Код-ревью (из 362 ревью) от 01.12.2024 08:11

Файл: MessengerBackendTests/Test1.cs

Строка 7

- **Комментарий:**

  Пустая строка после объявления класса. Удалите пустую строку.

- **Текущий код:**

```
namespace MessengerBackendTests
{
    public class Test1
    {
    }
```

Строка 10

- **Комментарий:**

  Пустая строка перед закрывающей фигурной скобкой класса. Удалите пустую строку.

- **Текущий код:**

```
{
    }
}
```

Файл: MessengerBackend/Startup.cs

Строка 37

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class Startup
    {
        private readonly CryptoService _cryptoService;
        private readonly IWebHostEnvironment _env;
```

Строка 45

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
{
            Configuration = configuration;
            _env = env;
            _cryptoService = new CryptoService(configuration);
        }
```

Строка 54

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
// This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddOptions();
            services.AddMemoryCache();
```

Строка 59

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
services.AddMemoryCache();

            // Configure rate limiting
            {
                //load general configuration from appsettings.json
```

Строка 79

- **Комментарий:**

  Метод AddMvc устарел. Используйте AddControllersWithViews или AddControllers.

- **Текущий код:**

```
// the clientId/clientIp resolvers use it.
                services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

                // configuration (resolvers, counter key builders)
                services.AddSingleton<IRateLimitConfiguration, RateLimitConfiguration>();
```

Строка 86

- **Комментарий:**

  Регистрация IHttpContextAccessor уже происходит на строке 136. Удалите дублирование.

- **Текущий код:**

```
services.AddSingleton(_cryptoService);

            services.AddDbContext<MessengerDBContext>(builder =>
            {
                builder
```

Строка 108

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
services.AddSingleton<IAuthorizationHandler, IPCheckHandler>();

            services.AddAuthorization(options =>
            {
                options.DefaultPolicy = new AuthorizationPolicyBuilder()
```

Строка 130

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
services.AddScoped<UserService>();
            services.AddScoped<AuthService>();
            services.AddScoped<ChatService>();
            services.AddScoped<MessageProcessService>();
```

Строка 134

- **Комментарий:**

  Регистрация VerificationService как Singleton может привести к проблемам с состоянием. Рассмотрите возможность использования Scoped или Transient.

- **Текущий код:**

```
services.AddScoped<ChatService>();
            services.AddScoped<MessageProcessService>();

            services.AddSwaggerDocument();
```

Строка 136

- **Комментарий:**

  Регистрация RealTimeServer как Singleton может привести к проблемам с состоянием. Рассмотрите возможность использования Scoped или Transient.

- **Текущий код:**

```
    services.AddSwaggerDocument();
```

Строка 146

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    services.Configure<KestrelServerOptions>(options => { options.AllowSynchronousIO = true; });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request p
```

Строка 153

- **Комментарий:**

  Разрешение синхронного ввода-вывода может привести к проблемам с производительностью и блокировкой потоков. Рассмотрите возможность использования асинхронных операций.

- **Текущий код:**

```
    {
            NpgsqlLogManager.Provider = new SerilogLoggingProvider(Log.Logger);


            app.UseSerilogRequestLogging(); // <-- Add this line
```

Строка 162

- **Комментарий:**

  Инициализация NpgsqlLogManager.Provider лучше выполнять в ConfigureServices для согласованности конфигурации.

- **Текущий код:**

```
app.UseDeveloperExceptionPage();
                app.UseDatabaseErrorPage();
                IdentityModelEventSource.ShowPII = true;
            }
            else
```

Строка 165

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
}

            else
            {
                app.UseHttpsRedirection();
```

Строка 182

- **Комментарий:**

  Использование двух асинхронных middleware для обработки 404 и исключений может привести к непредсказуемому поведению. Объедините их в один middleware.

- **Текущий код:**

```
var originalPath = ctx.Request.Path.Value;
                    ctx.Items["originalPath"] = originalPath;
                    ctx.Request.Path = "/error/404";
```

```
            await next();
        }
```

Строка 197

- **Комментарий:**

  Второй middleware для обработки исключений может перехватывать исключения, обработанные первым middleware. Рассмотрите возможность объединения их в один middleware.

- **Текущий код:**

```
{
                    ctx.Response.StatusCode = ex.HttpStatusCode;
                    ctx.Response.ContentType = "application/json";
                    foreach (var (key, value) in ex.HttpHeaders)
                    {
```

Строка 210

- **Комментарий:**

  Использование ctx.Items для хранения данных может быть небезопасным и запутанным. Рассмотрите возможность использования другого механизма для передачи данных между middleware.

- **Текущий код:**

```
summary = ex.Summary,
                        details = ex.Message
                    }));
                }
                catch (ApiException ex)
```

Строка 219

- **Комментарий:**

Поле details может содержать конфиденциальную информацию. Убедитесь, что оно безопасно для вывода в ответе.

- **Текущий код:**

```
await ctx.Response.WriteAsync(JsonConvert.SerializeObject(new
                    {
                        type = "twilio",
                        twilioErrorCode = ex.Code,
                        // details = ex.Message,
```

Строка 232

- **Комментарий:**

Раскомментируйте строку details, если она действительно нужна для ответа.

- **Текущий код:**

```
KeepAliveInterval = TimeSpan.FromMinutes(2.0)
            });

            app.UseMiddleware<WebSocketMiddleware>();
```

Строка 246

- **Комментарий:**

Метод UseRouting должен вызываться до UseAuthentication и UseAuthorization для правильного маршрутизирования запросов.

- **Текущий код:**

```
app.UseEndpoints(endpoints => { endpoints.MapControllers(); });
        }
    }
}
```

Файл: MessengerBackend/Extensions.cs

Строка 19

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{

    public static class Extensions
    {
        public static string GetString(this Stream s)
        {
```

Строка 29

- **Комментарий:**

Метод удаляет только пробелы, но не другие виды пробельных символов (табуляции, переводы строки и т.д.). Рассмотрите использование Regex или String.Concat с Where для удаления всех пробельных символов.

- **Текущий код:**

```
public static string RemoveWhitespace(this string input) => input.Replace(" ", "");

        public static byte[] ToByteArray(this DerAsnBitString bitString) => bitString.Encode(null)

        public static bool EqualsAnyString(this string self, params string[] args) =>
```

Строка 33

- **Комментарий:**

Метод пропускает первый байт без объяснения причины. Убедитесь, что это действительно необходимо и добавьте комментарий, объясняющий, почему пропускается первый байт.

- **Текущий код:**

```
public static bool EqualsAnyString(this string self, params string[] args) =>
            args.Any(arg => arg.Equals(self));

        public static async Task<(WebSocketReceiveResult, byte[])> ReceiveFrameAsync(
```

Строка 37

- **Комментарий:**

Порядок аргументов в методе EqualsAnyString может быть интуитивно непонятен. Рассмотрите возможность изменения сигнатуры на (this string self, params string[] values) для большей ясности.

- **Текущий код:**

```
public static async Task<(WebSocketReceiveResult, byte[])> ReceiveFrameAsync(
            this WebSocket socket, CancellationToken cancelToken)
        {
            WebSocketReceiveResult response;
```

Строка 43

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
var message = new List<byte>();

            var buffer = new byte[4096];
            do
            {
```

Строка 56

- **Комментарий:**

  Метод AddRange добавляет все 4096 байт из буфера, даже если фактически получено меньше. Используйте response.Count для добавления только полученных данных.

- **Текущий код:**

```
// Obtain the custom attribute for the method.
        // The value returned contains the StateMachineType property.
        // Null is returned if the attribute isn't present for the method.
        public static bool IsAsync(this MethodInfo methodInfo) =>
            methodInfo.GetCustomAttribute(
```

Строка 65

- **Комментарий:**

  Избыточные комментарии. Удалите или перенесите комментарии в регион документации метода.

- **Текущий код:**

```
{
            var sw = Stopwatch.StartNew();
            action();
            sw.Stop();
            return sw.Elapsed;
```

Строка 70

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
return sw.Elapsed;
        }

        public static IDisposable BeginNamedScope(this ILogger logger,
            string name, params ValueTuple<string, object>[] properties)
```

Строка 73

- **Комментарий:**

  Метод GetCustomAttribute вызывается без указания флага BindingFlags. Рассмотрите возможность добавления BindingFlags.Instance | BindingFlags.Static для более точного поиска атрибута.

- **Текущий код:**

```
    public static IDisposable BeginNamedScope(this ILogger logger,
                string name, params ValueTuple<string, object>[] properties)
            {
                var dictionary = properties.ToDictionary(p => p.Item1, p => p.Item2);
```

Строка 79

- **Комментарий:**

  Создание словаря внутри метода может быть неэффективным при большом количестве свойств. Рассмотрите возможность передачи словаря напрямую, если это возможно.

- **Текущий код:**

```
    return logger.BeginScope(dictionary);
            }
        }
```

Строка 80

- **Комментарий:**

  Использование ValueTuple в параметрах метода может быть менее читаемым. Рассмотрите возможность создания специального класса или структуры для передачи свойств.

- **Текущий код:**

```
    }
        }
    }
```

Строка 81

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    }
    }
```

Файл: MessengerBackend/Program.cs

Строка 17

- **Комментарий:**

  Использование null! может быть опасным и неочевидным. Лучше инициализировать Configuration при объявлении или использовать явную проверку на null перед использованием.

- **Текущий код:**

```
    {
            public static LoggerConfiguration Configuration { get; private set; } = null!;

            public static async Task<int> Main(string[] args)
            {
```

Строка 24

- **Комментарий:**

Метод CreateHostBuilder вызывается без сохранения результата в переменную. Убедитесь, что это не является ошибкой.

- **Текущий код:**

```
Configuration = new LoggerConfiguration()
                .MinimumLevel.Override("Microsoft", LogEventLevel.Information)
                .MinimumLevel.Override("Microsoft.AspNetCore", LogEventLevel.Warning)
                .MinimumLevel.Override("Npgsql", LogEventLevel.Information)
```

Строка 34

- **Комментарий:**

Создание логгера происходит после создания хоста. Рассмотрите возможность создания логгера раньше, чтобы логировать этапы инициализации хоста.

- **Текущий код:**

```
try
            {
                Log.Information("Starting web host");
                await host.RunAsync();
```

Строка 57

- **Комментарий:**

Использование .UseKestrel() избыточно, так как .ConfigureWebHostDefaults уже настраивает Kestrel по умолчанию. Удалите .UseKestrel() для упрощения кода.

- **Текущий код:**

```
.ConfigureWebHostDefaults(webBuilder => webBuilder.UseStartup<Startup>().UseKestrel());
        }
    }
}
```

Файл: MessengerBackend/Database/MessengerDBContext.cs

Строка 17

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class MessengerDBContext : DbContext
    {
        public MessengerDBContext(DbContextOptions<MessengerDBContext> options
            // , ILogger<MessengerDBContext> logger
```

Строка 29

- **Комментарий:**

Метод OnConfiguring не должен использоваться для настройки сущностей модели. Перенесите настройки сущностей в метод OnModelCreating.

- **Текущий код:**

```
optionsBuilder.UseSnakeCaseNamingConvention();
            NpgsqlConnection.GlobalTypeMapper.MapEnum<RoomType>();
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
```

Строка 33

- **Комментарий:**

Метод UseSnakeCaseNamingConvention не существует в DbContextOptionsBuilder. Используйте HasNamingConvention или настройте именование вручную.

- **Текущий код:**

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
```

```
        {
            modelBuilder
```

Строка 35

- **Комментарий:**

  Глобальное отображение типов с помощью NpgsqlConnection.GlobalTypeMapper не рекомендуется. Используйте modelBuilder.HasPostgresEnum вместо этого.

