



Создание первого клиентского веб-приложения. Знакомство с Blazor.

Overview

Современные сайты отличаются от тех, которыми мы пользовались раньше: они более динамичны. С их помощью можно покупать товары, общаться с друзьями или писать комментарии.

Изменилась и структура сайтов. Изначально они состояли из набора отдельных HTML-страниц. Сейчас они представляют собой один HTML-документ, внешний вид которого меняется динамически через JavaScript.

Такой подход получил название SPA — single-page application, или одностраничное приложение. SPA делает сайты полноценными приложениями, которые не перезагружаются между переходами по страницам. Чтобы выполнить переход, нужно полностью перерисовать контент документа с помощью JavaScript и полученных данных.

Сайты на чистом JavaScript давно стали редкостью: обычно при разработке используют библиотеки и фреймворки. Один из таких фреймворков который мы рассмотрим — Blazor.

Что такое Blazor и для чего он нужен?

Blazor представляет UI-фреймворк для создания интерактивных приложений, которые могут работать как на стороне сервера, так и на стороне клиента, на платформе .NET.

В своем развитии фреймворк Blazor испытал большое влияние современных фреймворков для создания клиентских приложений - Angular, React, VueJS.

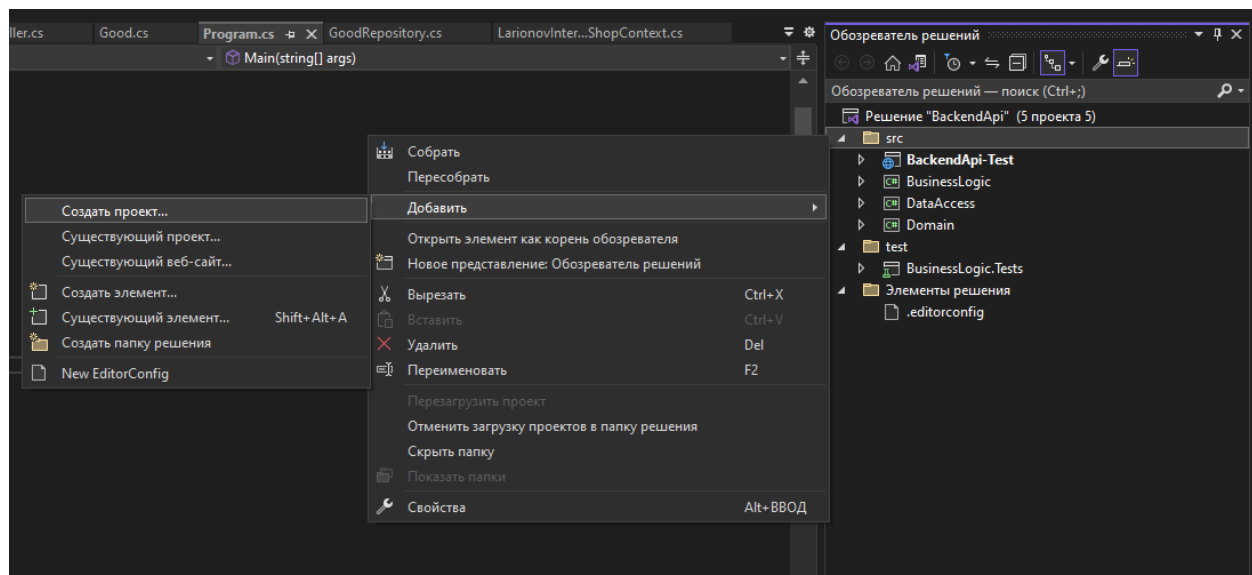
В частности, это проявляется в роли компонентов при построении пользовательского интерфейса. В то же время и на стороне клиента, и на стороне сервера при определении кода в качестве языка программирования применяется C#, вместо JavaScript. А для описания визуального интерфейса используются стандартные HTML и CSS.

Почему именно этот фреймворк может вам приглянуться для разработки:

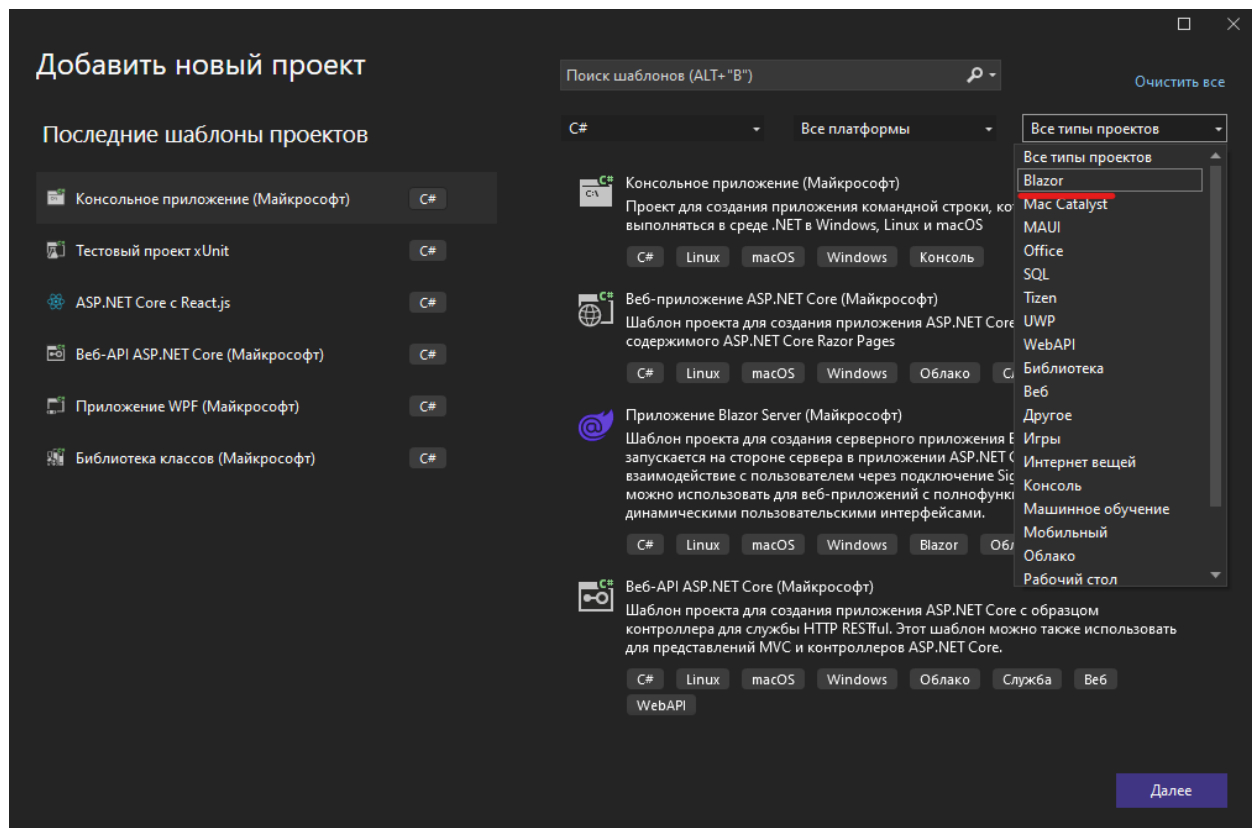
- Написание исполняемого кода с помощью C# вместо JavaScript
- Использование библиотек и инструментов .NET, например: Linq, EF, Mapster и т.д.
- Visual Studio позволяет создавать шаблон проекта Blazor на столько же просто, как вы это делаете при создании десктопных приложений (Таких как WPF, WinForms и т.д.)

Создание проекта Blazor и разбор элементов проекта

Откройте проект с вашим API и откройте окно для создания нового проекта



Укажите тип проекта “Blazor” и выберем подходящий из них



Выбор подходящей подсистемы

Функционально на текущий момент Blazor подразделяется на несколько подсистем:

- **Blazor Server**: позволяет создавать серверные приложения и поддерживается ASP.NET
- **Blazor WebAssembly**: позволяет создавать одностраничные интерактивные приложения клиентской стороны, которые запускаются в браузере пользователя и работают с помощью технологии WebAssembly
- **Blazor Hybrid**: позволяет создавать десктопные и мобильные приложения поверх технологии .NET MAUI

Следует отметить, что каждый из них имеет свои недостатки. Поэтому следует выбирать один из таких решений в зависимости от решаемой задачи.



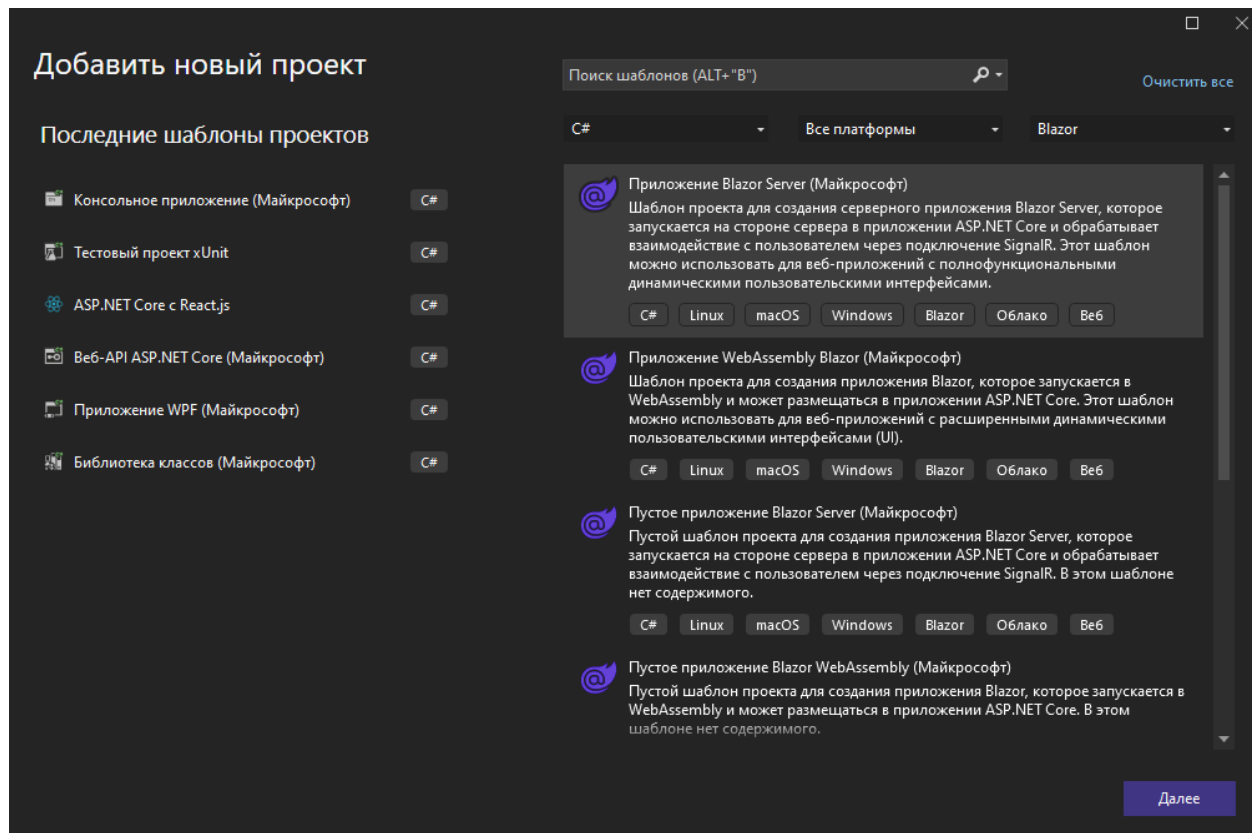
Наша же задача познакомиться с веб-разработкой и сделать таковую максимально быстрой (Например для курсовых или диплома)

Так как стоит продемонстрировать преимущества многослойной архитектуры - воспользуемся подсистемой Blazor Server, который будет генерировать страницы со стороны сервера.

