            if (ReferenceEquals(null, other)) return 1;
            return string.Compare(Id, other.Id, StringComparison.Ordinal);
        }
    }
}
```

DataContext.cs

```
using DAL.Entities;
using Microsoft.EntityFrameworkCore;

namespace DAL.Repositories.EFCore
{
    public class DataContext : DbContext
    {
        public DataContext(DbContextOptions<DataContext> options) : base(options)
        {
            Database.EnsureCreated();
            ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
        }

        public DbSet<CalendarEvent> CalendarEvents { get; set; }

        public DbSet<Group> Groups { get; set; }

        public DbSet<Message> Messages { get; set; }

        public DbSet<Project> Projects { get; set; }

        public DbSet<ProjectTask> ProjectTasks { get; set; }

        public DbSet<Tag> Tags { get; set; }

        public DbSet<User> Users { get; set; }
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    #region MessaagesConfig

    modelBuilder.Entity<Message>()
        .HasOne<User>()
        .WithMany()
        .HasForeignKey(e => e.Sender);
    modelBuilder.Entity<Message>()
        .HasOne<User>()
        .WithMany()
        .HasForeignKey(e => e.Recipient);

    #endregion

    #region CalendarEventsConfig

    modelBuilder.Entity<CalendarEvent>()
        .HasOne<User>()
        .WithMany()
        .HasForeignKey(e => e.OwnerId)
        .OnDelete(DeleteBehavior.Cascade);
    // modelBuilder.Entity<CalendarEvent>()
    //     .HasMany(e => e.Tags)
    //     .WithMany(e => e.CalendarEvents)
    //     .UsingEntity(e => e.ToTable("EventTag"));

    #endregion

    #region GroupsConfig

    modelBuilder.Entity<Group>()
        .HasMany<User>(e => e.GroupParticipants)
        .WithMany(e => e.Groups)
        .UsingEntity(e => e.ToTable("GroupUser"));
    modelBuilder.Entity<Group>()
        .HasOne<User>()
        .WithMany()
        .HasForeignKey(e => e.CommandOwner);

    #endregion

    #region ProjectsConfig

    modelBuilder.Entity<Project>()
        .HasMany(e => e.Participants)
        .WithMany(e => e.InProjects)
        .UsingEntity(e => e.ToTable("ProjectUser"));
    modelBuilder.Entity<Project>()
        .HasOne<User>()
        .WithMany(e => e.UserProjects)
        .HasForeignKey(e => e.ProjectOwner)
        .OnDelete(DeleteBehavior.Cascade);
    modelBuilder.Entity<Project>()
        .HasMany<ProjectTask>(e => e.Tasks)
        .WithOne()
```

```
    .HasForeignKey(e => e.ProjectId);

#endregion

#region ProjectTaskConfig

modelBuilder.Entity<ProjectTask>()
    .HasMany(e => e.Participants)
    .WithMany(e => e.Tasks)
    .UsingEntity(e => e.ToTable("ProjectTaskUser"));
modelBuilder.Entity<ProjectTask>()
    .HasMany(e => e.Tags)
    .WithMany(e => e.ProjectTasks)
    .UsingEntity(e => e.ToTable("TagProjectTask"));
modelBuilder.Entity<ProjectTask>()
    .HasOne<Project>()
    .WithMany(e => e.Tasks)
    .HasForeignKey(e => e.ProjectId)
    .OnDelete(DeleteBehavior.Cascade);

#endregion

#region TagsConfig

modelBuilder.Entity<Tag>()
    .HasMany(e => e.ProjectTasks)
    .WithMany(e => e.Tags)
    .UsingEntity(e => e.ToTable("ProjectTaskTag"));
modelBuilder.Entity<Tag>()
    .HasMany(e => e.CalendarEvents)
    .WithMany(e => e.Tags)
    .UsingEntity(e => e.ToTable("CalendarEventTag"));
modelBuilder.Entity<Tag>()
    .HasOne<User>()
    .WithMany()
    .HasForeignKey(e => e.OwnerId);

#endregion

#region UsersConfig

modelBuilder.Entity<User>()
    .HasMany(e => e.Groups)
    .WithMany(e => e.GroupParticipants)
    .UsingEntity(e => e.ToTable("UserGroup"));
modelBuilder.Entity<User>()
    .HasMany(e => e.Tasks)
    .WithMany(e => e.Participants)
    .UsingEntity(e => e.ToTable("UserProjectTask"));
modelBuilder.Entity<User>()
    .HasMany(e => e.InProjects)
    .WithMany(e => e.Participants)
    .UsingEntity(e => e.ToTable("UserProject"));
modelBuilder.Entity<User>()
    .HasMany(e => e.UserProjects)
    .WithOne()
    .HasForeignKey(e => e.ProjectOwner);
```

```
        #endregion
    }
  }
}
```

GroupsRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DAL.Repositories.EFCore
{
  public class GroupsRepository : IGroupsRepository
  {
    private readonly DataContext _context;
    private readonly ILogger<GroupsRepository> _logger;

    public GroupsRepository(ILogger<GroupsRepository> logger, DataContext context)
    {
      _logger = logger;
      _context = context;
    }

    public async Task<Group> Create(Group value)
    {
      var res = await _context.Groups.AddAsync(value);
      var saveRes = await _context.SaveChangesAsync();
      return res?.Entity;
    }

    public async Task<Group> Update(Group value)
    {
      var res = _context.Groups.Update(value);
      var saveRes = await _context.SaveChangesAsync();
      _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
      return res?.Entity;
    }

    public async Task<bool> Delete(object key)
    {
      var res = _context.Groups.Remove(GetById(key).Result);
      var saveRes = await _context.SaveChangesAsync();
      _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
      return true;
    }

    public async Task<ICollection<Group>> ReadAll()
    {
      return await _context.Groups.ToListAsync();
    }
```

```
        public async Task<ICollection<Group>> ReadAllInclude()
        {
            return await _context.Groups.Include(e => e.GroupParticipants).ToListAsync();
        }

        public async Task<ICollection<Group>> GetBySelector(Func<Group, bool> selector)
        {
            return await Task.Run(() => _context
                .Groups
                .Include(e => e.GroupParticipants)
                .Where(selector)
                .ToList());
        }

        public async Task<Group> GetById(object id)
        {
            return await _context.Groups.Include(e => e.GroupParticipants).FirstOrDefaultAsync(e =>
e.Id.Equals(id));
        }
    }
}
```

