**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

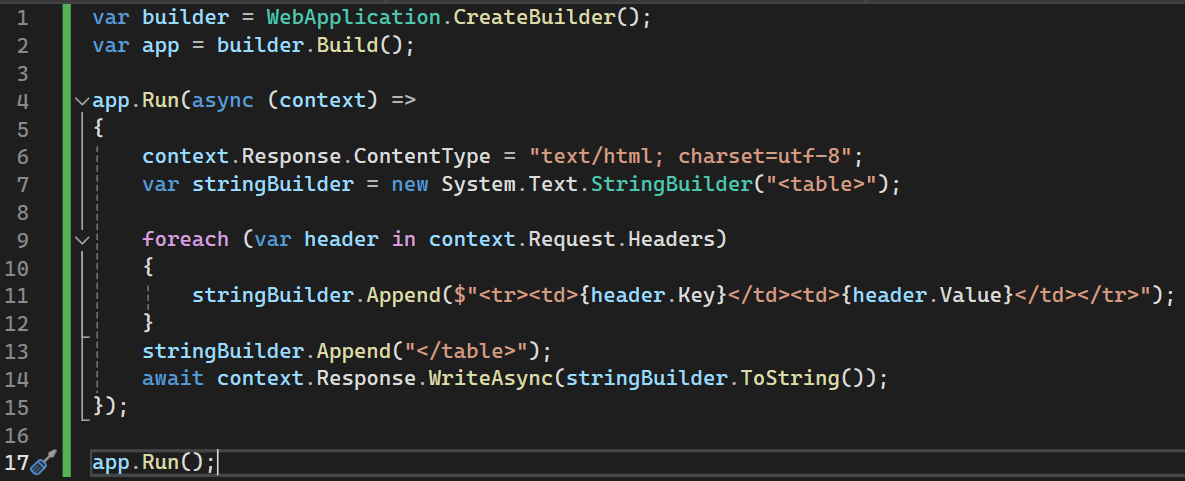
**RouteValues:** получает данные маршрута для текущего запроса

**Scheme:** получает или устанавливает схему запроса HTTP

Рассмотрим применение некоторых из этих свойств.

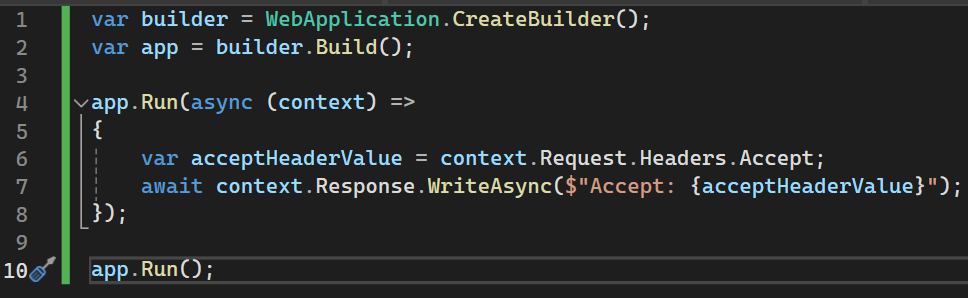
**Получение заголовков запроса**

Для получения заголовков применяется свойство Headers, которое представляет тип IHeaderDictionary. Например, получим все заголовки запроса и выведем их на веб-страницу: (пример 1)





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: (пример 2)





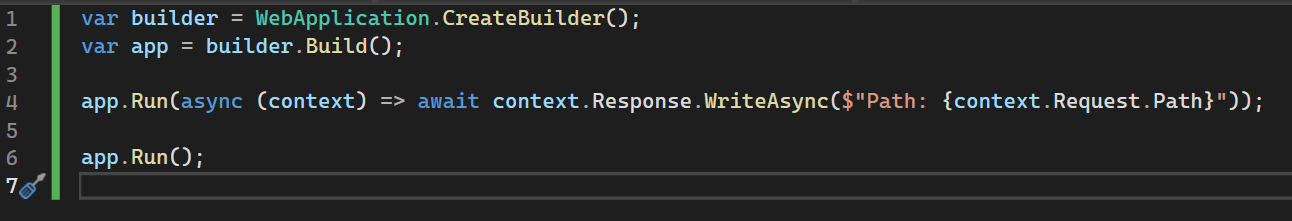
Также подобые заголовки, а также какие-то кастомные заголовки, для которых не определены подобные свойства, можно получить как и любой дугой элемент словаря:

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

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

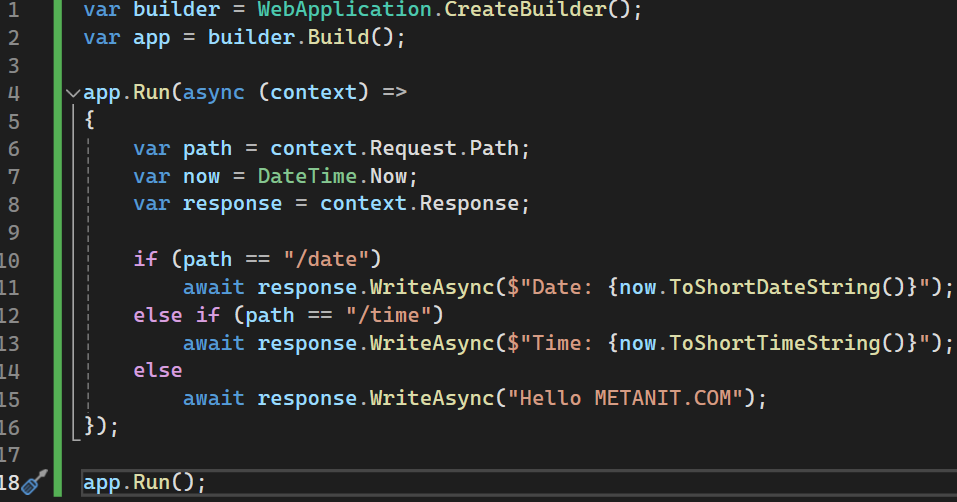
**Получение пути запроса**

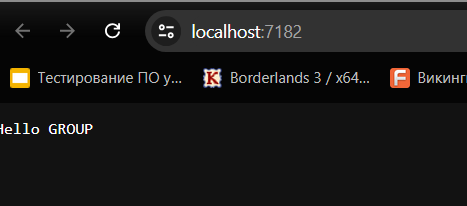
Свойство path позволяет получить запрошенный путь, то есть адрес, к которому обращается клиент: (пример 3)



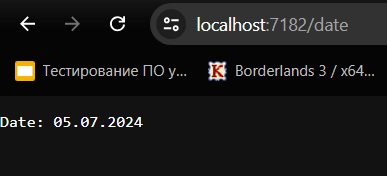
Это свойство позволяет нам узнать, по какому адресу обращается пользователю. Например, мы можем определить условную обработку запроса в зависимости от запрошенного адреса:

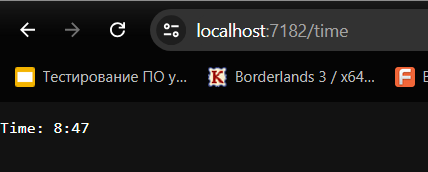
Свойство path позволяет получить запрошенный путь, то есть адрес, к которому обращается клиент: (пример 4)





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:





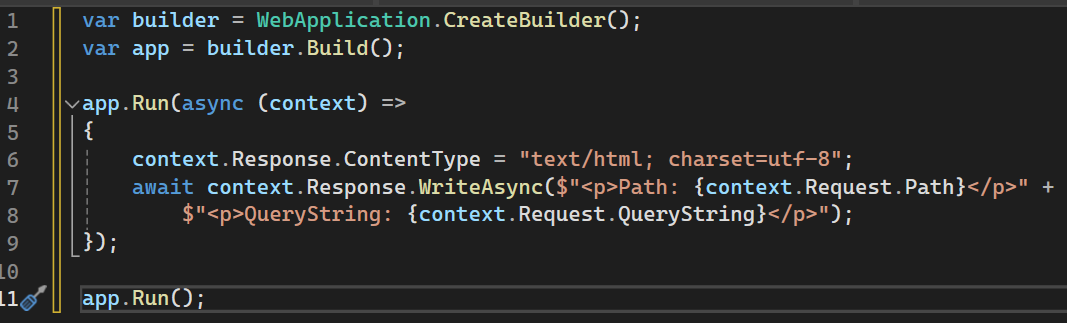
Подобным образом можно определить свою систему маршрутизации, однако в ASP.NET Core по умолчанию есть инструменты, которые проще использовать для создания системы маршрутизации в приложении и которые будут рассмотрены в последующих статьях.