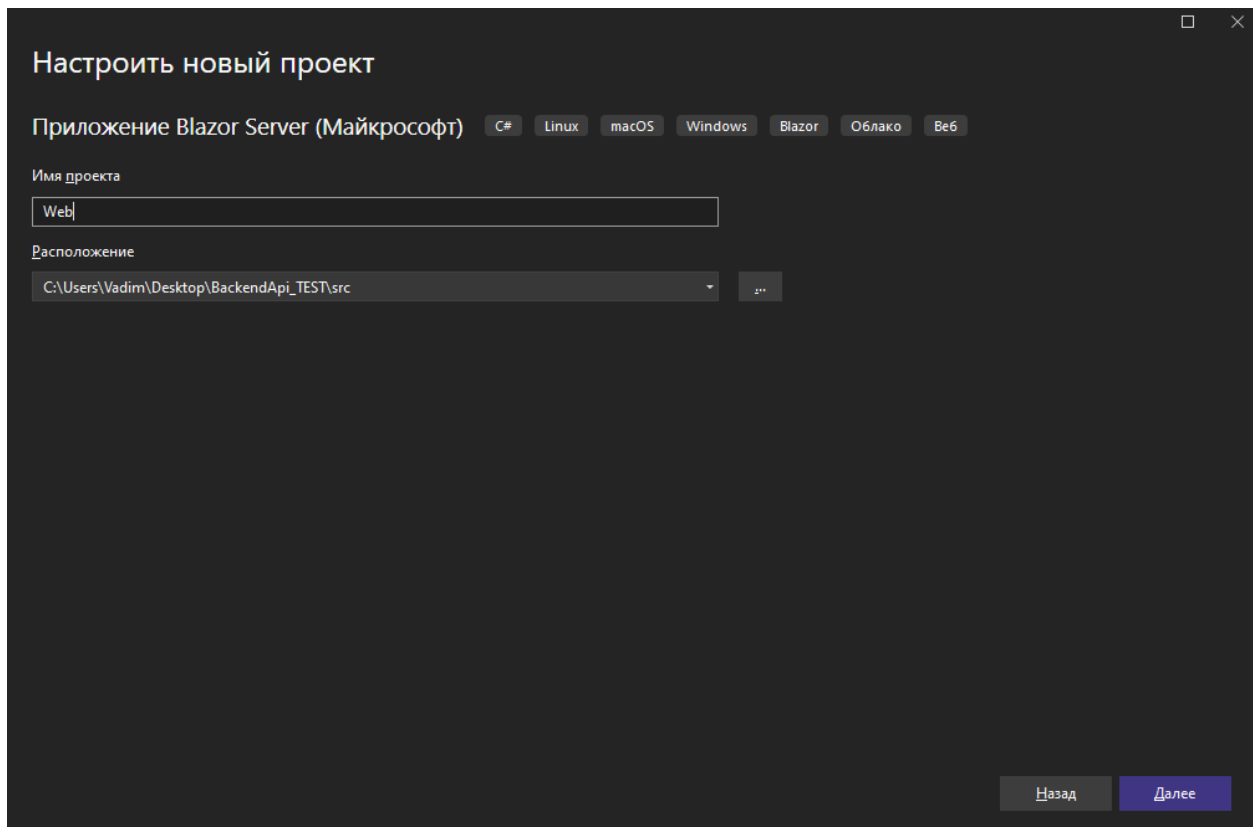


WebAssembly для такой задачи не подойдет, потому что страницы будут генерироваться со стороны клиента

Выберем шаблон “Приложение Blazor Server”



Укажите наиболее подходящее название для данного проекта (Например: Web, WebServer, BlazorServer и т.д.)



Настроить новый проект

Приложение Blazor Server (Майкрософт) C# Linux macOS Windows Blazor Облако Be6

Имя проекта

Web

Расположение

C:\Users\Vadim\Desktop\BackendApi_TEST\src

Назад Далее

Также стоит поставить галочку “Не использовать операторы верхнего уровня”

Дополнительные сведения

Приложение Blazor Server (Майкрософт) C# Linux macOS Windows Blazor Облако Be6

Платформа ⓘ

.NET 6.0 (долгосрочная поддержка)

Тип проверки подлинности ⓘ

Нет

☒ Настроить для HTTPS ⓘ

☐ Включить Docker ⓘ

Операционная система Docker ⓘ

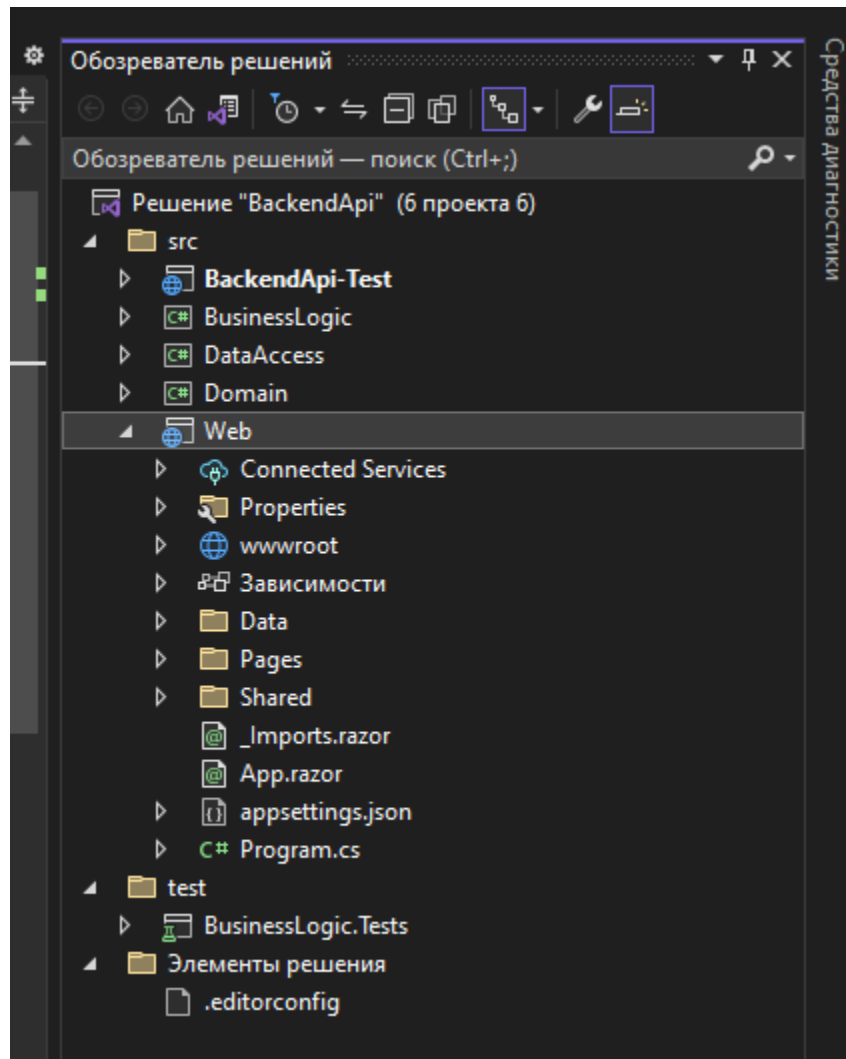
Linux

☒ Не использовать операторы верхнего уровня ⓘ

Назад

Создать

Теперь у нас появился проект который имеет следующий набор элементов



Содержание проекта

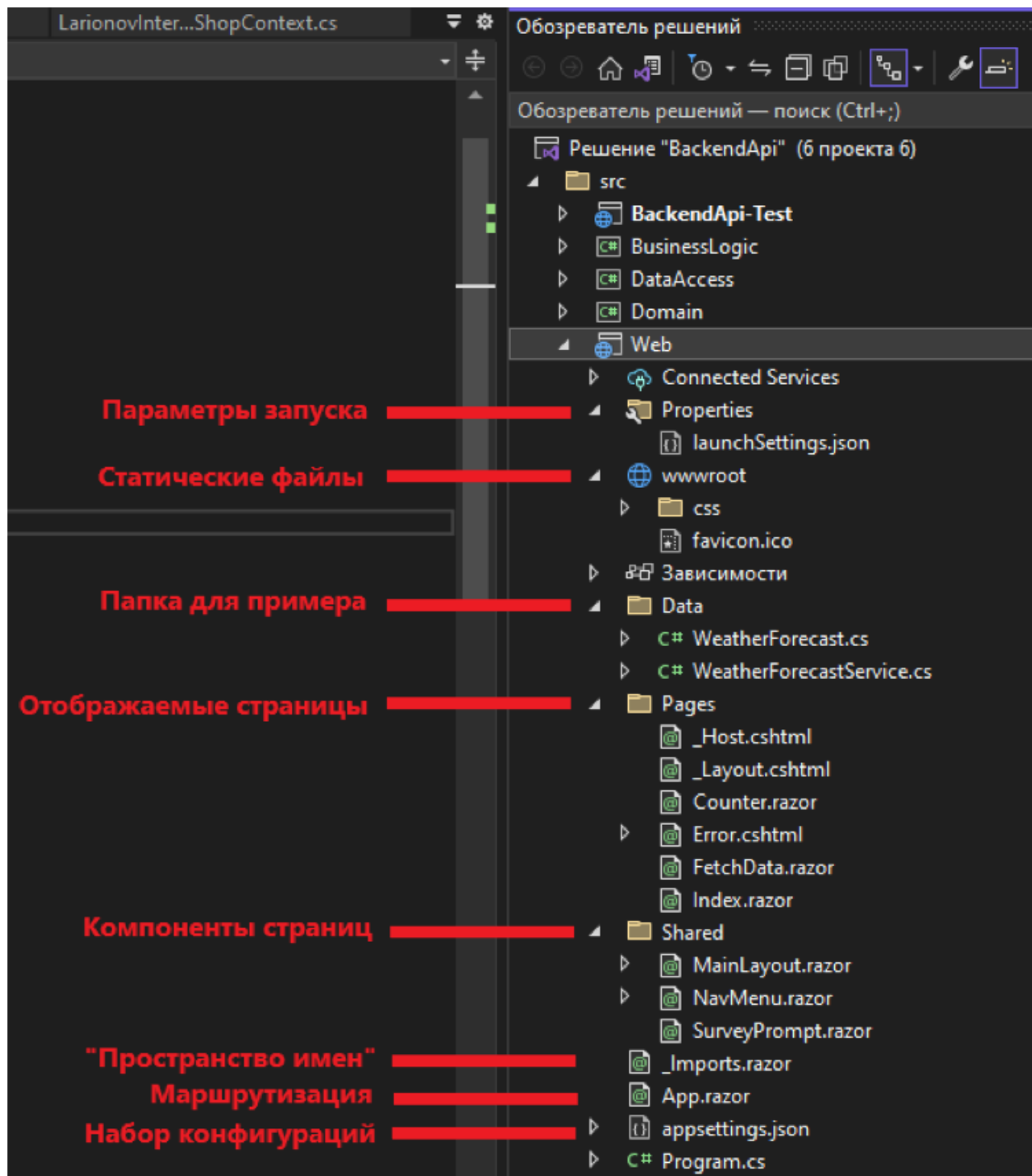
Можно отметить, что структура проекта Blazor похожа на ту, что и в Web-API. По сути мы имеем дело с проектом приложения ASP.NET Core, в рамках которого разворачивается функциональность фреймворка Blazor.

Разберем элементы проекта:

- Папка `wwwroot` для хранения статических файлов, по умолчанию хранит используемые файлы CSS, в частности, файлы фреймворка `bootstrap`.
- Папка `Data` хранит классы C#, которые описывают используемые данные (класс `WeatherForecast`) и сервисы (класс `WeatherForecastService`).
- Папка `Pages` содержит страницы Razor Pages, определяющих визуальную часть приложения и его логику, а также компоненты Razor (располагаются в

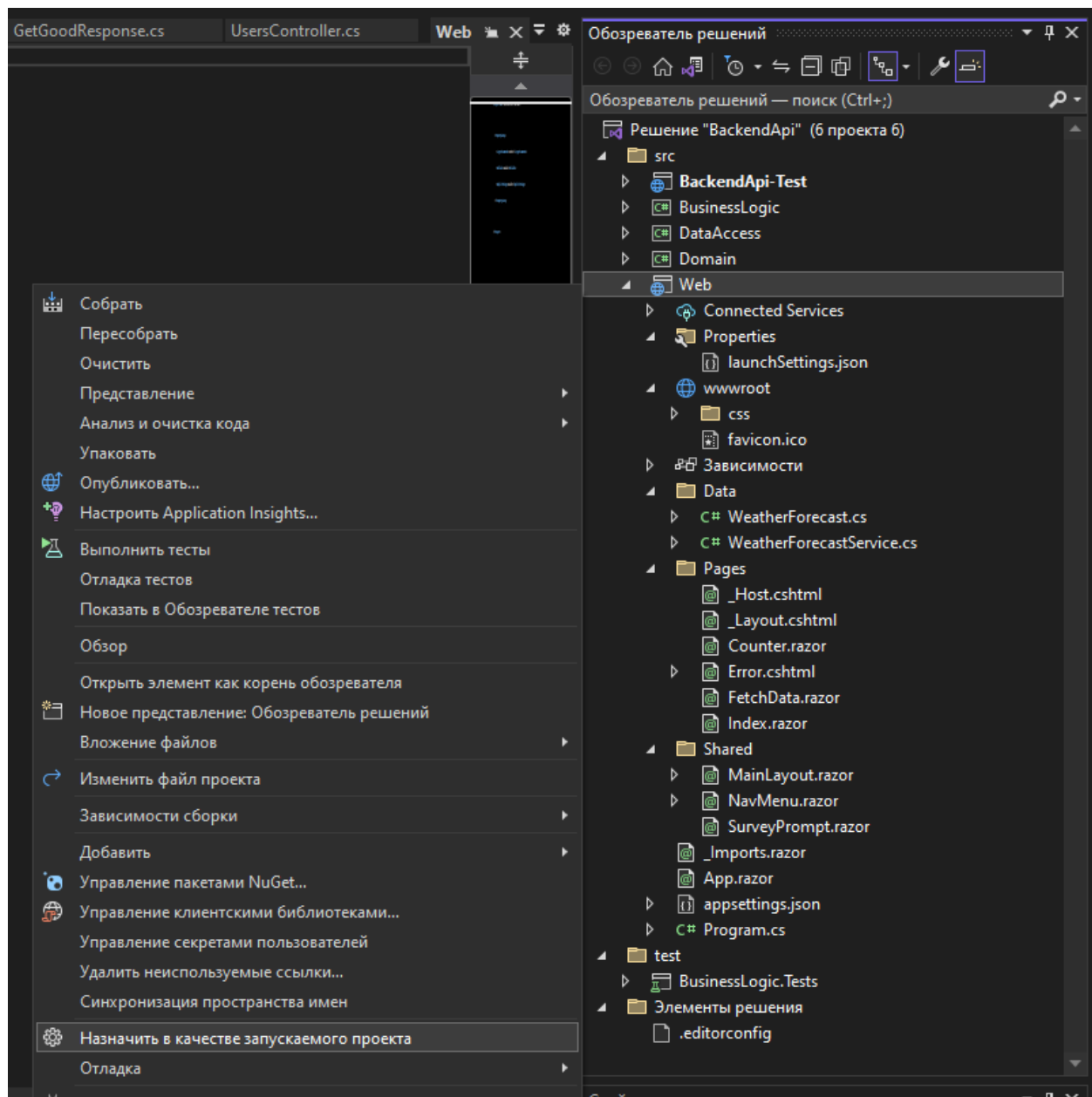
файлах с расширением *.razor), которые представляют основное содержание страницы.

