**Метод MAP**

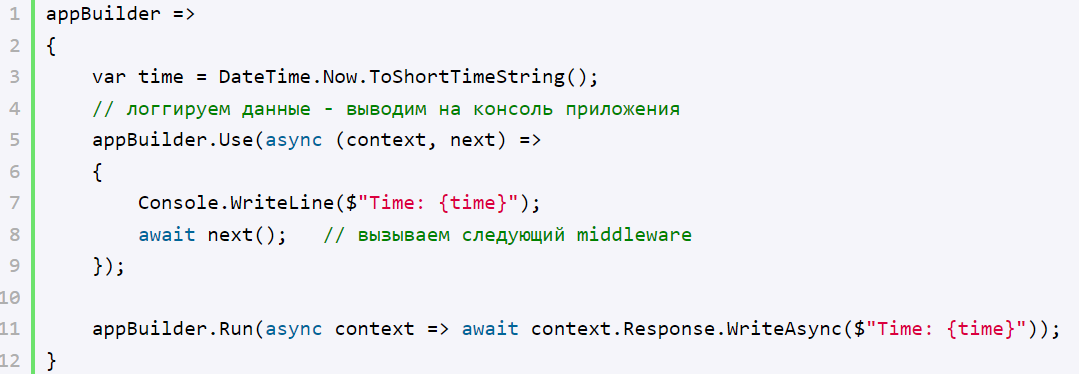
Метод Map() применяется для создания ветки конвейера, которая будет обрабатывать запрос по определенному пути. Этот метод реализован как метод расширения для типа IApplicationBuilder и имеет ряд перегруженных версий. Например:

**public static IApplicationBuilder Map (this IApplicationBuilder app, string pathMatch, Action<IApplicationBuilder> configuration);**

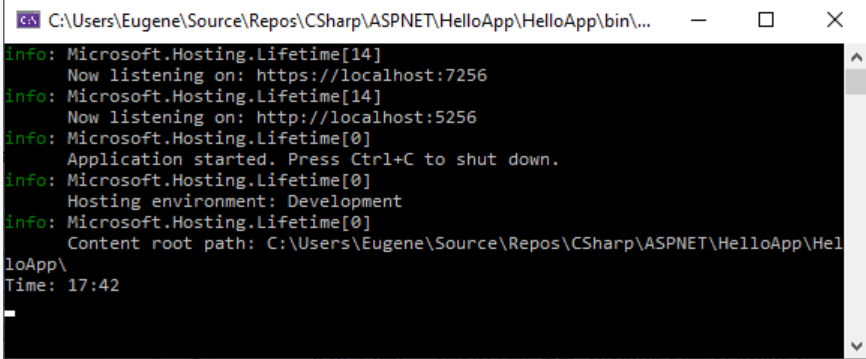
В качестве параметра pathMatch метод принимает путь запроса, с которым будет сопоставляться ветка. А параметр configuration представляет делегат, в который передается объект IApplicationBuilder и в котором будет создаваться ветка конвейера.

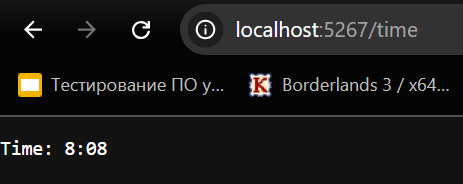
Рассмотрим простой пример: (pr7)

В данном случае метод app.Map() создает ответвление конвейера, которое будет обрабатывать запросы по пути "/time":



Созданная ветка конвейера содержит два middleware, встраиваемые с помощью методов Use() и Run(). Вначале получаем текущее время и в первом middleware логгируем это время на консоль. Во втором - терминальном компоненте middleware отправляем информацию о времени в ответ клиенту.