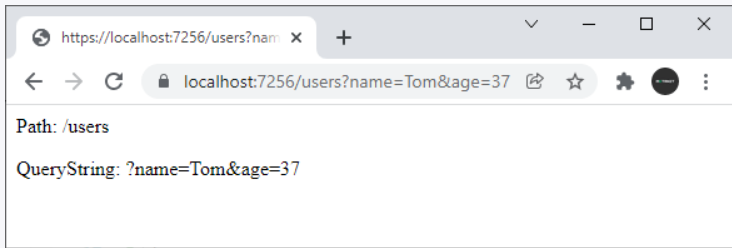
**Строка запроса**

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): (пример 5)





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 ?.

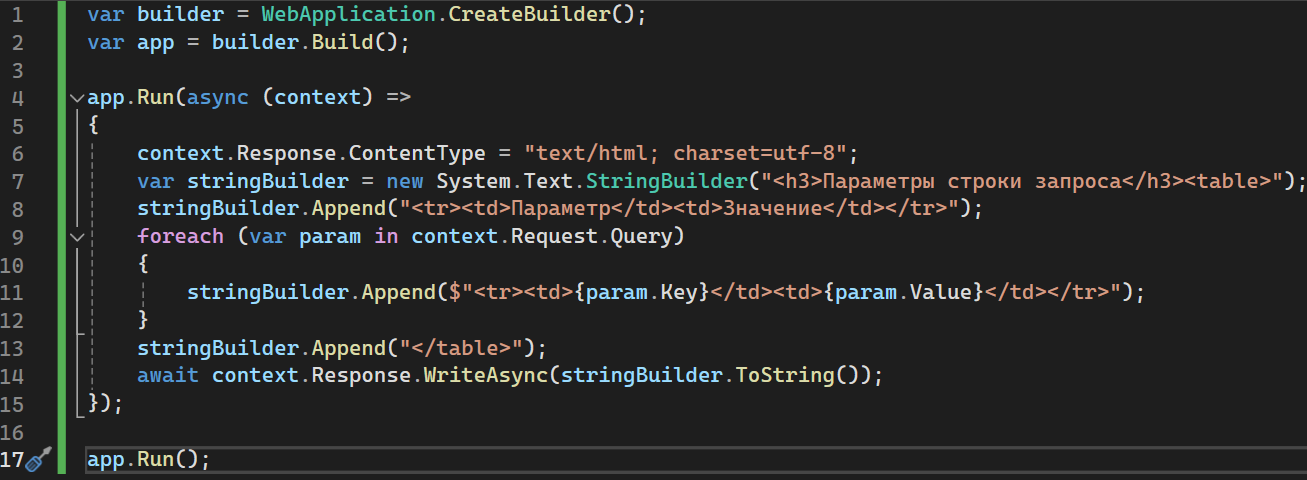
**/users**

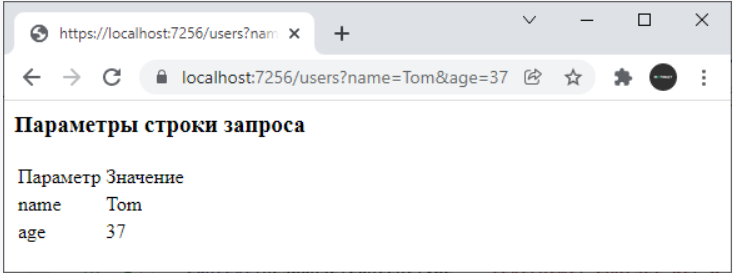
Строка запроса или query string представляет ту часть адреса, которая идет начиная с символа ?.