CalendarEvents.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DAL.Repositories.EFCore
{
    public class CalendarEventsRepository : ICalendarEventsRepository
    {
        private readonly DataContext _context;
        private readonly ILogger<CalendarEventsRepository> _logger;

        public CalendarEventsRepository(DataContext context,
            ILogger<CalendarEventsRepository> logger)
        {
            _context = context;
            _logger = logger;
        }

        public async Task<CalendarEvent> Create(CalendarEvent value)
        {
            var res = await _context.CalendarEvents.AddAsync(value);
            var saveRes = await _context.SaveChangesAsync();
            return res?.Entity;
        }

        public async Task<CalendarEvent> Update(CalendarEvent value)
        {
```

```
            var res = _context.CalendarEvents.Update(value);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return res?.Entity;
        }

        public async Task<bool> Delete(object key)
        {
            var res = _context.CalendarEvents.Remove(GetById(key).Result);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return true;
        }

        public async Task<ICollection<CalendarEvent>> ReadAll()
        {
            return await _context.CalendarEvents.ToListAsync();
        }

        public async Task<ICollection<CalendarEvent>> ReadAllInclude()
        {
            return await _context.CalendarEvents.Include(e => e.Tags).ToListAsync();
        }

        public async Task<ICollection<CalendarEvent>> GetBySelector(Func<CalendarEvent, bool> selector)
        {
            return await Task.Run(()=>_context.CalendarEvents.Where(selector).ToList());
        }

        public async Task<CalendarEvent> GetById(object id)
        {
            return await _context.CalendarEvents.FirstOrDefaultAsync(e=>e.OwnerId.Equals(id));
        }
    }
}
```

MessageRespository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DAL.Repositories.EFCore
{
    public class MessagesRepository : IMessagesRepository
    {
        private readonly DataContext _context;
        private readonly ILogger<MessagesRepository> _logger;

        public MessagesRepository(ILogger<MessagesRepository> logger, DataContext context)
        {
            _logger = logger;
```

```
            _context = context;
        }

        public async Task<Message> Create(Message value)
        {
            var res = await _context.Messages.AddAsync(value);
            var saveRes = await _context.SaveChangesAsync();
            return res?.Entity;
        }

        public async Task<Message> Update(Message value)
        {
            var res = _context.Messages.Update(value);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return res?.Entity;
        }

        public async Task<bool> Delete(object key)
        {
            var res = _context.Messages.Remove(GetById(key).Result);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return true;
        }

        public async Task<ICollection<Message>> ReadAll()
        {
            return await _context.Messages.ToListAsync();
        }

        public async Task<ICollection<Message>> ReadAllInclude()
        {
            return await _context.Messages.ToListAsync();
        }

        public async Task<ICollection<Message>> GetBySelector(Func<Message, bool> selector)
        {
            return await Task.Run(() => _context.Messages.Where(selector).ToList());
        }

        public async Task<Message> GetById(object id)
        {
            return await _context.Messages.FirstOrDefaultAsync(e => e.Id.Equals(id));
        }
    }
}
```

ProjectsRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.Data.Sqlite;
```

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DAL.Repositories.EFCore
{
   public class ProjectsRepository : IProjectsRepository
   {
      private readonly DataContext _context;
      private readonly ILogger<ProjectsRepository> _logger;

      public ProjectsRepository(ILogger<ProjectsRepository> logger,
         DataContext context)
      {
         _logger = logger;
         _context = context;
      }

      public async Task<Project> Create(Project value)
      {
         var res = await _context.Projects.AddAsync(value);
         var saveRes = await _context.SaveChangesAsync();
         return res?.Entity;
      }

      public async Task<Project> Update(Project value)
      {
         var res = _context.Projects.Update(value);
         var saveRes = await _context.SaveChangesAsync();
         _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
         return res?.Entity;
      }

      public async Task<bool> Delete(object key)
      {
         var res = _context.Projects.Remove(GetById(key).Result);
         var saveRes = await _context.SaveChangesAsync();
         _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
         return true;
      }

      public async Task<ICollection<Project>> ReadAll()
      {
         return await _context.Projects.ToListAsync();
      }

      public async Task<ICollection<Project>> ReadAllInclude()
      {
         return await _context
            .Projects
            .Include(e => e.Participants)
            .Include(e => e.Tasks)
            .ToListAsync();
      }

      public async Task<ICollection<Project>> GetBySelector(Func<Project, bool> selector)
      {
         return await Task.Run(() => _context.Projects
```

```
                .Include(e => e.Participants)
                .Include(e => e.Tasks)
                .Where(selector)
                .ToList());
        }

        public async Task<Project> GetById(object id)
        {
            return await _context.Projects
                .Include(e => e.Participants)
                .Include(e => e.Tasks)
                .FirstOrDefaultAsync(e => e.Id.Equals(id));
        }

        public async Task<Project> AddUserToProject(int projectId, object uid)
        {
            SqliteParameter pId = new SqliteParameter("pId", projectId);
            SqliteParameter uId = new SqliteParameter("uId", uid);
            await _context.Database
                .ExecuteSqlRawAsync(@"INSERT INTO UserProject (InProjectsId, ParticipantsId) VALUES
(@pId, @uId)", pId,
                    uId);
            await _context.SaveChangesAsync();
            return await _context.Projects.FirstOrDefaultAsync(e => e.Id == projectId);
        }
    }
}
```

ProjectTasksRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.Data.Sqlite;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DAL.Repositories.EFCore
{
    public class ProjectTasksRepository : IProjectTasksRepository
    {
        private readonly DataContext _context;
        private readonly ILogger<ProjectTasksRepository> _logger;

        public ProjectTasksRepository(DataContext context,
            ILogger<ProjectTasksRepository> logger)
        {
            _context = context;
            _logger = logger;
        }

        public async Task<ProjectTask> Create(ProjectTask value)
        {
            var res = await _context.ProjectTasks.AddAsync(value);
```