- **Текущий код:**

```
    {
            modelBuilder
                .HasPostgresEnum<RoomType>()
                .HasPostgresEnum<SubscriptionType>()
```

Строка 68

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    .HasValueGenerator<NowGenerator>();
                mb.Property(m => m.MessagePID).ValueGeneratedOnAdd()
                    .HasValueGenerator<PIDGenerator>();
                mb.HasOne(m => m.TargetRoom)
                    .WithMany(r => r.Messages)
```

Строка 92

- **Комментарий:**

  Дублирование индекса RoomPID. Удалите повторяющийся вызов HasIndex для RoomPID.

- **Текущий код:**

```
// modelBuilder.Entity<Bot>(b =>
            // {
            //     b.ToTable("bots").HasAlternateKey(e => e.BotUsername);
            //     b.Property(p => p.JoinedAt).ValueGeneratedOnAdd().HasValueGenerator<NowGenerato
            // });
```

Строка 103

- **Комментарий:**

Комментарий о том, что EF Core не поддерживает многие-ко-многим отношения в версии 3.0, устарел. EF Core 5.0 и выше поддерживают многие-ко-многим отношения напрямую.

- **Текущий код:**

```
rpb.HasOne(rp => rp.Room)
                    .WithMany(r => r.Participants)
                    .HasForeignKey(rp => rp.RoomID);
                rpb.HasOne(rp => rp.User)
                    .WithMany(u => u.RoomsParticipants)
```

Строка 136

- **Комментарий:**

Метод UseHiLo не используется для настройки сущностей модели. Возможно, имелось в виду использование HiLo для генерации ключей? Если да, то настройте его для конкретных сущностей или используйте HasSequence и HasDefaultValueSql.

- **Текущий код:**

```
public DbSet<Message> Messages { get; set; }
        public DbSet<Room> Rooms { get; set; }
        public DbSet<Session> Sessions { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<RoomParticipant> RoomParticipants { get; set; }
```

Строка 139

- **Комментарий:**

  Использование директивы #nullable disable и #nullable restore может быть избыточным, если проект уже настроен на использование nullable reference types. Убедитесь, что это необходимо.

- **Текущий код:**

```
public DbSet<User> Users { get; set; }
        public DbSet<RoomParticipant> RoomParticipants { get; set; }
        public DbSet<Subscription> Subscriptions { get; set; }
        public DbSet<Event> Events { get; set; }
```

Строка 142

- **Комментарий:**

  Закомментированный код может быть удален или перемещен в отдельный файл с документацией, если он нужен для будущих разработок.

- **Текущий код:**

```
public DbSet<Subscription> Subscriptions { get; set; }
        public DbSet<Event> Events { get; set; }
#nullable restore
    }
```

Строка 150

- **Комментарий:**

  Неправильный отступ класса. Используйте отступ в 4 пробела.

- **Текущий код:**

```
{
        public class PIDGenerator : ValueGenerator
        {
            public override bool GeneratesTemporaryValues => false;
```

Строка 156

- **Комментарий:**

  Неправильный отступ метода. Используйте отступ в 4 пробела.

- **Текущий код:**

```
protected override object NextValue(EntityEntry entry) => entry.Entity switch
            {
                User _ => CryptoService.GeneratePID("U"),
                Room _ => CryptoService.GeneratePID("R"),
                Message _ => CryptoService.GeneratePID("M"),
```

Строка 160

- **Комментарий:**

  Неправильный отступ внутри метода. Используйте отступ в 4 пробела.

- **Текущий код:**

```
Room _ => CryptoService.GeneratePID("R"),
                Message _ => CryptoService.GeneratePID("M"),
                Subscription _ => CryptoService.GeneratePID("S"),
                Event _ => CryptoService.GeneratePID("E"),
```

Строка 162

- **Комментарий:**

  Использование подчеркивания в шаблонном выражении switch может быть заменено на более понятное решение, например, на использование переменной, если она используется в будущем.

- **Текущий код:**

```
Subscription _ => CryptoService.GeneratePID("S"),
                Event _ => CryptoService.GeneratePID("E"),
                // Bot _ => CryptoService.GeneratePID("B"),
```

```
            _ => throw new ArgumentOutOfRangeException(
```

Строка 165

- **Комментарий:**

  То же самое применимо к этому и последующим шаблонам switch.

- **Текущий код:**

```
    // Bot _ => CryptoService.GeneratePID("B"),
                    _ => throw new ArgumentOutOfRangeException(
                        $"No PID generation available for {entry.Entity.GetType()}")
                };
```

Строка 171

- **Комментарий:**

  Пустая строка между объявлением класса и открывающей фигурной скобкой не нужна. Удалите ее.

- **Текущий код:**

```
    }

            public class RefreshTokenGenerator : ValueGenerator
            {
                public override bool GeneratesTemporaryValues => false;
```

Строка 174

- **Комментарий:**

  Избыточные пустые строки снижают читаемость кода. Оставьте только одну пустую строку между членами класса.

- **Текущий код:**

```
{
            public override bool GeneratesTemporaryValues => false;
```

Строка 176

- **Комментарий:**

  Закомментированный код лучше удалить или оставить с объяснением, если он нужен для будущих доработок.

- **Текущий код:**

```
public override bool GeneratesTemporaryValues => false;

            protected override object NextValue(EntityEntry entry) => CryptoService.GenerateRefres
        }
```

Строка 179

- **Комментарий:**

  Неправильный отступ строки с throw. Используйте отступ в 4 пробела.

- **Текущий код:**

```
protected override object NextValue(EntityEntry entry) => CryptoService.GenerateRefreshToken();
        }
```

Строка 181

- **Комментарий:**

  Использование стрелочного синтаксиса для метода NextValue делает его менее читаемым в данном контексте. Рассмотрите возможность использования традиционного блочного синтаксиса.

- **Текущий код:**

```
}
```

```
public class NowGenerator : ValueGenerator
{
```

Строка 184

- **Комментарий:**

  Неправильный отступ. Класс должен начинаться с отступа в 4 пробела.

- **Текущий код:**

```
    public class NowGenerator : ValueGenerator
        {
            public override bool GeneratesTemporaryValues => false;
            protected override object NextValue(EntityEntry entry) => DateTime.UtcNow;
```

Строка 189

- **Комментарий:**

  Используйте явный тип DateTime вместо object для возвращаемого значения метода NextValue для большей читаемости и типобезопасности.

- **Текущий код:**

```
    protected override object NextValue(EntityEntry entry) => DateTime.UtcNow;
        }
    }
}
```

Файл: MessengerBackend/Utils/EfficientInvoker.cs

Строка 42

- **Комментарий:**

Избыточные отступы перед открывающей фигурной скобкой класса. Удалите лишние пробелы.

- **Текущий код:**

```
{
        internal sealed class EfficientInvoker
        {
            private static readonly ConcurrentDictionary<ConstructorInfo, Func<object[], object>>
                ConstructorToWrapperMap
```

Строка 45

- **Комментарий:**

Избыточные отступы перед объявлением поля. Удалите лишние пробелы.

- **Текущий код:**

```
private static readonly ConcurrentDictionary<ConstructorInfo, Func<object[], object>>
                ConstructorToWrapperMap
                    = new ConcurrentDictionary<ConstructorInfo, Func<object[], object>>();
```

Строка 53

- **Комментарий:**

Избыточные отступы перед объявлением поля. Удалите лишние пробелы.

- **Текущий код:**

```
private static readonly ConcurrentDictionary<MethodKey, EfficientInvoker> MethodToWrapperMap
                = new ConcurrentDictionary<MethodKey, EfficientInvoker>(MethodKeyComparer.Instance

            private readonly Func<object, object[], object> _func;
```

Строка 59

- **Комментарий:**

Избыточные отступы перед объявлением поля. Удалите лишние пробелы.

- **Текущий код:**

```
public EfficientInvoker(Func<object, object[], object> func) => _func = func;

            public static Func<object[], object> ForConstructor(ConstructorInfo constructor)
            {
```

Строка 65

- **Комментарий:**

Избыточные отступы перед объявлением поля. Удалите лишние пробелы.

- **Текущий код:**

```
if (constructor == null)
                {
                    throw new ArgumentNullException(nameof(constructor));
                }
```

Строка 69

- **Комментарий:**

Избыточные отступы перед объявлением конструктора. Удалите лишние пробелы.

- **Текущий код:**

```
}

                return ConstructorToWrapperMap.GetOrAdd(constructor, t =>
                {
                    CreateParamsExpressions(constructor, out var argsExp, out var paramsExps);
```

Строка 73

- **Комментарий:**

Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
{
                    CreateParamsExpressions(constructor, out var argsExp, out var paramsExps);

                    var newExp = Expression.New(constructor, paramsExps);
                    var resultExp = Expression.Convert(newExp, typeof(object));
```

Строка 82

- **Комментарий:**

Метод GetOrAdd может выбросить исключение при попытке добавления значения. Рассмотрите возможность использования TryAdd или обработки исключения.

- **Текущий код:**

```
});
            }

            public static EfficientInvoker ForDelegate(Delegate del)
            {
```

Строка 95

- **Комментарий:**

Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
{
                    var method = del.GetMethodInfo();
                    var wrapper = CreateMethodWrapper(t, method, true);
                    return new EfficientInvoker(wrapper);
                });
```

Строка 104

- **Комментарий:**

  Метод GetType() устарел в пользу GetTypeInfo(). Рассмотрите возможность использования GetTypeInfo().

- **Текущий код:**

```
{
                if (type == null)
                {
                    throw new ArgumentNullException(nameof(type));
                }
```

Строка 114

- **Комментарий:**

  Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
}

                var key = new MethodKey(type, methodName);
                return MethodToWrapperMap.GetOrAdd(key, k =>
                {
```

Строка 128

- **Комментарий:**

  Избыточные отступы перед объявлением переменной. Удалите лишние пробелы.

- **Текущий код:**

```
if (type == null)
                {
                    throw new ArgumentNullException(nameof(type));
                }
```

Строка 133

- **Комментарий:**

  Метод GetMethod() может вернуть null, если метод не найден. Рассмотрите возможность проверки на null.

- **Текущий код:**

```
if (propertyName == null)
                {
                    throw new ArgumentNullException(nameof(propertyName));
                }
```

Строка 142

- **Комментарий:**

  Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
{
                    var wrapper = CreatePropertyWrapper(type, propertyName);
                    return new EfficientInvoker(wrapper);
                });
            }
```

Строка 153

- **Комментарий:**

  Избыточные отступы перед объявлением переменной. Удалите лишние пробелы.

- **Текущий код:**

```
var result = _func(target, args);
                if (!(result is Task task))
                {
                    return result;
                }
```

Строка 157

- **Комментарий:**

  Метод GetRuntimeProperty() может вернуть null, если свойство не найдено. Рассмотрите возможность проверки на null.

- **Текущий код:**

```
    return result;
                }

                if (!task.IsCompleted)
                {
```

Строка 167

- **Комментарий:**

  Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
    }

            public async Task<T> InvokeGenericAsync<T>(object target, params object[] args) =>
                _func(target, args) switch
                {
```

Строка 170

- **Комментарий:**

  Метод Invoke может выбросить исключение при неправильных аргументах. Рассмотрите возможность обработки исключения.

- **Текущий код:**

```
    _func(target, args) switch
                {
```

```
                Task<T> task => await task.ConfigureAwait(false),
                T sync => sync,
```

Строка 177

- **Комментарий:**

  Проверка IsCompleted не нужна перед await, так как await сам ждет завершения задачи.

- **Текущий код:**

```
    };

                public static Func<object, object[], object> CreateMethodWrapper(Type type, MethodInfo
                    bool isDelegate)
                {
```

Строка 183

- **Комментарий:**

  Метод GetResult() может выбросить исключение, если задача завершилась с ошибкой. Рассмотрите возможность использования await.

- **Текущий код:**

```
    CreateParamsExpressions(method, out var argsExp, out var paramsExps);

                    var targetExp = Expression.Parameter(typeof(object), "target");
                    var castTargetExp = Expression.Convert(targetExp, type);
                    var invokeExp = isDelegate
```

Строка 188

- **Комментарий:**

  Избыточные отступы перед выражением switch. Удалите лишние пробелы.

- **Текущий код:**

```
var invokeExp = isDelegate
                    ? (Expression) Expression.Invoke(castTargetExp, paramsExps)
                    : Expression.Call(castTargetExp, method, paramsExps);

                LambdaExpression lambdaExp;
```

Строка 194

- **Комментарий:**

Избыточные отступы перед кейсами switch. Удалите лишние пробелы.

- **Текущий код:**

```
if (method.ReturnType != typeof(void))
                {
                    var resultExp = Expression.Convert(invokeExp, typeof(object));
                    lambdaExp = Expression.Lambda(resultExp, targetExp, argsExp);
```

Строка 197

- **Комментарий:**

Избыточные отступы перед кейсами switch. Удалите лишние пробелы.

- **Текущий код:**

```
var resultExp = Expression.Convert(invokeExp, typeof(object));
                    lambdaExp = Expression.Lambda(resultExp, targetExp, argsExp);
                }
                else
```

Строка 200

- **Комментарий:**

Избыточные отступы перед кейсами switch. Удалите лишние пробелы.

- **Текущий код:**

```
    }
                else
                {
                    var constExp = Expression.Constant(null, typeof(object));
```

Строка 206

- **Комментарий:**

Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
    var blockExp = Expression.Block(invokeExp, constExp);
                    lambdaExp = Expression.Lambda(blockExp, targetExp, argsExp);
                }

                var lambda = lambdaExp.Compile();
```

Строка 213

- **Комментарий:**

Избыточные отступы перед объявлением переменной. Удалите лишние пробелы.

- **Текущий код:**

```
    }

            private static void CreateParamsExpressions(MethodBase method, out ParameterExpression
                out Expression[] paramsExps)
            {
```

Строка 218

- **Комментарий:**

Избыточные отступы перед тернарным оператором. Удалите лишние пробелы.

- **Текущий код:**

```
    {
                    var parameters = method.GetParameterTypes();

                    argsExp = Expression.Parameter(typeof(object[]), "args");
                    paramsExps = new Expression[parameters.Count];
```

Строка 240

- **Комментарий:**

Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
    var castPropExp = Expression.Convert(propExp, typeof(object));
                    var lambdaExp = Expression.Lambda(castPropExp, targetExp, argsExp);
                    var lambda = lambdaExp.Compile();
                    return (Func<object, object[], object>) lambda;
                }
```

Строка 246

- **Комментарий:**

Метод GetParameterTypes() не существует. Используйте method.GetParameters() и получите типы параметров из PropertyInfo[].