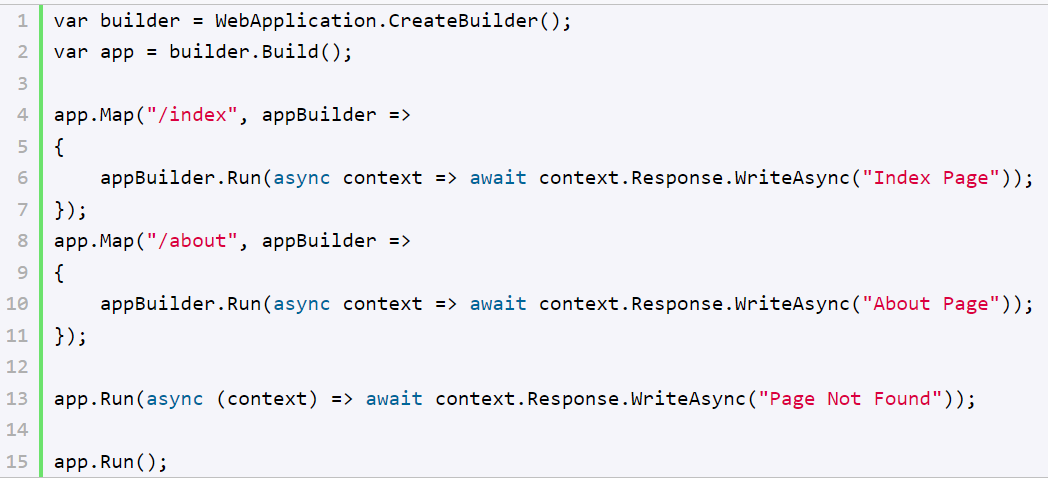


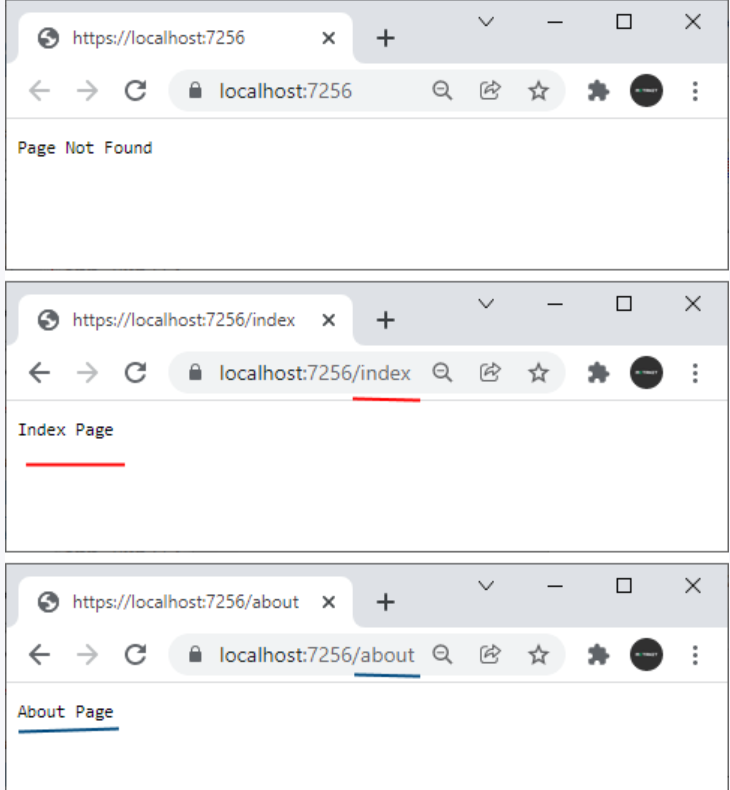


При других путях запросах, отличных от "/time", запрос будет обрабатываться основным потоком конвейера, который состоит в данном случае из одного компонента:

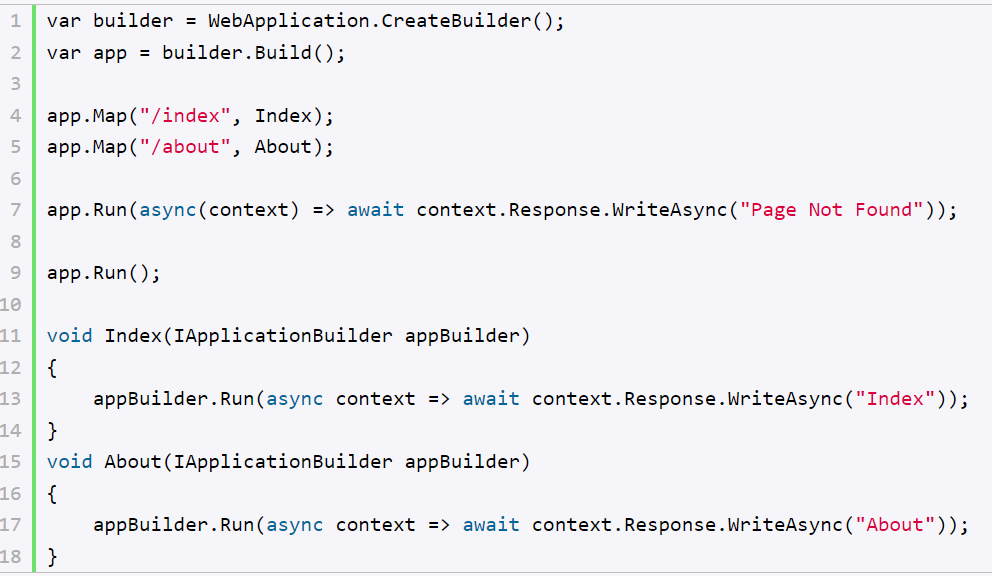
**app.Run(async (context) => await context.Response.WriteAsync("Hello ASP NET"));**

Подобным образом можно создавать ветки для разных путей(pr8)



