

Конфигурации, логирование

пользовательские секреты, Serilog, Polly

Ошибки DI

Не выполняйте работу за DI контейнер

```
public class MyService
{
    private readonly HttpClient _httpClient;

    public MyService(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public void DoSomething()
    {
        //_httpClient.Send(); ...
        _httpClient.Dispose();
    }
}
```

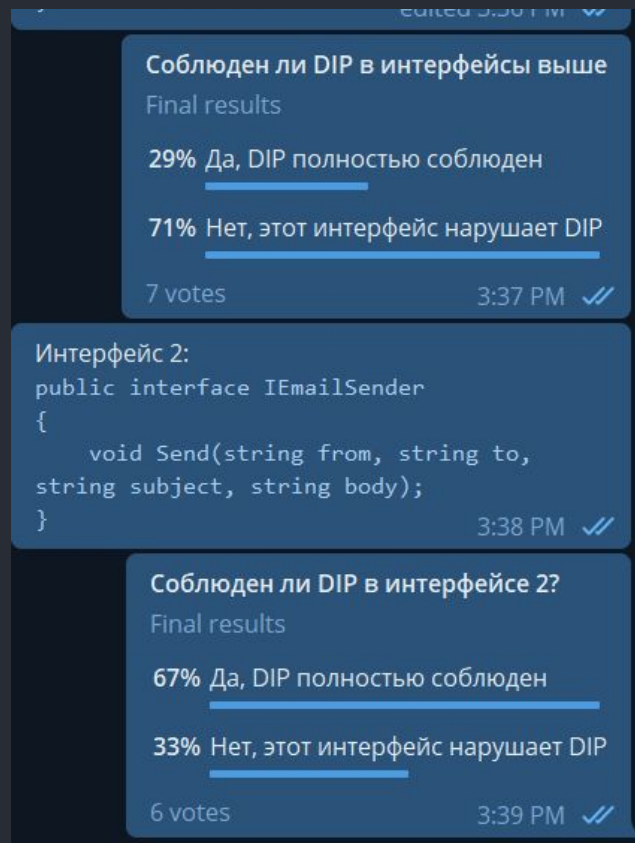
Валидируйте аргументы (Fail Fast)

```
public CatalogController(ICatalog catalog)
{
    _catalog = catalog ?? throw new ArgumentNullException(nameof(catalog));
}
```

```
public CatalogController(ICatalog catalog)
{
    ArgumentNullException.ThrowIfNull(catalog); // .NET 6+
    _catalog = catalog;
}
```

Грани DIP

Опрос



Оборотная сторона DIP

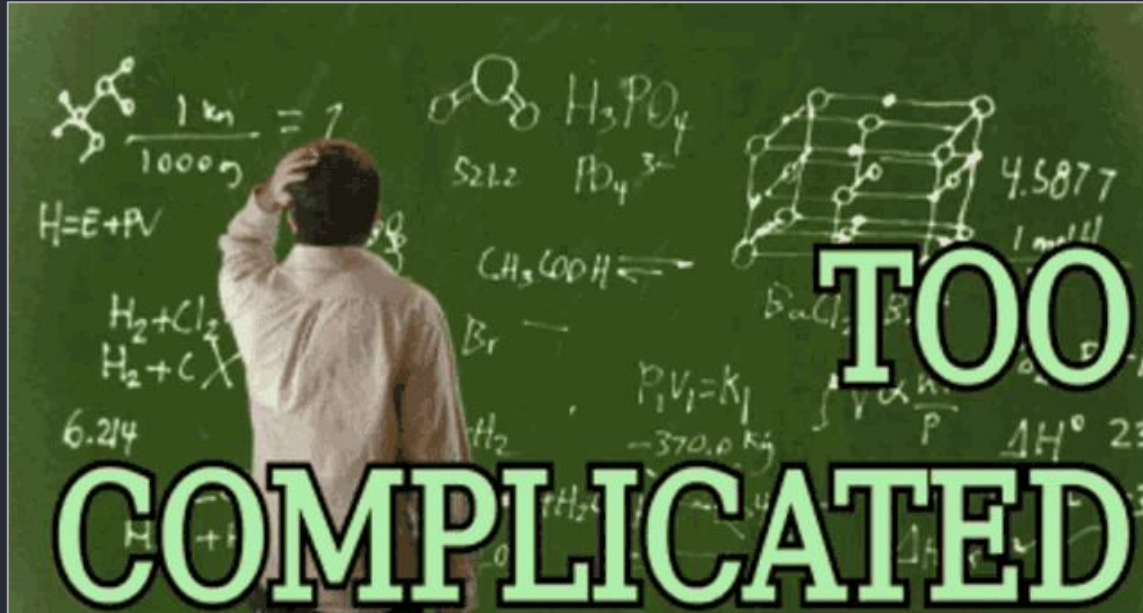
- Наличие интерфейсов образует дополнительный уровень абстракции, что затрудняет понимание системы. Понять логику исполнения становится довольно сложно.
- Было бы странно `List<T>` появилась бы зависимость вида `ICollectionPolicy`, которая отвечала бы за способ роста списка! Подобные решения обеспечивают гибкость не там, где нужно, и могут подрывать инкапсуляцию.