```
            var saveRes = await _context.SaveChangesAsync();
            return res?.Entity;
        }

        public async Task<ProjectTask> Update(ProjectTask value)
        {
            var res = _context.ProjectTasks.Update(value);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return res?.Entity;
        }

        public async Task<bool> Delete(object key)
        {
            var res = _context.ProjectTasks.Remove(GetById(key).Result);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return true;
        }

        public async Task<ICollection<ProjectTask>> ReadAll()
        {
            return await _context.ProjectTasks.ToListAsync();
        }

        public async Task<ICollection<ProjectTask>> ReadAllInclude()
        {
            return await _context.ProjectTasks.Include(e => e.Tags).ToListAsync();
        }

        public async Task<ICollection<ProjectTask>> GetBySelector(Func<ProjectTask, bool> selector)
        {
            return await Task.Run(() => _context.ProjectTasks.Where(selector).ToList());
        }

        public async Task<ProjectTask> GetById(object id)
        {
            return await _context.ProjectTasks.FirstOrDefaultAsync(e => e.Id.Equals(id));
        }

        public async Task<ProjectTask> AddUserToTask(int taskId, object uid)
        {
            SqliteParameter tId = new SqliteParameter("tId", taskId);
            SqliteParameter uId = new SqliteParameter("uId", uid);
            await _context.Database
                .ExecuteSqlRawAsync(@"INSERT INTO UserProjectTask (ParticipantsId, TasksId) VALUES
(@tId, @uId)",
                    tId, uId);
            await _context.SaveChangesAsync();
            return await _context.ProjectTasks.FirstOrDefaultAsync(e => e.Id == taskId);
        }
    }
}
```

TagsRepository.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DAL.Repositories.EFCore
{
   public class TagsRepository : ITagsRepository
   {
      private readonly DataContext _context;
      private readonly ILogger<TagsRepository> _logger;

      public TagsRepository(DataContext context, ILogger<TagsRepository> logger)
      {
         _context = context;
         _logger = logger;
      }

      public async Task<Tag> Create(Tag value)
      {
         var res = await _context.Tags.AddAsync(value);
         var saveRes = await _context.SaveChangesAsync();
         return res?.Entity;
      }

      public async Task<Tag> Update(Tag value)
      {
         var res = _context.Tags.Update(value);
         var saveRes = await _context.SaveChangesAsync();
         _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
         return res?.Entity;
      }

      public async Task<bool> Delete(object key)
      {
         var res = _context.Tags.Remove(GetById(key).Result);
         var saveRes = await _context.SaveChangesAsync();
         _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
         return true;
      }

      public async Task<ICollection<Tag>> ReadAll()
      {
         return await _context.Tags.ToListAsync();
      }

      public async Task<ICollection<Tag>> ReadAllInclude()
      {
         return await _context.Tags
            .Include(e => e.CalendarEvents)
            .Include(e => e.ProjectTasks)
            .ToListAsync();
      }
```

```
        public async Task<ICollection<Tag>> GetBySelector(Func<Tag, bool> selector)
        {
            return await Task.Run(() => _context.Tags
                .Include(e => e.CalendarEvents)
                .Include(e => e.ProjectTasks)
                .Where(selector).ToList());
        }

        public async Task<Tag> GetById(object id)
        {
            return await _context.Tags
                .Include(e => e.CalendarEvents)
                .Include(e => e.ProjectTasks)
                .FirstOrDefaultAsync(e => e.OwnerId.Equals(id));
        }
    }
}
```

UserRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;

namespace DAL.Repositories.EFCore
{
    public class UsersRepository : IUserRepository
    {
        private readonly DataContext _context;
        private readonly ILogger<UsersRepository> _logger;

        public UsersRepository(ILogger<UsersRepository> logger, DataContext context)
        {
            _logger = logger;
            _context = context;
        }

        public async Task<User> Create(User value)
        {
            var res = await _context.Users.AddAsync(value);
            var saveRes = await _context.SaveChangesAsync();
            return res?.Entity;
        }

        public async Task<User> Update(User value)
        {
            var res = _context.Users.Update(value);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return res?.Entity;
        }
```

```
        public async Task<bool> Delete(object key)
        {
            var res = _context.Users.Remove(GetById(key).Result);
            var saveRes = await _context.SaveChangesAsync();
            _logger.LogDebug(new EventId(1212), res?.DebugView?.LongView);
            return true;
        }

        public async Task<ICollection<User>> ReadAll()
        {
            return await _context.Users.ToListAsync();
        }

        public async Task<ICollection<User>> ReadAllInclude()
        {
            return await _context.Users.ToListAsync();
        }

        public async Task<ICollection<User>> GetBySelector(Func<User, bool> selector)
        {
            return await Task.Run(() => _context.Users.Where(selector).ToList());
        }

        public async Task<User> GetById(object id)
        {
            return await _context.Users.FirstOrDefaultAsync(e => e.Id.Equals(id));
        }
    }
}
```

Calendar.cs

```
using System;
using System.Collections.Generic;

namespace BLL.Calendar
{
    public class Calendar
    {
        public DateTime Month { get; set; }

        public ICollection<CalendarDay> CalendarDays { get; set; }
    }
}

using System;

namespace BLL.Calendar
{
    public enum Quarter
    {
        First = 1,
        Second = 2,
        Third = 3,
        Fourth = 4
    }
```

```
public enum Month
{
   January = 1,
   February = 2,
   March = 3,
   April = 4,
   May = 5,
   June = 6,
   July = 7,
   August = 8,
   September = 9,
   October = 10,
   November = 11,
   December = 12
}

/// <summary>
/// Common DateTime Methods.
/// </summary>
///
public static class CalendarUtils
{
   #region Quarters

   public static DateTime GetStartOfQuarter(int year, Quarter qtr)
   {
      return qtr switch
      {
         // 1st Quarter = January 1 to March 31
         Quarter.First => new DateTime(year, 1, 1, 0, 0, 0, 0),
         // 2nd Quarter = April 1 to June 30
         Quarter.Second => new DateTime(year, 4, 1, 0, 0, 0, 0),
         // 3rd Quarter = July 1 to September 30
         Quarter.Third => new DateTime(year, 7, 1, 0, 0, 0, 0),
         _ => new DateTime(year, 10, 1, 0, 0, 0, 0)
      };
   }

   public static DateTime GetEndOfQuarter(int year, Quarter qtr)
   {
      return qtr switch
      {
         // 1st Quarter = January 1 to March 31
         Quarter.First => new DateTime(year, 3, DateTime.DaysInMonth(year, 3), 23, 59, 59, 999),
         // 2nd Quarter = April 1 to June 30
         Quarter.Second => new DateTime(year, 6, DateTime.DaysInMonth(year, 6), 23, 59, 59, 999),
         // 3rd Quarter = July 1 to September 30
         Quarter.Third => new DateTime(year, 9, DateTime.DaysInMonth(year, 9), 23, 59, 59, 999),
         _ => new DateTime(year, 12, DateTime.DaysInMonth(year, 12), 23, 59, 59, 999)
      };
   }

   public static Quarter GetQuarter(Month month)
   {
      return month switch
      {
         // 1st Quarter = January 1 to March 31
```