**?name=Tom&age=37**

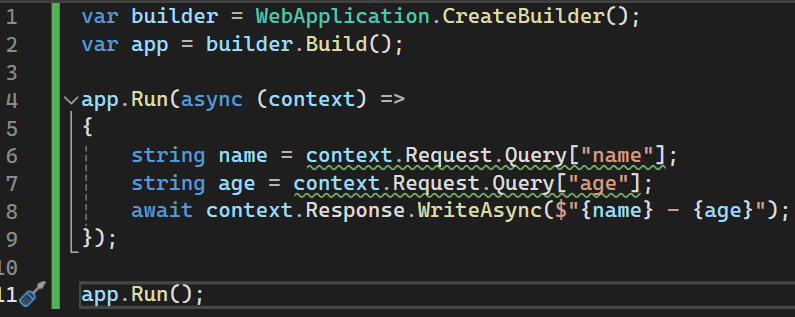
То есть в данном случае через строку запроса передаются два параметра. Первый параметр - name имеет значение "Tom". Bторой параметр - age имеет значение 37.

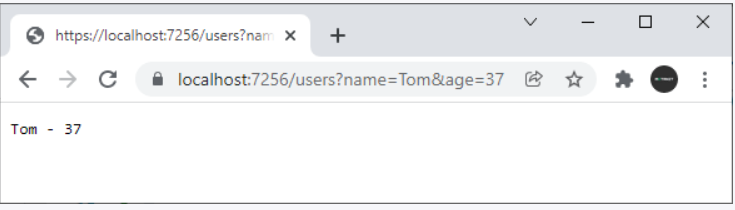
С помощью свойства Query можно получить все параметры строки запроса в виде словаря: (пример 6)





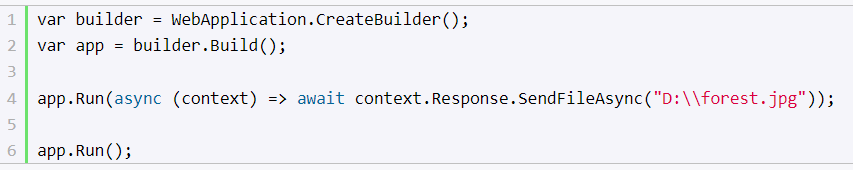
Соответственно можно вытащить из словаря Query значения отдельных параметров: (пример 7)



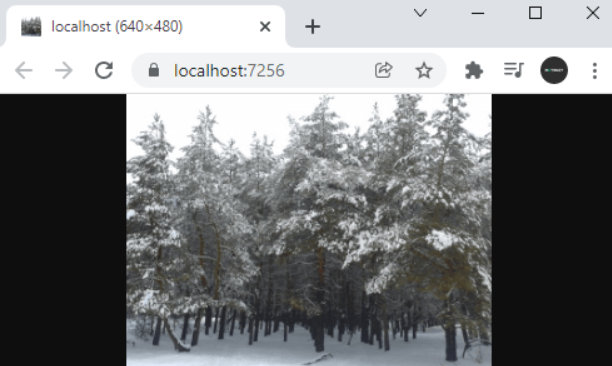


**Отправка файлов**

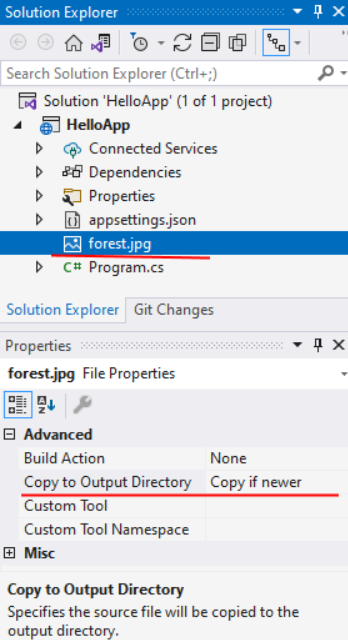
Для отправки файлов применяется метод SendFileAsync(), который получает либо путь к файлу в виде строки, либо информацию о файле в виде объекта IFileInfo. Например, допустим нам надо отправить файл по адресу "D:\\forest.jpg": (пример 8)