Anti-DIP

Принцип инверсии сознания, или DI головного мозга.
Интерфейсы выделяются для каждого класса и пачками передаются через конструкторы.

Anti-DIP

Понять, где находится логика, становится практически невозможно.



Где грань DIP?

- На этапе проектирования интерфейсы могут сильно усложнять процесс проектирования
- Для доменных сервисов, вероятно, абстракции не нужны
- Думаем и решаем грозит ли сервису изменение
- В стартапах

Решение 1: Параметры в конструкторе класса

```
public SmtplibEmailSender(string host, string userName, string password, int port = 25)
{
    _host = host;
    _userName = userName;
    _password = password;
    _port = port;
}
```

Но как теперь зарегистрировать такую зависимость?

Регистрация конкретного экземпляра зависимости

```
builder.Services.AddSingleton<IEmailSender>(
    new SmtпEmailSender("smtp.ya.ru", "mail@ya.ru", "passwd")
);
```

Регистрация конкретного экземпляра зависимости

```
Services.AddScoped<IEmailSender>(provider => new SmtpEmailSender("params..."));
```

```
Services.AddTransient<IEmailSender>(provider => new SmtpEmailSender("params..."));
```

provider - это IServiceProvider

Регистрация конкретного экземпляра зависимости

Хоть и удобно, но так лучше не делать. Причины:

- Легко вытащить данные из декомпилированного приложения
- Для изменения таких данных нужно перекомпилировать приложение
- Невозможно изменить конфигурацию во время работы приложения
- Исходники обычно хранятся в VCS (Git), поэтому к ним может получить доступ третье лицо

Конфигурация в .NET

Конфигурация в .NET

Задается в файле appsettings.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```


Конфигурация в .NET

Добавление параметра

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "AllowedHosts": "*",  
  "SmtpHost": "smtp.beget.com"  
}
```

Конфигурация в .NET

Получение значения параметра

```
var builder = WebApplication.CreateBuilder(args);  
...  
string host = builder.Configuration["SmtpHost"];
```

Секция конфигурации

```
{  
  "SmtpCredentials": {  
    "Host": "smtp.beget.com",  
    "UserName": "a@b.ru",  
    "Password": "000000"  
  }  
}
```

Получение значения из секции конфигурации

```
var builder = WebApplication.CreateBuilder(args);  
...  
string host = builder.Configuration["SmtpCredentials:Host"];
```

Привязка секции конфигурации к классу

```
var builder = WebApplication.CreateBuilder(args);  
...  
SmtpCredentials credentials = builder.Configuration  
    .GetSection("SmtpCredentials")  
    .Get<SmtpCredentials>();  
  
var host = credentials.Host;
```

Промежуточный итог

```
var credentials = builder.Configuration  
    .GetSection("SmtpCredentials")  
    .Get<SmtpCredentials>();
```

```
builder.Services.AddSingleton<IEmailSender>(new SmtpEmailSender(credentials));
```

Подход через IOptions

```
public SmtpEmailSender(IOptions<SmtpCredentials> options)
{
    _smtpCredentials = options.Value;
}
```

Подход через IOptions