- `_Host.cshtml` - главная страница (Razor Page) приложения, в рамках которой будут разворачиваться приложение.
- `Counter.razor` хранит код компонента `Counter`, суть которого в определение счетчика, значение которого увеличивается при нажатии на кнопку.
- `Error.razor` хранит код компонента `Error`, который применяется для вывода сообщения об ошибке.
- `FetchData.razor` хранит код компонента `FetchData`, который с помощью сервиса `WeatherForecastService` получает некоторые данные и выводит их на веб-страницу
- `Index.razor` хранит код компонента `Index`.
- Папка `Shared` хранит дополнительные компоненты Razor
 - `MainLayout.razor` хранит код компонента `MainLayout`, который определяет структуру или компоновку страницы.
 - `NavMenu.razor` хранит код компонента `NavMenu`, который определяет элементы навигации
- `_Imports.razor` содержит подключения пространств имен с помощью директивы `using`, которые будут подключаться в компоненты Razor (файлы с расширением `.razor`).
- `App.razor` содержит определение корневого компонента приложения, который позволяет установить маршрутизацию между вложенными компонентами с помощью другого встроенного компонента `Router`.
- Файл `appsettings.json` хранит конфигурацию приложения.
- Файл `Program.cs` содержит класс `Program`, который представляет точку входа в приложение. В данном случае это стандартный для приложения ASP.NET Core класс `Program`, который запускает и конфигурирует хост, в рамках которого разворачивается приложение с Blazor.

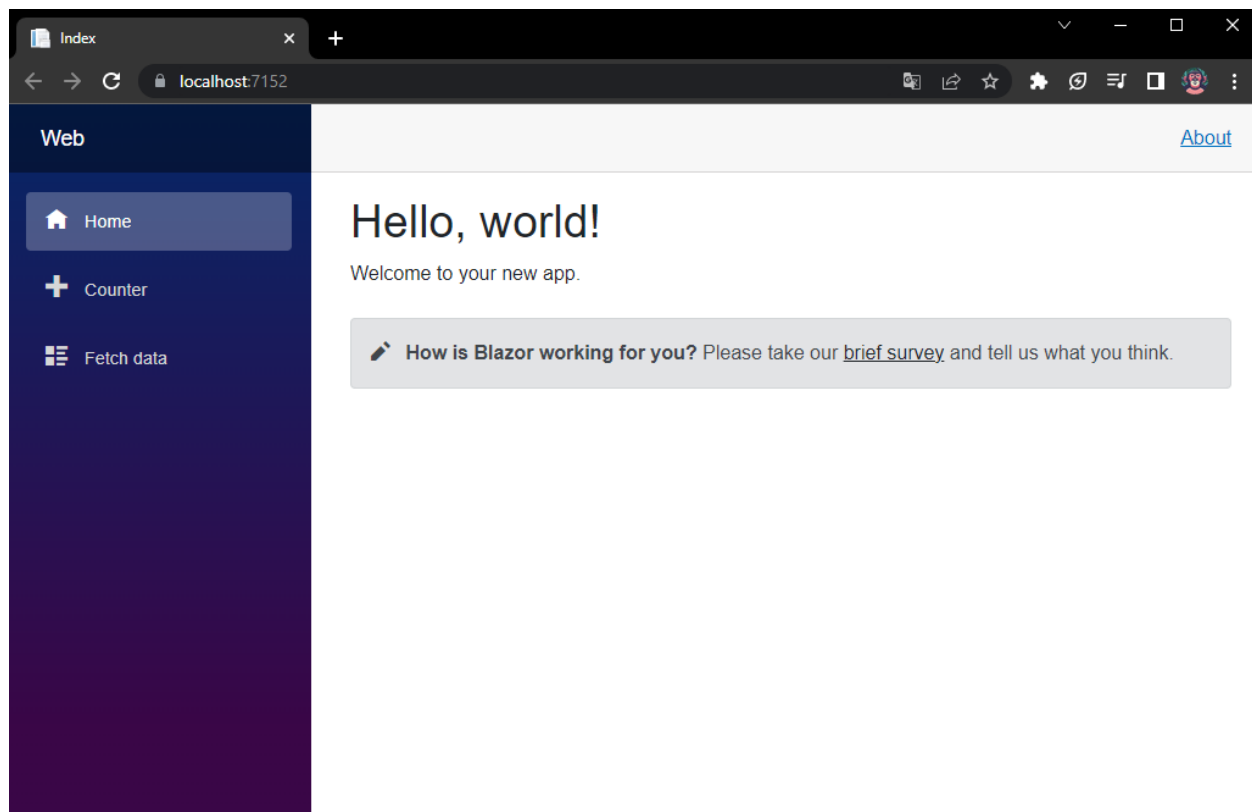


Запуск проекта. Настройка запуска нескольких проектов.

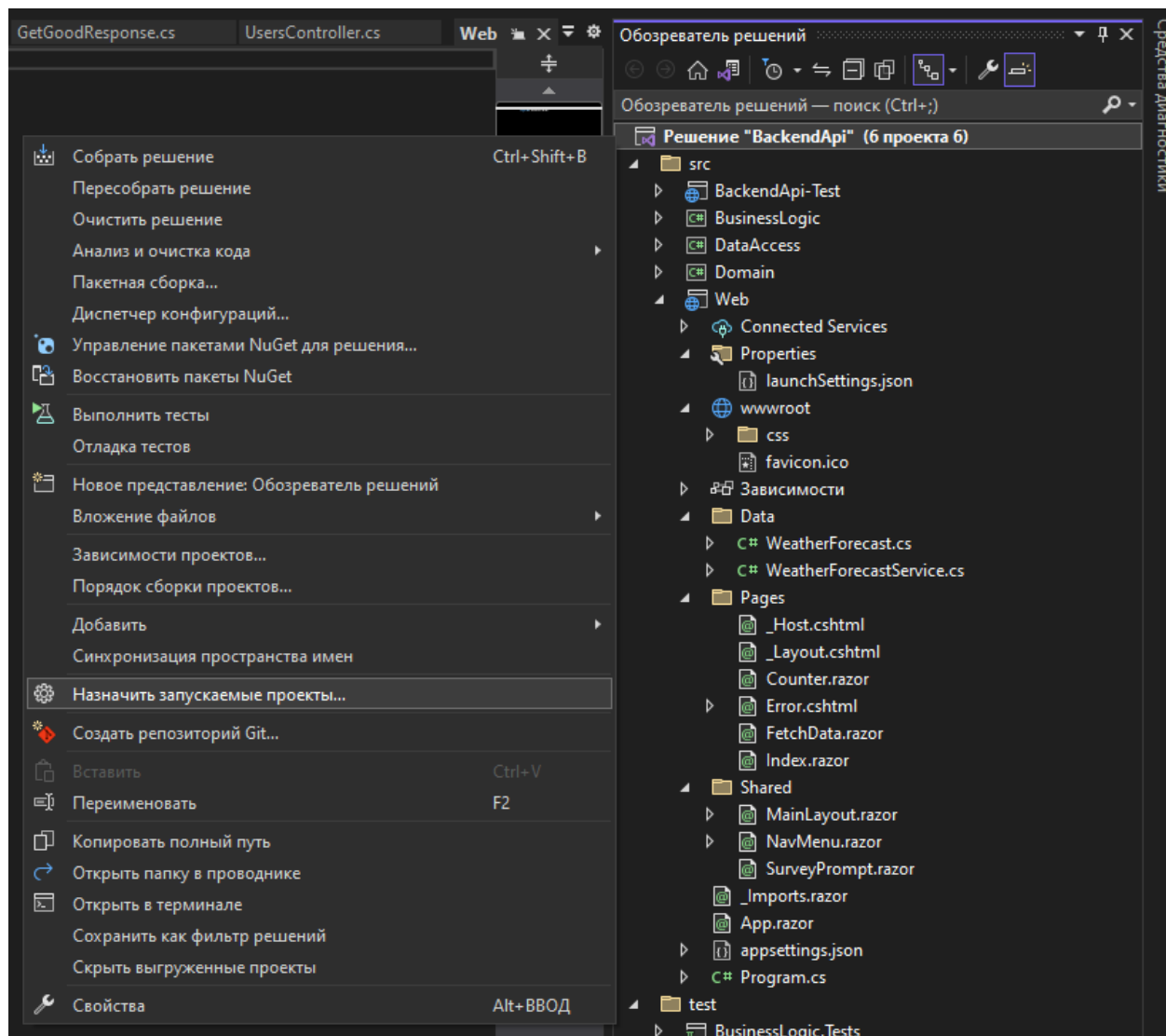
Для того, чтобы запустить этот проект - необходимо указать его в качестве запускаемого.



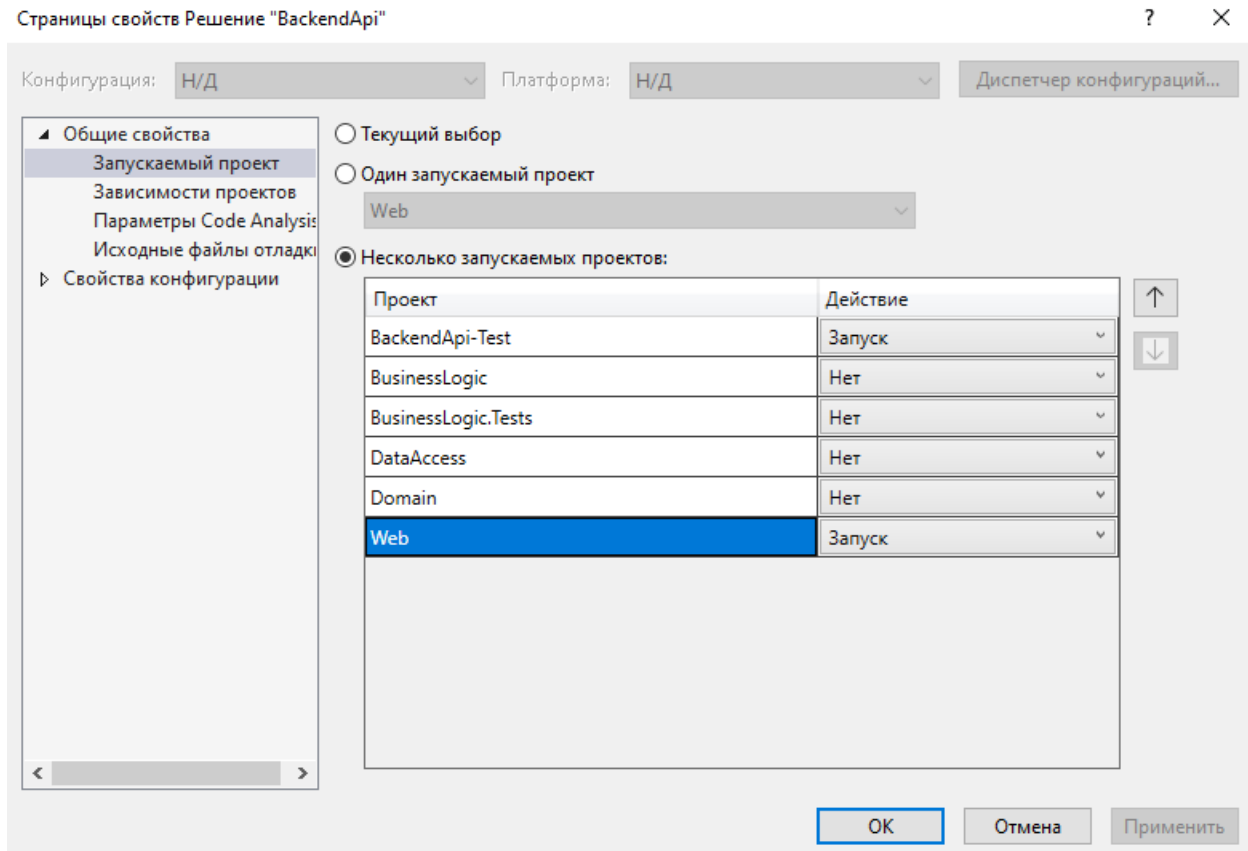
Проверьте что созданный проект запускается корректно



Однако, теперь будет запускаться только одно веб-приложение Blazor, а API не будет запущен. Чтобы запускать параллельно и API и Blazor - нажмите ПКМ по решению и выберите “Назначить запускаемые проекты...”