- **Текущий код:**

```
    private class MethodKeyComparer : IEqualityComparer<MethodKey>
                {
                    public static readonly MethodKeyComparer Instance = new MethodKeyComparer();
```

Строка 252

- **Комментарий:**

Избыточные отступы перед циклом for. Удалите лишние пробелы.

- **Текущий код:**

```
public bool Equals(MethodKey x, MethodKey y) =>
                    x.Type == y.Type &&
                    StringComparer.Ordinal.Equals(x.Name, y.Name);

                public int GetHashCode(MethodKey obj)
```

Строка 255

- **Комментарий:**

Избыточные отступы перед объявлением переменной. Удалите лишние пробелы.

- **Текущий код:**

```
public int GetHashCode(MethodKey obj)
                {
```

Строка 257

- **Комментарий:**

Избыточные отступы перед объявлением переменной. Удалите лишние пробелы.

- **Текущий код:**

```
public int GetHashCode(MethodKey obj)
                {
                    var typeCode = obj.Type.GetHashCode();
```

Строка 259

- **Комментарий:**

Избыточные отступы перед присваиванием значения. Удалите лишние пробелы.

- **Текущий код:**

```
{
                    var typeCode = obj.Type.GetHashCode();
                    var methodCode = obj.Name.GetHashCode();
```

Строка 261

- **Комментарий:**

  Избыточные отступы перед присваиванием значения. Удалите лишние пробелы.

- **Текущий код:**

```
var typeCode = obj.Type.GetHashCode();
                    var methodCode = obj.Name.GetHashCode();
                    return CombineHashCodes(typeCode, methodCode);
                }
```

Строка 265

- **Комментарий:**

  Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
return CombineHashCodes(typeCode, methodCode);
                }

                // From System.Web.Util.HashCodeCombiner
                private static int CombineHashCodes(int h1, int h2) => ((h1 << 5) + h1) ^ h2;
```

Строка 269

- **Комментарий:**

  Метод GetRuntimeProperty() может вернуть null, если свойство не найдено. Рассмотрите возможность проверки на null.

- **Текущий код:**

```
    // From System.Web.Util.HashCodeCombiner
                    private static int CombineHashCodes(int h1, int h2) => ((h1 << 5) + h1) ^ h2;
                }

                private struct MethodKey
```

Строка 274

- **Комментарий:**

  Использование оператора ! для подавления предупреждений может привести к ошибкам во время выполнения. Рассмотрите возможность проверки на null.

- **Текущий код:**

```
    private struct MethodKey
                {
                    public MethodKey(Type type, string name)
                    {
                        Type = type;
```

Строка 282

- **Комментарий:**

  Избыточные отступы перед объявлением вложенного класса. Удалите лишние пробелы.

- **Текущий код:**

```
    public readonly Type Type;
                    public readonly string Name;
                }
            }
```

Строка 287

- **Комментарий:**

Избыточные отступы перед объявлением статического поля. Удалите лишние пробелы.

- **Текущий код:**

```
}
    }

    internal static class EfficientExtensions
    {
```

Строка 290

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
internal static class EfficientExtensions
    {
        private const string CompleteTaskMessage = "Task must be complete";
```

Строка 292

- **Комментарий:**

Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
{
        private const string CompleteTaskMessage = "Task must be complete";

        private const string ResultPropertyName = "Result";
```

Строка 296

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    private const string ResultPropertyName = "Result";

            private static readonly Type GenericTaskType = typeof(Task<>);
```

Строка 298

- **Комментарий:**

Избыточные отступы перед возвращаемым значением. Удалите лишние пробелы.

- **Текущий код:**

```
    private static readonly Type GenericTaskType = typeof(Task<>);

            private static readonly ConcurrentDictionary<Type, bool> GenericTaskTypeMap =
```

Строка 301

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    private static readonly ConcurrentDictionary<Type, bool> GenericTaskTypeMap =
                new ConcurrentDictionary<Type, bool>();
```

Строка 303

- **Комментарий:**

Избыточные отступы перед объявлением метода. Удалите лишние пробелы.

- **Текущий код:**

```
    private static readonly ConcurrentDictionary<Type, bool> GenericTaskTypeMap =
                new ConcurrentDictionary<Type, bool>();

            internal static readonly ConcurrentDictionary<MethodBase, IReadOnlyList<Type>> ParameterMa
```

Строка 306

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    internal static readonly ConcurrentDictionary<MethodBase, IReadOnlyList<Type>> ParameterMap =
                new ConcurrentDictionary<MethodBase, IReadOnlyList<Type>>();
```

Строка 309

- **Комментарий:**

  Избыточные отступы перед объявлением переменной. Удалите лишние пробелы.

- **Текущий код:**

```
    new ConcurrentDictionary<MethodBase, IReadOnlyList<Type>>();

            public static EfficientInvoker GetMethodInvoker(this Type type, string methodName) =>
```

Строка 311

- **Комментарий:**

  Избыточные отступы перед возвращаемым значением. Удалите лишние пробелы.Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    public static EfficientInvoker GetMethodInvoker(this Type type, string methodName) =>
                EfficientInvoker.ForMethod(type, methodName);
```

Строка 315

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
EfficientInvoker.ForMethod(type, methodName);

        public static EfficientInvoker GetPropertyInvoker(this Type type, string propertyName) =>
            EfficientInvoker.ForProperty(type, propertyName);
```

Строка 318

- **Комментарий:**

  Избыточные отступы перед объявлением метода. Удалите лишние пробелы.Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
EfficientInvoker.ForProperty(type, propertyName);

        public static object GetResult(this Task task)
        {
```

Строка 322

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
public static object GetResult(this Task task)
        {
            if (task == null)
            {
                throw new ArgumentNullException(nameof(task));
```

Строка 325

- **Комментарий:**

  Избыточные отступы перед объявлением структуры. Удалите лишние пробелы.

- **Текущий код:**

```
    {
                    throw new ArgumentNullException(nameof(task));
                }
```

Строка 328

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    }

                if (!task.IsCompleted)
                {
```

Строка 331

- **Комментарий:**

  Избыточные отступы перед объявлением конструктора. Удалите лишние пробелы.

- **Текущий код:**

```
    if (!task.IsCompleted)
                {
                    throw new ArgumentException(CompleteTaskMessage, nameof(task));
```

Строка 333

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    {
                    throw new ArgumentException(CompleteTaskMessage, nameof(task));
                }
```

Строка 336

- **Комментарий:**

  Избыточные отступы перед присваиванием значения. Удалите лишние пробелы.

- **Текущий код:**

```
    }

                var type = task.GetType();
```

Строка 338

- **Комментарий:**

  Избыточные отступы перед присваиванием значения. Удалите лишние пробелы.Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    var type = task.GetType();
                var isGenericTaskType = GenericTaskTypeMap.GetOrAdd(type,
                    t => t.GetGenericTypeDefinition() == GenericTaskType);
```

Строка 341

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    var isGenericTaskType = GenericTaskTypeMap.GetOrAdd(type,
                    t => t.GetGenericTypeDefinition() == GenericTaskType);
```

Строка 343

- **Комментарий:**

  Избыточные отступы перед объявлением поля. Удалите лишние пробелы.

- **Текущий код:**

```
    t => t.GetGenericTypeDefinition() == GenericTaskType);

                return isGenericTaskType
                    ? type.GetPropertyInvoker(ResultPropertyName).Invoke(task)
```

Строка 346

- **Комментарий:**

  Избыточные отступы перед объявлением поля. Удалите лишние пробелы.

- **Текущий код:**

```
    return isGenericTaskType
                    ? type.GetPropertyInvoker(ResultPropertyName).Invoke(task)
                    : null;
            }
```

Строка 355

- **Комментарий:**

Получение типа через GetType() может быть заменено на более эффективное использование IsGenericType и GetGenericTypeDefinition() для проверки типа задачи.

- **Текущий код:**

```
return ParameterMap.GetOrAdd(method, c =>
                c.GetParameters().Select(p => p.ParameterType).ToArray());
        }
    }
```

Строка 360

- **Комментарий:**

Метод Invoke вызывается напрямую из GetPropertyInvoker, что может быть неэффективно. Рассмотрите возможность кэширования делегатов для свойств.Неправильное форматирование кода. Открывающая фигурная скобка должна быть на той же строке, что и объявление класса.

- **Текущий код:**

```
public class EfficientReflectionDelegate<T>
    {
        private readonly EfficientInvoker _invoker;
```

Строка 366

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.Неправильное форматирование кода. Открывающая фигурная скобка должна быть на той же строке, что и объявление конструктора.

- **Текущий код:**

```
protected EfficientReflectionDelegate(
            Type type,
            MethodInfo methodInfo,
            bool authenticated)
        {
```

Строка 369

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.Пустые строки между параметрами метода не нужны. Удалите лишние пробелы.

- **Текущий код:**

```
    bool authenticated)
            {
                Info = methodInfo;
```

Строка 371

- **Комментарий:**

  Метод GetParameters().Select(p => p.ParameterType).ToArray() может быть заменен на более эффективный метод GetParameterTypes(), если он доступен в вашей версии .NET.

- **Текущий код:**

```
    {
                Info = methodInfo;
                _invoker = new EfficientInvoker(EfficientInvoker
```

Строка 373

- **Комментарий:**

  Рассмотрите возможность использования Lazy<T> для кэширования параметров метода, чтобы избежать повторного вычисления.

- **Текущий код:**

```
    Info = methodInfo;
                _invoker = new EfficientInvoker(EfficientInvoker
                    .CreateMethodWrapper(type, methodInfo, false));
            }
```

Строка 378

- **Комментарий:**

  Неправильное форматирование кода. Открывающая фигурная скобка должна быть на той же строке, что и объявление конструктора.

- **Текущий код:**

```
    }

        public MethodInfo Info { get; }

        protected async Task<T> Invoke(object target, object[] parameters) =>
```

Строка 383

- **Комментарий:**

  Параметр authenticated не используется в конструкторе. Удалите его, если он не нужен.

- **Текущий код:**

```
    protected async Task<T> Invoke(object target, object[] parameters) =>
            await _invoker.InvokeGenericAsync<T>(target, parameters).ConfigureAwait(false);
    }
```

Строка 384

- **Комментарий:**

  Неправильное форматирование кода. Открывающая фигурная скобка должна быть на той же строке, что и объявление метода.

- **Текущий код:**

```
    await _invoker.InvokeGenericAsync<T>(target, parameters).ConfigureAwait(false);
    }
```

```
    }
```

Строка 385

- **Комментарий:**

  Публичное свойство Info должно быть readonly. Используйте readonly для свойств, которые устанавливаются только один раз.

- **Текущий код:**

```
    }
    }
```

Файл: MessengerBackend/Utils/PhoneNumberHelper.cs

Строка 9

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    {
        public class PhoneNumberHelper
        {
            private readonly PhoneNumberUtil _phoneNumberUtil = PhoneNumberUtil.GetInstance();
```

Строка 12

- **Комментарий:**

  Использование null в качестве регионального кода может привести к ошибкам. Рассмотрите возможность передачи регионального кода в метод ParseNumber.

- **Текущий код:**

```
    private readonly PhoneNumberUtil _phoneNumberUtil = PhoneNumberUtil.GetInstance();

            public string ParseNumber(string rawNumber)
            {
```

Строка 21

- **Комментарий:**

Передача null в метод Parse может вызвать исключение. Убедитесь, что rawNumber всегда содержит корректные данные или используйте регион по умолчанию.

- **Текущий код:**

```
    if (!_phoneNumberUtil.IsValidNumber(number))
                    {
                        throw new NumberParseException(
                            ErrorType.NOT_A_NUMBER, "Validation failed");
                    }
```

Строка 25

- **Комментарий:**

Исключение NumberParseException создается с общим сообщением об ошибке. Рассмотрите возможность передачи более детальной информации об ошибке.

- **Текущий код:**

```
    ErrorType.NOT_A_NUMBER, "Validation failed");
                    }

                    return _phoneNumberUtil.Format(number, PhoneNumberFormat.E164);
                }
```

Строка 34

- **Комментарий:**

Исключение InvalidNumberException создается на основе исключения NumberParseException. Убедитесь, что InvalidNumberException содержит все необходимые детали из исходного исключения.

- **Текущий код:**

```
}
        }
    }
}
```

Файл: MessengerBackend/Utils/PemReader.cs

Строка 34

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class PemReader : IDisposable
    {
        private static readonly int[] RsaIdentifier = { 1, 2, 840, 113549, 1, 1, 1 };
        private readonly bool _disposeStream;
```

Строка 48

- **Комментарий:**

Добавьте проверку на null для _stringReader в конструкторе PemReader(StringReader stringReader).

