Распределенные информационно-аналитические системы

Практическое занятие № 3. «Основы в ASP.NET Core. Часть 3»

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные вопросы:

1. **HttpResponse. Отправка ответа.**
2. **HttpRequest. Получение данных запроса.**
3. **Отправка файлов.**

1. HttpResponse. Отправка ответа

Все данные запроса передаются в **middleware** через объект **Microsoft.AspNetCore.Http.HttpContext**. Этот объект инкапсулирует информацию о запросе, позволяет управлять ответом и, кроме того, имеет еще много другой функциональности. Например, возьмем простейшее приложение:

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

app.Run(async(context) => await context.Response.WriteAsync("Hello METANIT.COM"));

app.Run();
```

Здесь параметр context, который передается в **middleware** в методе **app.Run()** как раз представляет объект **HttpContext**. И через этот объект, точнее через его свойство **Response** мы можем отправить клиенту некоторый ответ:

context.Response.WriteAsync($"Hello METANIT.COM").

Свойство **Response** объекта **HttpContext** представляет объект **HttpResponse** и устанавливает, что будет отравляться в виде ответа. Для установки различных аспектов ответа класс **HttpResponse** определяет следующие свойства:

Body: получает или устанавливает тело ответа в виде объекта **Stream**.

BodyWriter: возвращает объект типа **PipeWriter** для записи ответа.

ContentLength: получает или устанавливает заголовок **Content-Length**.

ContentType: получает или устанавливает заголовок **Content-Type**.

Cookies: возвращает куки, отправляемые в ответе.

HasStarted: возвращает true, если отправка ответа уже началась.

Headers: возвращает заголовки ответа.

Host: получает или устанавливает заголовок **Host**.

HttpContext: возвращает объект **HttpContext**, связанный с данным объектом Response.

StatusCode: возвращает или устанавливает статусный код ответа.

Чтобы отправить ответ, мы можем использовать ряд **методов** класса **HttpResponse**:

Redirect(): выполняет переадресацию (временную или постоянную) на другой ресурс.

WriteAsJson()/WriteAsJsonAsync(): отправляет ответ в виде объектов в формате JSON.

WriteAsync(): отправляет некоторое содержимое. Одна из версий метода позволяет указать кодировку. Если кодировка не указана, то по умолчанию применяется кодировка UTF-8.

SendFileAsync(): отправляет файл.

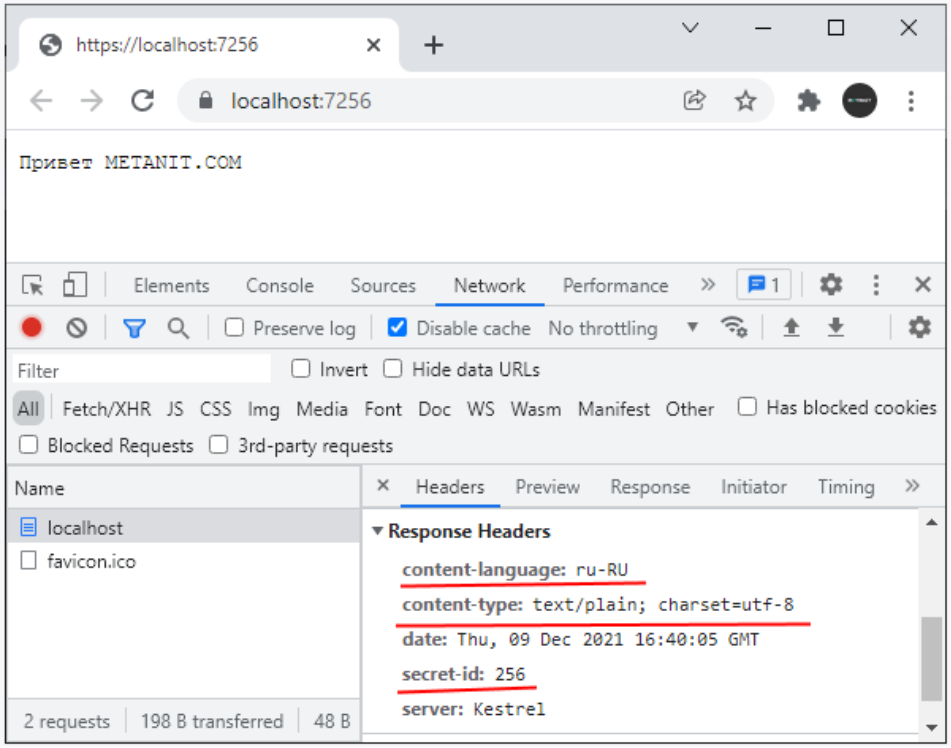
Самый простой способ отправки ответа представляет метод **WriteAsync()**, в который передается отправляемые данные. В качестве дополнительного параметра мы можем указать кодировку:

```
app.Run(async (context) =>
{
    await context.Response.WriteAsync("Hello METANIT.COM", System.Text.Encoding.Default);
});
```

Установка заголовков

Для установки заголовков применяется свойство **Headers**, которое представляет тип **IHeaderDictionary**. Для большинства стандартных заголовков HTTP в этом интерфейсе определены одноименные свойства, например, для заголовка **"content-type"** определено свойство **ContentType**. Другие, в том числе свои кастомные заголовки можно добавить через метод **Append()**. Например:

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

app.Run(async (context) =>
{
    var response = context.Response;
    response.Headers.ContentLanguage = "ru-RU";
    response.Headers.ContentType = "text/plain; charset=utf-8";
    response.Headers.Append("secret-id", "256");    // добавление кастомного заголовка
    await response.WriteAsync("Привет METANIT.COM");
});

app.Run();
```



Установка заголовков в ответе в ASP.NET Core и C#

Стоит отметить, что для вывода кириллицы желательно устанавливать заголовок **ContentType**, в том числе кодировку, которая применяется в отправляемом содержимом (в примере выше это "text/plain; charset=utf-8").