Укажите проекты Blazor и API в качестве запускаемых



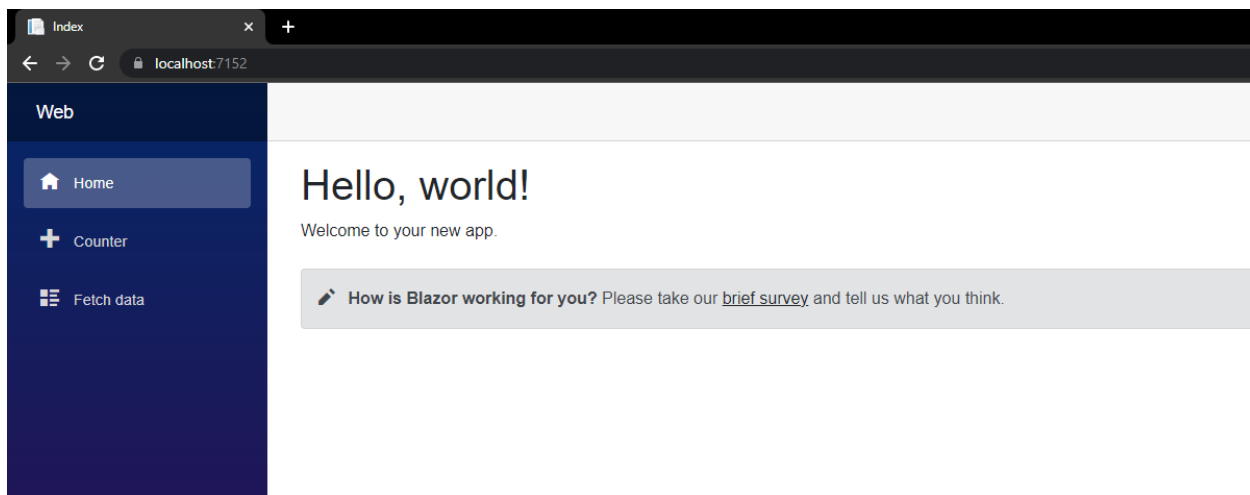
Теперь проекты запускаются одновременно.

Это будет полезно, если бы проект Blazor являлся подсистемой WebAssembly и требовалось обращаться к API для получения данных.

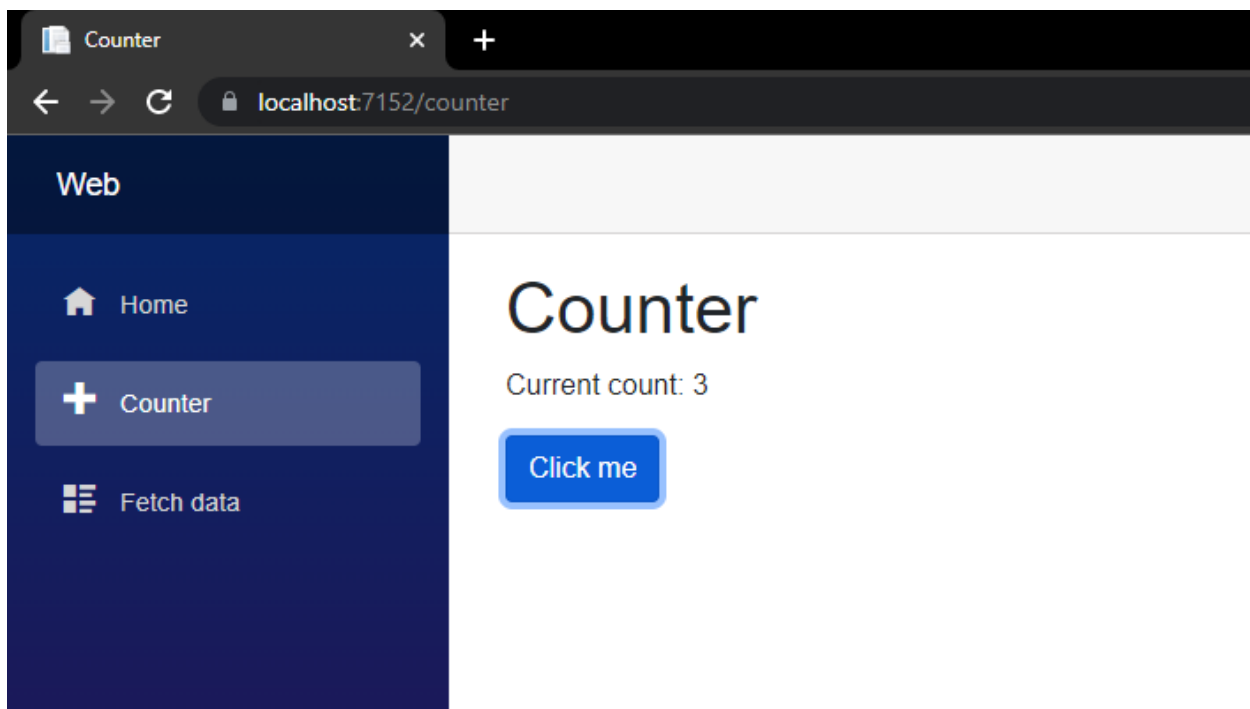
Анализ сформированных компонентов. Знакомство с .razor.

Страница счётчика нажатий по кнопке

Запустите проект Blazor ещё раз и изучите страницы, которые были созданы в качестве шаблона.



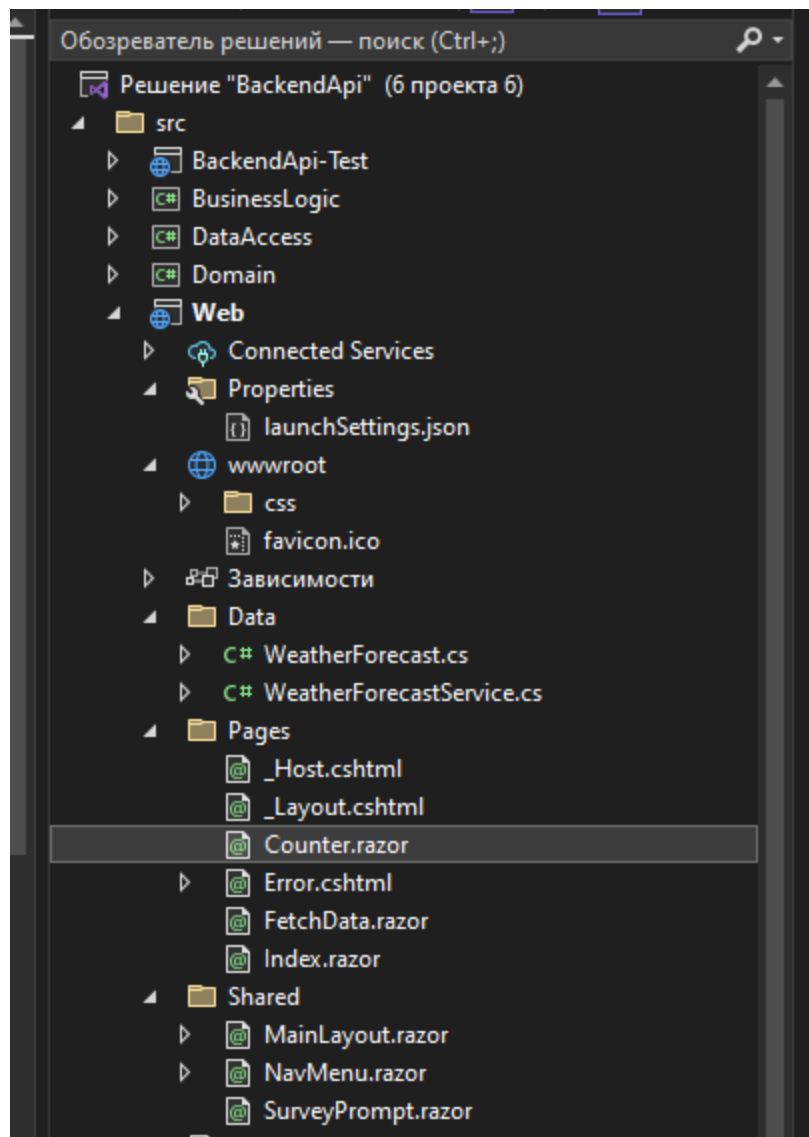
Перейдите во вкладку Counter и нажмите несколько раз на кнопку Click me



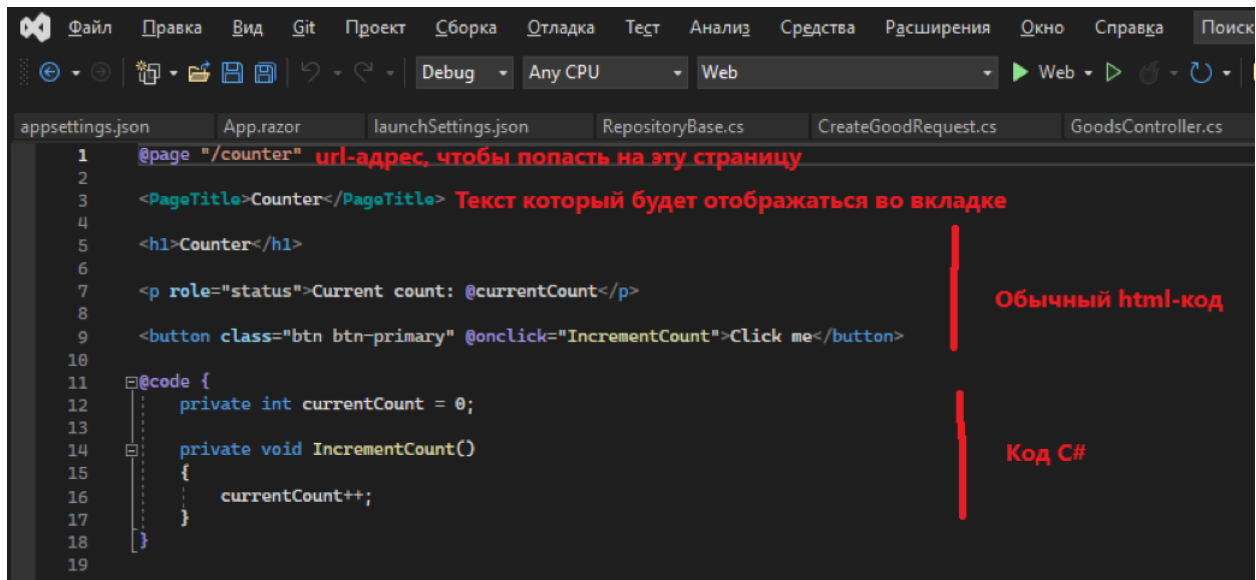
Давайте попробуем изменить страницу Counter и добавим ещё несколько элементов.

Так как Counter - это страница на которую мы перешли при помощи навигационной панели. Значит код этой страницы должен располагаться в папке Pages.

Найдите в проекте Blazor папку под названием Pages. Если страница называется как Counter - то будет логично открыть файл под данным названием.



Выделим основные особенности файлов-razor



Директива `@page "/counter"` указывает, что этот компонент будет сопоставляться с запросами по пути `"/counter"`, например, <https://localhost:44304/counter>.