- **Текущий код:**

```
_encoding = encoding ?? Encoding.UTF8;
        }

        public PemReader(StringReader stringReader) => _stringReader = stringReader;
```

Строка 69

- **Комментарий:**

  Добавьте документацию для метода ReadRsaKey().

- **Текущий код:**

```
{
                throw new InvalidOperationException($"Header/footer format mismatch: {headerFormat
            }

            var derData = Convert.FromBase64String(parts.Body);
```

Строка 77

- **Комментарий:**

  Используйте оператор == для сравнения значений enum PemFormat вместо Equals.

- **Текущий код:**

```
if (headerFormat.Equals(PemFormat.Public))
            {
                return ReadPublicKey(der);
            }
```

Строка 91

- **Комментарий:**

  Используйте оператор == для сравнения значений enum PemFormat вместо Equals.

- **Текущий код:**

```
private PemParts ReadPemParts()
        {
            if (_stringReader != null)
            {
                return ExtractPemParts(_stringReader.ReadToEnd());
```

Строка 125

- **Комментарий:**

  Метод RemoveWhitespace() не является стандартным методом класса String. Убедитесь, что он определен или используйте другой подход для удаления пробелов.

- **Текущий код:**

```
    {
                    throw new InvalidOperationException($"Unrecognized {beginOrEnd}: {headerOrFooter}"
                }

                return PemFormat.Parse(match.Groups["format"].Value.Trim());
```

Строка 134

- **Комментарий:**

  Используйте StringComparison.OrdinalIgnoreCase для игнорирования регистра при сравнении строк.

- **Текущий код:**

```
    {
                if (der == null)
                {
                    throw new ArgumentNullException(nameof(der));
                }
```

Строка 142

- **Комментарий:**

  Метод Parse() не является стандартным методом для enum. Убедитесь, что он определен или используйте другой подход для парсинга строки в enum.

- **Текущий код:**

```
if (outerSequence == null)
            {
                throw new ArgumentException($"{nameof(der)} is not a sequence");
            }
```

Строка 165

- **Комментарий:**

  Используйте оператор is для проверки типа вместо приведения к типу и последующей проверки на null.

- **Текущий код:**

```
var objectIdentifier = headerSequence.Value[0] as DerAsnObjectIdentifier;
            if (objectIdentifier == null)
            {
                throw new InvalidOperationException("First part of header sequence must be an obje
            }
```

Строка 178

- **Комментарий:**

  Используйте оператор is для проверки типа вместо приведения к типу и последующей проверки на null.

- **Текущий код:**

```
if (!(headerSequence.Value[1] is DerAsnNull))
            {
                throw new InvalidOperationException("Second part of header sequence must be a null
            }
```

Строка 196

- **Комментарий:**

  Используйте оператор is для проверки типа вместо приведения к типу и последующей проверки на null.

- **Текущий код:**

```
if (innerSequence.Value.Length < 2)
            {
                throw new InvalidOperationException(
                    "Inner sequence must at least contain 2 parts (modulus and exponent)");
```

Строка 204

- **Комментарий:**

Используйте оператор is для проверки типа вместо приведения к типу и последующей проверки на null.

- **Текущий код:**

```
return new RSAParameters
            {
                Modulus = GetIntegerData(innerSequence.Value[0]),
                Exponent = GetIntegerData(innerSequence.Value[1])
            };
```

Строка 258

- **Комментарий:**

Используйте оператор is для проверки типа вместо приведения к типу и последующей проверки на null.

- **Текущий код:**

```
private class PemParts
        {
            public string Header { get; set; }
            public string Body { get; set; }
            public string Footer { get; set; }
```

Файл: MessengerBackend/Utils/NpgsqlLogProvider.cs

Строка 9

- **Комментарий:**

  Класс SerilogLoggingProvider объявлен как internal, что ограничивает его использование только внутри сборки. Если планируется использовать его за пределами сборки, следует изменить модификатор доступа на public.

- **Текущий код:**

```
namespace MessengerBackend.Utils
{
    internal class SerilogLoggingProvider : INpgsqlLoggingProvider
    {
```

Строка 16

- **Комментарий:**

  Конструктор использует стрелочную запись, что хорошо для краткости, но если тело конструктора усложнится, лучше использовать традиционный блок конструктора для улучшения читаемости.

- **Текущий код:**

```
    internal SerilogLoggingProvider(ILogger logger) => _logger = logger;

        public NpgsqlLogger CreateLogger(string name) => new NpgsqlSerilogLogger(name, _logger);
    }
```

Строка 20

- **Комментарий:**

  Метод CreateLogger использует стрелочную запись, что хорошо для краткости, но если тело метода усложнится, лучше использовать традиционный блок метода для улучшения читаемости.

- **Текущий код:**

```
}
```

```
internal class NpgsqlSerilogLogger : NpgsqlLogger
{
```

Строка 22

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
internal class NpgsqlSerilogLogger : NpgsqlLogger
    {
        private readonly ILogger _logger;
```

Строка 28

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
internal NpgsqlSerilogLogger(string name, ILogger logger)
        {
            NpgsqlLogManager.IsParameterLoggingEnabled = Serilog.Log.IsEnabled(LogEventLevel.Debug
            _logger = logger.ForContext<NpgsqlSerilogLogger>().ForContext("NpgsqlName", name);
        }
```

Строка 39

- **Комментарий:**

  Комментарий TODO FIXME не информативен. Уточните, что именно нужно сделать или исправить.

- **Текущий код:**

```
_logger.Write(ToSerilogLevel(level), exception, msg);
            }
        }

        public override bool IsEnabled(NpgsqlLogLevel level) => _logger.IsEnabled(ToSerilogLevel(l
```

Строка 42

- **Комментарий:**

  Используйте именованные аргументы для улучшения читаемости кода: _logger.Write(exception: exception, messageTemplate: msg, level: ToSerilogLevel(level));

- **Текущий код:**

```
public override bool IsEnabled(NpgsqlLogLevel level) => _logger.IsEnabled(ToSerilogLevel(level));

        private static LogEventLevel ToSerilogLevel(NpgsqlLogLevel level)
```

Строка 47

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
private static LogEventLevel ToSerilogLevel(NpgsqlLogLevel level)
        {
            return level switch
            {
                NpgsqlLogLevel.Debug => LogEventLevel.Debug,
```

Строка 59

- **Комментарий:**

  Исключение ArgumentOutOfRangeException лучше выбросить с использованием nameof(level) для улучшения отладки: throw new

ArgumentOutOfRangeException(nameof(level), level, "Level out of range");

- **Текущий код:**

```
};
        }
    }
}
```

Файл: MessengerBackend/Models/Session.cs

Строка 13

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
#nullable disable
    public class Session
    {
        [Required] public DateTime CreatedAt { get; set; }
```

Строка 16

- **Комментарий:**

Атрибут [Required] не имеет смысла для типа DateTime, так как он не может быть null. Удалите атрибут [Required] для свойства CreatedAt.

- **Текущий код:**

```
[Required] public DateTime CreatedAt { get; set; }

        [Required] public DateTime ExpiresAt { get; set; }
```

Строка 20

- **Комментарий:**

  Атрибут [Required] не имеет смысла для типа DateTime, так как он не может быть null. Удалите атрибут [Required] для свойства ExpiresAt.

- **Текущий код:**

```
[Required] public DateTime ExpiresAt { get; set; }

        public string Fingerprint { get; set; }

        [Column(TypeName = "bigint")] public long SessionID { get; set; }
```

Строка 27

- **Комментарий:**

  Атрибут [Column(TypeName = "bigint")] избыточен, если тип свойства long соответствует типу bigint в базе данных. Удалите атрибут [Column(TypeName = "bigint")] для свойства SessionID.

- **Текущий код:**

```
public byte[] IPHash { get; set; }

        [Required] public string RefreshToken { get; set; }

        [Required] public DateTime UpdatedAt { get; set; }
```

Строка 31

- **Комментарий:**

  Свойство IPHash имеет тип byte[], что может быть неудобно для хранения и обработки. Рассмотрите возможность использования строки или другого более удобного типа.

- **Текущий код:**

```
[Required] public DateTime UpdatedAt { get; set; }
```

```
        [Required] public User User { get; set; }
        public int UserID { get; set; }
```

Строка 35

- **Комментарий:**

  Атрибут [Required] не имеет смысла для типа string, если свойство может быть пустым. Удалите атрибут [Required] для свойства RefreshToken, если пустое значение допустимо.

- **Текущий код:**

```
    [Required] public User User { get; set; }
            public int UserID { get; set; }

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

Строка 39

- **Комментарий:**

  Свойство UserID является внешним ключом для свойства User, но оно не имеет атрибута [Required]. Добавьте атрибут [Required] для свойства UserID, чтобы указать, что оно не может быть null.

- **Текущий код:**

```
    public string UserAgent { get; set; }
        }
    #nullable enable
```

Строка 40

- **Комментарий:**

  Свойство User используется как навигационное свойство в Entity Framework, но оно не имеет атрибута [ForeignKey("UserID")]. Добавьте атрибут [ForeignKey("UserID")] для

свойства User.

- **Текущий код:**

```
}
#nullable enable
}
```

Строка 41

- **Комментарий:**

Атрибут [Required] не имеет смысла для типа DateTime, так как он не может быть null. Удалите атрибут [Required] для свойства UpdatedAt.

- **Текущий код:**

```
#nullable enable
}
```

Файл: MessengerBackend/Models/Message.cs

Строка 13

- **Комментарий:**

Ненужные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class Message
    {
        public int MessageID { get; set; }
```

Строка 19

- **Комментарий:**

  Атрибут [Column(TypeName = "char(11)")] может быть непереносимым. Рассмотрите возможность использования стандартных типов данных Entity Framework.

- **Текущий код:**

```
    [Column(TypeName = "char(11)")]
            [Required]
            public string MessagePID { get; set; }

            [Required] public virtual Room TargetRoom { get; set; }
```

Строка 24

- **Комментарий:**

  Свойство TargetRoom и TargetRoomID дублируют информацию. Удалите одно из свойств для избежания несоответствий данных.

- **Текущий код:**

```
    [Required] public virtual Room TargetRoom { get; set; }

            public int TargetRoomID { get; set; }

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

Строка 31

- **Комментарий:**

  Свойство SentAt не инициализировано по умолчанию. Рассмотрите возможность установки значения по умолчанию, например, DateTime.Now.

- **Текущий код:**

```
    public DateTime SentAt { get; set; }
            public Message ReplyTo { get; set; }
        }
    }
```

Строка 32

- **Комментарий:**

  Отсутствует атрибут [Required] для свойства Sender. Добавьте его, если поле обязательно.

- **Текущий код:**

```
public Message ReplyTo { get; set; }
    }
}
```

Файл: MessengerBackend/Models/User.cs

Строка 16

- **Комментарий:**

  Неправильное форматирование. Открывающая фигурная скобка должна быть на той же строке, что и объявление класса.

- **Текущий код:**

```
{
    public class User
    {
        public List<RoomParticipant> RoomsParticipants { get; set; }
```

Строка 19

- **Комментарий:**

  Использование List вместо ICollection для навигационного свойства может ограничивать гибкость в будущем. Рассмотрите возможность использования ICollection<RoomParticipant>.

- **Текущий код:**

```
public List<RoomParticipant> RoomsParticipants { get; set; }

        [NotMapped] public IEnumerable<Room> Rooms => RoomsParticipants?.Select(p => p.Room);
```

Строка 23

- **Комментарий:**

  Использование IEnumerable для свойства Rooms может привести к многократному выполнению запроса. Рассмотрите возможность использования ICollection<Room> или IReadOnlyCollection<Room>.

- **Текущий код:**

```
[NotMapped] public IEnumerable<Room> Rooms => RoomsParticipants?.Select(p => p.Room);


        public List<Session> Sessions { get; set; }
```

Строка 31

- **Комментарий:**

  Использование [Column(TypeName = "int")] для свойства UserID избыточно, так как int является типом по умолчанию в Entity Framework. Удалите атрибут [Column(TypeName = "int")]

- **Текущий код:**

```
[Column(TypeName = "varchar(32)")] public string Username { get; set; }

        public string AvatarUrl { get; set; }

        [Required]
```

Строка 35

- **Комментарий:**

Использование [Column(TypeName = "varchar(32)")] для свойства Username может быть избыточным, если длина строки уже ограничена другими средствами (например, валидацией). Рассмотрите возможность удаления атрибута [Column(TypeName = "varchar(32)")]

- **Текущий код:**

```
    [Required]
            [Column(TypeName = "varchar(18)")]
            public string Number { get; set; }
```

Строка 42

- **Комментарий:**

Использование [Column(TypeName = "varchar(18)")] для свойства Number может быть избыточным, если длина строки уже ограничена другими средствами (например, валидацией). Рассмотрите возможность удаления атрибута [Column(TypeName = "varchar(18)")]

- **Текущий код:**

```
    [Column(TypeName = "varchar(100)")] public string LastName { get; set; }

            [Column(TypeName = "varchar(256)")] public string Bio { get; set; }
```

Строка 50

- **Комментарий:**

Использование [Column(TypeName = "varchar(100)")] для свойства FirstName может быть избыточным, если длина строки уже ограничена другими средствами (например, валидацией). Рассмотрите возможность удаления атрибута [Column(TypeName = "varchar(100)")]

- **Текущий код:**

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

            public DateTime JoinedAt { get; set; }
        }
```

Строка 54

- **Комментарий:**

  Свойство JoinedAt не имеет атрибута [Required], что может привести к сохранению незаполненного значения. Рассмотрите возможность добавления атрибута [Required] или установки значения по умолчанию, например, DateTime.UtcNow.