```
        <= Month.March => Quarter.First,
        // 2nd Quarter = April 1 to June 30
        >= Month.April and <= Month.June => Quarter.Second,
        // 3rd Quarter = July 1 to September 30
        >= Month.July and <= Month.September => Quarter.Third,
        _ => Quarter.Fourth
    };
}

public static DateTime GetEndOfLastQuarter()
{
    if ((Month) DateTime.Now.Month <= Month.March)
        //go to last quarter of previous year
        return GetEndOfQuarter(DateTime.Now.Year - 1, Quarter.Fourth);
    else //return last quarter of current year
        return GetEndOfQuarter(DateTime.Now.Year,
            GetQuarter((Month) DateTime.Now.Month));
}

public static DateTime GetStartOfLastQuarter()
{
    if ((Month) DateTime.Now.Month <= Month.March)
        //go to last quarter of previous year
        return GetStartOfQuarter(DateTime.Now.Year - 1, Quarter.Fourth);
    else //return last quarter of current year
        return GetStartOfQuarter(DateTime.Now.Year,
            GetQuarter((Month) DateTime.Now.Month));
}

public static DateTime GetStartOfCurrentQuarter()
{
    return GetStartOfQuarter(DateTime.Now.Year,
        GetQuarter((Month) DateTime.Now.Month));
}

public static DateTime GetEndOfCurrentQuarter()
{
    return GetEndOfQuarter(DateTime.Now.Year,
        GetQuarter((Month) DateTime.Now.Month));
}

#endregion

#region Weeks

public static DateTime GetStartOfLastWeek()
{
    int daysToSubtract = (int) DateTime.Now.DayOfWeek + 7;
    DateTime dt =
        DateTime.Now.Subtract(System.TimeSpan.FromDays(daysToSubtract));
    return new DateTime(dt.Year, dt.Month, dt.Day, 0, 0, 0, 0);
}

public static DateTime GetEndOfLastWeek()
{
    DateTime dt = GetStartOfLastWeek().AddDays(6);
    return new DateTime(dt.Year, dt.Month, dt.Day, 23, 59, 59, 999);
```

```
}

public static DateTime GetStartOfCurrentWeek()
{
   int daysToSubtract = (int) DateTime.Now.DayOfWeek;
   DateTime dt =
      DateTime.Now.Subtract(System.TimeSpan.FromDays(daysToSubtract));
   return new DateTime(dt.Year, dt.Month, dt.Day, 0, 0, 0, 0);
}

public static DateTime GetEndOfCurrentWeek()
{
   DateTime dt = GetStartOfCurrentWeek().AddDays(6);
   return new DateTime(dt.Year, dt.Month, dt.Day, 23, 59, 59, 999);
}

#endregion

#region Months

public static DateTime GetStartOfMonth(Month month, int year)
{
   return new DateTime(year, (int) month, 1, 0, 0, 0, 0);
}

public static DateTime GetEndOfMonth(Month month, int year)
{
   return new DateTime(year, (int) month,
      DateTime.DaysInMonth(year, (int) month), 23, 59, 59, 999);
}

public static DateTime GetStartOfLastMonth()
{
   if (DateTime.Now.Month == 1)
      return GetStartOfMonth((Month) 12, DateTime.Now.Year - 1);
   else
      return GetStartOfMonth((Month) (DateTime.Now.Month - 1), DateTime.Now.Year);
}

public static DateTime GetEndOfLastMonth()
{
   if (DateTime.Now.Month == 1)
      return GetEndOfMonth((Month) 12, DateTime.Now.Year - 1);
   else
      return GetEndOfMonth((Month) (DateTime.Now.Month - 1), DateTime.Now.Year);
}

public static DateTime GetStartOfCurrentMonth()
{
   return GetStartOfMonth((Month) DateTime.Now.Month, DateTime.Now.Year);
}

public static DateTime GetEndOfCurrentMonth()
{
   return GetEndOfMonth((Month) DateTime.Now.Month, DateTime.Now.Year);
}
```

```
        #endregion

        #region Years

        public static DateTime GetStartOfYear(int year)
        {
            return new DateTime(year, 1, 1, 0, 0, 0, 0);
        }

        public static DateTime GetEndOfYear(int year)
        {
            return new DateTime(year, 12,
                DateTime.DaysInMonth(year, 12), 23, 59, 59, 999);
        }

        public static DateTime GetStartOfLastYear()
        {
            return GetStartOfYear(DateTime.Now.Year - 1);
        }

        public static DateTime GetEndOfLastYear()
        {
            return GetEndOfYear(DateTime.Now.Year - 1);
        }

        public static DateTime GetStartOfCurrentYear()
        {
            return GetStartOfYear(DateTime.Now.Year);
        }

        public static DateTime GetEndOfCurrentYear()
        {
            return GetEndOfYear(DateTime.Now.Year);
        }

        #endregion

        #region Days

        public static DateTime GetStartOfDay(DateTime date)
        {
            return new DateTime(date.Year, date.Month, date.Day, 0, 0, 0, 0);
        }

        public static DateTime GetEndOfDay(DateTime date)
        {
            return new DateTime(date.Year, date.Month,
                date.Day, 23, 59, 59, 999);
        }

        #endregion
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading.Tasks;
using AutoMapper;
using BLL.DTO;
using DAL.Entities;
using DAL.Repositories.Interfaces;

namespace BLL.Calendar
{
   public class CalendarService : ICalendarService
   {
      private readonly ICalendarEventsRepository _calendarEvents;
      private readonly IMapper _mapper;
      private readonly IProjectsRepository _projects;
      private readonly IProjectTasksRepository _projectTasks;

      public CalendarService(ICalendarEventsRepository calendarEvents,
         IProjectTasksRepository projectTasks,
         IProjectsRepository projects,
         IMapper mapper)
      {
         _calendarEvents = calendarEvents;
         _projectTasks = projectTasks;
         _projects = projects;
         _mapper = mapper;
      }

      public async Task<Calendar> GetCalendarForUser(object userId, DateTime month)
      {
         Calendar calendar = new Calendar();
         calendar.Month = CalendarUtils.GetStartOfMonth((Month) month.Month, month.Year);
         int days = DateTime.DaysInMonth(month.Year, month.Month);
         calendar.CalendarDays = new List<CalendarDay>(days);
         for (int j = 0; j < days; j++)
         {
            calendar.CalendarDays.Add(
               new CalendarDay()
               {
                  Day = calendar.Month.AddDays(j)
               });
         }