На данной странице описана некоторая логика для счёта. При помощи директивы `@code` были определены переменная `currentCount` и метод `IncrementCount`, который увеличивает значение переменной.

```
private int currentCount = 0;

private void IncrementCount()
{
    currentCount++;
}
```

В коде html вы можете заметить установку привязки к переменным и методам компонента:

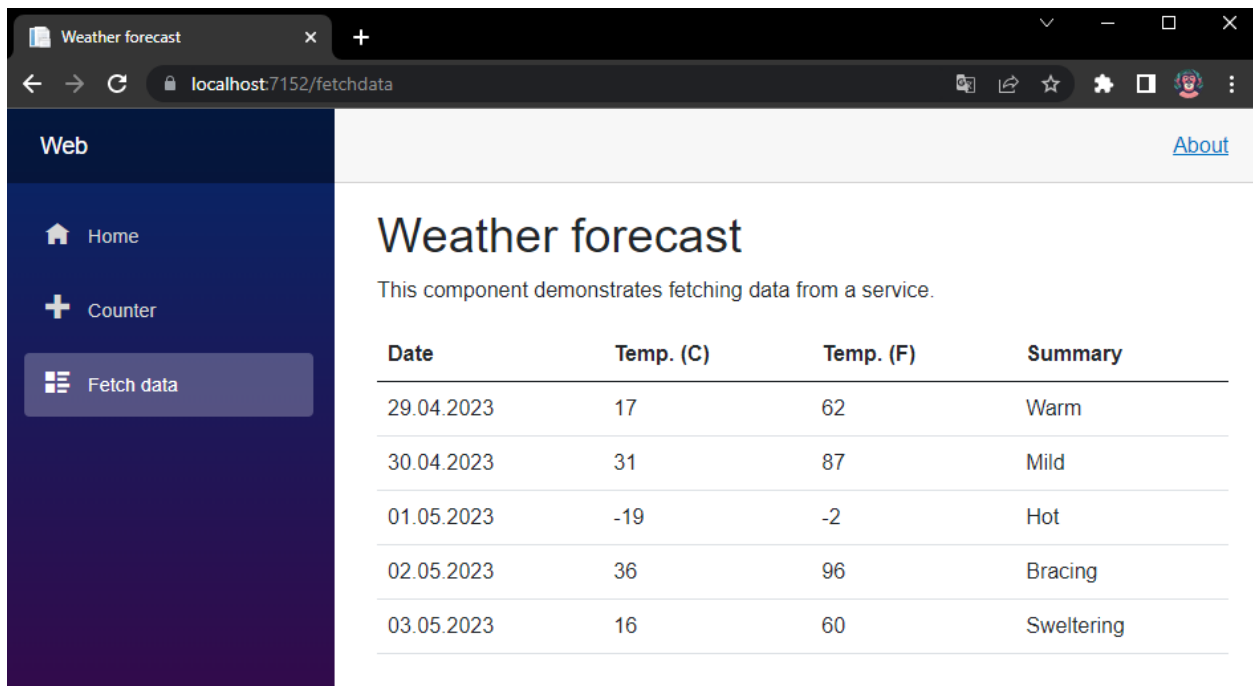
```
<p>Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
```

Метод привязывается к событию кнопки `onclick`, благодаря чему при нажатии на кнопку будет срабатывать метод `IncrementCount`, и пользователь увидит новое значение `currentCount`.

Страница с отображением прогноза погоды

Перейдите во вкладку Fetch Data и изучите содержимое страницы. Запомните количество строк в таблице.



The screenshot shows a web browser window with the address bar displaying `localhost:7152/fetchdata`. The page has a dark sidebar on the left with the following navigation items:

- Home
- Counter
- Fetch data (selected)

The main content area displays the title "Weather forecast" and a subtitle "This component demonstrates fetching data from a service." Below this is a table with the following data:

Date	Temp. (C)	Temp. (F)	Summary
29.04.2023	17	62	Warm
30.04.2023	31	87	Mild
01.05.2023	-19	-2	Hot
02.05.2023	36	96	Bracing
03.05.2023	16	60	Sweltering

Так как это страница Fetch data - то откроем соответствующий файл. В данном файле также есть много особенностей .razor. Одна из таковых особенностей - это отображение элементов по условиям или циклам.

```

FetchData.razor  X appsettings.json App.razor launchSettings.json RepositoryBase.cs CreateGoodReque
1  @page "/fetchdata"
2
3  <PageTitle>Weather forecast</PageTitle>
4
5  @using Web.Data
6  @inject WeatherForecastService ForecastService
7
8  <h1>Weather forecast</h1>
9
10 <p>This component demonstrates fetching data from a service.</p>
11
12 @if (forecasts == null)
13 {
14     <p><em>Loading...</em></p>
15 }
16 else
17 {
18     <table class="table">
19     <thead>
20     <tr>
21         <th>Date</th>
22         <th>Temp. (C)</th>
23         <th>Temp. (F)</th>
24         <th>Summary</th>
25     </tr>
26     </thead>
27     <tbody>
28         @foreach (var forecast in forecasts)
29         {
30             <tr>
31                 <td>@forecast.Date.ToShortDateString()</td>
32                 <td>@forecast.TemperatureC</td>
33                 <td>@forecast.TemperatureF</td>
34                 <td>@forecast.Summary</td>
35             </tr>
36         }
37     </tbody>
38     </table>
39 }
40
41 @code {
42     private WeatherForecast[]? forecasts;
43
44     protected override async Task OnInitializedAsync()
45     {
46         forecasts = await ForecastService.GetForecastAsync(DateTime.Now);
47     }
48 }

```

Пространство имён тоже используются

Подключение сервисов

Можно генерировать страницы в зависимости от условий

В данном случае выводится список прогнозов погоды если массив не null

Циклы позволяют генерировать ровно столько элементов - сколько есть в массиве

nullable переменная

Событие при открытии страницы

Давайте рассмотрим работу цикла, если записей будет немного больше.

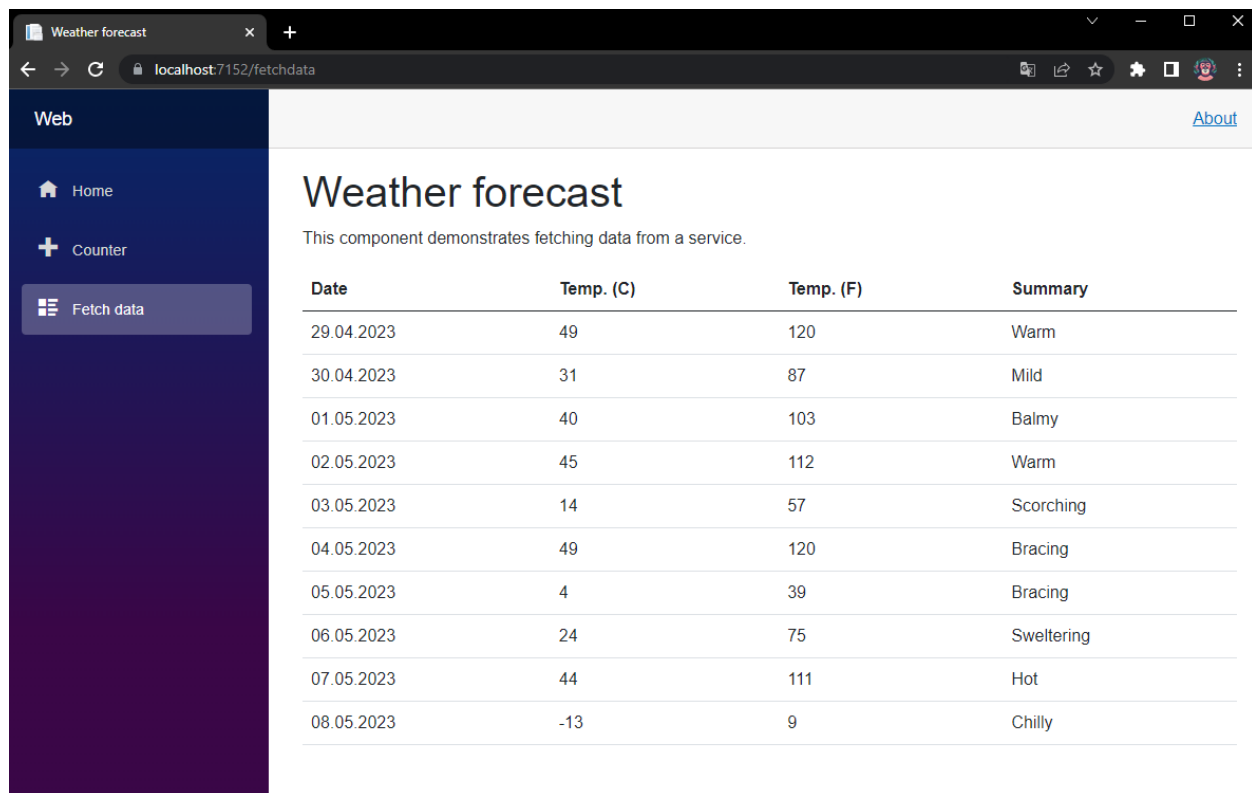
Перейдите в сервис прогнозы погоды.

```
1 namespace Web.Data
2 {
3     Ссылка 3
4     public class WeatherForecastService
5     {
6         private static readonly string[] Summaries = new[]
7         {
8             "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
9         };
10
11         Ссылка 0
12         public Task<WeatherForecast[]> GetForecastAsync(DateTime startDate)
13         {
14             return Task.FromResult(Enumerable.Range(1, 5).Select(index => new WeatherForecast
15             {
16                 Date = startDate.AddDays(index),
17                 TemperatureC = Random.Shared.Next(-20, 55),
18                 Summary = Summaries[Random.Shared.Next(Summaries.Length)]
19             }).ToArray());
20         }
21     }
22 }
```

Изменим количество записей с 5 до 10

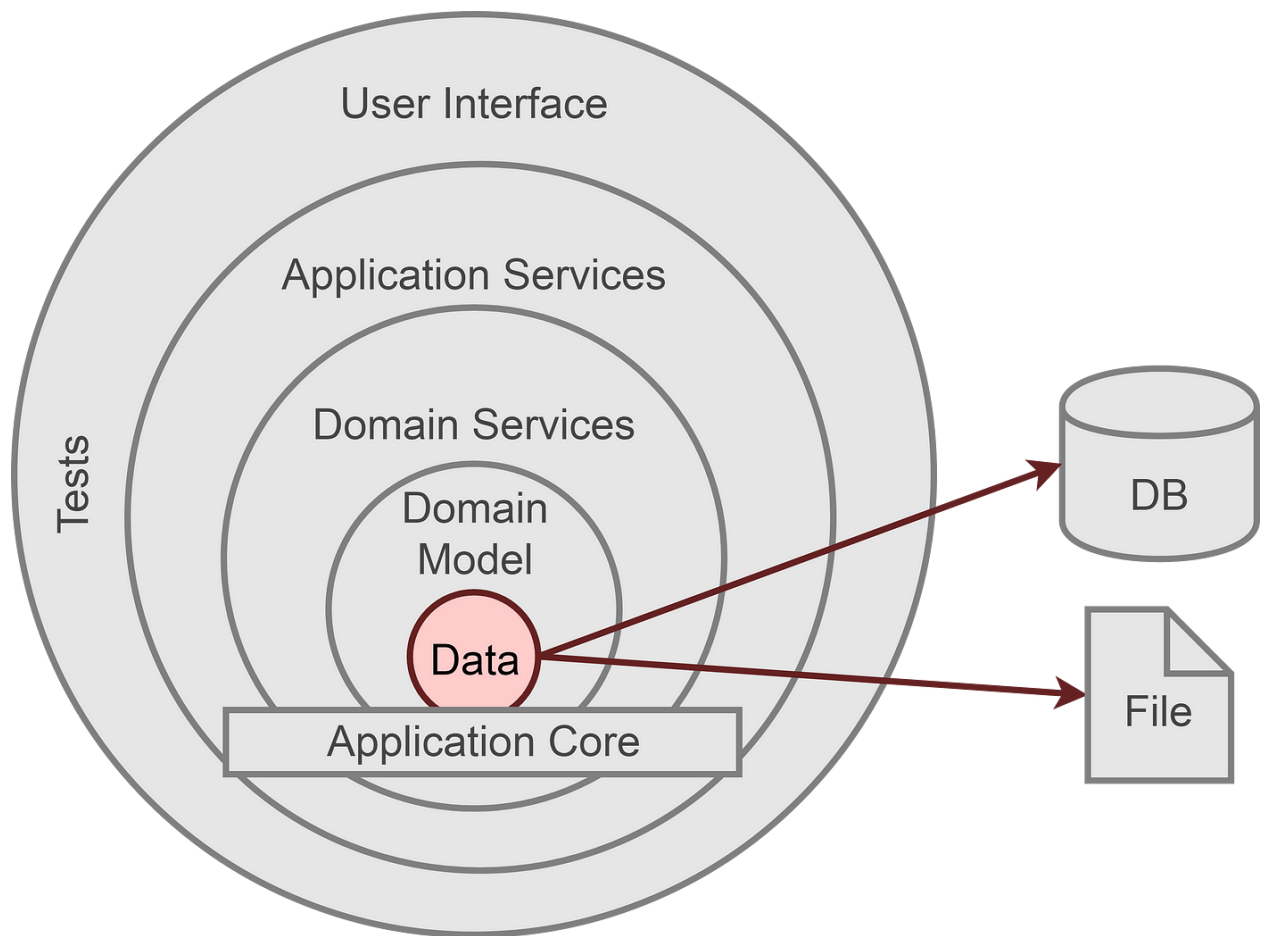
```
Ссылка 0
public Task<WeatherForecast[]> GetForecastAsync(DateTime startDate)
{
    return Task.FromResult(Enumerable.Range(1, 10).Select(index => new WeatherForecast
    {
        Date = startDate.AddDays(index),
        TemperatureC = Random.Shared.Next(-20, 55),
        Summary = Summaries[Random.Shared.Next(Summaries.Length)]
    }).ToArray());
}
```

Благодаря циклу в компоненте FetchData.razor записей в таблице стало ровно 10



Внедрение нового Presentation-слоя в текущее решение

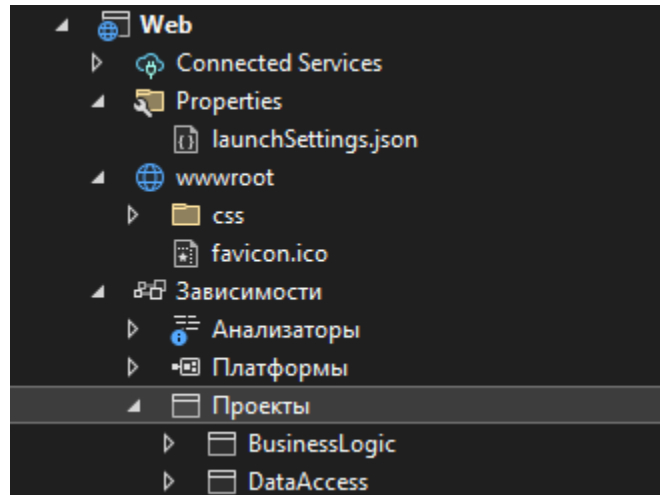
Так как Blazor-проект является интерфейсом для взаимодействия с пользователем - то такие проекты называют слоем Presentation или же UI.