- **Текущий код:**

```
}

    // Bots are on roadmap, however I am not implementing them now
```

Строка 55

- **Комментарий:**

  Использование [Column(TypeName = "char(11)")] для свойства UserPID может быть избыточным, если длина строки уже ограничена другими средствами (например, валидацией). Рассмотрите возможность удаления атрибута [Column(TypeName = "char(11)")]

- **Текущий код:**

```
// Bots are on roadmap, however I am not implementing them now
```

Строка 56

- **Комментарий:**

  Использование [Column(TypeName = "varchar(256)")] для свойства Bio может быть избыточным, если длина строки уже ограничена другими средствами (например, валидацией). Рассмотрите возможность удаления атрибута [Column(TypeName = "varchar(256)")]

- **Текущий код:**

```
// Bots are on roadmap, however I am not implementing them now
```

```
}
```

Строка 57

- **Комментарий:**

  Использование [Column(TypeName = "varchar(100)")] для свойства LastName может быть избыточным, если длина строки уже ограничена другими средствами (например, валидацией). Рассмотрите возможность удаления атрибута [Column(TypeName = "varchar(100)")]

- **Текущий код:**

```
}
```

Файл: MessengerBackend/Models/Event.cs

Строка 9

- **Комментарий:**

  Пустая строка после объявления класса не нужна.

- **Текущий код:**

```
namespace MessengerBackend.Models
{
    public class Event
    {
        public long EventID { get; set; }
```

Строка 12

- **Комментарий:**

  Пустая строка после открывающей фигурной скобки класса не нужна.

- **Текущий код:**

```
{
        public long EventID { get; set; }
        public string EventPID { get; set; }
        public DateTime DeliveredAt { get; set; }
```

Строка 16

- **Комментарий:**

Согласно общепринятым соглашениям об именовании в C#, свойства должны начинаться с заглавной буквы. Рекомендуется переименовать EventPID в EventPid.

- **Текущий код:**

```
public string EventPID { get; set; }
        public DateTime DeliveredAt { get; set; }

        [Required] public DateTime OccuredAt { get; set; }
```

Строка 21

- **Комментарий:**

Пустая строка между свойствами не нужна. Удалите лишние пустые строки.

- **Текущий код:**

```
[Required] public Message Message { get; set; }

        [Required] public Subscription Subscription { get; set; }
```

Строка 23

- **Комментарий:**

Атрибут [Required] используется для валидации данных в контексте Entity Framework или других ORM. Если это не требуется, уберите атрибут.

- **Текущий код:**

```
    [Required] public Subscription Subscription { get; set; }
            public long SubscriptionID { get; set; }
        }
```

Строка 26

- **Комментарий:**

Свойство SubscriptionID является избыточным, если Subscription уже содержит необходимый идентификатор. Удалите SubscriptionID, если оно не используется.

- **Текущий код:**

```
    public long SubscriptionID { get; set; }
        }
```

Строка 27

- **Комментарий:**

Атрибут [Required] используется для валидации данных в контексте Entity Framework или других ORM. Если это не требуется, уберите атрибут.

- **Текущий код:**

```
    public long SubscriptionID { get; set; }
        }
    }
```

Строка 28

- **Комментарий:**

Атрибут [Required] используется для валидации данных в контексте Entity Framework или других ORM. Если это не требуется, уберите атрибут.

- **Текущий код:**

```
}
}
```

Файл: MessengerBackend/Models/Subscription.cs

Строка 12

- **Комментарий:**

Ненужные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class Subscription
    {
        public long SubscriptionID { get; set; }
```

Строка 17

- **Комментарий:**

Ненужные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
public User User { get; set; }

        public Room Room { get; set; }
        public SubscriptionType Type { get; set; }
```

Строка 21

- **Комментарий:**

Ненужные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
public Room Room { get; set; }
        public SubscriptionType Type { get; set; }
        public DateTime SubscribedAt { get; set; }
        public DateTime LastEventOccuredAt { get; set; }
        public IEnumerable<Event> Events { get; set; }
```

Строка 29

- **Комментарий:**

Неправильное написание свойства. Используйте PascalCase для имен свойств. Правильное название: LastEventOccurredAt.

- **Текущий код:**

```
}

    public enum SubscriptionType
    {
        Message
```

Строка 32

- **Комментарий:**

Неправильный тип свойства. IEnumerable<Event> не подходит для хранения коллекции событий в модели. Рассмотрите использование ICollection<Event> или List<Event>.

- **Текущий код:**

```
{
        Message
    }
```

Строка 34

- **Комментарий:**

  Ненужные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
Message
    }
}
```

Файл: MessengerBackend/Models/Room.cs

Строка 15

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class Room
    {
        public int RoomID { get; set; }
        [Column(TypeName = "char(11)")] public string RoomPID { get; set; }
```

Строка 20

- **Комментарий:**

  Использование типа char(11) для строки может быть неэффективным и ограничивать возможности. Рассмотрите использование типа string без ограничений или varchar с необходимой длиной.

- **Текущий код:**

```
[Column(TypeName = "char(11)")] public string RoomPID { get; set; }
        public IEnumerable<Message> Messages { get; set; }
        public RoomType Type { get; set; }
        public IEnumerable<RoomParticipant> Participants { get; set; }
```

Строка 23

- **Комментарий:**

  Свойство Messages имеет тип IEnumerable<Message>, что может быть неудобно для добавления и удаления сообщений. Рассмотрите использование ICollection<Message> или List<Message>.

- **Текущий код:**

```
public IEnumerable<RoomParticipant> Participants { get; set; }

        [NotMapped]
        public IEnumerable<User> Users =>
```

Строка 28

- **Комментарий:**

  Свойство Participants имеет тип IEnumerable<RoomParticipant>, что также может быть неудобно для добавления и удаления участников. Рассмотрите использование ICollection<RoomParticipant> или List<RoomParticipant>.

- **Текущий код:**

```
public IEnumerable<User> Users =>
            Participants?.Select(p => p.User);

        public DateTime CreatedAt { get; set; }
        public string Link { get; set; }
```

Строка 34

- **Комментарий:**

  Свойство Users использует выражение с отложенным выполнением. Если это свойство будет использоваться несколько раз, это может привести к повторному выполнению запроса. Рассмотрите использование материализации запроса, например, с помощью .ToList().

- **Текущий код:**

```
    public string Name { get; set; }
            public string RoomAvatar { get; set; }
        }

        public enum RoomType
```

Строка 42

- **Комментарий:**

Свойство Link может быть необязательным. Рассмотрите использование типа string? или добавление атрибута [Required] в зависимости от бизнес-логики.

- **Текущий код:**

```
    Group,
            Channel
        }

        public class RoomParticipant
```

Строка 45

- **Комментарий:**

Свойство Name может быть необязательным. Рассмотрите использование типа string? или добавление атрибута [Required] в зависимости от бизнес-логики.Пустая строка после объявления класса не нужна. Удалите ее.

- **Текущий код:**

```
    public class RoomParticipant
        {
            public int UserID { get; set; }
```

Строка 48

- **Комментарий:**

  Свойство RoomAvatar может быть необязательным. Рассмотрите использование типа string? или добавление атрибута [Required] в зависимости от бизнес-логики.Пустая строка после открытия скобки класса не нужна. Удалите ее.

- **Текущий код:**

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

        [Required] public User User { get; set; }
```

Строка 52

- **Комментарий:**

  Пустая строка между свойствами не нужна. Удалите ее.

- **Текущий код:**

```
[Required] public User User { get; set; }

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

Строка 56

- **Комментарий:**

  Пустая строка между свойствами не нужна. Удалите ее.

- **Текущий код:**

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

        [Required] public Room Room { get; set; }

        public ParticipantRole Role { get; set; }
```

Строка 63

- **Комментарий:**

  Пустая строка между свойствами не нужна. Удалите ее.

- **Текущий код:**

```
public enum ParticipantRole
    {
        Creator,
        Participant
```

Строка 67

- **Комментарий:**

  Пустая строка перед закрытием скобки класса не нужна. Удалите ее.

- **Текущий код:**

```
Creator,
        Participant
    }
}
```

Файл: MessengerBackend/Policies/IPCheckRequirement.cs

Строка 10

- **Комментарий:**

  Класс IPCheckRequirement имеет публичное свойство IpClaimRequired с модификатором set что может привести к изменению состояния объекта после его создания. Рекомендуется сделать это свойство только для чтения, убрав модификатор set.

- **Текущий код:**

```
namespace MessengerBackend.Policies
{
    public class IPCheckRequirement : IAuthorizationRequirement
    {
```

Строка 13

- **Комментарий:**

  Избыточные пустые строки ухудшают читаемость кода. Удалите лишние пробелы.

- **Текущий код:**

```
public class IPCheckRequirement : IAuthorizationRequirement
    {
        public IPCheckRequirement(bool required) => IpClaimRequired = required;
        public bool IpClaimRequired { get; set; }
```

Строка 16

- **Комментарий:**

  Использование стрелочного синтаксиса для конструктора допустимо но может быть менее понятным для некоторых разработчиков. Рассмотрите возможность использования традиционного блочного синтаксиса для конструктора.

- **Текущий код:**

```
public IPCheckRequirement(bool required) => IpClaimRequired = required;
        public bool IpClaimRequired { get; set; }
    }
```

Строка 21

- **Комментарий:**

  Неправильное форматирование кода. Открывающая фигурная скобка должна быть на той же строке, что и объявление класса.

- **Текущий код:**

```
public class IPCheckHandler : AuthorizationHandler<IPCheckRequirement>
    {
        private readonly CryptoService _cryptoService;
```

Строка 27

- **Комментарий:**

  Неправильное форматирование кода. Параметры конструктора должны быть на одной строке без лишних пробелов.

- **Текущий код:**

```
public IPCheckHandler(IHttpContextAccessor httpContextAccessor, CryptoService cryptoService)
        {
            HttpContextAccessor = httpContextAccessor ?? throw new ArgumentNullException(nameof(ht
            _cryptoService = cryptoService;
        }
```

Строка 31

- **Комментарий:**

  Использование свойства HttpContextAccessor без приватного поля. Рекомендуется использовать приватное поле для хранения IHttpContextAccessor.

- **Текущий код:**

```
_cryptoService = cryptoService;
        }

        private IHttpContextAccessor HttpContextAccessor { get; }
        private HttpContext HttpContext => HttpContextAccessor.HttpContext;
```

Строка 41

- **Комментарий:**

  Неправильное форматирование кода. Слишком много пустых строк.

- **Текущий код:**

```
{
            var ipClaim = context.User.FindFirst(claim => claim.Type == "ip");
```

```
            // No claim existing set and and its configured as optional so skip the check
            if (ipClaim == null && !requirement.IpClaimRequired)
```

Строка 48

- **Комментарий:**

  Использование лямбда-выражения в методе FindFirst может быть заменено на константу ClaimTypes для улучшения читаемости и поддерживаемости кода.

- **Текущий код:**

```
    // This allows next Handle to succeed. If we call Fail() the access will be denied, even if handle
                // evaluated after this one do succeed
            {
                return Task.CompletedTask;
```

Строка 50

- **Комментарий:**

  Дублирование комментария. Первая часть комментария объясняет логику, а вторая часть является дублированием информации, уже содержащейся в коде. Удалите или упростите комментарий.

- **Текущий код:**

```
    {
                return Task.CompletedTask;
            }
```

Строка 61

- **Комментарий:**

  Проверка IP-адреса должна быть обернута в try-catch для обработки возможных исключений, связанных с HttpContext.Connection.RemoteIpAddress.

- **Текущий код:**

```
// Only call fail, to guarantee a failure, even if further handlers may succeed
            {
                context.Fail();
            }
```

Строка 67

- **Комментарий:**

Неверный возврат. Метод должен возвращать результат выполнения асинхронной задачи, а не Task.CompletedTask. В данном случае следует возвращать результат HandleRequirementAsync.

