




# Minimal web API на ASP.NET Core 7



Андрей Порожняков,  
.NET разработчик



# План доклада

---

- Пример приложения Minimal API
- Эволюция web API в .NET, обозначение проблемы
- Похожие решения в других языках
- Что такое minimal API: возможности, конструкции и примеры
- Выводы и личные рекомендации по использованию

# Пример приложения Minimal API

---

```
var app = WebApplication.Create(args);  
  
app.MapGet("/", () => "Hello World!");  
  
app.Run();
```

**Minimal API** — это подход создания HTTP API с помощью ASP.NET Core.

Он позволяет получить полностью функционирующие endpoints REST с минимальным кодом и настройкой.

\*Пример взят из [Microsoft Learn](#)

# Эволюция web API в .NET

## ASP.NET MVC

демонстрация  
первой версии

2007

2012

## Микросервисная архитектура

распространяется  
с середины 2010-х

2022

## Minimal API

с выходом .NET 6

# Ruby + Sinatra

---

```
# myapp.rb  
require 'sinatra'  
get '/' do  
  'Hello world!'  
end
```

\*Пример взят из [GitHub](#)



# Python + Flask

---

```
from flask import Flask

app = Flask(__name__)

@app.route("/")

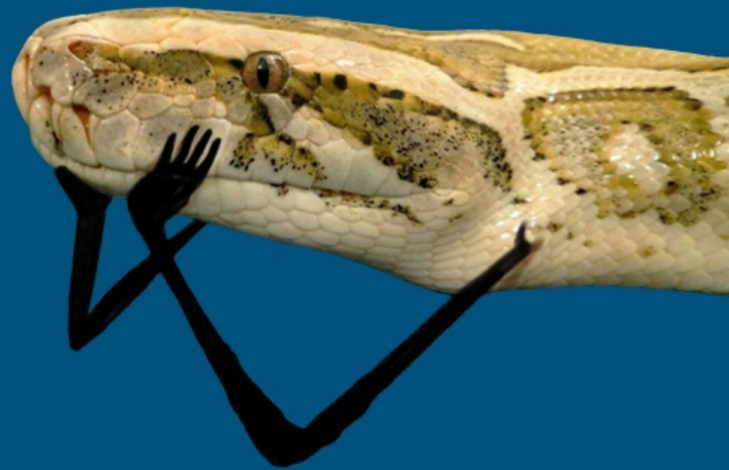
def hello():

    return "Hello World!"

if __name__ == '__main__':

    app.run(debug=True)
```

\*Пример взят из [Medium](#)



# Поддержка Swagger и OpenAPI

Добавляем ссылки на nuget пакеты

```
<PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />  
<PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.2" />
```

Содержимое файла Program.cs

```
var builder = WebApplication.CreateBuilder(args);  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
var app = builder.Build();  
app.UseSwagger();  
app.UseSwaggerUI();  
app.MapGet("/", () => "Hello World!").WithOpenApi(operation => new(operation)  
{ Summary = "This is a summary", Description = "This is a description" });  
app.Run();
```

# Типы HTTP-запросов

---

## Существующие методы для различных типов запросов

```
app.MapGet("/", () => "Hello from GET!");  
app.MapPost("/", () => "Hello from POST!");  
app.MapPut("/", () => "Hello from PUT!");  
app.MapDelete("/", () => "Hello from DELETE!");  
app.MapPatch("/", () => "Hello from PATCH!"); // .NET 7  
app.MapMethods("/", new[] { "OPTIONS" }, () => "Hello from OPTIONS!");
```



# Маршруты и параметры

```
var app = WebApplication.Create(args);

app.MapPost("/route/example/{foo}", (int foo, //[FromRoute]
    double? bar, //[FromQuery]
    [FromHeader(Name = "X-CUSTOM-HEADER")] string customHeader,
    Baz bazFromBody, //[FromBody]
    [AsParameters] Baz bazFromQuery)
    => $"Foo: {foo}, Bar: {bar}, Custom header {customHeader}, Baz from body value:
{bazFromBody.Value}, Baz from query value: {bazFromQuery.Value}");

app.Run();

record Baz(string Value);
```

# Загрузка файлов (.NET 7)

```
var app = WebApplication.Create(args);

app.MapPost("/upload", async (IFormFile file) => {

    // Логика работы с файлом

});

app.MapPost("/upload_many", async (IFormFileCollection myFiles) => {

    foreach (var file in myFiles) {

        // Логика работы с файлом

    });

});

app.Run();
```

# Возвращение результата

---

```
app.MapGet("/hi", () => "Hi"); //Строка

app.MapGet("/hi", () => new { Message = "Hi" }); //Любой тип данных, в т.ч. анонимный

app.MapGet("/hi", () => Results.Ok(new Message() { Text = "Hi" })) //IResult<T>

    .Produces<Message>(); //предоставляет метаданные типа ответа для OpenAPI

app.MapGet("/hi", () => TypedResults.Ok(new Message() { Text = "Hi" })); // ASP.NET
Core 7, фабрика для IResult, предоставляет метаданные типа ответа для OpenAPI
```