```
builder.Services.Configure<SmtpCredentials>(
    builder.Configuration.GetSection("SmtpCredentials"));
builder.Services.AddSingleton<IEmailSender, SmtpEmailSender>();
```

IOptions создается один раз при запуске приложения и не изменяется

IOptionsSnapshot

- Позволяет изменять опции “на горячую”, т. е. конфигурация автоматически обновляется при изменении `appsettings.json`, к примеру
- Время жизни `Scoped`, поэтому не получится так просто передать `IOptionsSnapshot` в `Singleton`

Поставщики конфигурации (configuration providers)

Это специальные «классы», которые могут загружать конфигурацию из различных источников и предоставлять её в виде пар «ключ-значение».

Примеры источников конфигурации:

- Файлы параметров, например, appsettings.json
- Переменные среды (environment variables)
- Аргументы командной строки
- Справочные файлы, например, .ini, .xml, .config
- **Пользовательские секреты (user secrets)**
- Объекты .NET в памяти (Dictionary)
- Кастомный поставщик конфигурации, реализованный через интерфейс IConfigurationProvider

Configuration providers: Настройка источников конфигурации

```
 IConfigurationRoot config =  
     new ConfigurationBuilder()  
         .SetBasePath(Directory.GetCurrentDirectory())  
         .AddJsonFile("appsettings.json", true)  
         .AddUserSecrets<Program>(true)  
         .AddEnvironmentVariables()  
         .AddCommandLine(args)  
         .Build();
```

Пользовательские секреты

User Secrets

appsettings.json небезопасен для секретов

Перемещение данных авторизации в файл конфигурации не спасает данные от утечки, т. к. файл конфигурации обычно включается в репозиторий.

Secret Manager

- Хранит данные локально, за пределами файлов проекта приложения
 - В Windows: %APPDATA%\Microsoft\UserSecrets\\secrets.json
 - В Linux: ~/.microsoft/usersecrets/<user_secrets_id>/secrets.json
- Секреты не зашифрованы, поэтому их нельзя считать безопасными
- Требуется дополнительная настройка, в отличие от конфигураций
- Конфигурации можно комбинировать

Файл secrets.json

```
{  
  "SmtpCredentials:Port": "25",  
  "SmtpCredentials:Host": "smtp.ya.ru"  
}
```

Секреты в Production

- Azure Key Vault – это сервис, позволяющий безопасно хранить секреты в облаке. Ваше приложение извлекает секреты из Azure Key Vault во время выполнения, вызывая API и предоставляя идентификатор клиента и секрет.
- Еще один популярный вариант – Vault от компании Hashicorp (vaultproject.io), который можно запустить локально или в облаке.

Добавление секретов

1. В консоли перейдите в папку с проектом
2. Активируйте секретное хранилище:

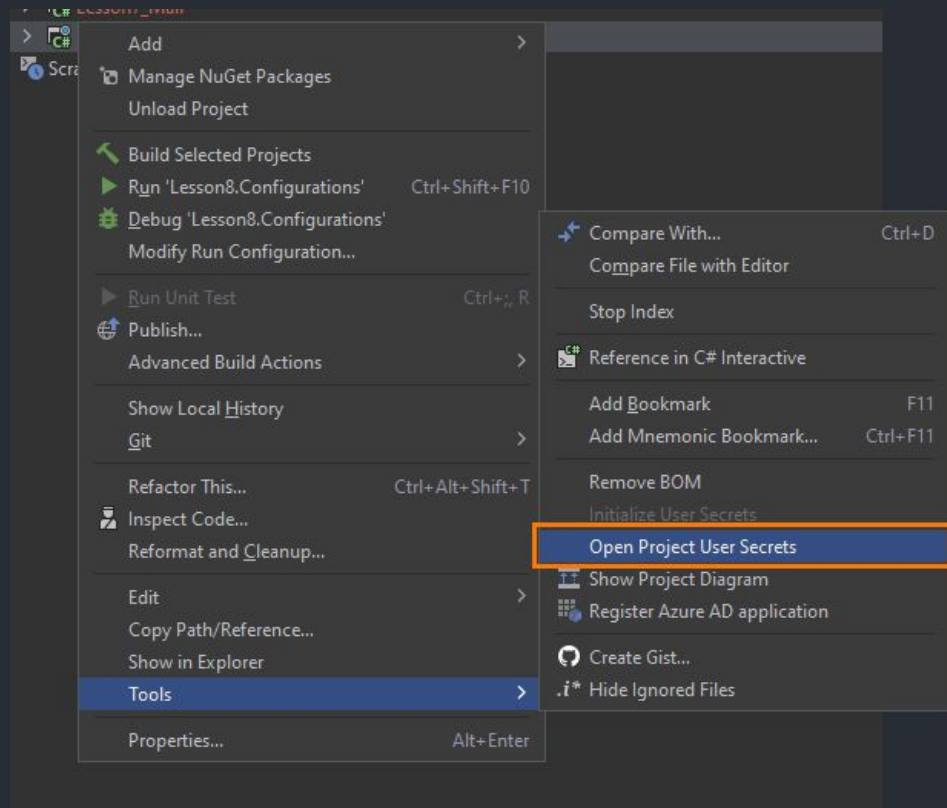
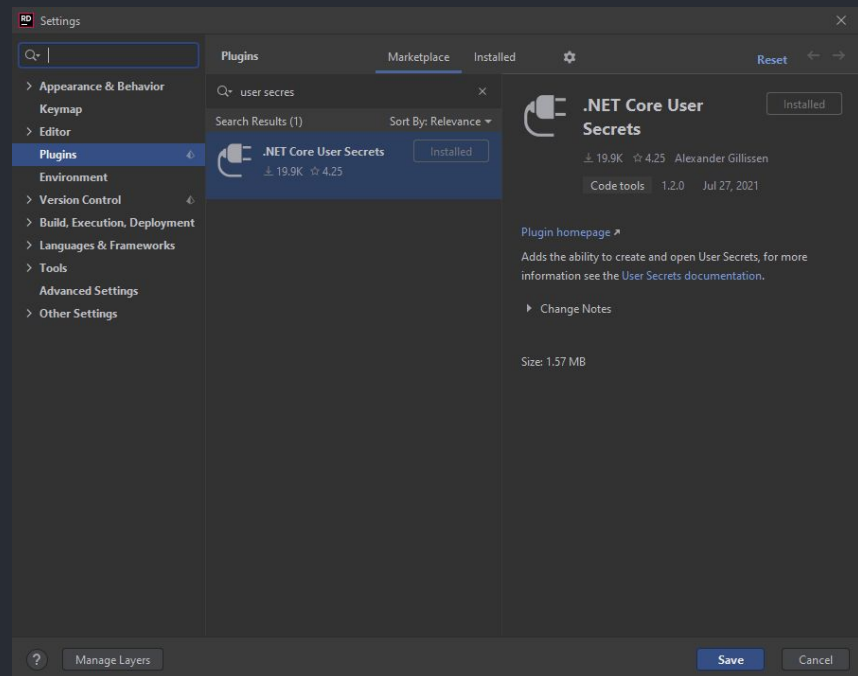
```
dotnet user-secrets init
```

3. Добавьте секрет в хранилище:

```
dotnet user-secrets set "Movies:ApiKey" "12345"
```

4. Теперь секреты доступны как часть конфигурации (например, `Configuration["Movies:ApiKey"]`)

Плагин .NET Core User Secrets для Rider



Логирование

Console.WriteLine: Недостатки

- По умолчанию пишет только в консоль
- Нет поддержки уровней логирования
- Статический класс - значит, трудно тестировать
- Не поддерживает структурное логирование (об этом дальше)

Лог в файл

- Пишет только в файл
- Потоконебезопасно
- Нетестируемо

Сторонние библиотеки

- log4net
- Serilog
- NLog

Недостаток: привязка к определенной библиотеке - трудно переехать

Microsoft.Extensions.Logging (MEL)

- Появился в ASP.NET Core
- Добавляет универсальный интерфейс логера ILogger

ILogger использование

```
private readonly AppDbContext _dbContext;  
private readonly ILogger<OrderRepository> _logger;  
  
public OrderRepository(AppDbContext dbContext, ILogger<OrderRepository> logger)  
{  
    _dbContext = dbContext;  
    _logger = logger;  
    _logger.LogInformation("OrderRepository has been created");  
}
```


А что в логе?

```
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\rodio\RiderProjects\
info: Lesson6.Data.OrderRepository[0]
      OrderRepository has been created
```

Уровни логирования



Уровни логирования для категорий

В appsettings.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "OrderRepository": "Warning"
    }
  }
}
```

Означает, что в категории OrderRepository будут фиксироваться только события уровня Warning и выше.

Среда

В файле: launchSettings.json

```
"environmentVariables": {  
  "ASPNETCORE_ENVIRONMENT": "Development"  
}
```

Либо в переменных окружения ОС.