- **Текущий код:**

```
return Task.CompletedTask;
        }
    }
}
```

Файл: MessengerBackend/RealTime/Message.cs

Строка 9

- **Комментарий:**

Интерфейсы не должны содержать модификаторов доступа. Удалите public перед uint ID.

- **Текущий код:**

```
namespace MessengerBackend.RealTime
{
    public interface IMessage
    {
```

Строка 12

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
public interface IMessage
    {
        public uint ID { get; set; }
    }
```

Строка 17

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
}

    [DataContract]
    public class InboundMessage : IMessage
    {
```

Строка 45

- **Комментарий:**

  Поле Type должно быть закрытым. Предлагаю использовать свойство с закрытым полем.

- **Текущий код:**

```
[DataMember(Order = 0)] public OutboundMessageType Type;
        [DataMember(Order = 2)] public bool IsSuccess { get; set; }

        // [JsonConverter(typeof(JsonInt32Converter))]
        [DataMember(Order = 3)] public Dictionary<string, object?>? Data { get; set; }
```

Строка 53

- **Комментарий:**

  Использование Dictionary<string, object?> может привести к проблемам с сериализацией и читаемостью кода. Рассмотрите возможность создания специализированного класса для данных.

- **Текущий код:**

```
}

    public enum OutboundMessageType
    {
        Response,
```

Строка 57

- **Комментарий:**

  Тип uint может быть неудобен для некоторых платформ и сериализаторов. Рассмотрите возможность использования int.

- **Текущий код:**

```
{
        Response,
        Event
    }
}
```

Файл: MessengerBackend/RealTime/RealTimeServer.cs

Строка 14

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public sealed class RealTimeServer : IDisposable
    {
        private readonly CryptoService _cryptoService;
        private readonly ILogger<RealTimeServer> _logger;
```

Строка 22

- **Комментарий:**

Инициализация _random вне конструктора может привести к непредсказуемому поведению при многопоточном доступе. Рассмотрите возможность инициализации в конструкторе или использования ThreadSafeRandom.

- **Текущий код:**

```
private readonly CancellationTokenSource _shutdown = new CancellationTokenSource();

        public readonly ConcurrentDictionary<ulong, Connection> Connections =
            new ConcurrentDictionary<ulong, Connection>();
```

Строка 26

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
public readonly ConcurrentDictionary<ulong, Connection> Connections =
            new ConcurrentDictionary<ulong, Connection>();

        public RealTimeServer(CryptoService cryptoService, ILogger<RealTimeServer> logger)
        {
```

Строка 30

- **Комментарий:**

  Публичное поле Connections может быть уязвимым для внешнего изменения. Рассмотрите возможность использования свойства с приватным сеттером.

- **Текущий код:**

```
public RealTimeServer(CryptoService cryptoService, ILogger<RealTimeServer> logger)
        {
            _cryptoService = cryptoService;
            _logger = logger;
        }
```

Строка 36

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
public void Dispose()
        {
            _shutdown.Cancel();
            foreach (var connection in Connections)
```

Строка 53

- **Комментарий:**

  Анонимный метод внутри Connect может быть заменен на лямбду или отдельный метод для улучшения читаемости и поддерживаемости кода.

- **Текущий код:**

```
await conn.StartPolling();
        }
    }
```

Строка 54

- **Комментарий:**

  Использование оператора присваивания здесь может привести к ConcurrentDictionary.UpdateException, если два потока попытаются добавить запись с одинаковым ключом одновременно. Рассмотрите использование TryAdd.

- **Текущий код:**

```
    }
        }
```

Строка 55

- **Комментарий:**

  Использование NextDouble() для генерации уникального идентификатора может привести к коллизиям. Рассмотрите использование Guid или другого метода генерации уникальных идентификаторов.

- **Текущий код:**

```
    }
    }
```

Строка 56

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    }
```

Файл: MessengerBackend/RealTime/WebSocketMiddleware.cs

Строка 12

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class WebSocketMiddleware
    {
        private readonly ILogger<WebSocketMiddleware> _logger;
        private readonly RequestDelegate _next;
```

Строка 30

- **Комментарий:**

Параметр messageProcessService передается напрямую в метод InvokeAsync. Рассмотрите возможность внедрения этого сервиса через конструктор для соблюдения принципа Dependency Injection.

- **Текущий код:**

```
if (ctx.Request.Path.StartsWithSegments("/ws"))
            {
                if (ctx.WebSockets.IsWebSocketRequest)
                {
                    try
```

Строка 36

- **Комментарий:**

Используйте StringComparison.OrdinalIgnoreCase для метода StartsWithSegments, чтобы сделать проверку пути нечувствительной к регистру.

- **Текущий код:**

```
{
                        messageProcessService.Connections = _srv.Connections;
                        await _srv.Connect(await ctx.WebSockets.AcceptWebSocketAsync(),
                            messageProcessService);
                        await _next(ctx);
```

Строка 43

- **Комментарий:**

Присваивание _srv.Connections напрямую в методе InvokeAsync может привести к проблемам с потокобезопасностью. Рассмотрите возможность использования потокобезопасных коллекций или синхронизации.

- **Текущий код:**

```
catch (WebSocketException e) when (e.WebSocketErrorCode ==
                                                    WebSocketError.ConnectionClosedPrematurely)
                    {
                        await _next(ctx);
                    }
```

Строка 52

- **Комментарий:**

Вызов _next(ctx) в блоке catch для WebSocketException с кодом WebSocketError.ConnectionClosedPrematurely не нужен, так как соединение уже закрыто. Удалите этот вызов.

- **Текущий код:**

```
ctx.Response.StatusCode = 400;
                }
            }
            else
            {
```

Строка 59

- **Комментарий:**

  Вызов _next(ctx) для запросов, не начинающихся с "/ws", может быть нежелательным, если вы хотите обрабатывать только WebSocket-запросы. Убедитесь, что это поведение ожидаемо.

- **Текущий код:**

```
}
        }
    }
}
```

Файл: MessengerBackend/RealTime/Verification.cs

Строка 10

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
namespace MessengerBackend.RealTime
{
    public class VerificationBuilder
    {
```

Строка 12

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
public class VerificationBuilder
    {
```

```
        private readonly List<Func<object?, string?>> _argumentPredicates = new List<Func<object?,
        private readonly ILogger _logger;
```

Строка 21

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
{
            _message = message;
            _logger = logger;
        }
```

Строка 41

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
Argument<T>(null, false);

        public VerificationBuilder Argument<T>(Func<T, string?>? customRequirement, bool required)
        {
            var args = _argumentPredicates.Count + 1;
```

Строка 46

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
var args = _argumentPredicates.Count + 1;
```

```
if (required)
{
    _argumentPredicates.Add(arg =>
    {
```

Строка 52

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    if (arg == null)

                    {
                        return $"Argument {args} must be specified";
                    }
```

Строка 67

- **Комментарий:**

  Используйте is вместо GetType() для сравнения типов. Это более безопасно и эффективно.

- **Текущий код:**

```
    else
            {
                _argumentPredicates.Add(arg =>
                {
                    if (arg == null)
```

Строка 114

- **Комментарий:**

  Используйте is вместо GetType() для сравнения типов. Это более безопасно и эффективно.

- **Текущий код:**

```
_argumentPredicates.Add(arg =>
                {
                    if (arg == null)
                    {
                        return null;
```

Строка 144

- **Комментарий:**

Метод Build() должен возвращать результат проверки, но не должен записывать информацию об ошибке в лог. Логирование следует вынести за пределы этого метода.

- **Текущий код:**

```
for (var i = 0; i < _argumentPredicates.Count; i++)
            {
                object? arg;
                try
                {
```

Строка 150

- **Комментарий:**

Избыточное использование ToString() для перечисления. Можно использовать интерполяцию строк напрямую.

- **Текущий код:**

```
arg = _message.Params[i];
                }
                catch (ArgumentOutOfRangeException)
                {
```

Строка 152

- **Комментарий:**

  Избыточное использование ToString() для перечисления. Можно использовать интерполяцию строк напрямую.

- **Текущий код:**

```
    catch (ArgumentOutOfRangeException)
                    {
                        arg = null;
```

Строка 154

- **Комментарий:**

  Избыточное использование ToString() для перечисления. Можно использовать интерполяцию строк напрямую.

- **Текущий код:**

```
    {
                        arg = null;
                    }
```

Строка 164

- **Комментарий:**

  Исключение ArgumentOutOfRangeException может возникнуть только при некорректном использовании кода. Лучше проверять длину массива заранее.

- **Текущий код:**

```
    }
                }

                _logger.Verbose("Verification succeeded for message ID {MessageID}", _message.ID);
                return null;
```

Строка 169

- **Комментарий:**

  Исключение ArgumentOutOfRangeException может возникнуть только при некорректном использовании кода. Лучше проверять длину массива заранее.

- **Текущий код:**

```
return null;
        }
    }
}
```

Файл: MessengerBackend/Errors/Exceptions.cs

Строка 9

- **Комментарий:**

  Класс ApiErrorException следует сделать селективно доступным (internal), если он используется только внутри сборки. Это улучшит инкапсуляцию.

- **Текущий код:**

```
namespace MessengerBackend.Errors
{
    //TODO Organize codes
    public abstract class ApiErrorException : Exception
    {
```

Строка 14

- **Комментарий:**

  Поле Details должно быть доступно для чтения и записи (например, protected internal) или иметь свойство для доступа к нему, если требуется инкапсуляция.

- **Текущий код:**

```
{
        protected string? Details;
        public abstract int Code { get; }
        public abstract string Summary { get; }
        public abstract int HttpStatusCode { get; }
```

Строка 21

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
}

    public class InvalidNumberException : ApiErrorException
    {
        private readonly ErrorType _type;
```

Строка 24

- **Комментарий:**

  Использование new Dictionary<string, string>() в каждом вызове HttpHeaders может привести к созданию лишних экземпляров. Рассмотрите возможность использования readonly поле или инициализации в конструкторе.

- **Текущий код:**

```
{
        private readonly ErrorType _type;

        public InvalidNumberException(NumberParseException ex)
```

Строка 29

- **Комментарий:**

Конструктор InvalidNumberException(NumberParseException ex) может быть улучшен путем вызова базового конструктора ApiErrorException с соответствующими параметрами.

- **Текущий код:**

```
public InvalidNumberException(NumberParseException ex)
        {
            _type = ex.ErrorType;
            Details = ex.Message;
        }
```

Строка 37

- **Комментарий:**

Конструктор InvalidNumberException(string details) также может вызывать базовый конструктор ApiErrorException для обеспечения согласованности.

- **Текущий код:**

```
public override int Code => 1100;
        public override int HttpStatusCode => 400;
        public override string Summary => "Invalid Number";
```

Строка 48

- **Комментарий:**

Использование switch expression для формирования сообщения допустимо, но можно улучшить читаемость, удалив лишние пустые строки.

- **Текущий код:**

```
ErrorType.TOO_SHORT_AFTER_IDD => "Too short after IDD",
            ErrorType.INVALID_COUNTRY_CODE => "Invalid country code",
            _ => "Unknown Error"
        } + "; " + Details;
    }
```

Строка 53

- **Комментарий:**

  Пустая строка между объявлением класса и открывающей фигурной скобкой не требуется. Удалите пустую строку.

- **Текущий код:**

```
}

    public class TooManyRequestsException : ApiErrorException
    {
        public override int Code => 1200;
```

Строка 56

- **Комментарий:**

  Пустая строка между открывающей фигурной скобкой и первым свойством класса не требуется. Удалите пустую строку.

- **Текущий код:**

```
{
        public override int Code => 1200;
        public override int HttpStatusCode => 429;
        public override string Summary => "Too Many Requests";
```

Строка 59

- **Комментарий:**

  Пустая строка между свойствами класса не требуется. Удалите пустую строку.

- **Текущий код:**

```
public override int HttpStatusCode => 429;
        public override string Summary => "Too Many Requests";
    }
```

Строка 62

- **Комментарий:**

  Пустая строка между свойствами класса не требуется. Удалите пустую строку.

- **Текущий код:**

```
}

    public class WrongTokenException : ApiErrorException
```

Строка 64

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
public class WrongTokenException : ApiErrorException
    {
```

Строка 66

- **Комментарий:**

  Пустая строка между последним свойством и закрывающей фигурной скобкой не требуется. Удалите пустую строку.

- **Текущий код:**

```
public class WrongTokenException : ApiErrorException
    {
        private readonly string? _actualType;
```

Строка 68

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
{
        private readonly string? _actualType;
```

Строка 70

- **Комментарий:**

  Пустая строка после закрывающей фигурной скобки класса не требуется. Удалите пустую строку.