         // int i = 0;
         foreach (var day in calendar.CalendarDays)
         {
            // day.Day = calendar.Month.AddDays(i);
            var calendarEvents = await _calendarEvents
               .GetBySelector(e => e.EventDate >= day.Day
                           && e.EventDate < day.Day.AddDays(1)
                           && e.OwnerId.Equals(userId));
            day.CalendarEvents = await Task
               .Run(() => calendarEvents.Select(_mapper.Map<CalendarEventDTO>)
                  .ToList());
            var projectTasks = await _projectTasks.GetBySelector(e => e.TaskStart <= day.Day
                                                   && e.TaskEnd >= day.Day.AddDays(1)
                                                   && e.Participants != null
                                                   && e.Participants.Contains(new User()
                                                      {Id = userId as string}));
```

```
                day.ProjectTasks = await Task.Run(() => projectTasks.Select(_mapper.Map<Project-
TaskDTO>).ToList());
                // i++;
            }

            return calendar;
        }

        public async Task<CalendarDay> GetCalendarDay(object userId, DateTime date)
        {
            CalendarDay day = new();

            day.Day = date;
            var calendarEvents = await _calendarEvents
                .GetBySelector(e => e.EventDate >= day.Day
                                && e.EventDate < day.Day.AddDays(1)
                                && e.OwnerId.Equals(userId));
            day.CalendarEvents = await Task
                .Run(() => calendarEvents.Select(_mapper.Map<CalendarEventDTO>)
                    .ToList());
            var projectTasks = await _projectTasks.GetBySelector(e => e.TaskStart <= day.Day
                                                    && e.TaskEnd >= day.Day.AddDays(1)
                                                    && e.Participants.Contains(new User()
                                                        {Id = userId as string}));
            day.ProjectTasks = await Task.Run(() => projectTasks.Select(_mapper.Map<Project-
TaskDTO>).ToList());

            return day;
        }

        public async Task<CalendarEventDTO> AddCalemdarEvent(CalendarEventDTO value)
        {
            return _mapper.Map<CalendarEventDTO>(
                await _calendarEvents.Create(_mapper.Map<CalendarEvent>(value))
            );
        }
    }
}

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BLL.DTO;
using DAL.Entities;
using DAL.Repositories.Interfaces;

namespace BLL.Services
{
    public class GroupService : IGroupService
    {
        private readonly IGroupsRepository _groupsRepository;
        private readonly IMapper _mapper;
        private readonly IUserRepository _userRepository;

        public GroupService(IGroupsRepository groupsRepository,
            IMapper mapper,
```

```
    IUserRepository userRepository)
{
    _groupsRepository = groupsRepository;
    _mapper = mapper;
    _userRepository = userRepository;
}

public async Task<bool> AddUserToGroup(int groupId, string userId)
{
    bool result = true;
    try
    {
        var user = await _userRepository.GetById(userId);
        var group = await _groupsRepository.GetById(groupId);
        group.GroupParticipants.Add(user);
        group = await _groupsRepository.Update(group);
    }
    catch
    {
        result = false;
    }

    return result;
}

public async Task<ICollection<GroupDTO>> GetUserGroups(string userId)
{
    var groups = await _groupsRepository.GetBySelector(e => e.CommandOwner.Equals(userId));
    return groups.Select(_mapper.Map<GroupDTO>).ToList();
}

public async Task<bool> DeleteGroup(int id)
{
    return await _groupsRepository.Delete(id);
}

public async Task<GroupDTO> UpdateGroup(GroupDTO group)
{
    var ngroup = await _groupsRepository.Update(_mapper.Map<Group>(group));
    return _mapper.Map<GroupDTO>(ngroup);
}

public async Task<bool> RemoveUserFromGroup(string userId, int groupId)
{
    var user = await _userRepository.GetById(userId);
    var group = await _groupsRepository.GetById(groupId);
    group.GroupParticipants.Remove(user);
    group = await _groupsRepository.Update(group);
    return true;
}

public async Task<GroupDTO> AddNewGroup(GroupDTO group)
{
    var ngroup = await _groupsRepository.Create(_mapper.Map<Group>(group));
    return _mapper.Map<GroupDTO>(group);
}
```

```
        public async Task<GroupDTO> GetById(int id)
        {
            var group = await _groupsRepository.GetById(id);
            return _mapper.Map<GroupDTO>(group);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BLL.DTO;
using DAL.Entities;
using DAL.Repositories.Interfaces;

namespace BLL.Services
{
    public class MessagesService : IMessagesService
    {
        private readonly IMapper _mapper;
        private readonly IMessagesRepository _messagesRepository;
        private readonly IUserRepository _userRepository;

        public MessagesService(IMessagesRepository messagesRepository,
            IUserRepository userRepository,
            IMapper mapper)
        {
            _messagesRepository = messagesRepository;
            _userRepository = userRepository;
            _mapper = mapper;
        }

        public async Task<ICollection<UserDTO>> GetConversationsList(object userId)
        {
            var msgs = await _messagesRepository.GetBySelector(e => e.Sender.Equals(userId)
                                                    || e.Recipient.Equals(userId));
            var conversationsWith = await Task.Run(() => msgs?
                .SelectMany(e => new string[] {e?.Recipient, e?.Sender})
                .Distinct()
                .Except(new[] {userId as string}).Select(e => _userRepository.GetById(e).Result)
                .Select(e => _mapper.Map<UserDTO>(e))
                .ToList());
            return conversationsWith;
        }

        public async Task<ICollection<MessageDTO>> GetConversationBetween(object userA, object userB)
        {
            try
            {
                var conversation = await _messagesRepository
                    .GetBySelector(e => e.Sender.Equals(userA) && e.Recipient.Equals(userB)
                                || e.Sender.Equals(userB) && e.Recipient.Equals(userA));
                return await Task.Run(() => conversation?
                    .Select(e => _mapper.Map<MessageDTO>(e))
                    .ToList());
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
                return new List<MessageDTO>();
            }
        }