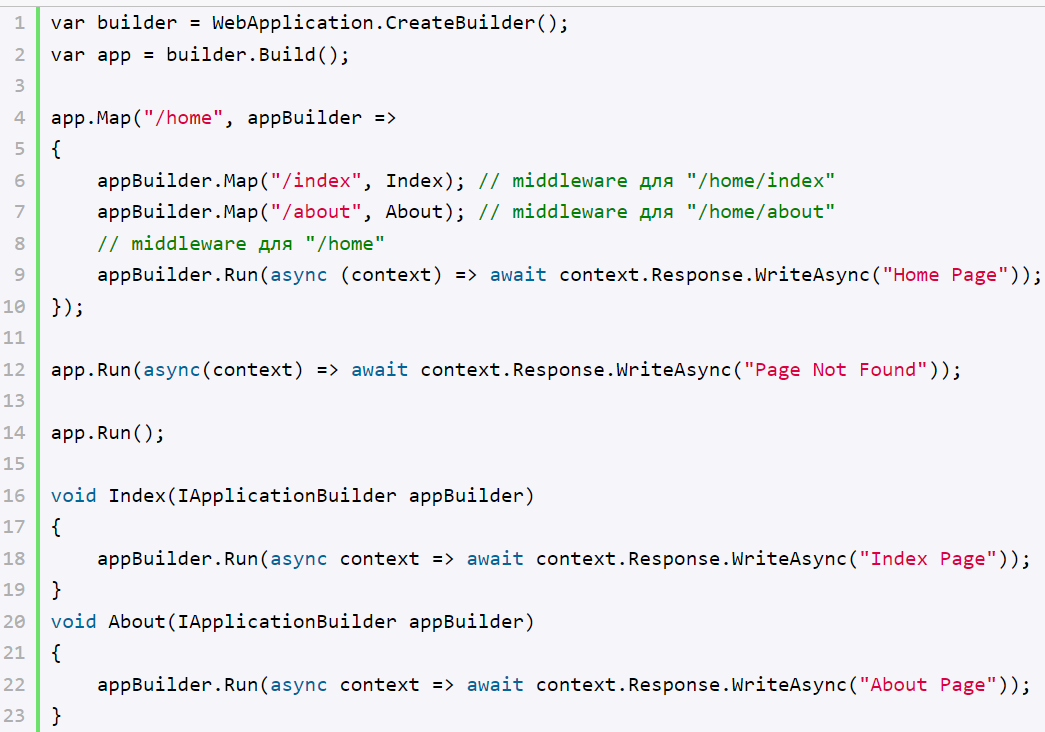


При необходимости создание веток конвейера можно вынести в отдельные методы: (pr9)



**Вложенные методы Map**

Ветка конвейера, которая создается в методе Map(), может иметь вложенные ветки, которые обрабатывают подзапросы. Например (pr10):

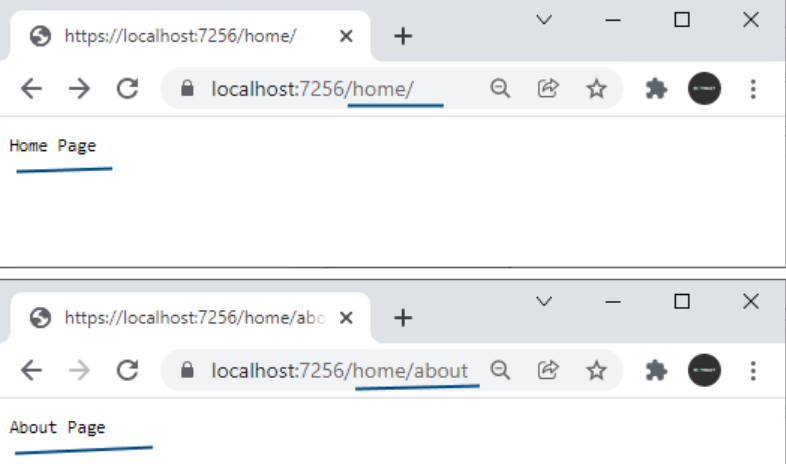


Здесь ветка создается с помощью вызова



Эта ветка будет обрабатывать запросы по пути "/home".

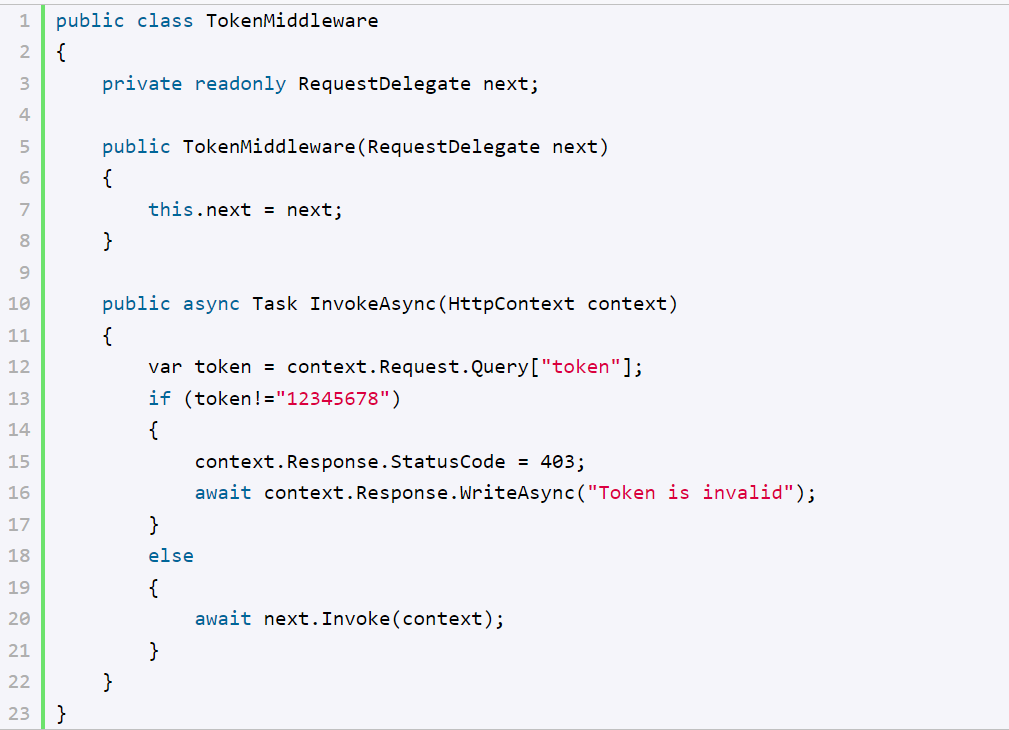
Внутри этой ветки создаются две вложенные ветки, которые будут обрабатывать запросы по путям относительно пути основной ветки. То есть теперь метод About будет обрабатывать запрос по пути "/home/about", а не "/about".