- **Текущий код:**

```
private readonly string? _actualType;

        private readonly string? _requiredType;
```

Строка 72

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
private readonly string? _requiredType;
```

Строка 74

- **Комментарий:**

  Конкатенация строк через + может быть заменена на string interpolation для улучшения читаемости кода.

- **Текущий код:**

```
    private readonly string? _requiredType;

            public WrongTokenException(string? requiredType, string? actualType)
```

Строка 76

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    public WrongTokenException(string? requiredType, string? actualType)
            {
                _requiredType = requiredType;
```

Строка 79

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
    {
                _requiredType = requiredType;
                _actualType = actualType;
            }
```

Строка 83

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
_actualType = actualType;
        }

        public override int Code => 3101;
        public override int HttpStatusCode => 403;
```

Строка 88

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
public override int HttpStatusCode => 403;
        public override string Summary => "Wrong token type";

        public override string Message
        {
```

Строка 96

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
if (_requiredType != null && _actualType != null)
                {
                    return "Type '" + _requiredType + "' was expected, instead got '" + _actualTyp
                }
```

Строка 100

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
}

                if (_requiredType != null && _actualType == null)
                {
                    return "Type '" + _requiredType + "' was expected";
```

Строка 105

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
return "Type '" + _requiredType + "' was expected";
                }

                if (_requiredType == null && _actualType != null)
                {
```

Строка 108

- **Комментарий:**

Используйте интерполяцию строк для улучшения читаемости кода. Пример: return $"Type '{_requiredType}' was expected, instead got '{_actualType}'";

- **Текущий код:**

```
if (_requiredType == null && _actualType != null)
                {
                    return "Type '" + _actualType + "' unexpected";
                }
```

Строка 113

- **Комментарий:**

Используйте интерполяцию строк для улучшения читаемости кода. Пример: return $"Type '{_requiredType}' was expected";

- **Текущий код:**

```
    }

                return "";
            }
        }
```

Строка 119

- **Комментарий:**

Используйте интерполяцию строк для улучшения читаемости кода. Пример: return $"Type '{_actualType}' unexpected";

- **Текущий код:**

```
    }

    #nullable disable
        public class TokenVerificationFailedException : ApiErrorException
        {
```

Строка 122

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    public class TokenVerificationFailedException : ApiErrorException
        {
            public TokenVerificationFailedException(string message) => Message = message;
```

Строка 125

- **Комментарий:**

  Конструктор с параметром message не вызывает базовый конструктор с сообщением. Используйте base(message) для передачи сообщения в базовый класс.

- **Текущий код:**

```
public TokenVerificationFailedException(string message) => Message = message;

        public TokenVerificationFailedException()
```

Строка 127

- **Комментарий:**

  Возвращайте более информативное сообщение, например, "Token type is not specified".

- **Текущий код:**

```
public TokenVerificationFailedException()
        {
        }
```

Строка 130

- **Комментарий:**

  Пустой конструктор без параметров не нужен, если у вас уже есть конструктор с параметром message. Удалите его, если он не используется.

- **Текущий код:**

```
{
        }
```

Строка 132

- **Комментарий:**

  Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
}

        public override int Code => 3102;
        public override int HttpStatusCode => 403;
```

Строка 140

- **Комментарий:**

  Свойство Message переопределяет только геттер, что может привести к ошибкам, так как базовый класс ApiErrorException ожидает, что это свойство будет доступно для записи. Переопределите свойство с использованием базового конструктора или сделайте его доступным для записи.

- **Текущий код:**

```
public override string Message { get; }
    }
#nullable restore
}
```

Файл: MessengerBackend/Controllers/UserController.cs

Строка 17

- **Комментарий:**

  Наследование от Controller вместо ControllerBase для API контроллеров. Рекомендуется использовать ControllerBase для уменьшения размера ответа и исключения лишних функций представления.

- **Текущий код:**

```
public class UserController : Controller
    {
```

```
    private readonly UserService _userService;

    public UserController(UserService userService) => _userService = userService;
```

Строка 35

- **Комментарий:**

  Использование SingleOrDefaultAsync без обработки возможного исключения ArgumentNullException. Рекомендуется проверять наличие HttpContext.User.FindFirst("uid").Value перед использованием.

- **Текущий код:**

```
return Ok(new
            {
                username = user.Username ?? "",
                firstName = user.FirstName,
                lastName = user.LastName,
```

Строка 43

- **Комментарий:**

  Возврат анонимного типа из контроллера. Рекомендуется создать DTO (Data Transfer Object) для структурированного возврата данных.

- **Текущий код:**

```
avatarUrl = user.AvatarUrl ?? ""
            });
        }

        [HttpPost("me")]
```

Строка 61

- **Комментарий:**

Метод принимает параметры напрямую из строки запроса, что не является хорошей практикой для больших или сложных объектов. Рекомендуется использовать модель для передачи данных.

- **Текущий код:**

```
{
                return NotFound();
            }

            if (firstName != null)
```

Строка 70

- **Комментарий:**

Использование SingleOrDefaultAsync без обработки возможного исключения ArgumentNullException. Рекомендуется проверять наличие HttpContext.User.FindFirst("uid").Value перед использованием.

- **Текущий код:**

```
if (lastName != null)
            {
                user.LastName = lastName;
            }
```

Строка 92

- **Комментарий:**

Возврат Forbid() в случае неудачного сохранения пользователя. Рекомендуется возвращать BadRequest или другое подходящее сообщение об ошибке.

- **Текущий код:**

```
return Forbid();
        }
    }
}
```

Файл: MessengerBackend/Controllers/ErrorController.cs

Строка 6

- **Комментарий:**

  Использование абсолютного маршрута не рекомендуется. Лучше использовать атрибут [Route] на уровне контроллера без начального слеша.

- **Текущий код:**

```
namespace MessengerBackend.Controllers
{
    [Route("/error/")]
    public class ErrorController : Controller
```

Строка 13

- **Комментарий:**

  Использование относительного маршрута без начального слеша может быть более понятным. Рассмотрите вариант [Route("404")] без начального слеша.

- **Текущий код:**

```
[Route("404")]
        public IActionResult PageNotFound() =>
            View();
    }
```

Строка 15

- **Комментарий:**

  Рассмотрите возможность передачи модели в представление для более гибкой обработки ошибок.

- **Текущий код:**

```
View();
    }
}
```

Строка 16

- **Комментарий:**

  Метод PageNotFound может быть улучшен, если будет явно указывать, что это обработка ошибки 404. Рассмотрите использование [HttpGet("404")] и возврат NotFound().

- **Текущий код:**

```
}
}
```

Файл: MessengerBackend/Controllers/AuthController.cs

Строка 17

- **Комментарий:**

  Используйте [Route("api/auth")] без начального слэша для согласованности с соглашениями маршрутизации ASP.NET Core.

- **Текущий код:**

```
namespace MessengerBackend.Controllers
{
    [Route("/api/auth")]
    [ApiController]
```

Строка 22

- **Комментарий:**

Класс AuthController наследуется от Controller, что не подходит для API-контроллеров. Наследуйтесь от ControllerBase.

- **Текущий код:**

```
[ApiController]
    public class AuthController : Controller
    {
        private readonly AuthService _authService;
        private readonly CryptoService _cryptoService;
```

Строка 31

- **Комментарий:**

Создание экземпляра PhoneNumberHelper напрямую в контроллере не соответствует принципу внедрения зависимостей. Рассмотрите возможность внедрения PhoneNumberHelper через конструктор.

- **Текущий код:**

```
private readonly VerificationService _verificationService;

        public AuthController(
            UserService userService,
            VerificationService verificationService,
```

Строка 94

- **Комментарий:**

Проверка на null и вызов ToString() для StringValues не имеет смысла. Используйте fingerprint.FirstOrDefault() для получения значения.

- **Текущий код:**

```
var newUser = await _userService.AddUserAsync(
                HttpContext.User.FindFirst("num").Value, firstName, lastName, username);

            if (newUser == null)
            {
```

Строка 133

- **Комментарий:**

  Проверка на null и вызов ToString() для StringValues не имеет смысла. Используйте fingerprint.FirstOrDefault() для получения значения.

- **Текущий код:**

```
    {
                    return Forbid();
                }

                var fingerprint = Request.Headers["X-Fingerprint"];
```

Строка 172

- **Комментарий:**

  Проверка на null и вызов ToString() для StringValues не имеет смысла. Используйте fingerprint.FirstOrDefault() для получения значения.

- **Текущий код:**

```
    return Forbid();
                }

                if (session.Fingerprint != null && session.Fingerprint !=
                    (StringValues.IsNullOrEmpty(fingerprint) ? fingerprint.ToString() : null)
```

Строка 182

- **Комментарий:**

  Используйте DateTime.UtcNow вместо DateTime.Now для обеспечения согласованности времени в различных часовых поясах.

- **Текущий код:**

```
    return Forbid();
                }
```

```
                if (session.ExpiresAt >= DateTime.UtcNow)
                {
```

Строка 205

- **Комментарий:**

  Используйте DateTime.UtcNow вместо DateTime.Now для обеспечения согласованности времени в различных часовых поясах.

- **Текущий код:**

```
    });
            }
        }
    }
```

Строка 206

- **Комментарий:**

  Проверка на null и вызов ToString() для StringValues не имеет смысла. Используйте fingerprint.FirstOrDefault() для получения значения.

- **Текущий код:**

```
    }
        }
    }
```

Файл: MessengerBackend/Controllers/GeneralController.cs

Строка 11

- **Комментарий:**

Класс наследуется от Controller вместо ControllerBase. Для API контроллеров рекомендуется использовать ControllerBase.

- **Текущий код:**

```
[ApiController]
    public class GeneralController : Controller
    {
        private readonly CryptoService _cryptoService;
```

Строка 24

- **Комментарий:**

Метод Index возвращает представление (View), что не соответствует RESTful API стилю. Для API контроллеров рекомендуется возвращать данные (например, JSON).

- **Текущий код:**

```
[return: Description("Gets RSA JWT Public Key in the PKCS#1 format")]
        [HttpGet("publicKey")]
        // Same
        public IActionResult GetPublicKey()
        {
```

Строка 31

- **Комментарий:**

Комментарий 'Same' не информативен. Удалите или замените на более осмысленный комментарий.

- **Текущий код:**

```
}
#endif
    }
}
```

Файл: MessengerBackend/Services/VerificationService.cs

Строка 20

- **Комментарий:**

  Поле ResendInterval должно быть private или protected, а не public. Предлагаю сделать его private.

- **Текущий код:**

```
public readonly ServiceResource TwilioService;

        public VerificationService(IConfiguration configuration, ILogger<VerificationService> logg
        {
            _logger = logger;
```

Строка 23

- **Комментарий:**

  Поле TwilioService должно быть private или protected, а не public. Предлагаю сделать его private.

- **Текущий код:**

```
{
            _logger = logger;
            ResendInterval = configuration.GetValue<double>("SMSVerification:ResendInterval");
            _twilioConfig = new TwilioConfig(
```

Строка 43

- **Комментарий:**

  Использование стрелочного синтаксиса для метода StartVerificationAsync может быть менее читаемым. Предлагаю использовать традиционный синтаксис метода.

- **Текущий код:**

```
pathServiceSid: _twilioConfig.ServiceSid
            );
```

```
        // {
```

Строка 64

- **Комментарий:**

  Закомментированный код должен быть удален или оставлен для будущего использования с соответствующими комментариями.

- **Текущий код:**

```
    var verificationCheckResource = await VerificationCheckResource.CreateAsync(
                    to: phoneNumber,
                    code: code,
                    pathServiceSid: _twilioConfig.ServiceSid
                );
```

Строка 83

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    public class TwilioConfig
        {
            public TwilioConfig(string accountSid, string authToken, string serviceSid)
            {
```

Строка 87

- **Комментарий:**

  Метод CheckVerificationAsync не обрабатывает исключения. Рекомендуется добавить обработку исключений, аналогично закомментированному коду.

- **Текущий код:**

```
public TwilioConfig(string accountSid, string authToken, string serviceSid)
        {
            AccountSid = accountSid;
            AuthToken = authToken;
            ServiceSid = serviceSid;
```

Строка 90

- **Комментарий:**

  Используйте автосвойства для упрощения кода. Автосвойства не требуют явного определения полей для хранения значений.

- **Текущий код:**

```
AuthToken = authToken;
            ServiceSid = serviceSid;
        }
```

Строка 96

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
public string AccountSid { get; set; }
        public string AuthToken { get; set; }
        public string ServiceSid { get; set; }
    }
```

Строка 98

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

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

Строка 99

- **Комментарий:**

Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    }
    }
```

Строка 100

- **Комментарий:**

Сравнение строк с использованием оператора == может быть ненадежным из-за чувствительности к регистру. Рекомендуется использовать StringComparison.OrdinalIgnoreCase.

- **Текущий код:**

```
    }
    }
```

Файл: MessengerBackend/Services/UserService.cs

Строка 12

- **Комментарий:**

Ненужные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class UserService
    {
        private readonly MessengerDBContext _dbContext;
```

Строка 20

- **Комментарий:**

Публичное свойство Users может быть небезопасным, так как оно предоставляет прямой доступ к DbSet. Рассмотрите возможность использования методов для работы с пользователями.