Также стоит отметить, что вместо

```
response.Headers.ContentType = "text/plain; charset=utf-8";
```

можно было бы написать

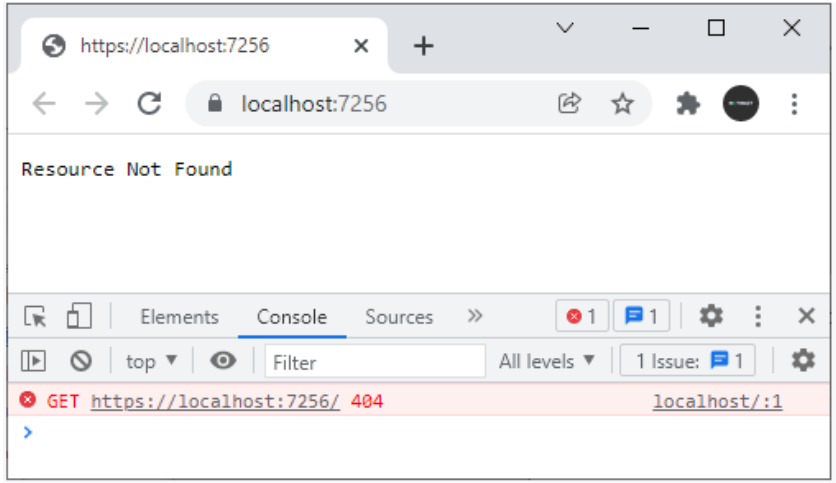
```
response.ContentType = "text/plain; charset=utf-8";
```

Установка кодов статуса

Для установки статусных кодов применяется свойство **StatusCode**, которому передается числовой код статуса:

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

app.Run(async(context) =>
{
    context.Response.StatusCode = 404;
    await context.Response.WriteAsync("Resource Not Found");
});

app.Run();
```

В данном случае устанавливается код 404, который указывает, что ресурс не найден.

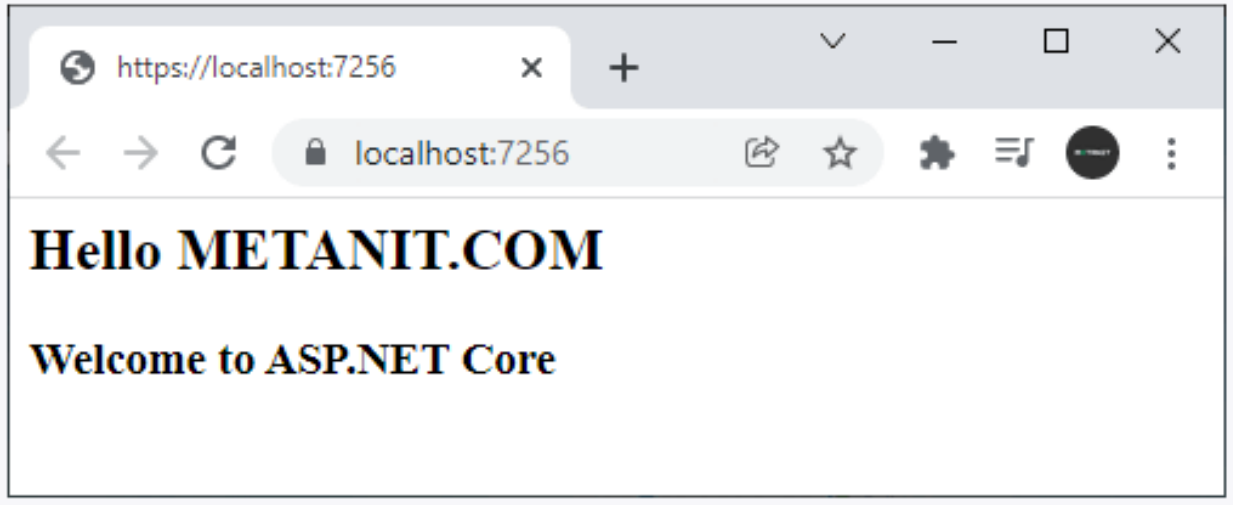


Установка кодов статуса в ASP.NET Core и C#

Отправка html-кода

Если необходимо отправить html-код, то для этого необходимо установить для заголовка **Content-Type** значение **text/html**:

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

app.Run(async (context) =>
{
    var response = context.Response;
    response.ContentType = "text/html; charset=utf-8";
    await response.WriteAsync("<h2>Hello METANIT.COM</h2><h3>Welcome to ASP.NET Core</h3>");
});

app.Run();
```



Отправка html кода из ASP.NET Core и C#

2. HttpRequest. Получение данных запроса

Свойство **Request** объекта **HttpContext** представляет объект **HttpRequest** и хранит информацию о запросе в виде следующих **свойств**:

Body: предоставляет тело запроса в виде объекта **Stream**.

BodyReader: возвращает объект типа **PipeReader** для чтения тела запроса.

ContentLength: получает или устанавливает заголовок **Content-Length**.

ContentType: получает или устанавливает заголовок **Content-Type**.

Cookies: возвращает коллекцию куки (объект **Cookies**), ассоциированных с данным запросом.

Form: получает или устанавливает тело запроса в виде форм.

HasFormContentType: проверяет наличие заголовка **Content-Type**.

Headers: возвращает заголовки запроса.

Host: получает или устанавливает заголовок **Host**.

HttpContext: возвращает связанный с данным запросом объект **HttpContext**.

IsHttps: возвращает true, если применяется протокол https.

Method: получает или устанавливает метод HTTP.

Path: получает или устанавливает путь запроса в виде объекта **RequestPath**.

PathBase: получает или устанавливает базовый путь запроса. Такой путь не должен содержать завершающий слеш.

Protocol: получает или устанавливает протокол, например, HTTP.

Query: возвращает коллекцию параметров из строки запроса.

QueryString: получает или устанавливает строку запроса.

RouteValues: получает данные маршрута для текущего запроса.