        public async Task SendMessage(object recipient, object sender, string message)
        {
            await _messagesRepository.Create(
                new Message()
                {
                    Id = Guid.NewGuid().ToString("D"),
                    MessageBody = message,
                    Recipient = recipient as string,
                    Sender = sender as string,
                    Sended = DateTime.Now
                });
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BLL.DTO;
using DAL.Entities;
using DAL.Repositories.Interfaces;
using Microsoft.Extensions.Logging;

namespace BLL.Services
{
    public class ProjectService : IProjectService
    {
        private readonly ILogger<ProjectService> _logger;
        private readonly IMapper _mapper;
        private readonly IProjectsRepository _projects;
        private readonly IProjectTasksRepository _projectTasks;
        private readonly IUserRepository _userRepository;

        public ProjectService(IProjectTasksRepository projectTasks, IProjectsRepository projects, IMapper mapper,
            ILogger<ProjectService> logger, IUserRepository userRepository)
        {
            _projectTasks = projectTasks;
            _projects = projects;
            _mapper = mapper;
            _logger = logger;
            _userRepository = userRepository;
        }

        public async Task<ICollection<ProjectDTO>> GetUsersProjects(object uid)
        {
            var projects = await _projects.GetBySelector(e => e.Participants.Any(f => f.Id.Equals(uid)));
            return projects.Select(_mapper.Map<ProjectDTO>).ToList();
```

```
}

public async Task<ICollection<ProjectDTO>> UserOwnedProjects(object uid)
{
   var projects = await _projects.GetBySelector(e => e.ProjectOwner.Equals(uid));
   return projects.Select(_mapper.Map<ProjectDTO>).ToList();
}

public async Task<ProjectDTO> GetProjectById(int projectId)
{
   var project = await _projects.GetById(projectId);
   return _mapper.Map<ProjectDTO>(project);
}

public async Task<ProjectTaskDTO> GetTaskById(int taskId)
{
   var task = await _projectTasks.GetById(taskId);
   return _mapper.Map<ProjectTaskDTO>(task);
}

public async Task<ProjectDTO> AddProject(ProjectDTO project)
{
   try
   {
      var source = await _projects.Create(_mapper.Map<Project>(project));
      return _mapper.Map<ProjectDTO>(source);
   }
   catch (Exception ex)
   {
      _logger.LogError(ex, "Exception in AddProject(ProjectDTO project)");
   }

   return project;
}

public async Task<ProjectDTO> UpdateProject(ProjectDTO project)
{
   try
   {
      var source = await _projects.Update(_mapper.Map<Project>(project));
      return _mapper.Map<ProjectDTO>(source);
   }
   catch (Exception ex)
   {
      _logger.LogError(ex, "Exception in UpdateProject(ProjectDTO project)");
   }

   return project;
}

public async Task<bool> RemoveProject(int projectId)
{
   bool res = true;
   try
   {
      var remRes = await _projects.Delete(projectId);
   }
```

```
    catch (Exception ex)
    {
        _logger.LogError(ex, "Exception in UpdateProject(ProjectDTO project)");
        res = false;
    }

    return res;
}

public async Task<ProjectTaskDTO> AddProjectTask(ProjectTaskDTO projectTask)
{
    try
    {
        var source = await _projectTasks.Create(_mapper.Map<ProjectTask>(projectTask));
        return _mapper.Map<ProjectTaskDTO>(source);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Exception in AddProjectTask(ProjectTaskDTO projectTask)");
    }

    return projectTask;
}

public async Task<ProjectTaskDTO> UpdateProjectTask(ProjectTaskDTO projectTask)
{
    try
    {
        var source = await _projectTasks.Update(_mapper.Map<ProjectTask>(projectTask));
        return _mapper.Map<ProjectTaskDTO>(source);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Exception in AddProjectTask(ProjectTaskDTO projectTask)");
    }

    return projectTask;
}

public async Task<bool> RemoveProjectTask(int taskId)
{
    bool res = true;
    try
    {
        var remRes = await _projectTasks.Delete(taskId);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Exception in UpdateProjectTask(ProjectTaskDTO project)");
        res = false;
    }
    return res;
}

public async Task<ProjectTaskDTO> AddUserToProjectTask(int taskId, object userId)
{
    var task = await _projectTasks.AddUserToTask(taskId, userId);
```

```
            return _mapper.Map<ProjectTaskDTO>(task);
        }

        public async Task<ProjectDTO> AddUserToProject(int projectId, object userId)
        {
            var nproject = await _projects.AddUserToProject(projectId, userId);
            return _mapper.Map<ProjectDTO>(nproject);
        }
    }
}

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BLL.DTO;
using DAL.Entities;
using DAL.Repositories.Interfaces;

namespace BLL.Services
{
    public class UserDataService : IUserDataService
    {
        private readonly IMapper _mapper;
        private readonly IUserRepository _userRepository;

        public UserDataService(IUserRepository userRepository,
            IMapper mapper)
        {
            _userRepository = userRepository;
            _mapper = mapper;
        }

        public async Task<UserDTO> AttachUserData(UserDTO value)
        {
            var user =  await _userRepository.Create(_mapper.Map<User>(value));
            return _mapper.Map<UserDTO>(user);
        }

        public async Task<UserDTO> Update(UserDTO value)
        {
            var user =  await _userRepository.Update(_mapper.Map<User>(value));
            return _mapper.Map<UserDTO>(user);
        }

        public async Task<bool> DeleteUserData(object key)
        {
            var result =  await _userRepository.Delete(key);
            return result;
        }

        public async Task<ICollection<UserDTO>> ReadAll()
        {
            var users =  await _userRepository.ReadAll();
            return await Task.Run(() => users.Select(_mapper.Map<UserDTO>).ToList());
        }
```