Параметры сообщения

```
public async Task Add(Order order)
```

```
{
```

```
    _logger.LogWarning("Adding order with Id: {OrderId}", order.Id);
```

```
    await _dbContext.Orders.AddAsync(order);
```

```
    await _dbContext.SaveChangesAsync();
```

```
}
```

```
info: Lesson6.Data.OrderRepository[0]  
      OrderRepository has been created
```

```
warn: Lesson6.Data.OrderRepository[0]  
      Adding order with Id: 76be4438-4342-4845-8e4d-cb1473ef18c4
```

Структурное логирование

- Полезно когда нужно логировать объекты
- В ASP.NET Core синтаксис поддерживается из коробки, но не функционал
- Чтобы заработало нужно добавить библиотеку, поддерживающую структурное логирование

Serilog

- Популярная библиотека, поддерживающая структурное логирование
- Поддерживает множество приёмников (sinks), в т. ч.:
 - В файл
 - В базу данных
 - В специализированные системы (ELK, Seq)

Пример структурного логирования

```
public async Task Add(Order order)
{
    _logger.LogWarning("Adding order: {@Order}", order);
    await _dbContext.Orders.AddAsync(order);
    await _dbContext.SaveChangesAsync();
}
```


Лог из Serilog

```
[04:38:19 INF] Request starting HTTP/2 POST https://localhost:7043/_blazor/negotiate?negotiateVersion=1 text/plain; charset=UTF-8 0
[04:38:19 INF] Executing endpoint '/_blazor/negotiate'
[04:38:19 INF] Executed endpoint '/_blazor/negotiate'
[04:38:19 INF] Request finished HTTP/2 POST https://localhost:7043/_blazor/negotiate?negotiateVersion=1 text/plain; charset=UTF-8 0 - 200 316
ation/json 4.5120ms
[04:38:19 INF] Request starting HTTP/1.1 GET https://localhost:7043/_blazor?id=JemG6yNdwnpnx6MiJEjSHQ - -
[04:38:19 INF] Executing endpoint '/_blazor'
[04:38:19 INF] OrderRepository has been created
[04:38:19 WRN] Adding order: {"Id": "ab1cebba-cb7c-4401-b778-6bab12c30804", "Phone": "+79999999999", "TotalPrice": 100, "$type": "Order"}
```

Обычный лог (без Serilog)

```
info: Microsoft.Hosting.Lifetime[0]  
      Hosting environment: Development  
info: Microsoft.Hosting.Lifetime[0]  
      Content root path: C:\Users\rodio\RiderProjects\  
info: Lesson6.Data.OrderRepository[0]  
      OrderRepository has been created  
warn: Lesson6.Data.OrderRepository[0]  
      Adding order: Lesson6.Data.Order
```

Логирование эксепшена

```
public async Task TryAddNewOrder()
{
    var order = new Order() { Id = Guid.NewGuid() };
    try
    {
        await _orderRepository.Add(order);
    }
    catch (Exception e)
    {
        _logger.LogError(e, "Order adding failed: {@Order}", order);
    }
}
```

Логирование эксепшена

```
[05:37:44 ERR] Order adding is failed: {"Id": "13404675-2a21-4f60-9ff8-b4bf05971191", "Phone": null, "TotalPrice": 0, "$type": "Order"}
Microsoft.EntityFrameworkCore.DbUpdateException: An error occurred while saving the entity changes. See the inner exception for details.
--> Microsoft.Data.Sqlite.SqliteException (0x80004005): SQLite Error 19: 'NOT NULL constraint failed: Orders.Phone'.
    at Microsoft.Data.Sqlite.SqliteException.ThrowExceptionForRC(Int32 rc, sqlite3 db)
    at Microsoft.Data.Sqlite.SqliteDataReader.NextResult()
    at Microsoft.Data.Sqlite.SqliteCommand.ExecuteReader(CommandBehavior behavior)
    at Microsoft.Data.Sqlite.SqliteCommand.ExecuteReaderAsync(CommandBehavior behavior, CancellationToken cancellationToken)
    at Microsoft.Data.Sqlite.SqliteCommand.ExecuteReaderAsync(CommandBehavior behavior, CancellationToken cancellationToken)
    at Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteReaderAsync(RelationalCommandParameterObject parameterObject, CancellationToken cancellationToken)
```

Логирование запросов к базе

```
builder.Services.AddDbContext<AppDbContext>(
    options => options
        .UseSqlite($"Data Source={dbPath}")
        .EnableDetailedErrors()
        .EnableSensitiveDataLogging()
);
```

Подключение Serilog

1. Добавить NuGet пакет [Serilog.AspNetCore](#)
2. Активировать:

```
builder.Host.UseSerilog((_, conf) => conf.WriteTo.Console());
```

Серилог добавлен и активирован. Но это еще не все.