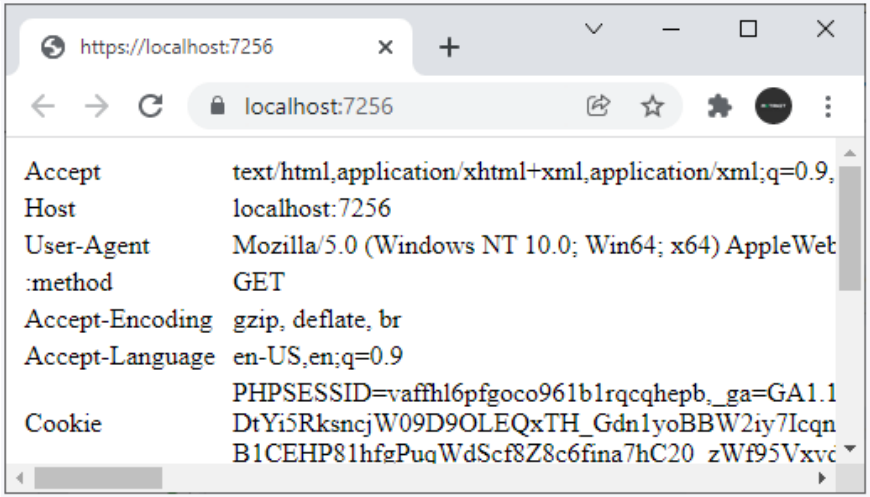
Scheme: получает или устанавливает схему запроса HTTP.

Рассмотрим применение некоторых из этих свойств.

Получение заголовков запроса

Для получения заголовков применяется свойство **Headers**, которое представляет тип **IHeaderDictionary**. Например, получим все заголовки запроса и выведем их на веб-страницу:

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

app.Run(async(context) =>
{
    context.Response.ContentType = "text/html; charset=utf-8";
    var stringBuilder = new System.Text.StringBuilder("<table>");

    foreach(var header in context.Request.Headers)
    {
        stringBuilder.Append($"<tr><td>{header.Key}</td><td>{header.Value}</td></tr>");
    }
    stringBuilder.Append("</table>");
    await context.Response.WriteAsync(stringBuilder.ToString());
});

app.Run();
```



Получение заголовков запроса в ASP.NET Core и C#

Для большинства стандартных заголовков HTTP в этом интерфейсе определены одноименные свойства, например, для заголовка **"content-type"** определено свойство **ContentType**, а для заголовка **"accept"** – свойство **Accept**:

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

app.Run(async(context) =>
{
    var acceptHeaderValue = context.Request.Headers.Accept;
    await context.Response.WriteAsync($"Accept: {acceptHeaderValue}");
});

app.Run();
```

Также подобные заголовки, а также какие-то кастомные заголовки, для которых не определены подобные свойства, можно получить, как и любой дугой элемент словаря:

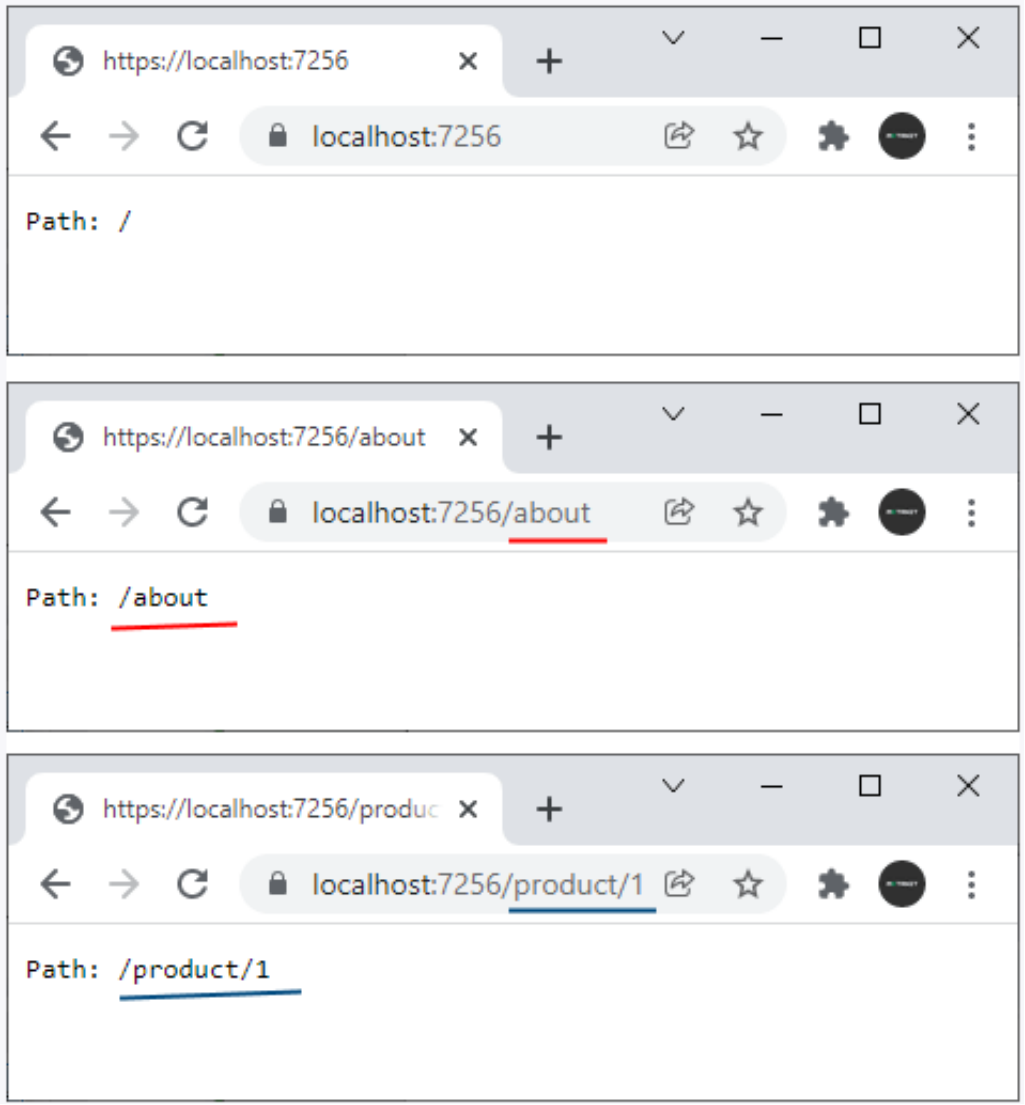
```
var acceptHeaderValue = context.Request.Headers["accept"];
```

Для ряда заголовков в классе **HttpRequest** определены отдельные свойства: **Host**, **Method**, **ContentType**, **ContentLength**.

