MAVZU: ASP.NET muhitida ishlash

Maqsad: Talabalar ASP.NET muhitida ishlash haqida tasavvurga ega bo’lish va undan foydalanib Web sahifalar hosil qilishni o’rganish.

REJA:

1. ASP.NET ga kirish.
2. ASP.NET modellari.
3. ASP.NET Razor haqida.
4. ASP.NET Razor operatorlari.

ADABIYOTLAR:

1. Карли Уотсон, Кристиан Нейгел, Якоб Хаммер Педерсен, Джон Д Рид, Морган Скиннер, Эрик Уайт. Visual C# 2008: базовый курс.: Пер. с англ. - М.: ООО "ИД. Вильяме", 2009.- 1216с.
2. Неш Трей. C # 2010: ускоренный курс для профессионалов. :Пер. с англ. - М.: ООО «И.Д. Вильямс», 2010.
3. Чамберс Джеймс, Пэкетт Дэвид, Тимме Саймон. ASP.NET Core. Разработка приложений.-Спб.: Питер, 2018.-464 с.
4. Б.И.Пахомов. C# для начинающих. – СПб.: БХВ-Петербург, 2014. – 432 с.
5. Эндрю Троелсон. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 2013. – 1312 с.
6. *http://mycsharp.ru*
7. *https://metanit.com*

ASP - "**A**ctive **S**erver **P**ages (faol server sahifalari)" degan ma'noni anglatadi.

ASP - bu web-sahifalarni yaratish uchun mo'ljallangan dastur.

ASP ko'plab turli xil ishlab chiqish modellarini qo'llab-quvvatlaydi:

- ✓ Classic ASP
- ✓ ASP.NET Web Forms
- ✓ ASP.NET MVC
- ✓ ASP.NET Web Pages
- ✓ ASP.NET API
- ✓ ASP.NET Core

ASP texnologiyasi

ASP va ASP.NET server tomonidan bajariladigan texnologiyalardir.

Ikkala texnologiya ham Internet-server tomonidan kompyuter kodini bajarishga imkon beradi.

Brauzer ASP yoki ASP.NET faylini so'raganda, ASP mexanizmi faylni o'qiydi, fayldagi istalgan kodni bajaradi va natijani brauzerga qaytaradi.

Klassik ASP - faol server sahifalari

ASP (Classic ASP akasi) 1998 yilda Microsoft-ning birinchi server tomonidagi skript tili sifatida taqdim etilgan.

Klassik ASP sahifalari **.asp** fayl kengaytmasiga ega va odatda **VBScript** da yoziladi.

ASP.NET

ASP.NET 2002 yilda Classic ASP ning vorisi sifatida ishlab chiqarilgan.

ASP.NET sahifalari **.aspx** kengaytmasiga ega va odatda **C# (C sharp)** tilida yoziladi.

ASP.NET 4.6 - ASP.NET ning eng so'nggi rasmiy versiyasidir.

ASP.NET 5 ASP.NET ning muhim qayta dizayni bo'lishi kutilgan edi.

Biroq, ASP.NET 5 ning rivojlanishi ASP.NET Core foydasi uchun to'xtatildi.

ASP.NET Web Pages – ASP.NET web-sahifalari

ASP.NET web-sahifalari - bu SPA(Single Page Application – Bir sahifali dastur) ilova modeli.

SPA modeli PHP va Classic ASP ga juda o'xshaydi.

ASP.NET Web Pages yangi ASP.NET Core bilan birlashtirilmoqda.

ASP.NET Web Pages – veb-sahifalari

Veb-sahifalar ASP.NET veb-saytlari va veb-ilovalarini yaratish uchun ko'plab dasturlash modellaridan biridir.

Veb-sahifalar HTML, CSS va server kodlarini birlashtirishning oson usulini taqdim etadi:

- O'rganish, tushunish va ishlatish oson
- SPA(Single Page Application-Bir sahifali ilova) ilova modelidan foydalanadi
- PHP va Classic ASP ga o'xshash
- VB (Visual Basic) yoki C# (C sharp) skript tillari

Bundan tashqari, veb-sahifalar ilovalari ma'lumotlar bazalari, videolar, grafikalar, ijtimoiy tarmoqlar va boshqalar uchun dasturlashtiriladigan yordamchilar bilan osongina kengaytirilishi mumkin.

ASP.NET Web API

ASP.NET API - bu API(Application Programming Interface – Ilova dasturlash interfeysi) ilova modeli.

ASP.NET API yangi ASP.NET Core bilan birlashtirilmoqda.

ASP.NET Web Forms - veb shakllari

ASP.NET Web Forms - bu hodisaga asoslangan dastur modeli.

ASP.NET Web Forms yangi ASP.NET Core tarkibiga kirmaydi.