**Классы middleware**

В прошлых темах используемые компоненты middleware фактически представляли методы, то есть так называемые inline middleware. Однако также ASP.NET Core позволяет определять middleware в виде отдельных классов.

Итак, добавим в проект новый класс, который назовем TokenMiddleware и который будет иметь следующий код: (pr 11)

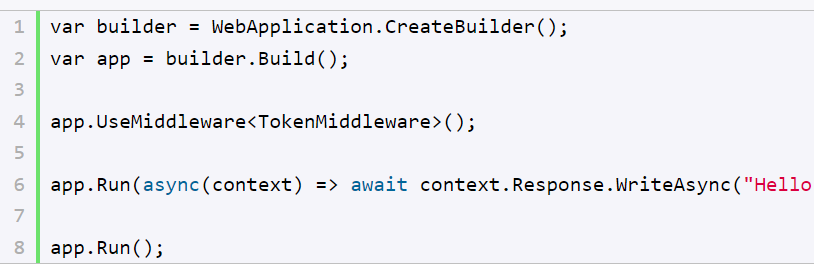


Класс middleware должен иметь конструктор, который принимает параметр типа RequestDelegate. Через этот параметр можно получить ссылку на тот делегат запроса, который стоит следующим в конвейере обработки запроса.

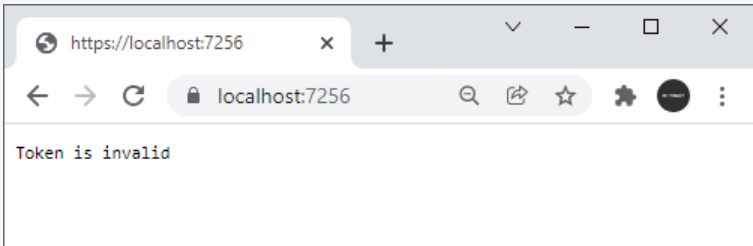
Также в классе должен быть определен метод, который должен называться либо Invoke, либо InvokeAsync. Причем этот метод должен возвращать объект Task и принимать в качестве параметра контекст запроса - объект HttpContext. Данный метод собственно и будет обрабатывать запрос.

Суть действия класса заключается в том, что мы получаем из запроса параметр "token". Если полученный токен равен строке "12345678", то передаем запрос дальше следующему компоненту, вызвав метод \_next.Invoke(). Иначе возвращаем пользователю сообщение об ошибке.