Такие проекты вы уже создавали, например WPF.

Так как логику работы мы уже описывали ранее (Слои Domain, DataAccess и BusinessLogic), просто подключимся к UI-слою.

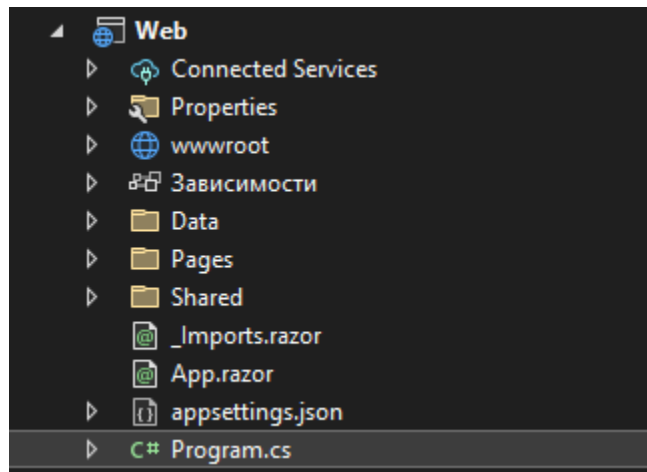
Для начала подключим слои, которыми обладает проект API



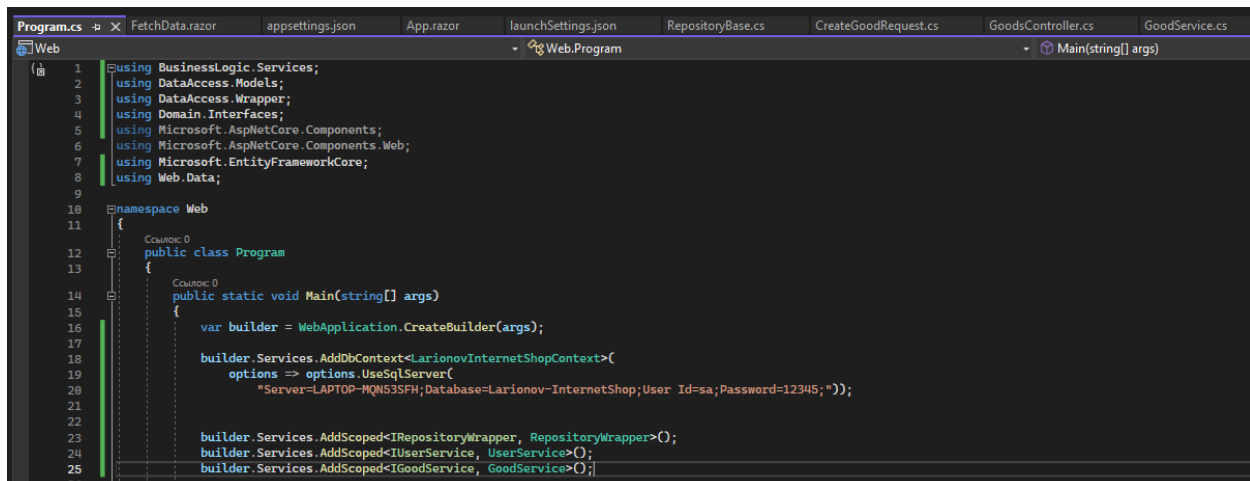
Далее перейдите в файл Program.cs в проекте API и скопируйте код, который сопоставляет реализацию слоев Data Access и Business Logic с интерфейсами из слоя Domain в контейнере зависимостей.

```
Program.cs  FetchData.razor  appsettings.json  App.razor  launchSettings.json  RepositoryBase.cs  CreateGoodRequest.cs
BackendApi-Test  BackendApi.Program
1  using BusinessLogic.Services;
2  using DataAccess.Models;
3  using DataAccess.Wrapper;
4  using Domain.Interfaces;
5  using Microsoft.EntityFrameworkCore;
6  using Microsoft.OpenApi.Models;
7  using System.Reflection;
8
9  namespace BackendApi
10 {
11     public class Program
12     {
13         public static void Main(string[] args)
14         {
15             var builder = WebApplication.CreateBuilder(args);
16
17             builder.Services.AddDbContext<LarionovInternetShopContext>(
18                 options => options.UseSqlServer(
19                     "Server=LAPTOP-MQN53SFH;Database=Larionov-InternetShop;User Id=sa;Password=12345;");
20
21             builder.Services.AddScoped<IRepositoryWrapper, RepositoryWrapper>();
22             builder.Services.AddScoped<IUserService, UserService>();
23             builder.Services.AddScoped<IGoodService, GoodService>();
24
25             builder.Services.AddSwaggerGen(options =>
```

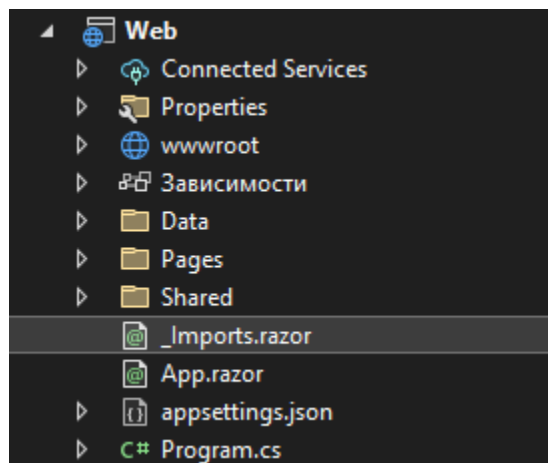
Теперь перейдите в файл Program.cs вашего Blazor-проекта



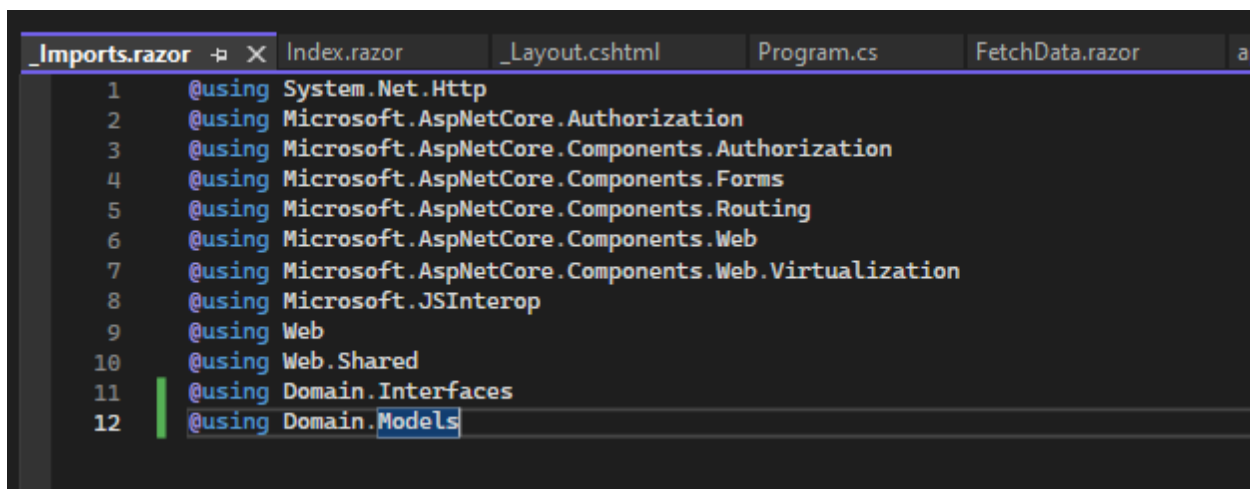
И вставьте ваш код в точно такое же место



Перейдите в файл _Imports.razor для указания используемых пространств имён



И добавьте используемые интерфейсы

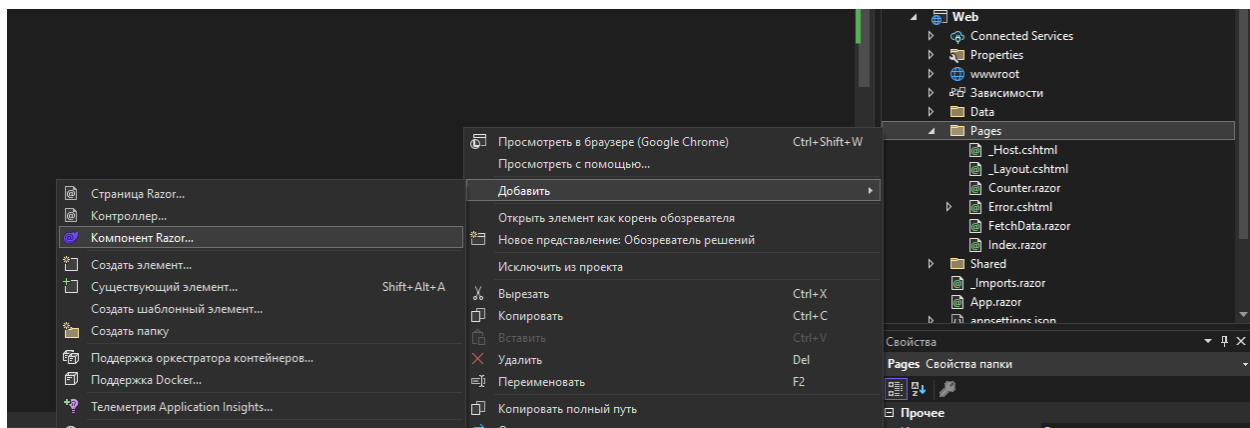


```
1 @using System.Net.Http
2 @using Microsoft.AspNetCore.Authorization
3 @using Microsoft.AspNetCore.Components.Authorization
4 @using Microsoft.AspNetCore.Components.Forms
5 @using Microsoft.AspNetCore.Components.Routing
6 @using Microsoft.AspNetCore.Components.Web
7 @using Microsoft.AspNetCore.Components.Web.Virtualization
8 @using Microsoft.JSInterop
9 @using Web
10 @using Web.Shared
11 @using Domain.Interfaces
12 @using Domain.Models
```