ASP.NET MVC

ASP.NET MVC – MVC(Model View Controller – Modelni ko'rish boshqaruvchisi) ilova modelidir.

ASP.NET MVC yangi ASP.NET Core bilan birlashtirilmoqda.

ASP.NET Core - yadrosi

ASP.NET Core 2016 yilda chiqarilgan.

ASP.NET Core ASP.NET MVC, ASP.NET Web API va ASP.NET veb-sahifalarini bitta dastur tizimiga birlashtiradi.

ASP.NET Razor – Belgilash

Razor dasturlash tili emas. Bu server tomonidan belgilash tilidir.

Razor bu veb-sahifalarga serverga asoslangan kodni (Visual Basic va C#) joylashtirish imkonini beruvchi belgilash sintaksisidir.

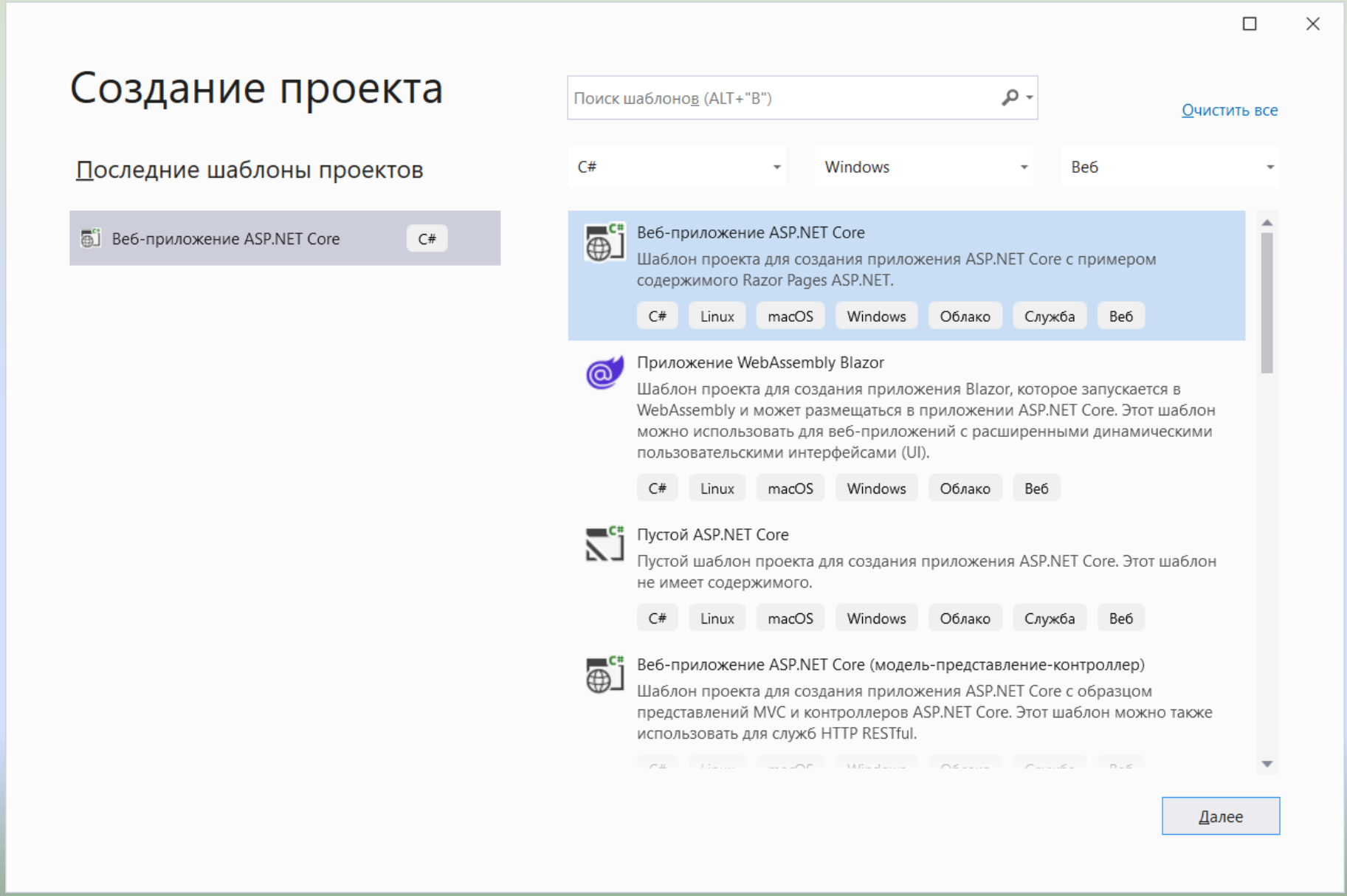
Serverga asoslangan kod tezda dinamik veb-kontentni yaratishi mumkin, shu bilan birga veb-sahifa brauzerga yoziladi. Veb-sahifa chaqirilganda, server sahifani brauzerga qaytarishdan oldin sahifa ichidagi serverga asoslangan kodni bajaradi. Serverda ishlash orqali kod ma'lumotlar bazalariga kirish kabi murakkab vazifalarni bajarishi mumkin. Razor ASP.NET-ga asoslangan va veb-ilovalarni yaratish uchun mo'ljallangan. U an'anaviy ASP.NET belgilash kuchiga ega, ammo undan foydalanish va o'rganish osonroq.