Логирование в файл

```
builder.Host.UseSerilog((_, conf) =>
{
    conf
        .WriteTo.Console()
        .WriteTo.File("log-.txt", RollingInterval.Day)
    ;
});
```

Опционально: Конфигурирование Serilog

```
{
  "Serilog": {
    "Using": [ "Serilog.Sinks.Console", "Serilog.Sinks.File" ],
    "MinimumLevel": "Information",
    "WriteTo": [
      { "Name": "Console" },
      { "Name": "File", "Args": { "path": "logs/log.txt" } },
    ],
    "MinimumLevel": "Debug"
  },
  "Enrich": [ "FromLogContext" ]
}
```


Добавление параметров логирования из конфигурации

```
builder.Host.UseSerilog((ctx, conf) =>
{
    conf
        .WriteTo.Console()
        .WriteTo.File("log-.txt", rollingInterval: RollingInterval.Day)
        .ReadFrom.Configuration(ctx.Configuration)
    ;
});
```

Настройка уровней логирования событий

```
builder.Host.UseSerilog((ctx, conf) =>
{
    conf
        .MinimumLevel.Debug() //← Минимальный уровень для всех приемников
        .WriteTo.File("log-.txt", rollingInterval: RollingInterval.Day)
        .WriteTo.Console(restrictedToMinimumLevel: LogEventLevel.Information)
        .ReadFrom.Configuration(ctx.Configuration)
    ;
});
```

* `restrictedToMinimumLevel` – минимальный уровень для конкретного получателя (для консоли в данном случае)

Корректное логирование ошибок запуска

```
using Serilog;

Log.Logger = new LoggerConfiguration()
    .WriteTo.Console()
    .CreateBootstrapLogger(); //означает, что глобальный логер будет заменен на вариант из Host.UseSerilog
Log.Information("Starting up");
try
{
    /*
     * тут должна находиться вся логика создания веб-приложения
     * (содержимое файла Program.cs: builder, app и тд)
     */
}
catch (Exception ex)
{
    Log.Fatal(ex, "Unhandled exception");
}
finally
{
    Log.Information("Shut down complete");
    Log.CloseAndFlush(); //перед выходом дожидаемся пока все логи будут записаны
}
```

Активация логирования запросов Serilog

```
var app = builder.Build();
```

```
app.UseStaticFiles();
```

```
app.UseSerilogRequestLogging();
```

```
app.MapGet("/", () => "Hello World!");
```

Standard logging from ASP.NET Core infrastructure

[14:15:44 INF] Request starting HTTP/2 GET https://localhost:5001/

[14:15:44 INF] Executing endpoint '/Index'

[14:15:45 INF] Route matched with {page = "/Index"}. Executing page /Index

[14:15:45 INF] Executing handler method SerilogRequestLogging.Pages.IndexModel.OnGet - ModelState is Valid

[14:15:45 INF] Executed handler method OnGet, returned result .

[14:15:45 INF] Executing an implicit handler method - ModelState is Valid

[14:15:45 INF] Executed an implicit handler method, returned result

Microsoft.AspNetCore.Mvc.RazorPages.PageResult.

[14:15:45 INF] Executed page /Index in 124.7462ms

[14:15:45 INF] Executed endpoint '/Index'

Additional Log from Serilog

[14:15:45 INF] **HTTP GET / responded 200 in 249.6985 ms**

Standard logging from ASP.NET Core infrastructure

[14:15:45 INF] Request finished in 274.283ms 200 text/html; charset=utf-8

Важно: Serilog не учитывает уровни для MEL

Отлично работает в связке с [Seq](#)

The screenshot displays the Seq logging application interface. The top navigation bar includes the Seq logo, the URL 'cafe-logs.example.com', and the application name 'Seq Café'. The main content area shows a list of log events. The selected event is a POST request from 'SeqCafe.Web.OrdersController' with a response time of 453.043 ms. The event details are expanded, showing a JSON payload for an order item. The sidebar on the right contains a search bar and a list of filters, including 'SIGNALS', 'Columns', 'Method', 'Source', 'Total', 'Message Bus', 'Orders Sent', 'SeqCafe.Web', 'All', 'Customers', 'Requests', and 'Revenue'. The bottom status bar shows the current time and the selected event details.