Получение пути запроса

Свойство path позволяет получить запрошенный путь, то есть адрес, к которому обращается клиент:

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

app.Run(async(context) => await context.Response.WriteAsync($"Path: {context.Request.Path}"));

app.Run();
```



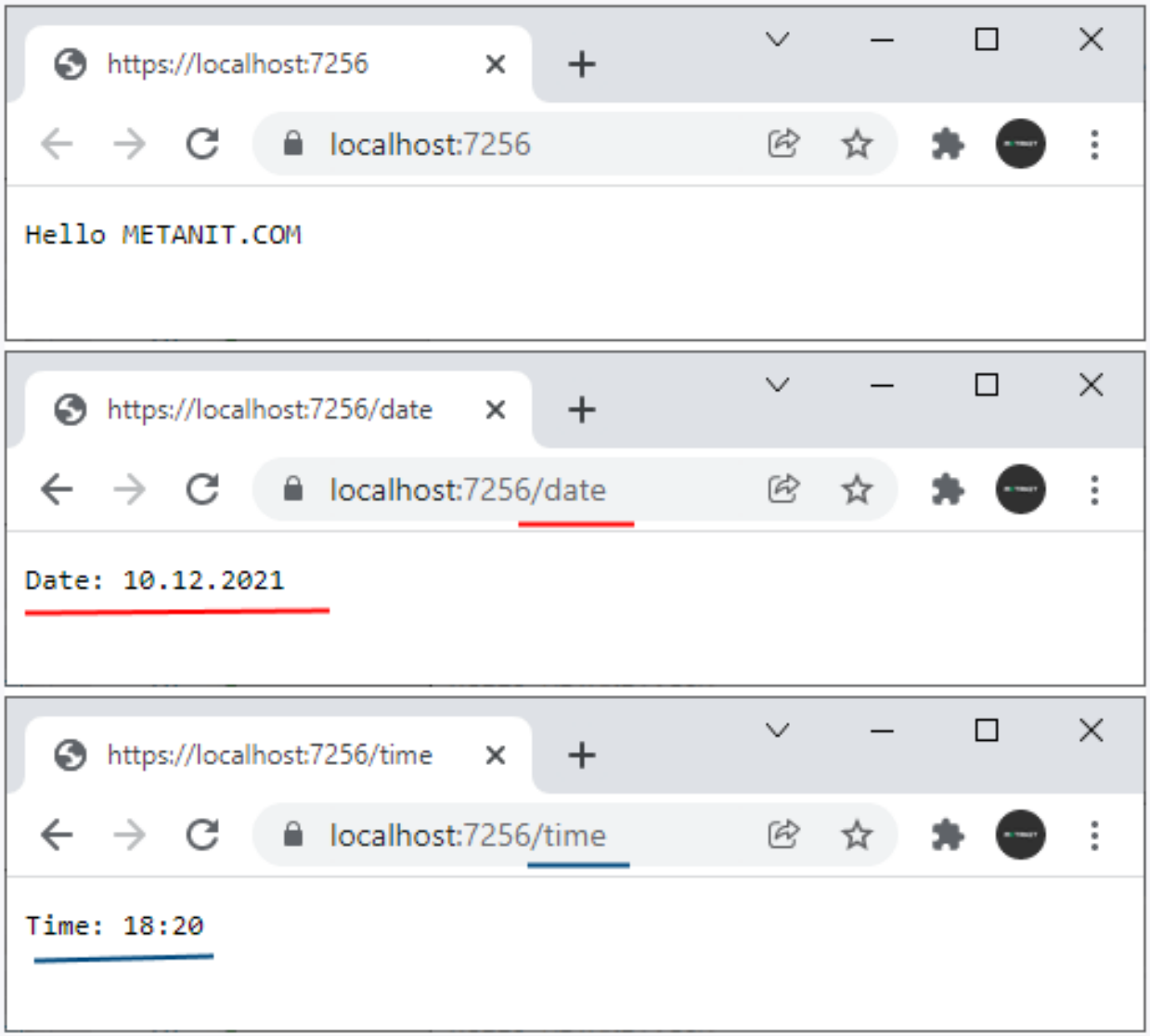
Получение пути запроса в ASP.NET Core и C#

Это свойство позволяет нам узнать, по какому адресу обращается пользователю. Например, мы можем определить условную обработку запроса в зависимости от запрошенного адреса.

Свойство **path** позволяет получить запрошенный путь, то есть адрес, к которому обращается клиент:

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

app.Run(async(context) =>
{
    var path = context.Request.Path;
    var now = DateTime.Now;
    var response = context.Response;

    if (path=="/date")
        await response.WriteAsync($"Date: {now.ToShortDateString()}");
    else if (path == "/time")
        await response.WriteAsync($"Time: {now.ToShortTimeString()}");
    else
        await response.WriteAsync("Hello METANIT.COM");
});

app.Run();
```

В данном случае, если пользователь обращается по адресу **"/date"**, то ему отображается текущая дата, а если обращается по адресу **"/time"** – текущее время. В остальных случаях отображается некоторое универсальное сообщение:



Получение адреса запроса и маршрутизация в ASP.NET Core и C#

Подобным образом можно определить свою систему маршрутизации, однако в ASP.NET Core по умолчанию есть инструменты, которые проще использовать для создания системы маршрутизации в приложении и которые будут рассмотрены на последующих занятиях.