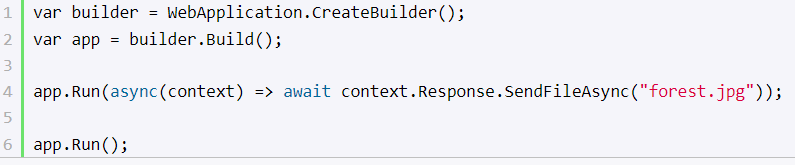
По умолчанию браузер попытается открыть файл. Так, в случае с изображениями они отображаются в браузере:



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

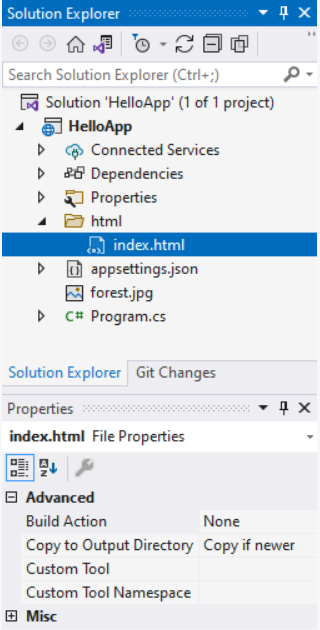


Для этого файла в окне свойств установим для опции Copy to Output Directory значение Copy if newer или Copy always, чтобы файл автоматически копировался в выходной каталог при построении приложения. И установим относительный путь относительно корня приложения: (пример 9)

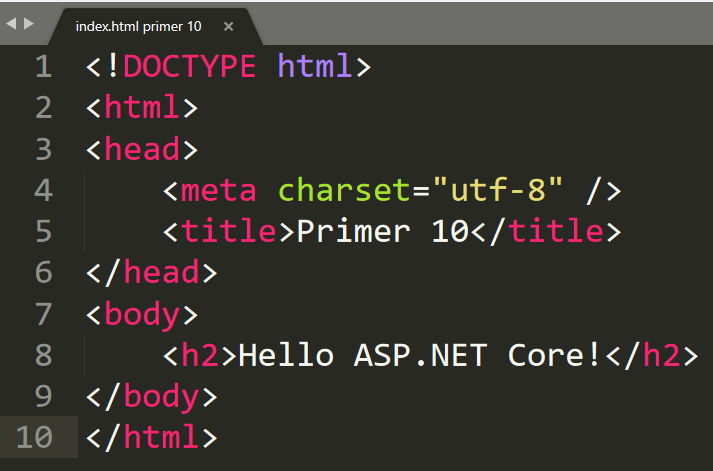


**Отправка html-страницы**

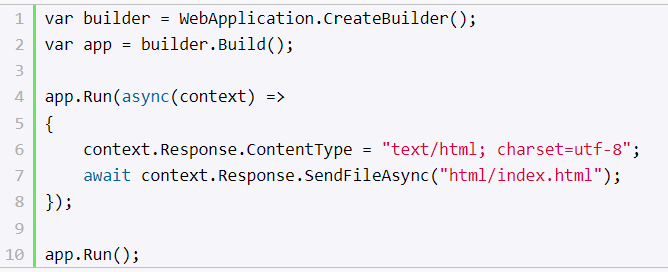
Подобным образом мы можно отправлять и другие типы файлов, например, html-страницу. Так, определим в проекте новую папку, которую назовем html. В эту папку добавим новый файл index.html:



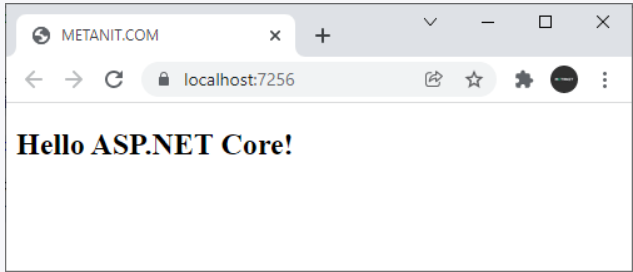
Определим в файле index.html следующий код:



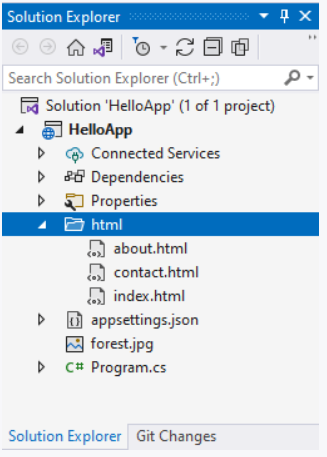
Определим для отправки веб-страницы следующий код: (пример 10)



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



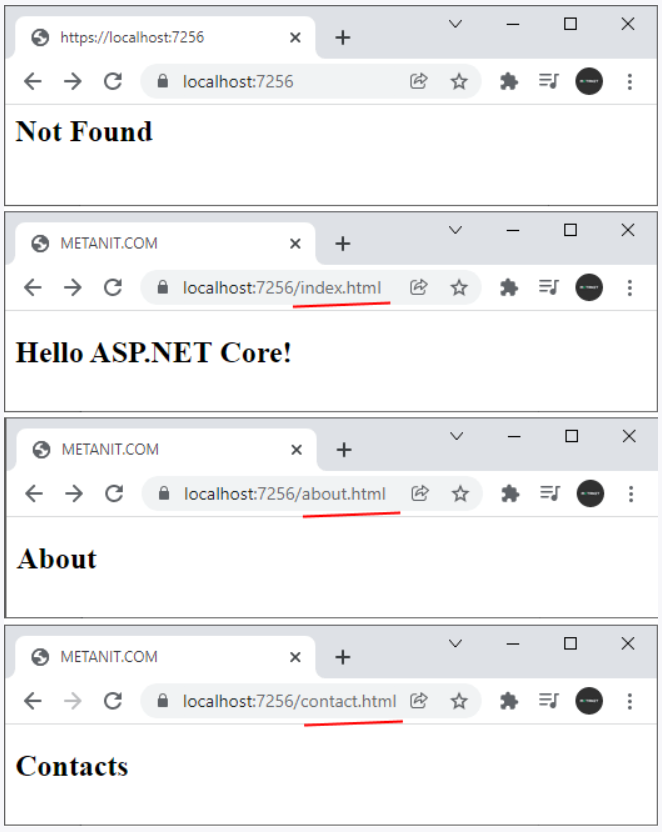
Теперь немного усложним задачу. Добавим в проект в папку html еще пару файлов. Назовем их, к примеру, about.html и contact.html.



Для отправки этих файлов определим следующий код: (пример 11)



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 ресус не найден:



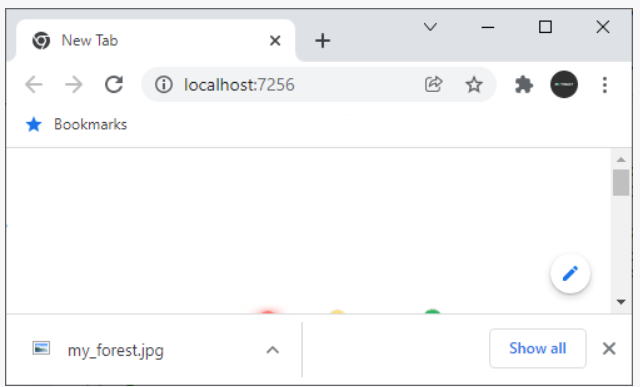
Стоит отметить, что в ASP.NET Core уже имеется встроенный middleware, который позволяет упростить работу со статическими файлами.

**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": (пример 12)



В этом случае загруженный файл получит имя "my\_forest.jpg"



**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: (пример 13)

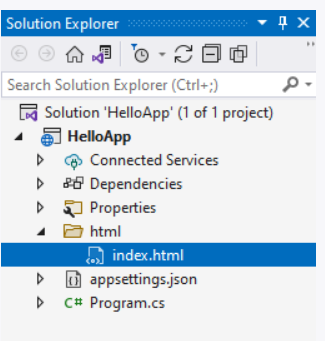


В этом случае сначала необходимо определить объект PhysicalFileProvider, конструктор которого получает каталог для поиска файлов. В его метод fileProvider.GetFileInfo() передается путь к файлу в рамках этого каталога. А результатом метода является объект IFileInfo, который передается в SendFileAsync().