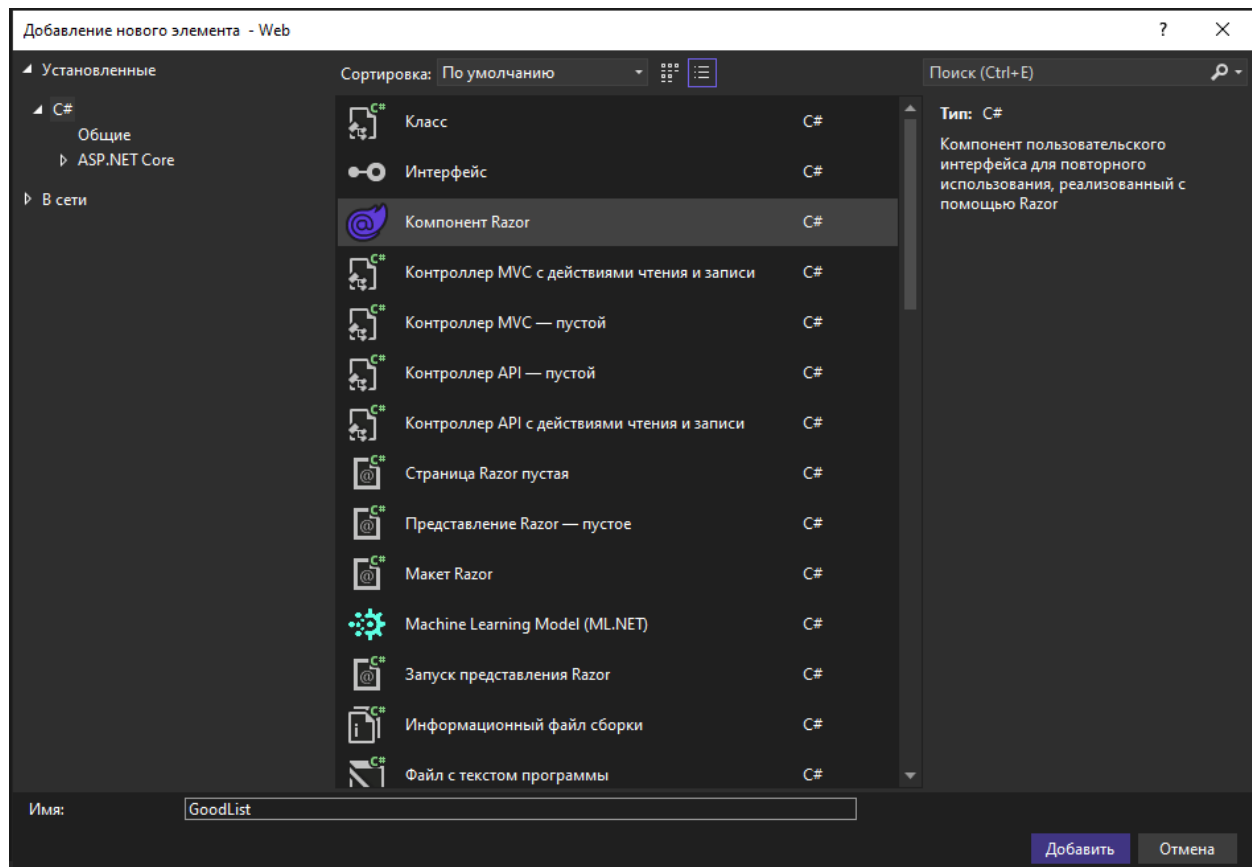
Создание новой страницы для отображения списка товаров

Теперь попробуем создать свою страницу, которая будет отображать например список товаров.

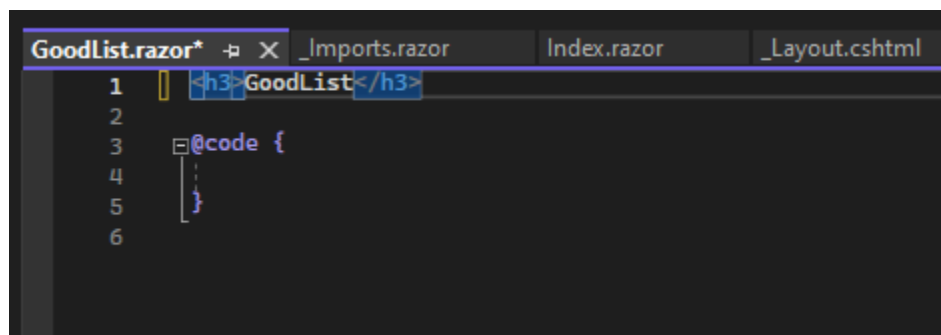
Создайте новый компонент Razor



Укажите подходящее название вашей страницы, так как страница будет выводить списком все товары - назову как GoodsList



Созданная страница имеет следующий вид:



Укажем адрес по которому будет отображаться данная страница - добавим директиву @page со значением "/goods-list"

```
GoodsList.razor  Imports.razor  Index.razor  _Layout.cshtml  Program.cs
1  @page "/goods-list"
2
3  <h3>GoodList</h3>
4
5  @code {
6      ...
7  }
8
```

Чтобы работать с товарами - подключим сервис IGoodService при помощи директивы @inject

```
GoodsList.razor*  Imports.razor  Index.razor  _Layout.cshtml
1  @page "/goods-list"
2  @inject IGoodService GoodService
3
4
5  <h3>GoodList</h3>
6
7  @code {
8      ...
9  }
10
```

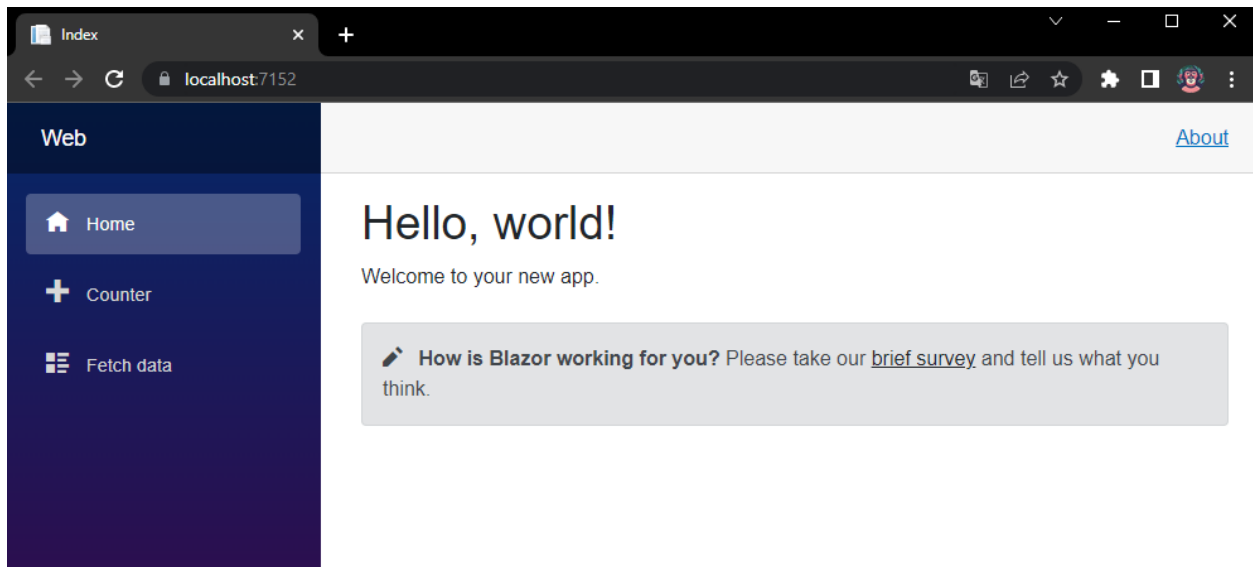
Теперь, внутри директивы @code напомним логику для получения всех записей о товарах

```
GoodsList.razor* X _Imports.razor Index.razor _Layout.cshtml
1 @page "/goods-list"
2 @inject IGoodService GoodService
3
4
5 <h3>GoodList</h3>
6
7
8
9 @code {
10     private List<Good> goods = new List<Good>();
11
12     protected override async Task OnInitializedAsync()
13     {
14         goods = await GoodService.GetAll();
15     }
16 }
17
```

И опишем код для отображения списка товаров на странице

```
GoodsList.razor X _Imports.razor Index.razor _Layout.cshtml Program.cs
1 @page "/goods-list"
2 @inject IGoodService GoodService
3
4
5 <h2>Список товаров:</h2>
6
7 @foreach(var good in goods)
8 {
9     <h3>Товар: @good.Name</h3>
10    <p>Цена: @good.Price</p>
11    <hr />
12 }
13
14
15 @code {
16     private List<Good> goods = new List<Good>();
17
18     protected override async Task OnInitializedAsync()
19     {
20         goods = await GoodService.GetAll();
21     }
22 }
23
```

Проверим работу страницы, для этого запустим проект

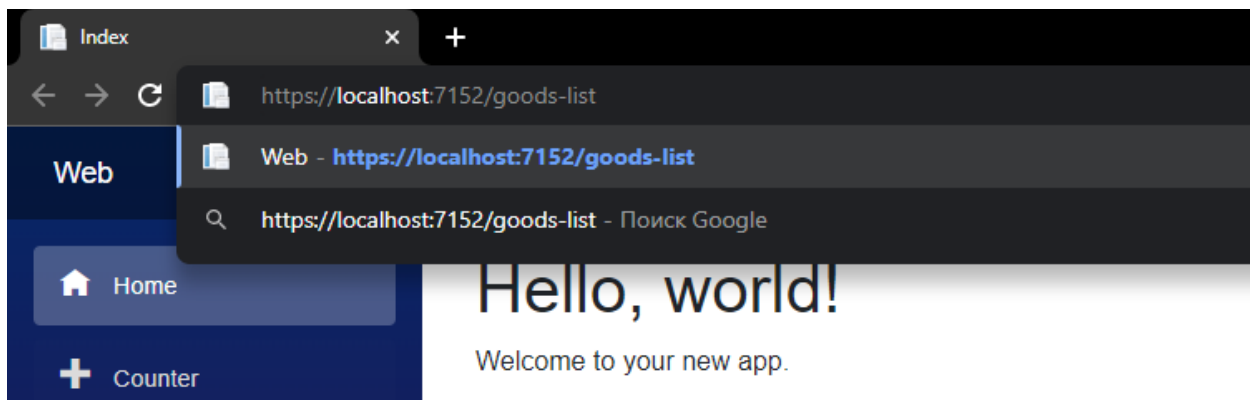


Так как на навигационной панели нет ссылки для перехода на страницу для просмотра списка товаров - мы можем самостоятельно перейти по этому пути

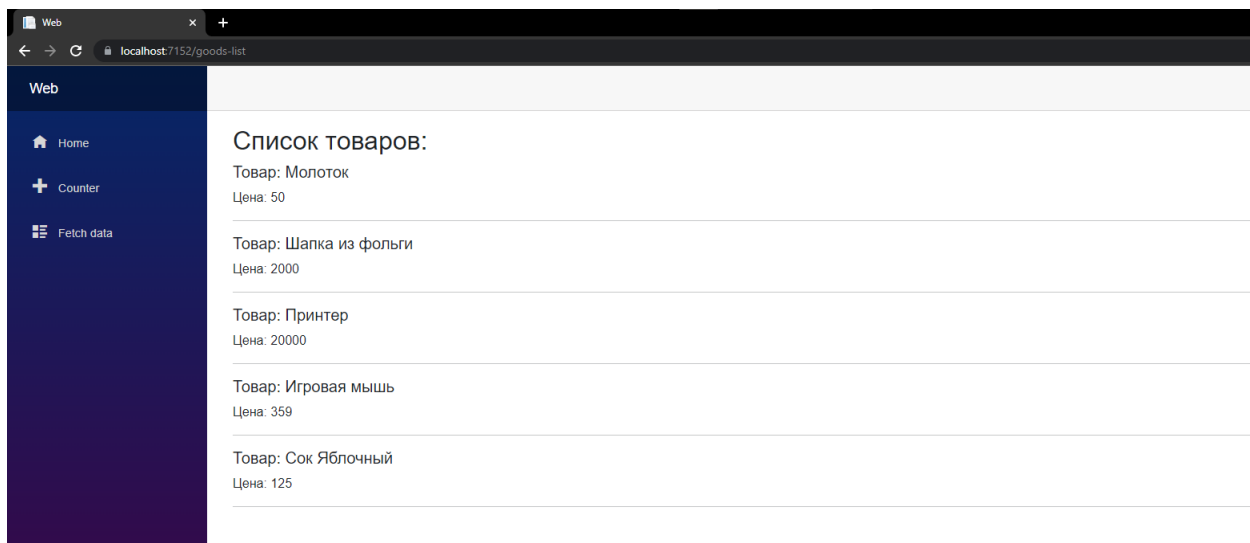
A screenshot of the Visual Studio Code editor. The file explorer on the left shows a project structure with files like '_Imports.razor', 'Index.razor', '_Layout.cshtml', and 'Program.cs'. The main editor window is open to 'Goodslist.razor'. The code is as follows:

```
1 @page "/goods-list"
2 @inject IGoodService GoodService
3
4
5 <h2>Список товаров:</h2>
6
7 @foreach(var good in goods)
8 {
9     <h5>Товар: @good.Name</h5>
10    <p>Цена: @good.Price</p>
11    <hr />
```

Добавим в URL-адрес данный текст

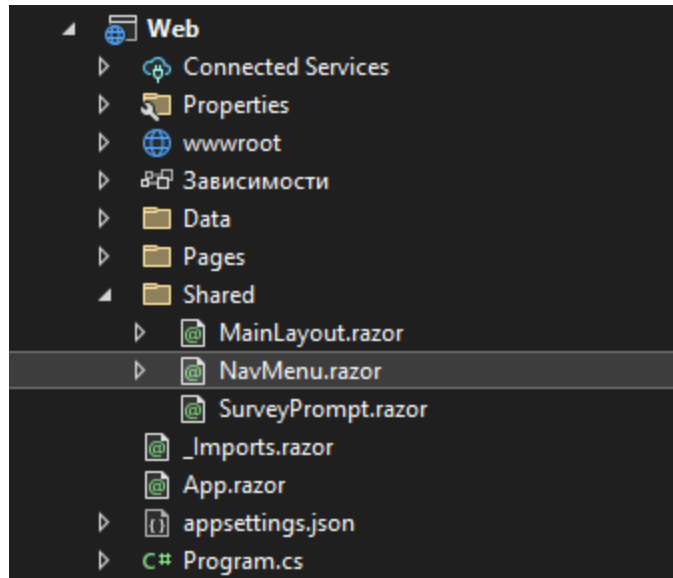


И в результате, мы попали на страницу со списком товаров



Добавление ссылки на созданную страницу. Использование стилей.