Строка запроса

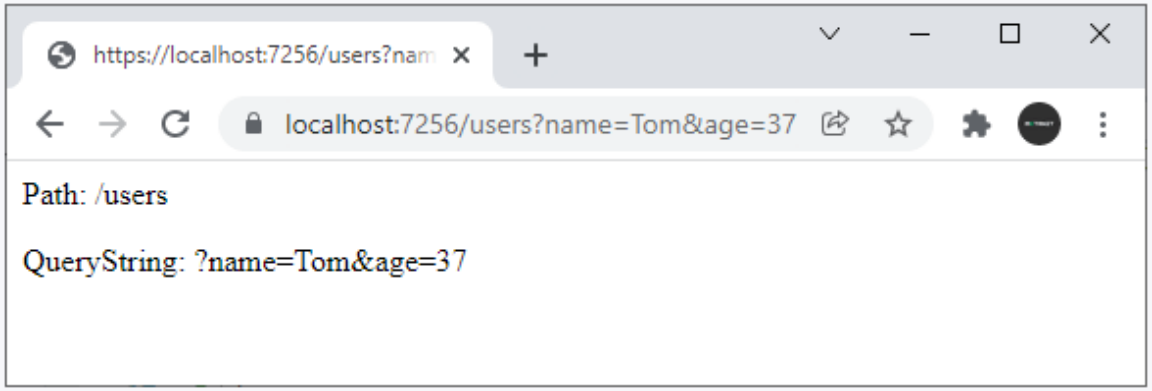
Свойство **QueryString** позволяет получить строку запроса. Строка запроса представляет ту часть запрошенного адреса, которая идет после символа **?** и представляет набор параметров, разделенных символом амперсанда **&**:

```
?параметр1=значение1&параметр2=значение2&параметр3=значение3
```

Каждому параметру с помощью знака равно передается некоторое значение.

Стоит отметить, что строка запроса (**query string**) НЕ входит в путь запроса (**path**):

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

app.Run(async(context) =>
{
    context.Response.ContentType = "text/html; charset=utf-8";
    await context.Response.WriteAsync($"<p>Path: {context.Request.Path}</p>" +
        $"<p>QueryString: {context.Request.QueryString}</p>");
});

app.Run();
```



Получение строки запроса в ASP.NET Core и C#

Так, в данном случае идет обращение по адресу

```
https://localhost:7256/users?name=Tom&age=37
```

Путь запроса или path представляет ту часть адреса, которая идет после домена/порта и до символа **?**.

```
/users
```

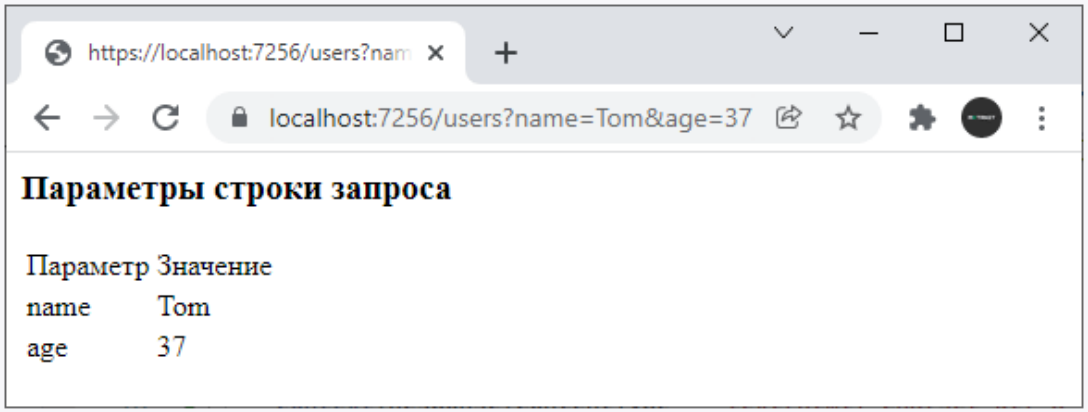
Строка запроса или query string представляет ту часть адреса, которая идет начиная с символа **?**.

```
?name=Tom&age=37
```

То есть в данном случае через строку запроса передаются два параметра. Первый параметр – **name** имеет значение "Tom". Второй параметр – **age** имеет значение 37.

С помощью свойства **Query** можно получить все параметры строки запроса в виде словаря:

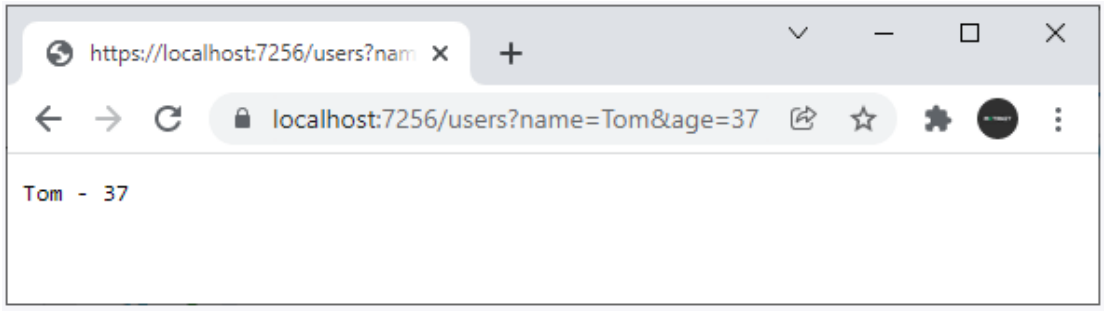
```
var builder = WebApplication.CreateBuilder();
var app = builder.Build();

app.Run(async(context) =>
{
    context.Response.ContentType = "text/html; charset=utf-8";
    var stringBuilder = new System.Text.StringBuilder("<h3>Параметры строки запроса</h3><table>");
    stringBuilder.Append("<tr><td>Параметр</td><td>Значение</td></tr>");
    foreach (var param in context.Request.Query)
    {
        stringBuilder.Append($"<tr><td>{param.Key}</td><td>{param.Value}</td></tr>");
    }
    stringBuilder.Append("</table>");
    await context.Response.WriteAsync(stringBuilder.ToString());
});

app.Run();
```



Парсинг строки запроса в ASP.NET Core и C#

Соответственно можно вытащить из словаря **Query** значения отдельных параметров:

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

app.Run(async(context) =>
{
    string name = context.Request.Query["name"];
    string age = context.Request.Query["age"];
    await context.Response.WriteAsync($"{name} - {age}");
});

app.Run();
```