**Отправка форм**

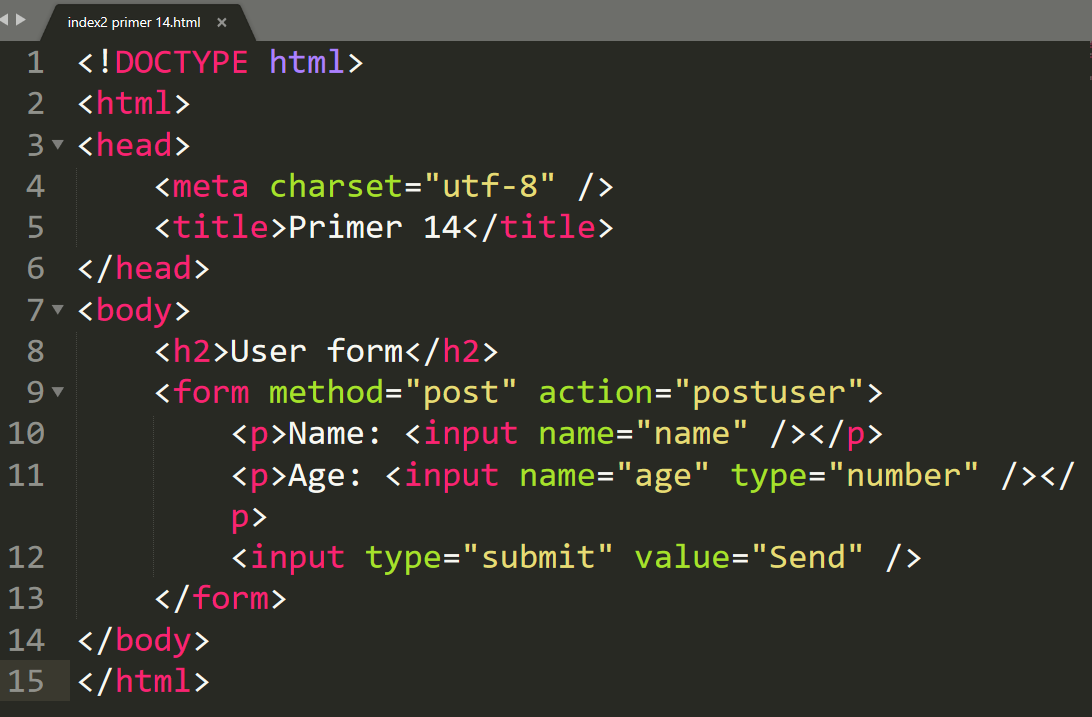
Нередко данные отправляются на сервер с помощью форм html, обычно в запросе типа POST. Для получения подобных данных в классе HttpRequest определено свойство Form. Рассмотрим, как мы можем получить подобные данные.

Прежде всего определим в проекте в папке html файл index2.html.



Определим в нем следующее содержимое:

Определим в нем следующее содержимое:



Здесь определена форма условно для ввода данных пользователя, которая в запросе типа POST (атрибут method="post") отправляет данные по адресу "postuser" (атрибут action="postuser")

На форме определены два поля ввода. Первое поле предназначено для ввода имени пользователя. Второе поле - для ввода возроста пользователя.

Для получения этих данных определим в файле Program.cs следующий код: (пример 14)



Здесь, если запрошен адрес "/postuser", то предполагается, что отправлена некоторая форма. Сначала получаем отправленную форму в переменную form:



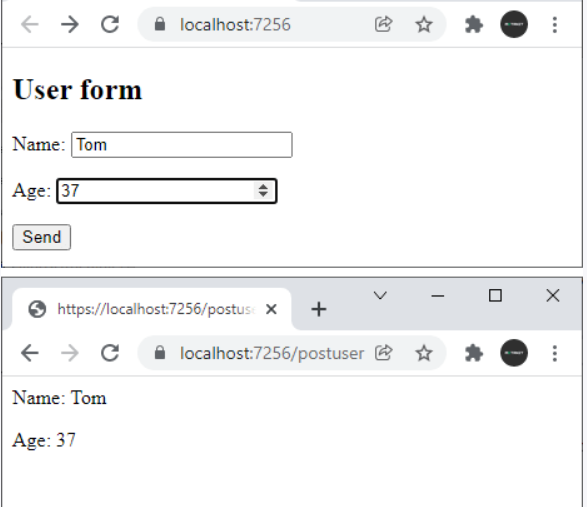
Свойство Request.Form возвращает объект IFormCollection - своего рода словарь, где по ключу можно получить значение элемента. При этом в качестве ключей выступает названия полей форм (значения атрибутов name элементов формы):