Seq café-logs.example.com | Seq Café

Events Dashboards Ingestion Settings Support

Find signals and queries

SIGNALS

- None
- @Level
- Errors
- Exceptions
- Warnings
- Columns
- Method
- Source
- Total
- Message Bus
- Orders Sent
- SeqCafe.Web
- All
- Customers
- Requests
- Revenue

25 Jun 2020 16:36:47.454 POST SeqCafe.Web.DemoMiddleware HTTP POST /api/orders responded 200 in 453.043 ms

25 Jun 2020 16:36:47.001 SeqCafe.Web.Middleware.ExceptionHandlingMiddleware The operation timed out

25 Jun 2020 16:36:47.001 SeqCafe.Web.OrdersController Received order f1be3140-e8b0-4def-bd8c-58b6d84e4c37 total \$3.00 for customers-1132831135

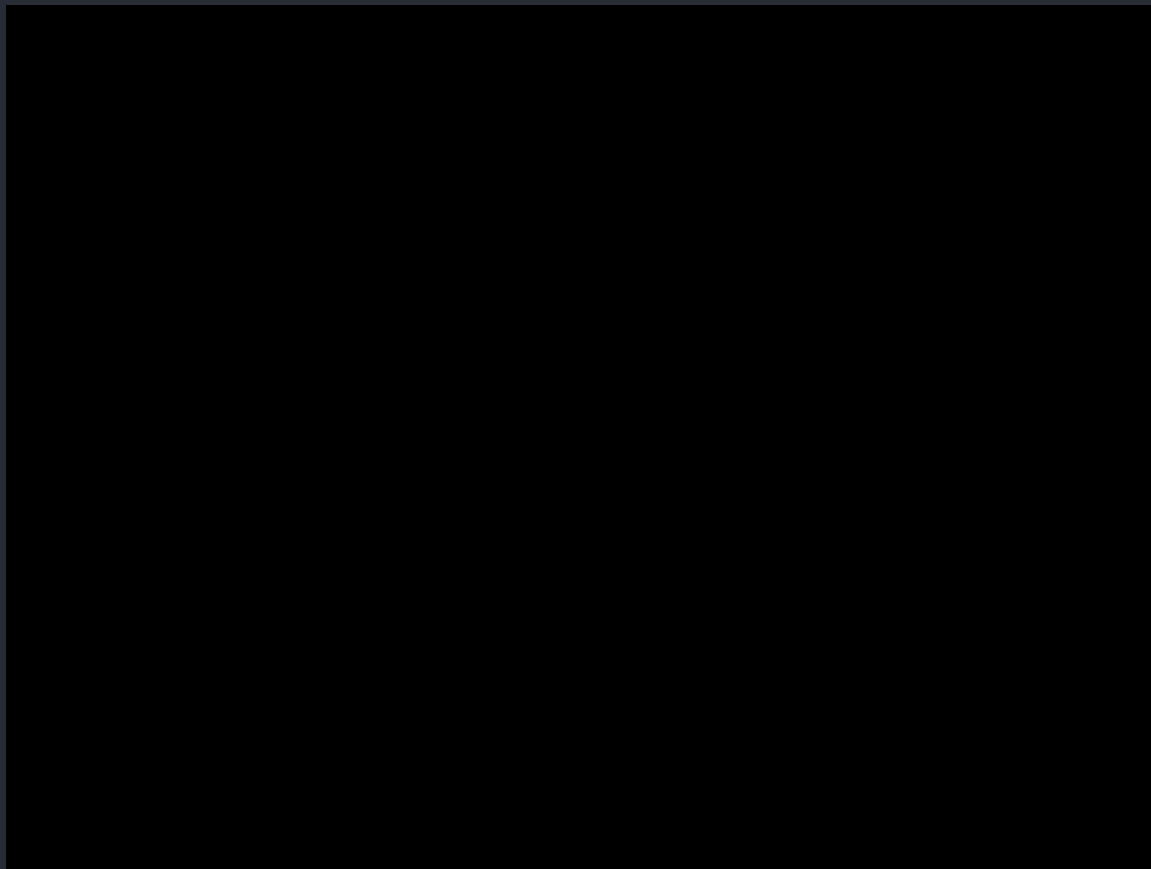
Event Level (Information) Type (0x107BD6BD) Retain Download Raw JSON