Параметры строки запроса в ASP.NET Core и C#

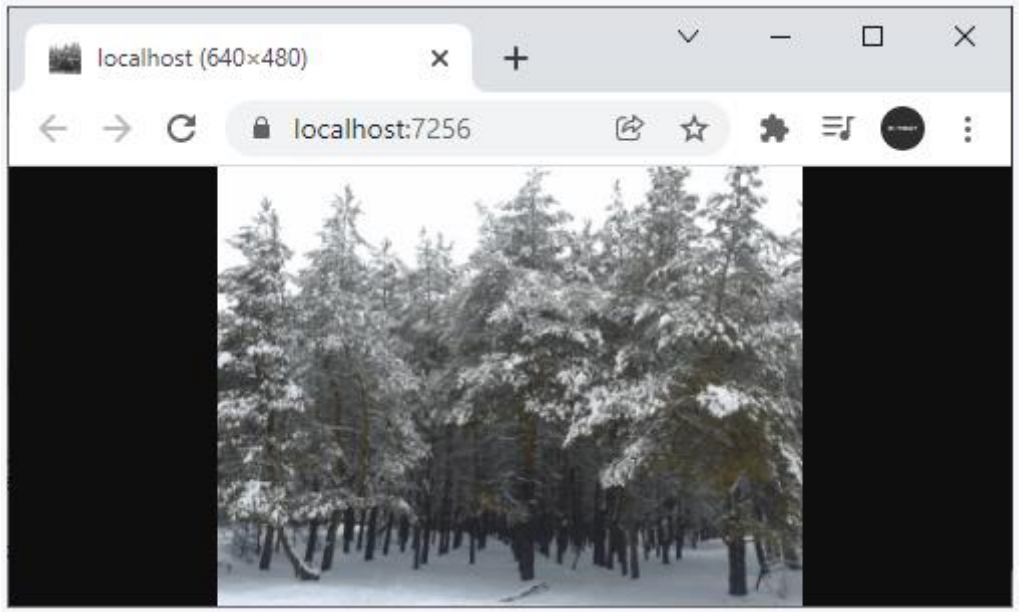
3. Отправка файлов

Для отправки файлов применяется метод **SendFileAsync()**, который получает либо путь к файлу в виде строки, либо информацию о файле в виде объекта **IFileInfo**.

Например, допустим нам надо отправить файл по адресу "D:\\forest.jpg":

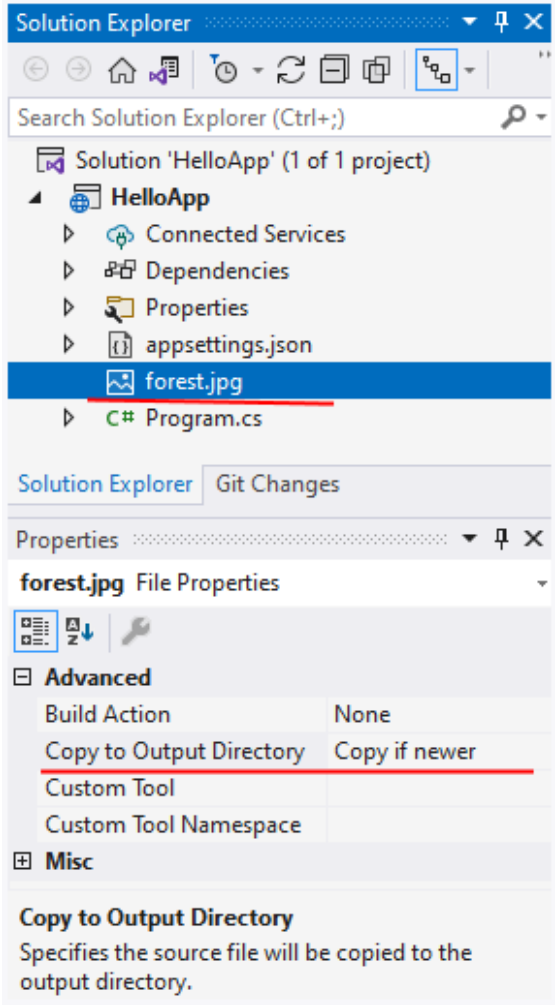
```
var builder = WebApplication.CreateBuilder();
var app = builder.Build();

app.Run(async (context) => await context.Response.SendFileAsync("D:\\forest.jpg"));

app.Run();
```

По умолчанию браузер попытается открыть файл. Так, в случае с изображениями они отображаются в браузере:



SendFileAsync и отправка файлов в ASP.NET Core и C#

Также мы можем использовать относительные пути. Например, добавим в проект какой-нибудь файл (в примере это файл forest.jpg):



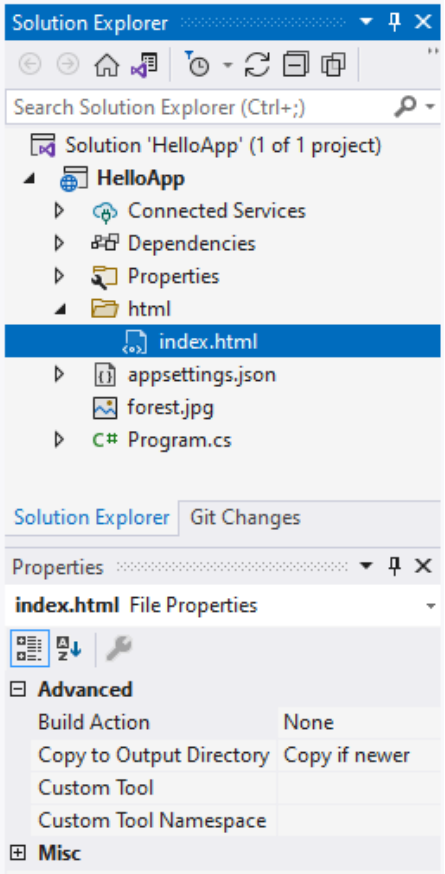
Отправка файлов и SendFileAsync в ASP.NET Core и C#

