



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 6

Дисциплина: «Backend»

Тема: *Конфигурация веб-приложения на основе ASP.NET Core*

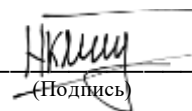
Выполнил: студент группы: 231-339

Карапетян Нвер Каренович

(Фамилия И.О.)

Дата, подпись: 14.04.25

(Дата)


(Подпись)

Проверил: _____

(Фамилия И.О., степень, звание)

(Оценка)

Дата, подпись _____

(Дата)

(Подпись)

Москва
2025

Цель:

Освоить процесс конфигурирования веб-приложений на платформе ASP.NET Core для эффективного управления параметрами приложения.

Задачи:

- Создать файлы конфигурации для различных сред, таких как разработка, тестирование и продуктивное окружение.
- Использовать различные источники конфигурации, такие как файлы JSON, переменные среды, настройки веб-сервера и другие.
- Настроить обработку и использование конфигурационных параметров в коде приложения.
- Протестировать работу приложения при изменении конфигурационных параметров для разных сред.

Ход работы

Конфигурационные файлы

Для разных окружений можно создавать отдельные файлы конфигурации, которые позволяют задавать специфические для каждой среды параметры, такие как строки подключения к базе данных, настройки логирования, параметры безопасности и т.д.

Исходный файл **appsettings.json** содержит общие настройки приложения, которые применяются для всех сред. Пример содержимого:

Листинг 1. Общий конфигурационный файл appsettings.

```
{
  "ConnectionStrings": {
    "Default": "Host=localhost;Port=5432;Database=Users;Username=postgres;Password=Spillett777"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
}
```

```

    "AllowedHosts": "*",
    "JwtSettings": {
      "SigningKey":
"d5be7e771f8fd195502ec156b43d76f4b2335da20d423cccc3f44df1b2404f70f48954e0096facb4cf
d19094eb71209b4b108182f8a29f664f9959a0e9aac9bd",
      "Issuer": "https://localhost:7137/",
      "Audience": "https://localhost:7137/"
    }
  }
}

```

Для различения окружений создаются дополнительные файлы:

- **appsettings.Development.json** – для режима разработки.
- **appsettings.Testing.json** – для тестового окружения.
- **appsettings.Production.json** – для продуктивного окружения.

Такая организация позволяет задавать разные строки подключения, уровни логирования и параметры безопасности для каждого окружения. Это особенно важно для продуктивного развертывания, где требования к безопасности и производительности значительно строже, чем в режиме разработки.

Использование различных источников конфигурации

Помимо файлов JSON, ASP.NET Core позволяет использовать другие источники для загрузки настроек.

Переменные среды

Метод `AddEnvironmentVariables()` позволяет включить в конфигурацию значения, заданные в системных переменных. Эти переменные автоматически обрабатываются как набор пар ключ-значение, при этом вложенные секции задаются посредством разделителя (например, «__»).

Таким образом, если на уровне ОС задана какая-либо переменная с помощью консольной команды «set», как в приведенном ниже листинге, то значение по ключу `"ConnectionStrings:Default"` будет переопределено значением переменной среды.

```
set ConnectionStrings__Default="Host=prod-db-server;Port=5432;Database=Us -
ersProd;Username=prod_user;Password=ProdPassword!"
```

Аргументы командной строки

Метод AddCommandLine(args) позволяет передавать параметры через аргументы командной строки при запуске приложения. Это особенно полезно для динамического переопределения конфигурационных значений без изменения файлов.

При запуске приложения с аргументами, например, `dotnet run -- Logging:LogLevel:Default="Debug"` значение "Debug" будет установлено для ключа Logging:LogLevel:Default, даже если в файле appsettings.json прописано другое значение. Таким образом, метод позволяет гибко изменять поведение приложения без изменения исходных конфигурационных файлов.

Конфигурационные INI-файлы

При необходимости можно подключить конфигурацию из файлов другого формата. Пример для INI-файла:

Листинг 3. Пример метода AddIniFile.

```
builder.Configuration.AddIniFile("config.ini", false);
```

Сами по себе ini-файлы — это обычные текстовые файлы, которые можно редактировать и просматривать при помощи любого текстового редактора. ini-файлы имеют следующий формат:

Листинг 4. Пример конфигурации ini-файлов.

```
[project]
name = orchard rental service (with app)
target region = "Bay Area"
; TODO: advertise vacant positions
legal team = (vacant)

[fruit "Apple"]
trademark issues = foreseeable
taste = known

[fruit.Date]
taste = novel
```

```
Trademark Issues = "truly unlikely"

[fruit "Raspberry"]
anticipated problems = "logistics (fragile fruit)"
Trademark Issues = possible

[fruit.raspberry.proponents.fred]
date = 2025-04-14, 01:01 +0900
comment = "I like red fruit."
```