Так, в данном случае название поля (значение атрибута name) равно "age". Соответственно в Request.Form по этому имени мы можем получить его значение:

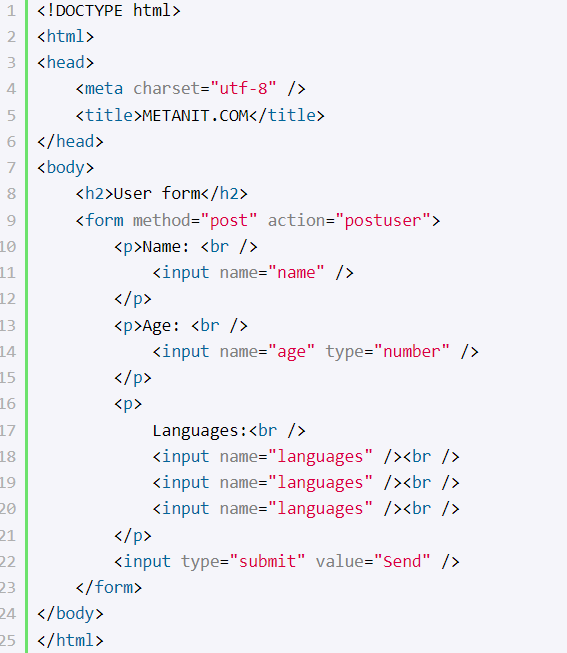


После получения данных формы они отправляются обратно клиенту:



**Получение массивов**

Усложним задачу и добавим в форму на странице index3.html несколько полей, которые будут представлять массив: (файл index3.html)



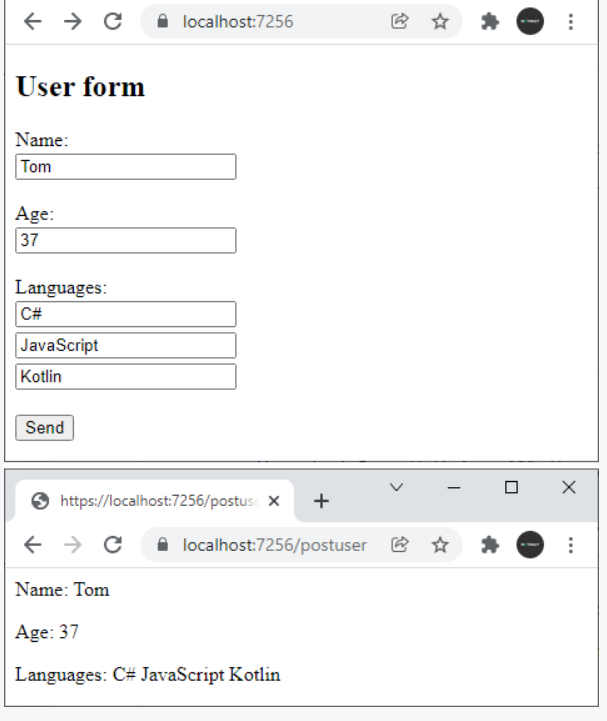
Здесь добавлено три поля ввода, которые имеют одно и то же имя. Поэтому при их отправке будет формироваться массив из трех значений. Теперь получим эти значения в коде C#:



Поскольку параметр "languages" представляет массив, то и сопоствляться он будет с массивом строк:



Для вывода на веб-страницу из этого массива формируется код html в виде строки:



Подобным образом можно передавать значения массива полей других типов, либо полей, которые представляют набор элементов, например, элемента select, который поддерживает множественный выбор: (index 4 html)