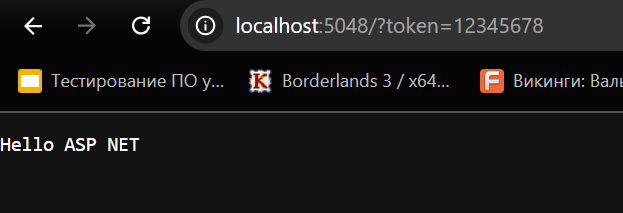
Для добавления компонента middleware, который представляет класс, в конвейер обработки запроса применяется метод UseMiddleware(). Так, изменим файл Program.cs следующим образом: (pr11)



Запустим проект. И если мы не передадим через строку запроса параметр token или передадим для него значение, отличное от "12345678", то браузер отобразит ошибку:

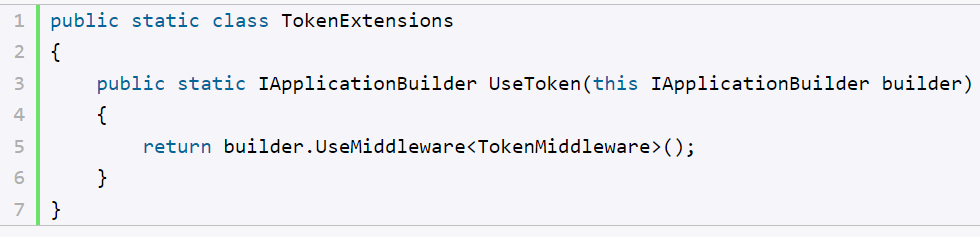


Если же будет передан корректный токен, то вызов app.UseMiddleware<TokenMiddleware>() передаст обработку запроса в компонент middleware из app.Run():



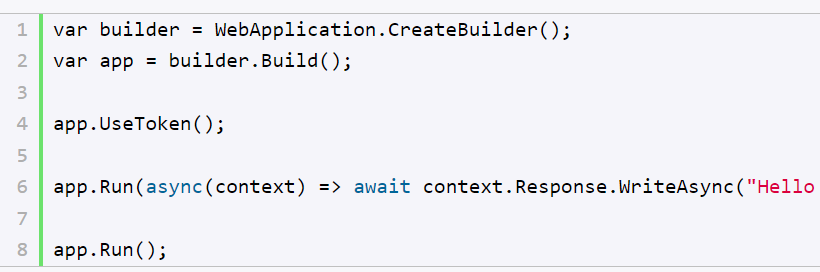
**Метод расширения для встраивания middleware**

Также нередко для встраивания подобных компонентов middleware определяются специальные методы расширения. Так, добавим в проект новый класс, который назовем TokenExtensions:



Здесь создается метод расширения для типа IApplicationBuilder. И этот метод встраивает компонент TokenMiddleware в конвейер обработки запроса. Как правило, подобные методы возвращают объект IApplicationBuilder.

Теперь применим этот метод в коде программы в Program.cs:

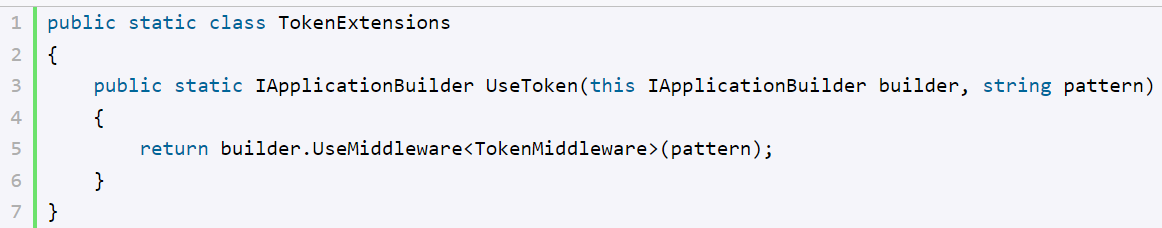


Передача параметров

Изменим класс TokenMiddleware, чтобы он извне получал образец токена для сравнения:



Образец токена, с которым идет сравнения, устанавливается через конструктор. Чтобы передать его в конструктор, изменим класс TokenExtensions:



В метод builder.UseMiddleware можно передать набор значений, которые передаются в конструктор компонента middleware.

И при вызове метода расширения UseToken в него можно передать конкретное значение: (pr12)