- **Текущий код:**

```
public async Task<User?> AddUserAsync(string number, string firstName, string lastName, string use
        {
            var newUser = new User
            {
```

Строка 24

- **Комментарий:**

Метод AddUserAsync возвращает null в случае ошибки уникальности. Это может быть непонятно для вызывающего кода. Рассмотрите возможность выброса исключения или возврата результата операции через специальный объект (например, Result<User>).

- **Текущий код:**

```
var newUser = new User
            {
                Number = number,
                FirstName = firstName,
                LastName = lastName,
```

Строка 37

- **Комментарий:**

  Метод AddAsync возвращает EntityEntry, который затем используется для получения добавленной сущности. Это лишнее присваивание. Можно сразу вернуть newUser после добавления.

- **Текущий код:**

```
    }
            catch (DbUpdateException e)
                when ((e.InnerException as PostgresException)?.SqlState == PostgresErrorCodes.Uniq
            {
                return null;
```

Строка 49

- **Комментарий:**

  Метод SaveUserAsync возвращает bool, что может быть недостаточно информативно для вызывающего кода. Рассмотрите возможность возврата результата операции через специальный объект (например, Result<bool>).

- **Текущий код:**

```
    {
                _dbContext.Users.Attach(user);
                await _dbContext.SaveChangesAsync();
                return true;
            }
```

Строка 60

- **Комментарий:**

  Использование метода EqualsAnyString не является стандартным в C#. Рассмотрите возможность использования Contains или других стандартных методов для проверки наличия значения в списке.

- **Текущий код:**

```
    return false;
            }
        }
```

```
    }
}
```

Файл: MessengerBackend/Services/CryptoService.cs

Строка 27

- **Комментарий:**

Использование препроцессорных директив (#if, #elif) может затруднить поддержку кода. Рассмотрите возможность использования конфигурации для выбора алгоритма.

- **Текущий код:**

```
public JwtBuilder JwtBuilder => new JwtBuilder()
            .WithAlgorithm(new RS256Algorithm(PublicKey, PrivateKey))
            .Issuer(JwtOptions.Issuer)
            .Audience(JwtOptions.Audience);
```

Строка 39

- **Комментарий:**

Поля PrivateKey и PublicKey объявлены как public readonly, что может быть небезопасно. Рассмотрите возможность использования private полей и предоставления методов для доступа к ним.

- **Текущий код:**

```
.WithSecret(_hmacKey)
            .Issuer(JwtOptions.Issuer)
            .Audience(JwtOptions.Audience);

        private readonly string _hmacKey;
```

Строка 43

- **Комментарий:**

  Поля PrivateKey и PublicKey объявлены как public readonly, что может быть небезопасно. Рассмотрите возможность использования private полей и предоставления методов для доступа к ним.

- **Текущий код:**

```
    private readonly string _hmacKey;
    #endif
            private readonly JwtSecurityTokenHandler _jwtSecurityTokenHandler = new JwtSecurityTokenHa
```

Строка 59

- **Комментарий:**

  Поле _hmacKey объявлено как private readonly, но используется в статическом контексте. Рассмотрите возможность использования конфигурации или передачи ключа через параметры методов.

- **Текущий код:**

```
    #if USERSA
                using (var sr = new StringReader(configuration["JWT:RSAPublicKey"]))
                {
                    PublicKey = new RSACryptoServiceProvider();
```

Строка 78

- **Комментарий:**

  Конструктор принимает IConfiguration, но не использует его для всех возможных алгоритмов. Убедитесь, что все необходимые параметры конфигурации доступны.

- **Текущий код:**

```
    ValidateLifetime = true,
                    ValidateIssuerSigningKey = true,
                    ValidateIssuer = true,
                    ValidIssuer = JwtOptions.Issuer,
                    ValidateAudience = true,
```

Строка 110

- **Комментарий:**

  Использование cryptoService.PublicKey в статическом контексте вызовет ошибку, так как cryptoService не определен. Используйте this.PublicKey.

- **Текущий код:**

```
    .ExpirationTime(DateTime.UtcNow.AddMinutes(JwtOptions.AccessTokenLifetimeMinutes))
                   .Encode();

            public string CreateAuthJwt(IPAddress ip, string number) =>
                JwtBuilder
```

Строка 140

- **Комментарий:**

  Использование _sha256 в статическом методе CreateAccessJwt вызовет ошибку, так как _sha256 является экземплярным полем. Передавайте _sha256 через параметры метода или сделайте его статическим.

- **Текущий код:**

```
    public static uint RandomUint()
            {
                var buf = new byte[4];
                Rng.GetNonZeroBytes(buf);
                return BitConverter.ToUInt32(buf);
```

Строка 160

- **Комментарий:**

  Использование _sha256 в статическом методе CreateAuthJwt вызовет ошибку, так как _sha256 является экземплярным полем. Передавайте _sha256 через параметры метода или сделайте его статическим.

- **Текущий код:**

```
    public static readonly int RefreshTokenLength = 24;
        }

        public TokenValidationParameters ValidationParameters { get; }
```

Строка 168

- **Комментарий:**

  Метод ValidateAccessJWT возвращает ClaimsPrincipal, но не обрабатывает возможные исключения. Рассмотрите возможность добавления обработки исключений для улучшения надежности.

- **Текущий код:**

```
    private const int PIDLength = 10;
    }
}
```

Строка 169

- **Комментарий:**

  Метод RandomUint использует GetNonZeroBytes, что может привести к некорректным значениям, так как GetNonZeroBytes не гарантирует равномерное распределение. Используйте GetBytes для генерации случайных чисел.

- **Текущий код:**

```
    private const int PIDLength = 10;
    }
}
```

Файл: MessengerBackend/Services/ChatService.cs

Строка 12

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class ChatService
    {
        private readonly MessengerDBContext _dbContext;
        public ChatService(MessengerDBContext dbContext) => _dbContext = dbContext;
```

Строка 27

- **Комментарий:**

  Метод AddAsync возвращает EntityEntry, а не добавленный объект. Используйте room вместо nRoom.Entity.

- **Текущий код:**

```
Room = room,
                User = user
            });
            await _dbContext.SaveChangesAsync();
            return nRoom.Entity;
```

Строка 64

- **Комментарий:**

  Используйте более конкретные имена переменных для улучшения читаемости кода. Например, subscription вместо sub.

- **Текущий код:**

```
_dbContext.Subscriptions.Remove(sub);
            await _dbContext.SaveChangesAsync();
            return sub;
        }
```

Строка 94

- **Комментарий:**

  Метод DeliverEvent может выбросить NullReferenceException, если событие не найдено. Добавьте проверку на null перед использованием ev.

- **Текущий код:**

```
ev.DeliveredAt = DateTime.UtcNow;
            _dbContext.Attach(ev);
            await _dbContext.SaveChangesAsync();
            return ev;
        }
```

Строка 98

- **Комментарий:**

  Метод Attach используется для отслеживания существующей сущности, которая не была загружена из базы данных. В данном случае лучше использовать Update для изменения существующей сущности.

- **Текущий код:**

```
return ev;
        }
    }
}
```

Файл: MessengerBackend/Services/MessageProcessService.cs

Строка 18

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class MessageProcessService
    {
        private readonly Random _rng = new Random();
        public readonly ChatService ChatService;
```

Строка 23

- **Комментарий:**

Поля ChatService и UserService должны быть readonly, но они инициализируются в конструкторе. Объявите их как private readonly.

- **Текущий код:**

```
public readonly ChatService ChatService;
        public readonly UserService UserService;

        public MessageProcessService(UserService userService, ChatService chatService)
        {
```

Строка 36

- **Комментарий:**

Использование null! может привести к NullReferenceException. Лучше инициализировать свойства в конструкторе или использовать явную проверку на null.

- **Текущий код:**

```
// All public methods here are callable by the websocket Method property
        // They must return Task<OutboundMessage> or OutboundMessage

        public OutboundMessage Echo(string data) => new OutboundMessage
```

Строка 39

- **Комментарий:**

То же самое применимо к свойству Current. Лучше инициализировать его в конструкторе или использовать явную проверку на null.

- **Текущий код:**

```
public OutboundMessage Echo(string data) => new OutboundMessage
        {
            Data = new Dictionary<string, object?> { { "Echo", data } },
```

Строка 74

- **Комментарий:**

Использование Current.CurrentUser! может привести к NullReferenceException. Убедитесь, что Current.CurrentUser всегда инициализирован или используйте безопасное обращение.

- **Текущий код:**

```
var targetUser = await
                UserService.Users.SingleOrDefaultAsync(u => u.UserPID == targetPID);
            if (targetUser == null)
            {
                return Fail("User not found");
```

Строка 93

- **Комментарий:**

Использование Current.CurrentUser! может привести к NullReferenceException. Убедитесь, что Current.CurrentUser всегда инициализирован или используйте безопасное обращение.

- **Текущий код:**

```
{
                Name = name,
                CreatedAt = DateTime.UtcNow,
                Type = RoomType.Channel,
                Link = link
```

Строка 147

- **Комментарий:**

  Использование Current.CurrentUser может привести к NullReferenceException. Убедитесь, что Current.CurrentUser всегда инициализирован или используйте безопасное обращение.

- **Текущий код:**

```
    await ChatService.DeliverEvent(ev)
                        );
                        await connection.OutboundMessages.Writer.WriteAsync(new OutboundMessage
                        {
                            Type = OutboundMessageType.Event,
```

Строка 154

- **Комментарий:**

  Условие room == null всегда будет false, так как room уже проверен на null выше. Удалите избыточное условие.

- **Текущий код:**

```
    {
                                    { "eventType", "message" },
                                    { "room", roomPID },
                                    { "subscription", sub.SubscriptionPID },
                                    { "replyTo", null },
```

Строка 170

- **Комментарий:**

  Использование CurrentMessageID без инициализации может привести к неожиданным результатам. Убедитесь, что CurrentMessageID всегда инициализирована перед использованием.

- **Текущий код:**

```
{ "messagePID", msg.MessagePID }
            });
        }

        private static OutboundMessage Fail(string error) => new OutboundMessage
```

Строка 192

- **Комментарий:**

  Метод Success() без параметра data может быть избыточным. Лучше использовать только один метод Success с параметром по умолчанию.

- **Текущий код:**

```
Type = OutboundMessageType.Response
        };
    }
}
```

Файл: MessengerBackend/Services/AuthService.cs

Строка 14

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
{
    public class AuthService
    {
        private readonly MessengerDBContext _dbContext;
```

Строка 20

- **Комментарий:**

  Конструктор AuthService принимает IConfiguration, но он не используется. Удалите этот параметр, если он не нужен.

- **Текущий код:**

```
public AuthService(MessengerDBContext dbContext, IConfiguration config) => _dbContext = dbContext;

        public Task<Session> GetSessionAsync(string token) =>
            _dbContext.Sessions.Where(s => s.RefreshToken == token).SingleOrDefaultAsync();
```

Строка 24

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
_dbContext.Sessions.Where(s => s.RefreshToken == token).SingleOrDefaultAsync();

        public async Task<Session?> GetAndDeleteSessionAsync(string token)
        {
            var session = await GetSessionAsync(token).ConfigureAwait(false);
```

Строка 30

- **Комментарий:**

  Метод GetAndDeleteSessionAsync возвращает nullable Task<Session?>. Рассмотрите возможность использования Task<Session> и выброса исключения, если сессия не найдена.

- **Текущий код:**

```
if (session == null)
            {
                return null;
            }
```

Строка 33

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    }

            _dbContext.Sessions.Remove(session);
            await _dbContext.SaveChangesAsync();
```

Строка 36

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    _dbContext.Sessions.Remove(session);
            await _dbContext.SaveChangesAsync();
            return session;
        }
```

Строка 42

- **Комментарий:**

  Метод Remove не возвращает значение, поэтому нет необходимости сохранять результат в переменную s.

- **Текущий код:**

```
    public async Task<EntityEntry<Session>> AddSessionAsync(Session session)
        {
            var s = await _dbContext.Sessions.AddAsync(session);
            await _dbContext.SaveChangesAsync();
```

Строка 48

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    return s;
            }
        }
```

Строка 49

- **Комментарий:**

  Переменная s используется только для возврата значения. Рассмотрите возможность возврата session напрямую.

- **Текущий код:**

```
    }
        }
```

Строка 50

- **Комментарий:**

  Избыточные пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
    }
    }
```

Строка 51

- **Комментарий:**

  Метод AddSessionAsync возвращает EntityEntry<Session>, что может быть избыточным для клиентов этого сервиса. Рассмотрите возможность возвращения только Session или void.

- **Текущий код:**

```
}
```