```
        public async Task<ICollection<UserDTO>> ReadAllInclude()
        {
            var users =  await _userRepository.ReadAllInclude();
            return await Task.Run(() => users.Select(_mapper.Map<UserDTO>).ToList());
        }


        public async Task<UserDTO> GetUserDataById(object id)
        {
            var result =  await _userRepository.GetById(id);
            return _mapper.Map<UserDTO>(result);
        }
    }
}
```

EmailService.cs

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Options;
using MailKit.Net.Smtp;
using MimeKit;


namespace DumbCalendar.Services.Email
{
    public class EmailService : IEmailSender
    {
        private readonly MessageSenderOption _options;

        public EmailService(IOptions<MessageSenderOption> options)
        {
            _options = options?.Value;
        }

        public Task SendEmailAsync(string email, string subject, string htmlMessage)
        {
            return Task.Run(() =>
            {
                var emailMessage = new MimeMessage();

                emailMessage.From.Add(new MailboxAddress("Site Bot", _options.EmailSenderLogin));
                emailMessage.To.Add(new MailboxAddress("User", email));
                emailMessage.Subject = subject;
                emailMessage.Body = new TextPart(MimeKit.Text.TextFormat.Html)
                {
                    Text = htmlMessage
                };

                using var client = new SmtpClient();
                client.ConnectAsync("smtp.yandex.ru", 465, true).Wait();
                client.AuthenticateAsync(_options.EmailSenderLogin, _options.EmailSenderPassword).Wait();
                client.SendAsync(emailMessage).Wait();

                client.DisconnectAsync(true).Wait();
            });
        }
```

```
    }
}

using System.Threading.Tasks;
using BLL.DTO;
using BLL.Services;
using DumbCalendar.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;

namespace DumbCalendar.Controllers
{
    [Authorize]
    public class GroupsController : Controller
    {
        private readonly IEmailSender _emailSender;
        private readonly IGroupService _groupService;
        private readonly UserManager<IdentityUser> _userManager;

        public GroupsController(IGroupService groupService,
            UserManager<IdentityUser> userManager, IEmailSender emailSender)
        {
            _groupService = groupService;
            _userManager = userManager;
            _emailSender = emailSender;
        }

        // GET
        public async Task<IActionResult> Index()
        {
            var groups = await _groupService.GetUserGroups(_userManager.GetUserId(User));
            return View(groups);
        }

        [HttpGet]
        public async Task<IActionResult> GroupInvite(int id)
        {
            var group = await _groupService.GetById(id);
            var userId = _userManager.GetUserId(User);
            //var res = await _groupService.AddUserToGroup(id, _userManager.GetUserId(User));
            return View(new GroupInviteModel {Group = group, UserId = userId});
        }

        [HttpPost]
        public async Task<IActionResult> GroupInvite(int groupId, string userId)
        {
            var res = await _groupService.AddUserToGroup(groupId, userId);
            var msg = new MessageViewModel()
            {
                ReturnUrl = "/Groups",
                Caption = res ? "Everything is ok!" : "Something went wrong!",
                Message = res
                    ? "You are added to group."
                    : "Internal error."
            };
```

```
            return View("Message", msg);
        }

        [HttpGet]
        public IActionResult SendInvite(int id)
        {
            return PartialView("_addUserToGroupDialog", id);
        }

        [HttpPost]
        public async Task<IActionResult> SendInvite(string email, int id)
        {
            await _emailSender.SendEmailAsync(email, "Group invite : Dumb Calendar",
                $"Your invite to group: <a href='localhost:5001/Groups/GroupInvite/{id}'>localhost:5001/Groups/GroupIn-
vite/{id}</a>");
            return RedirectToAction("Index");
        }

        public async Task<IActionResult> Participants(int id)
        {
            var group = await _groupService.GetById(id);
            return PartialView("_participantsList", group?.GroupParticipants);
        }

        public IActionResult AddGroup()
        {
            return PartialView("_addGroup", new BLL.DTO.GroupDTO());
        }

        [HttpPost]
        public async Task<IActionResult> AddGroup(GroupDTO group)
        {
            if (!ModelState.IsValid)
            {
                return PartialView("_addGroup", group);
            }

            await _groupService.AddNewGroup(group);
            return Content("Group added!");
        }

        public async Task<IActionResult> DeleteGroup(int id)
        {
            var group = await _groupService.GetById(id);
            string uid = _userManager.GetUserId(User);
            if (!group.CommandOwner.Equals(uid))
            {
                RedirectToAction("Index");
            }

            bool res = await _groupService.DeleteGroup(id);
            return View("Message", new MessageViewModel()
            {
                Caption = (res)
                    ? $"Group {group?.CommandName ?? "null"} deleted!"
                    : $"Group {group?.CommandName ?? "null"} not deleted!",
                Message = "",
```

```
            ReturnUrl = "/Groups"
        });
    }
  }
}

using DumbCalendar.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace DumbCalendar.Controllers
{
  public class HomeController : Controller
  {
    private readonly ILogger<HomeController> _logger;
    private readonly SignInManager<IdentityUser> _signInManager;

    public HomeController(ILogger<HomeController> logger,
      SignInManager<IdentityUser> signInManager)
    {
      _logger = logger;
      _signInManager = signInManager;
    }

    public IActionResult Index()
    {
      if (_signInManager.IsSignedIn(User))
      {
        return RedirectToAction("Index", "Calendar");
      }
      return View();
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
      return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
  }
}

using System;
using System.Threading.Tasks;
using BLL.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace DumbCalendar.Controllers
{
  // public class MessagesController : Controller
```

```
// {
//     // GET
//     public IActionResult Index()
//     {
//         return View();
//     }
//
//     public IActionResult Details()
//     {
//         throw new System.NotImplementedException();
//     }
//
//     public async Task<IActionResult> Send(MessageDTO)
//     {
//
//     }
// }
[Authorize]
public class MessagesController : Controller
{
    private readonly IMessagesService _messageService;
    private readonly UserManager<IdentityUser> _user;

    public MessagesController(IMessagesService messageService, UserManager<IdentityUser> user)
    {
        _messageService = messageService;
        _user = user;
    }

    public async Task<IActionResult> Index()
    {
        var id = _user.GetUserId(User);
        return View("Index", await _messageService.GetConversationsList(id));
    }

    public async Task<IActionResult> Details(string id)
    {
        ViewBag.Recipient = id;
        var uid = _user.GetUserId(User);
        var conversations = await _messageService.GetConversationBetween(uid, id);
        return View("Details", conversations);
    }

    [HttpPost]
    public async Task<IActionResult> Send(string messageBody, string sender, string recipient)
    {
        var res = new ContentResult {Content = "Message sent!"};