Для этого файла в окне свойств установим для опции **Copy to Output Directory** значение **Copy if newer** или **Copy always**, чтобы файл автоматически копировался в выходной каталог при построении приложения. И установим относительный путь относительно корня приложения:

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

app.Run(async(context) => await context.Response.SendFileAsync("forest.jpg"));

app.Run();
```

Отправка html-страницы

Подобным образом мы можно отправлять и другие типы файлов, например, html-страницу. Так, определим в проекте новую папку, которую назовем html. В эту папку добавим новый файл index.html:



Отправка html-страницы в ASP.NET Core и C#

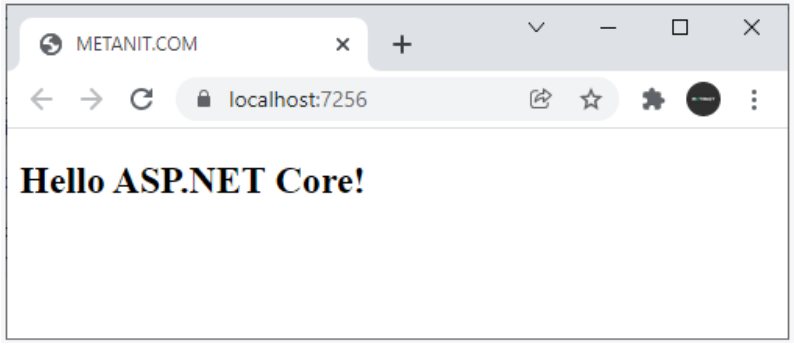
Определим в файле index.html следующий код:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
</head>
<body>
    <h2>Hello ASP.NET Core!</h2>
</body>
</html>
```

Определим для отправки веб-страницы следующий код:

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

app.Run(async(context) =>
{
    context.Response.ContentType = "text/html; charset=utf-8";
    await context.Response.SendFileAsync("html/index.html");
});

app.Run();
```

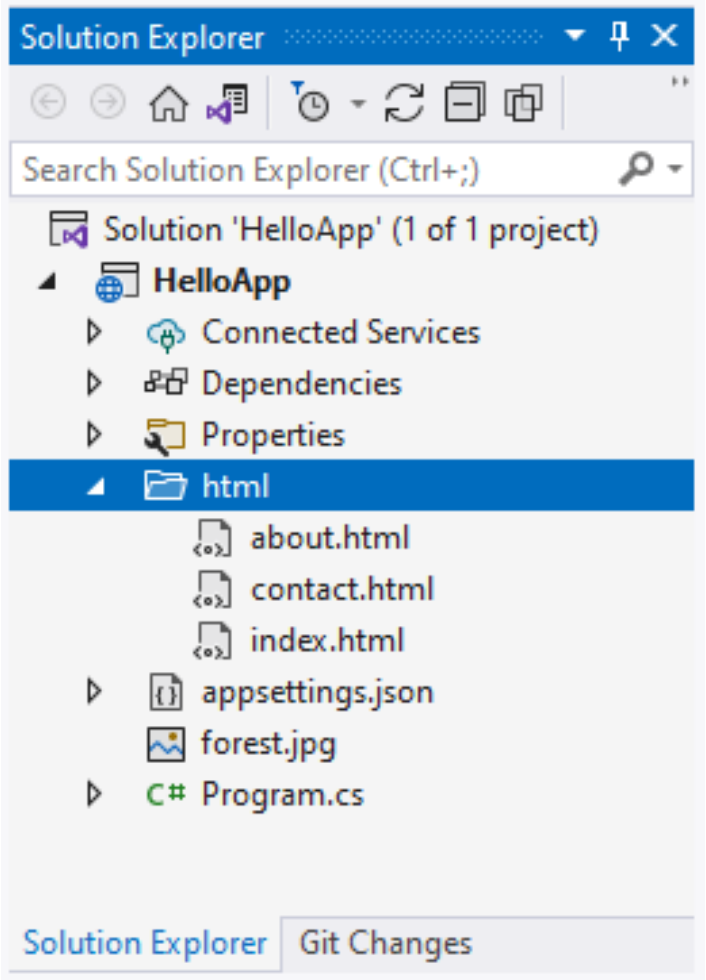
В итоге при обращении к приложению сервер возвратит страницу index.html:



Отправка страницы html с помощью SendFileAsync в ASP.NET Core и C#

Теперь немного усложним задачу. Добавим в проект в папку html еще пару файлов.

Назовем их, к примеру, about.html и contact.html.

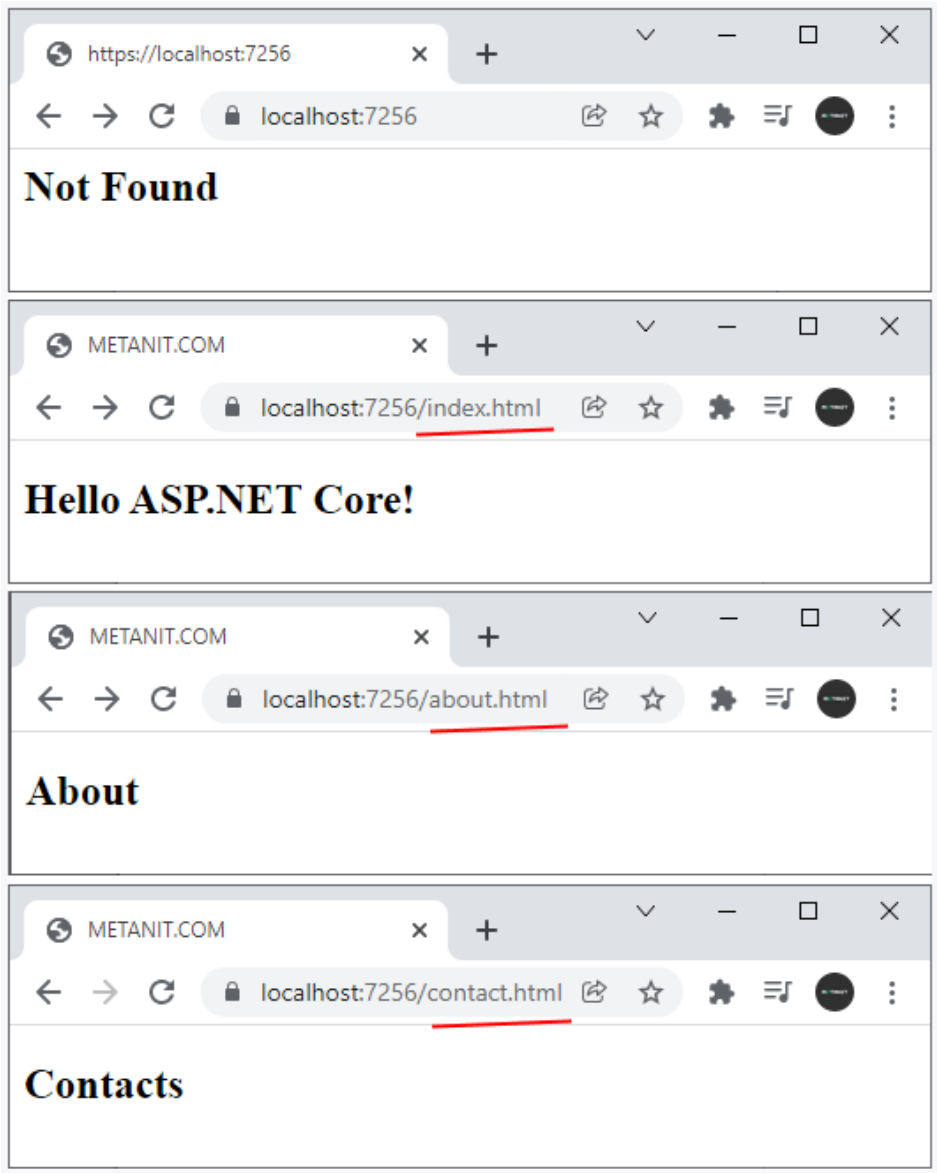


Отправка статических файлов с помощью SendFileAsync в ASP.NET Core и C#

Для отправки этих файлов определим следующий код:

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

app.Run(async(context) =>
{
    var path = context.Request.Path;
    var fullPath = $"html/{path}";
    var response = context.Response;

    response.ContentType = "text/html; charset=utf-8";
    if (File.Exists(fullPath))
    {
        await response.SendFileAsync(fullPath);
    }
    else
    {
        response.StatusCode = 404;
        await response.WriteAsync("<h2>Not Found</h2>");
    }
});

app.Run();
```