# Внедрение зависимостей

```
builder.Services.AddSingleton<Service>();  
builder.Services.AddDbContext<FooDb>(opt => opt.UseInMemoryDatabase("FooDb"));  
var app = builder.Build();  
app.MapGet("/service", (Service service) => service.Hello); // [FromServices]  
app.MapGet("/foo/{id}", async (int id, FooDb db) => await db.FooSet.FindAsync(id));  
app.MapPost("/foo", async (Foo foo, FooDb db) => {  
    db.FooSet.Add(foo);  
    return await db.SaveChangesAsync();  
});  
app.Run();  
  
class Service { public string Hello => "Hello from service!"; }
```

# Использование Middleware

---

```
app.UseSwagger();  
app.UseSwaggerUI();  
app.UseRouting();  
  
app.UseMiddleware<CustomMiddleware>(); // должен иметь InvokeAsync(HttpContext  
context)  
  
app.Use((context, next) => {  
    // логика  
    return next(context); });
```

# Аутентификация и авторизация

```
var builder = WebApplication.CreateBuilder(args);  
builder.Services.AddAuthentication();  
builder.Services.AddAuthorization();  
var app = builder.Build();  
app.UseAuthentication();  
app.UseAuthorization();  
app.MapGet("/with-authorization", () => "Hello from authorized  
user").RequireAuthorization(); // AuthorizeAttribute  
app.MapGet("/without-authorization", () => "Hello from  
anonymous").AllowAnonymous(); // AllowAnonymousAttribute  
app.Run();
```

# Аутентификация и авторизация

---

## Выборочное использование политик

```
var builder = WebApplication.CreateBuilder(args);  
  
builder.Services.AddAuthorizationBuilder().AddPolicy("admin_greetings",  
p => p.RequireRole("admin"));  
  
var app = builder.Build();  
  
app.MapGet("/hello", () => "Hello from  
endpoint").RequireAuthorization("admin_greetings");  
  
app.Run();
```

# Использование фильтров (.NET 7)

```
app.MapGet("/hello", (string value) => "Hello from endpoint")
    .AddEndpointFilter(async (invocationContext, next) => {
        //Логика фильтра

        return await next(invocationContext); })
    .AddEndpointFilter<CustomFilter>(); //Должен реализовывать IEndpointFilter

public class CustomFilter : IEndpointFilter {
    public async ValueTask<object?> InvokeAsync(EndpointFilterInvocationContext
efiContext, EndpointFilterDelegate next) {
        //Логика фильтра

        return await next(efiContext); }}
```



# Работа с группами (.NET 7)

---

```
var app = WebApplication.Create(args);  
  
var group = app.MapGroup("/say")  
    .AddEndpointFilter<CustomFilter>()  
    .RequireAuthorization();  
  
group.MapGet("/hello", () => "Hello!");  
  
group.MapGet("/bye", () => "Good bye, have a nice day!");  
  
app.Run();
```

# Плюсы и минусы Minimal API

Со мной ты сократишь объём кода,  
сделаешь компактные проекты

И вообще, потратишь меньше  
времени на прототип и разработку



**Minimal API**

Я смешаю логику обработки  
запросов с кодом конфигурации

А потом, с ростом функционала,  
меня будет сложно поддерживать



**Тоже Minimal API**

# Рекомендации по использованию

---



## Стоит использовать, если:

- нужно сделать «песочницу»
- нужно быстро сделать прототип
- мало времени на разработку (осознанный тех. долг)
- в разработке простой микросервис на несколько методов



## Не стоит использовать:

- в уже разрабатываемых проектах с существующими контроллерами
- в проектах, где предполагается большое количество методов и сложная логика, которую нужно разделять

# Темы, не вошедшие в доклад

---

- Развитие предшествующих технологий в .NET
- Конфигурация приложения через WebApplicationBuilder
- Логирование
- Обработка исключений
- Работа с DI-контейнером
- Использование Minimal API вместе с контроллерами
- Юнит и интеграционное тестирование
- Больше подробностей рассмотренных тем: развитие Web API, примеры из других языков, техническая часть

# Новые функции Minimal API в .NET 7

- Поддержка OpenAPI
- Метод MapPatch()
- Декомпозиция параметра [AsParameters]
- Загрузка файлов
- Возвращение результата TypedResults
- Фильтры
- Работа с группами



# Ссылки на источники

---

- [ASP.NET MVC - Wikipedia](#)
- [Microservices - Wikipedia](#)
- [Minimal APIs quick reference | Microsoft Learn](#)
- [Minimal APIs in .NET 6](#)
- [Build Minimal APIs In .NET 7 Using Entity Framework Core 7](#)