        if (messageBody?.Length == 0)
            res.Content = "Message must be > 0 symbols!";
        else
        {
            try
            {
                await _messageService.SendMessage(sender, recipient, messageBody);
            }
            catch (Exception e)
            {
```

```
                Console.WriteLine(e);
                res.Content = "Exception occures, try again later!";
            }
        }

        return res;
    }
  }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BLL.DTO;
using BLL.Services;
using DumbCalendar.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace DumbCalendar.Controllers
{
  [Authorize]
  public class ProjectsController : Controller
  {
    private readonly IGroupService _groupService;
    private readonly IProjectService _projectService;
    private readonly IUserDataService _userDataService;
    private readonly UserManager<IdentityUser> _userManager;

    public ProjectsController(UserManager<IdentityUser> userManager, IUserDataService userDataService,
      IProjectService projectService, IGroupService groupService)
    {
      _userManager = userManager;
      _userDataService = userDataService;
      _projectService = projectService;
      _groupService = groupService;
    }

    public async Task<IActionResult> Index()
    {
      var model = new ProjectsIndexViewModel();
      model.UserOwn = await _projectService.GetUsersProjects(_userManager.GetUserId(User));
      model.Participating = await _projectService.UserOwnedProjects(_userManager.GetUserId(User));
      return View(model);
    }

    public async Task<IActionResult> ProjectInfo(int id)
    {
      var project = await _projectService.GetProjectById(id);
      return PartialView("_projectDetails", project);
    }

    [HttpGet]
    public async Task<IActionResult> AddParticipantToProject(int id)
```

```
{
    var availableUsers = await _groupService.GetUserGroups(_userManager.GetUserId(User));
    var select = availableUsers.Select(e => e.GroupParticipants);
    var list = new List<UserDTO>();
    foreach (var col in select)
    {
        list.AddRange(col);
    }

    var options = list
        .Distinct()
        .Select(e => new {Id = e.Id, FullName = e.FullName})
        .ToList();
    ViewBag.SelectOpts = new MultiSelectList(options, "Id", "FullName");
    return PartialView("_addToProject", new AddParticipantViewModel()
    {
        AvailableUsers = list,
        SelectedUser = ViewBag.SelectOpts,
        TargetId = id
    });
}

[HttpPost]
public async Task<IActionResult> AddParticipantToProject(string[] id)
{
    int projectId = Int32.Parse(Request.Form["TargetId"]);
    id = id.Distinct().ToArray();
    foreach (var uid in id)
    {
        await _projectService.AddUserToProject(projectId, uid);
    }

    return RedirectToAction("Index");
}

[HttpGet]
public async Task<IActionResult> AddProjectTask(int id)
{
    var projects = await _projectService.UserOwnedProjects(_userManager.GetUserId(User));
    ViewBag.ProjectId = new MultiSelectList(
        projects.Select(e => new {Id = e.Id, Name = e.Name}),
        "Id",
        "Name"
    );
    return PartialView("_addProjectTask", new ProjectTaskDTO() {ProjectId = id});
}

[HttpPost]
public async Task<IActionResult> AddProjectTask(ProjectTaskDTO task)
{
    await _projectService.AddProjectTask(task);
    return RedirectToAction("Index");
}

[HttpGet]
public IActionResult AddProject()
{
```

```
            return PartialView("_addProject", new ProjectDTO() {ProjectOwner = _userManager.GetUserId(User)});
        }

        public async Task<IActionResult> AddProject(ProjectDTO project)
        {
            await _projectService.AddProject(project);
            return RedirectToAction("Index");
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BLL.Services;
using DumbCalendar.Models;
using Microsoft.AspNetCore.Mvc;

namespace DumbCalendar.Controllers
{
    public class ProjectTaskController : Controller
    {
        private readonly IProjectService _projectService;

        public ProjectTaskController(IProjectService projectService)
        {
            _projectService = projectService;
        }

        // public IActionResult Index()
        // {
        //     return View();
        // }

        public async Task<IActionResult> Details(int id)
        {
            var task = await _projectService.GetTaskById(id);
            return View(task);
        }

        public async Task<IActionResult> AddUserToTask(int id)
        {
            var task = await _projectService.GetTaskById(id);
            return PartialView("_addToTask", new AddParticipantViewModel(){TargetId = task.Id});
        }

        [HttpPost]
        public async Task<IActionResult> AddUserToTask(string[] id)
        {
            foreach (var i in id.Distinct())
            {
                await _projectService.AddUserToProjectTask(Int32.Parse(Request.Form["TargetId"]), i);
            }

            return View("Message", new MessageViewModel()
            {
```

```
            Caption = "Users are added!",
            Message = "",
            ReturnUrl = "/Projects"
        });
    }}}


using System;
using System.Threading.Tasks;
using BLL.Calendar;
using BLL.DTO;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace DumbCalendar.Controllers
{
    [Authorize]
    public class CalendarController : Controller
    {
        private readonly ICalendarService _calendarService;
        private readonly UserManager<IdentityUser> _userManager;

        public CalendarController(ICalendarService calendarService,
            UserManager<IdentityUser> userManager)
        {
            _calendarService = calendarService;
            _userManager = userManager;
        }

        [Authorize]
        public async Task<IActionResult> Index()
        {
            var userId = _userManager.GetUserId(User);
            return View(await _calendarService.GetCalendarForUser(userId, DateTime.Now));
        }

        [Authorize]
        public async Task<IActionResult> IndexSpecified(DateTime month)
        {
            var userId = _userManager.GetUserId(User);
            return View("Index", await _calendarService.GetCalendarForUser(userId,
                CalendarUtils.GetStartOfMonth((Month) month.Month, month.Year)));
        }

        [Authorize]
        public async Task<IActionResult> DayDetailed(DateTime day)
        {
            var userId = _userManager.GetUserId(User);
            var calendarDay = await _calendarService.GetCalendarDay(userId, day);
            return View(calendarDay);
        }

        [Authorize]
        public IActionResult AddCalendarEvent()
        {
            return View(new CalendarEventDTO());
```

```
        }

        [Authorize]
        [HttpPost]
        public async Task<IActionResult> AddCalendarEvent(CalendarEventDTO value)
        {
            if (!ModelState.IsValid)
            {
                ModelState.AddModelError(string.Empty, "Wrong input!");
                return View(value);
            }

            await _calendarService.AddCalemdarEvent(value);
            return RedirectToAction("DayDetailed", new {day = value.EventDate});
        }
    }
}

using System.Threading.Tasks;
using BLL.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace DumbCalendar.Controllers
{
    [Authorize]
    public class UserController : Controller
    {
        private readonly IUserDataService _userDataService;

        public UserController(IUserDataService userDataService)
        {
            _userDataService = userDataService;
        }

        public async Task<ViewResult> Details(string id)
        {
            var user = await _userDataService.GetUserDataById(id);
            return View(user);
        }
    }
}
```