- Application Cafe
- CustomerId customers-1132831135
- Environment Staging
- OrderId f1be3140-e8b0-4def-bd8c-58b6d84e4c37
- OrderItems
 - {
ProductCode: "SOL100",
UnitPrice: 3,
Quantity: 1,
_typeTag: "OrderItem"
}
- RequestId 8ce267d107e2473e
- SourceContext SeqCafe.Web.OrdersController
- Total 3

25 Jun 2020 16:36:46.107 GET SeqCafe.Web.DemoMiddleware HTTP GET /api/products/list responded 200 in 175.329 ms

Available Properties

Отлично работает в связке с [Seq](#)




Баг-трекер

Баг-трекер

- Агрегирует ошибки
- Позволяет группировать ошибки по
 - их частотности
 - устройству
 - браузеру
 - ОС
 - и другим критериям
- Умеет отправлять алерты на почту

Sentry

 SENTRY Детали


System.Exception: Realm accessed from incorrect thread.
C:\Users\rodio\RiderProjects\goodwins\Goodwins.Server\Services\ShopService.cs in ProductR <>c__DisplayClass16_...

СООБЩЕНИЕ

Connection id "0HMDQKB8B4PUM", Request id "0HMDQKB8B4PUM:00000001": An unhandled exception was thrown by the application.

ИСКЛЮЧЕНИЕ (последний вызов в начале) Только приложение Полностью Raw

System.Exception
Realm accessed from incorrect thread.

 void NativeException.ThrowIfNecessary(Func<RealmExceptionCodes, Exception> overrider) ^

Assembly: Realm Версия: 4.3.0.0 Culture: neutral PublicKeyToken: null

bool TableHandle.TryFind(SharedRealmHandle realmHandle, long? id, out ObjectHandle objectHandle) ▾

T Realm.Find<T>(long? primaryKey) ▾

ProductR <>c__DisplayClass16_0<T>.<GetMainProductsV2>g__TryGetProductCached|0(?)>+TryGetProductCached(int sku) at line 385:6 ▾

Task<List<T>> ShopService.GetMainProductsV2<T>(string shopId, int limit, AppUser user, bool enableFilter, HashSet<int> filterCategories)+ (int sku) => { } at line 405:6 ▾

bool Enumerator.MoveNext() at line 85:6 ▾

Called from: bool EnumerablePartition<TSource>.MoveNext() ▾

async Task<List<T>> ShopService.GetMainProductsV2<T>(string shopId, int limit, AppUser user, bool enableFilter, HashSet<int> filterCategories) at line 418:5 ▾

<https://sentry.io/share/issue/7b9961ba741a41b090c68d2db6a76bca/>

Polly

Polly: Политики обработки ошибок

- Позволяет задавать гибкие политики обработки ошибок
- Повтор операции (ретрай) с динамическим интервалом
- Таймаут выполнения операции
- Circuit Breaker (СВ или автоматический выключатель)
- Fallback (план Б)
- И многое другое:

<https://github.com/App-vNext/Polly#resilience-policies>

Подробнее: <https://habr.com/ru/company/dododev/blog/503376/>

Polly: Пример

```
AsyncRetryPolicy? policy = Policy
    .Handle<Exception>()
    .RetryAsync(retryCount, onRetry: (exception, retryAttempt) =>
    {
        _logger.LogWarning(
            exception, "Error while sending email. Retrying: {Attempt}", retryAttempt);
    });

PolicyResult? result = await policy.ExecuteAndCaptureAsync(
    token => _emailSender.SendAsync(..., token), stoppingToken);

if (result.Outcome == OutcomeType.Failure)
{
    _logger.LogError(result.FinalException, "There was an error while sending email");
}
```

Подключение Serilog

1. Добавить NuGet пакет [Serilog.AspNetCore](#)
2. Активировать:

```
builder.Host.UseSerilog((_, conf) => conf.WriteTo.Console());
```

Серилог добавлен и активирован. Но это еще не все.

ДЗ

1. Сделайте класс отправки email'ов настраиваемым через IOptions.
2. Логин и пароль от ящика сохраните в пользовательские секреты.
3. Корректно подключите Serilog к проекту
4. Добавьте 3 попытки отправки email'а о добавленном товаре при помощи Polly. Залоггируйте все попытки, обратите на уровни логирования.