Настройка обработки конфигурационных параметров в коде приложения

Ключевым моментом является правильное объединение источников конфигурации в файле `Program.cs`:

Листинг 5. Пример настройки конфигурации.

```
builder.Configuration
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
    .AddJsonFile($"appsettings.{builder.Environment.EnvironmentName}.json", optional: true, reloadOnChange: true)
    .AddEnvironmentVariables()
    .AddCommandLine(args)
    .AddIniFile("config.ini", optional: true, reloadOnChange: true);
```

Конфигурация приложения в ASP.NET Core может быть построена из различных источников, объединенных в единую систему, в которой значения с более высоким приоритетом могут переопределять ранее заданные. В представленной конфигурации используется цепочка методов, которая последовательно подключает эти источники в строго определенном порядке.

Сначала с помощью `SetBasePath(Directory.GetCurrentDirectory())` задается базовый путь, откуда будут считываться все конфигурационные файлы — обычно это корневая директория проекта. Далее подключается основной конфигурационный файл `appsettings.json`, который является обязательным и автоматически отслеживается на предмет изменений благодаря флагу `reloadOnChange: true`. После этого добавляется файл, специфичный для текущего окружения (например, `appsettings.Development.json` или `appsettings.Production.json`). Этот файл необязателен, но позволяет переопределить настройки, указанные в

общем конфигурационном файле, при запуске приложения в определенной среде.

Следующим источником конфигурации становятся переменные среды, подключаемые через метод `AddEnvironmentVariables()`. Они позволяют задавать параметры конфигурации вне кода и файлов, например, на уровне операционной системы или среды хостинга. Затем добавляются аргументы командной строки, переданные при запуске приложения (`AddCommandLine(args)`), которые имеют еще более высокий приоритет и могут использоваться для временной настройки параметров без изменения файлов или переменных среды. В завершение, в конфигурацию подключается INI-файл `config.ini`, который также может содержать дополнительные параметры и включается в общий набор настроек.

Таким образом, формируется иерархия конфигурационных источников, в которой каждый последующий источник может переопределять значения предыдущих. Это обеспечивает гибкость и удобство при управлении конфигурацией приложения в различных средах: значения из `appsettings.json` могут быть переопределены в `appsettings.{Environment}.json`, затем переменными среды, затем аргументами командной строки, и наконец — INI-файлом. Такой подход делает приложение легко настраиваемым и адаптируемым под разные сценарии развертывания.

Использование конфигурационных параметров в коде приложения

Обычно на практике конфигурация внедряется через механизм `Dependency Injection`, а именно через конструктор класса, посредством интерфейса `IConfiguration`:

Листинг 6. Пример внедрения конфигурации через DI.

```
private readonly UserDbContext _context;
private readonly IConfiguration _configuration;

public AuthService(UserDbContext context, IConfiguration configuration)
{
    _context = context;           // Подключение контекста БД через DI
    _configuration = configuration; // Подключение конфигурации через DI
}
```

Это позволяет обращаться к настройкам, заданным в файлах конфигурации (`appsettings.json`, `appsettings.{Environment}.json`) или переопределённым через переменные среды, аргументы командной строки и иные источники.

Ниже приведены примеры строк, в которых происходит чтение параметров, а также краткое описание их назначения:

Листинг 7. Примеры чтения параметров из конфигурационных файлов.

```
// Чтение секретного ключа для подписи JWT из секции "JwtSettings"
// Этот ключ используется для создания симметричного ключа, который затем применяется для подписи токена
var signingKey = _configuration.GetValue<string>("JwtSettings:SigningKey");

// Чтение значения Issuer (издатель токена) из конфигурации
// Параметр определяет, кто выдал токен, и используется при его валидации
var issuer = _configuration.GetValue<string>("JwtSettings:Issuer");

// Чтение значения Audience (получатель токена) из конфигурации
// Это значение указывает, для кого предназначен токен, и также участвует в проверке его корректности
var audience = _configuration.GetValue<string>("JwtSettings:Audience");
```

Эти строки располагаются в методе, отвечающем за создание токена. Сначала происходит получение строки ключа для подписи, что обеспечивает безопасность и целостность JWT-токена. Затем значения `Issuer` и `Audience` извлекаются для дальнейшей настройки параметров токена, гарантируя, что при валидации запроса сервер сможет убедиться, что токен выдан именно нужным источником и предназначен для корректного получателя.

Благодаря такому подходу, если параметры изменятся в конфигурационных файлах или будут переопределены через переменные среды/аргументы командной строки, приложение автоматически будет использовать обновлённые значения без необходимости внесения изменений в код.



Рисунок 1. Вывод параметров конфигурации в консоль.