Так как панель навигации - это компонент, то перейдем в папку Shared и откроем его.



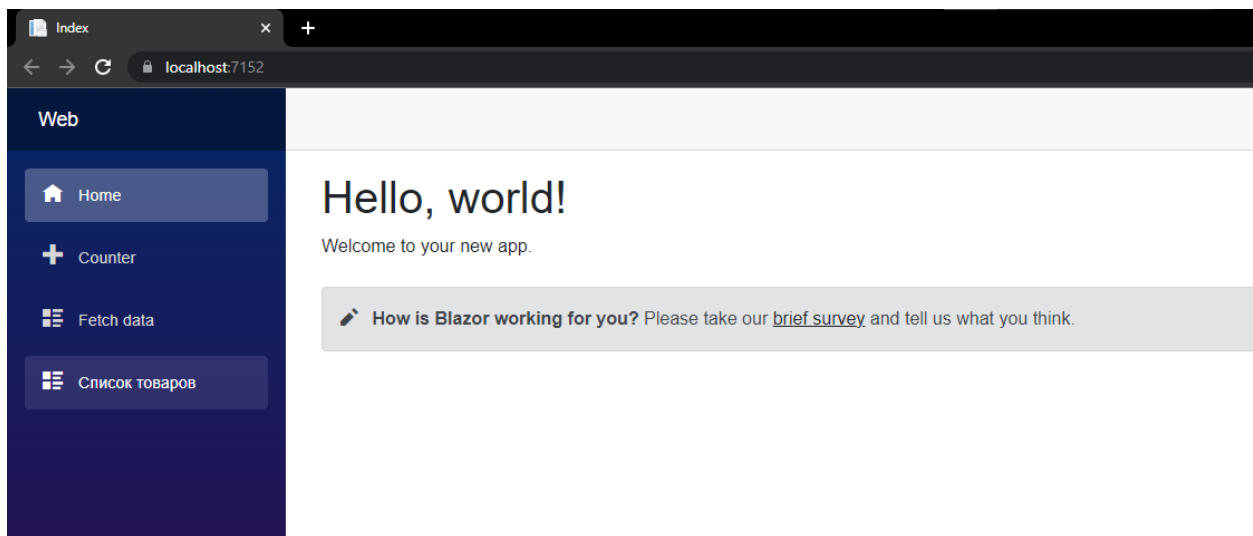
В данном файле код описан следующим образом

```
NavMenu.razor
1 <div class="top-row ps-3 navbar navbar-dark">
2   <div class="container-fluid">
3     <a class="navbar-brand" href="#">Web</a>
4     <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
5       <span class="navbar-toggler-icon"></span>
6     </button>
7   </div>
8 </div>
9
10 <div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
11   <nav class="flex-column">
12     <div class="nav-item px-3">
13       <NavLink class="nav-link" href="#" Match="NavLinkMatch.All">
14         <span class="oi oi-home" aria-hidden="true"></span> Home
15       </NavLink>
16     </div>
17     <div class="nav-item px-3">
18       <NavLink class="nav-link" href="counter">
19         <span class="oi oi-plus" aria-hidden="true"></span> Counter
20       </NavLink>
21     </div>
22     <div class="nav-item px-3">
23       <NavLink class="nav-link" href="fetchdata">
24         <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
25       </NavLink>
26     </div>
27   </nav>
28 </div>
29
30 @code {
31   private bool collapseNavMenu = true;
32   private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;
33
34   private void ToggleNavMenu()
35   {
36     collapseNavMenu = !collapseNavMenu;
37   }
38 }
39
40
```

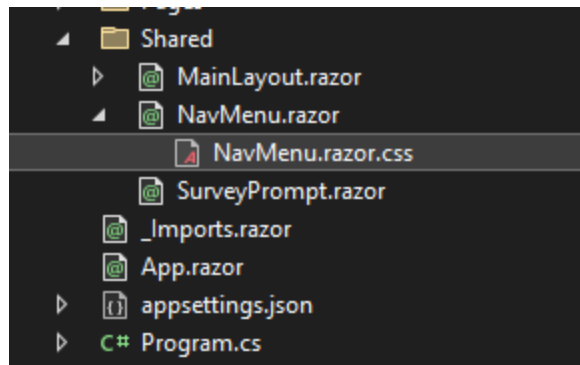
Добавим в панель навигации ссылку для перехода на созданную страницу. Не забудьте указать правильную ссылку

```
NavMenu.razor* X GoodsList.razor _Imports.razor Index.razor _Layout.cshtml Program.cs FetchData.razor appsettings.json App.razor
1 <div class="top-row ps-3 navbar navbar-dark">
2   <div class="container-fluid">
3     <a class="navbar-brand" href="#">Web</a>
4     <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
5       <span class="navbar-toggler-icon"></span>
6     </button>
7   </div>
8 </div>
9
10 <div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
11   <nav class="flex-column">
12     <div class="nav-item px-3">
13       <NavLink class="nav-link" href="#" Match="NavLinkMatch.All">
14         <span class="oi oi-home" aria-hidden="true"></span> Home
15       </NavLink>
16     </div>
17     <div class="nav-item px-3">
18       <NavLink class="nav-link" href="counter">
19         <span class="oi oi-plus" aria-hidden="true"></span> Counter
20       </NavLink>
21     </div>
22     <div class="nav-item px-3">
23       <NavLink class="nav-link" href="fetchdata">
24         <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
25       </NavLink>
26     </div>
27     <div class="nav-item px-3">
28       <NavLink class="nav-link" href="goods-list">
29         <span class="oi oi-list-rich" aria-hidden="true"></span> Список товаров
30       </NavLink>
31     </div>
32   </nav>
33 </div>
34
35 @code {
36   private bool collapseNavMenu = true;
37
38   private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;
39
40   private void ToggleNavMenu()
41   {
42     collapseNavMenu = !collapseNavMenu;
43   }
44 }
45
46
```

Если запустить проект - ссылка появится на навигационной панели

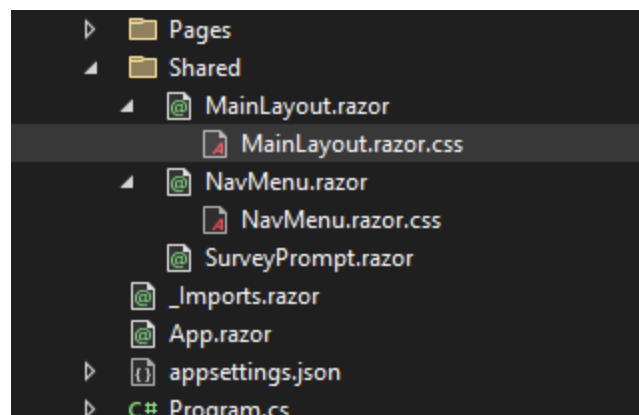


Для того, чтобы редактировать стилизацию элементов - вы можете развернуть содержимое .razor



Внутри данного файла вы можете заметить другой файл, который как раз хранит информацию по стилям

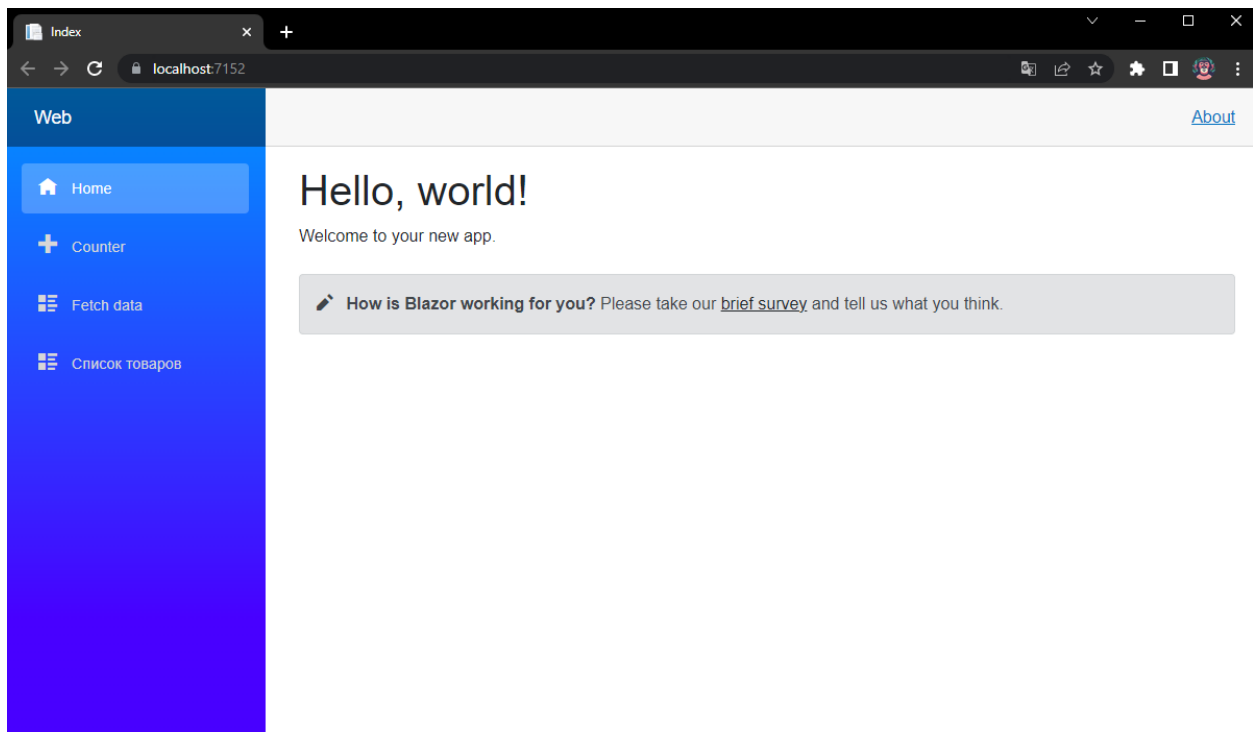
Давайте поменяем цвет для панели - поэтому перейдем к файлу MainLayout и развернем его.



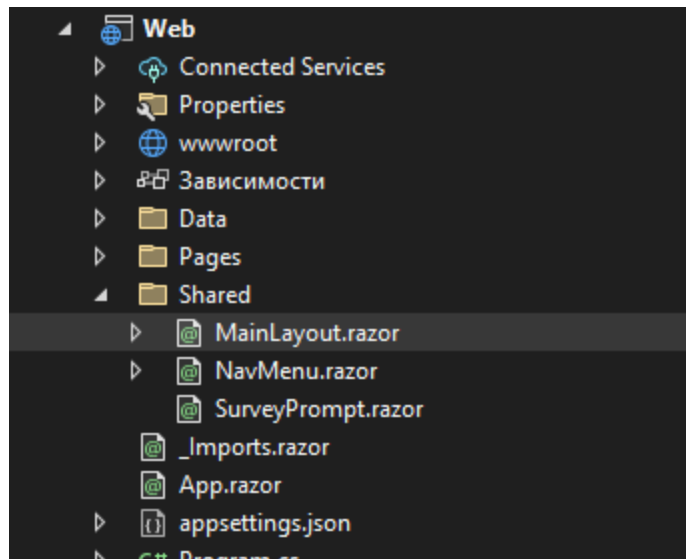
В данном файле вы сможете найти данный класс, которому нужно будет изменить цвета


```
MainLayout.razor.css*  x  site.css  NavMenu.razor.css*  MainLayout.razor  NavMenu.razor  GoodsList.razor
1  .page {
2      position: relative;
3      display: flex;
4      flex-direction: column;
5  }
6
7  .main {
8      flex: 1;
9  }
10
11  .sidebar {
12      background-image: linear-gradient(180deg, rgb(0, 148, 255) 0%, rgb(72, 0, 255) 70%);
13  }
14
15  .top-row {
16      background-color: #f7f7f7;
17      border-bottom: 1px solid #d6d6d6;
18      justify-content: flex-end;
19      height: 3.5rem;
20      display: flex;
21      align-items: center;
22  }
23
24  .top-row ::deep a, .top-row .btn-link {
```

С такими цветами сайт будет выглядеть так



Если панель навигации не нужна - ее можно убрать тут:



Если удалить данный фрагмент

```
MainLayout.razor.css  site.css  NavMenu.razor.css  MainLayout.razor  NavMenu.razor  GoodsList.razor  _Imports.razor
1  @inherits LayoutComponentBase
2
3  <PageTitle>Web</PageTitle>
4
5  <div class="page">
6      <div class="sidebar">
7          <NavMenu />
8      </div>
9
10     <main>
11         <div class="top-row px-4">
12             <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
13         </div>
14
15         <article class="content px-4">
16             @Body
17         </article>
18     </main>
19 </div>
20
```

То после запуска приложения страница будет выглядеть именно так