Когда приходит запрос, мы сопоставляем путь запроса (path) с файлами в папке html. То есть если path = about.html, то нам надо оправить в ответ файл about.html. При этом проверяем наличие файла. Если он есть в папке, то отправляем данный файл. Если нет, то отправляем статусный код 404 и сообщение, что ресурс не найден:



Отправка статических файлов в ASP.NET Core и C#

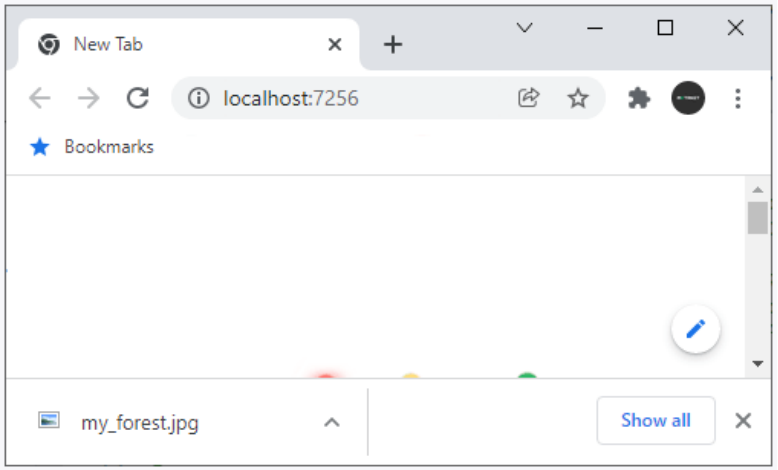
Стоит отметить, что в ASP.NET Core уже имеется встроенный middleware, который позволяет упростить работу со статическими файлами.

Загрузка файла

По умолчанию браузер пытается открыть отправляемый файл, что может быть полезно в случае файлов html – мы можем определить файл html и таким образом отправить клиенту веб-страницу. Но также может быть необходимо, чтобы браузер загружал файл без его открытия. В этом случае мы можем установить для заголовка **"Content-Disposition"** значение **"attachment"**:

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

app.Run(async (context) =>
{
    context.Response.Headers.ContentDisposition = "attachment; filename=my_forest.jpg";
    await context.Response.SendFileAsync("forest.jpg");
});

app.Run();
```

В этом случае загруженный файл получит имя "my_forest.jpg"



SendFileAsync и загрузка файлов в ASP.NET Core и C#

IFileInfo

В примерах выше применялась версия метода **SendFileAsync()**, которая получает путь к файлу в виде строки. Также можно использовать другую версию, которая получает информацию о файле в виде объекта **IFileInfo**:

```
using Microsoft.Extensions.FileProviders;

var builder = WebApplication.CreateBuilder();
var app = builder.Build();

app.Run(async (context) =>
{
    var fileProvider = new PhysicalFileProvider(Directory.GetCurrentDirectory());
    var fileinfo = fileProvider.GetFileInfo("forest.jpg");

    context.Response.Headers.ContentDisposition = "attachment; filename=my_forest2.jpg";
    await context.Response.SendFileAsync(fileinfo);
});

app.Run();
```

В этом случае сначала необходимо определить объект **PhysicalFileProvider**, конструктор которого получает каталог для поиска файлов. В его метод **fileProvider.GetFileInfo()** передается путь к файлу в рамках этого каталога. А результатом метода является объект **IFileInfo**, который передается в **SendFileAsync()**.