Razor sintaksisi

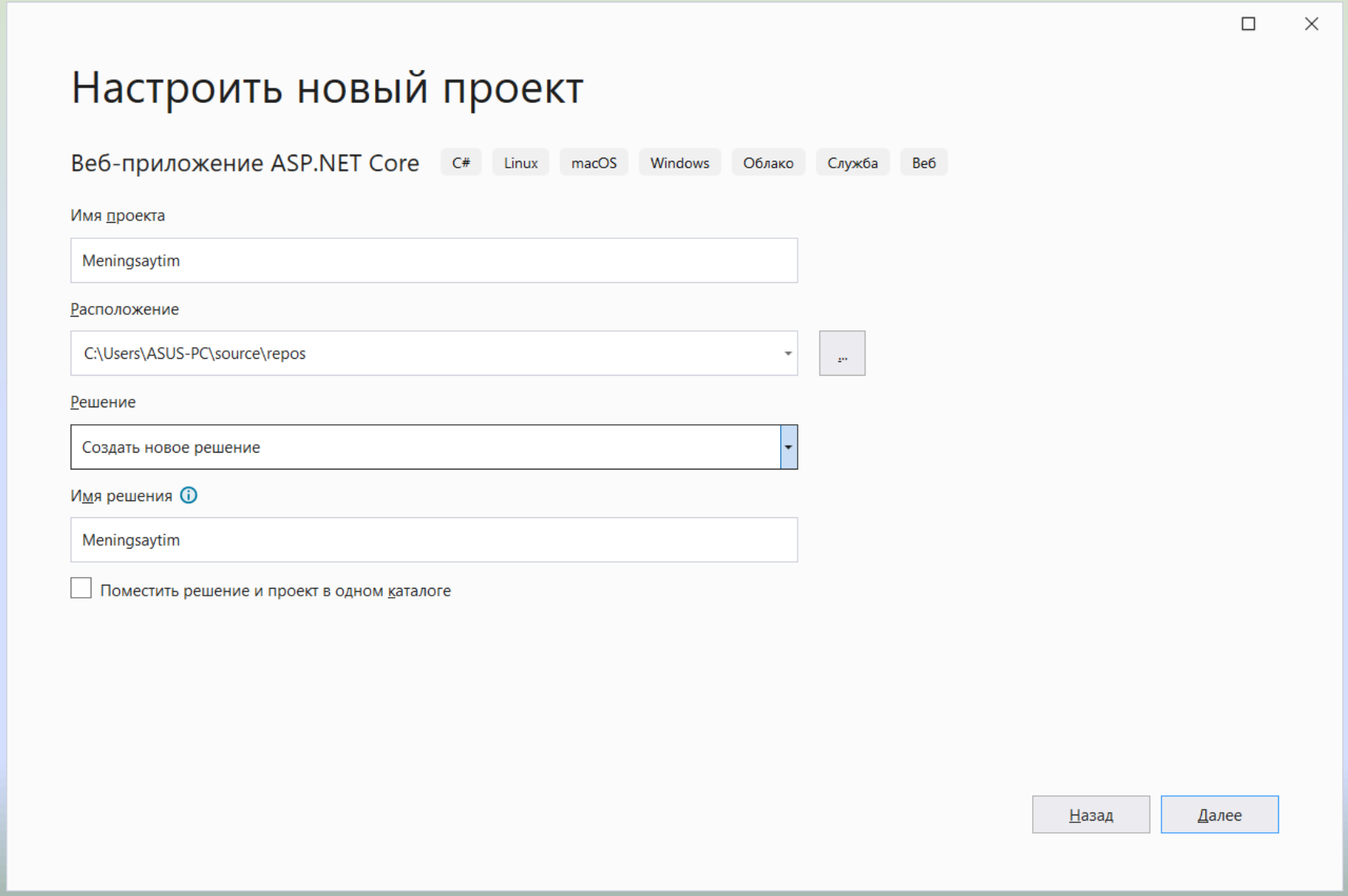
C# uchun asosiy Razor sintaksisi qoidalari:

- ✓ Razor kod bloklari **@{ ...}** ichiga kiritilgan
- ✓ Inline ifodalar (o‘zgaruvchilar va funksiyalar) **@** bilan boshlanadi
- ✓ Kod bayonotlari nuqta-vergul (;) bilan tugaydi
- ✓ O'zgaruvchilar **var** kalit so'zi bilan e'lon qilinadi
- ✓ Satrlar qo'shtirnoq bilan o'raladi
- ✓ C# kodi katta-kichik harflarga sezgirdir
- ✓ C# fayllari **.cshtml** kengaytmasiga ega

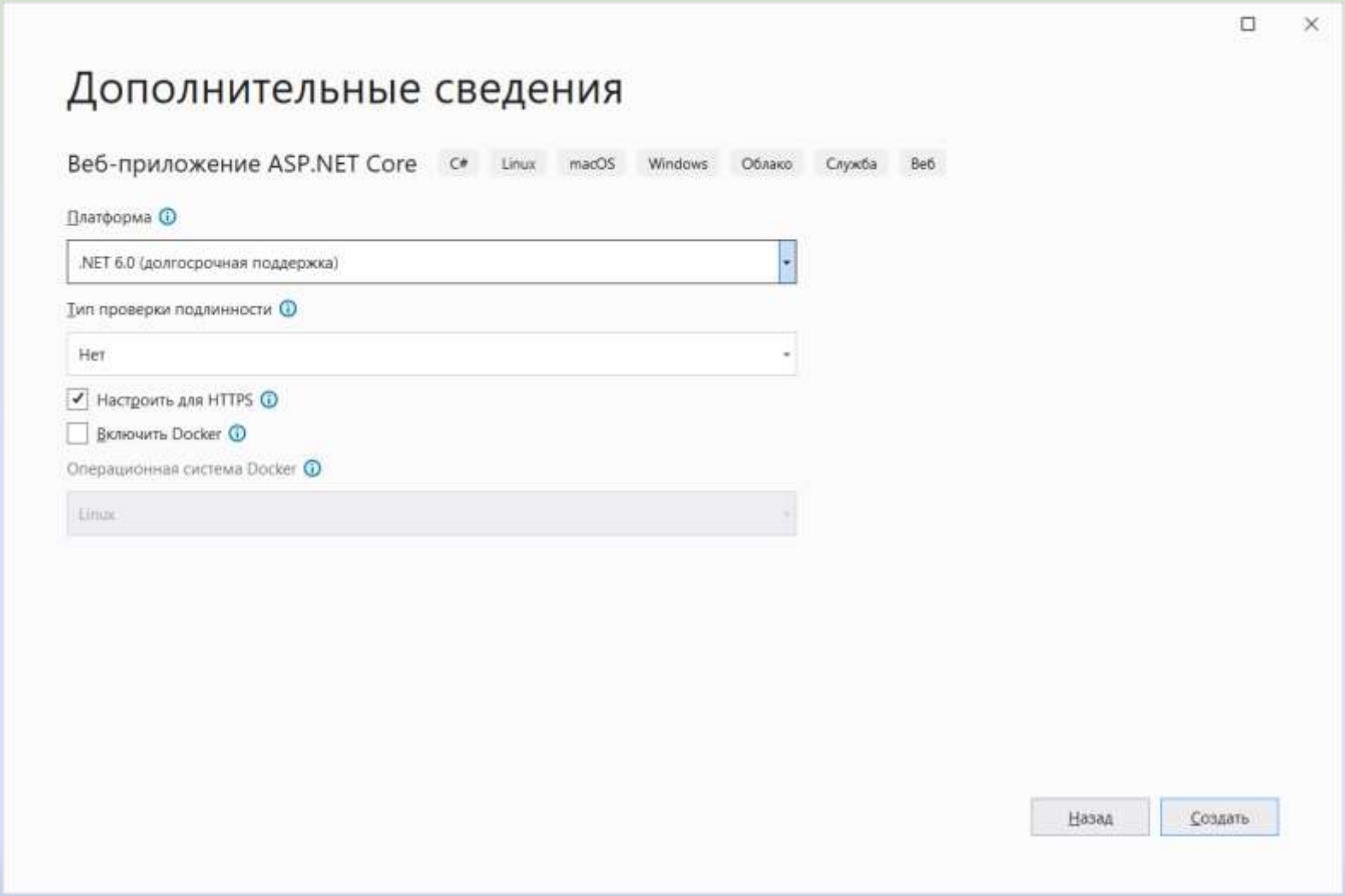
Microsoft Visual Studio 2022 da ASP.NET Core ishga tushirish oynasi quyidagicha:



Далее tugmasini bosib quyidagi oynaga o’tamiz:

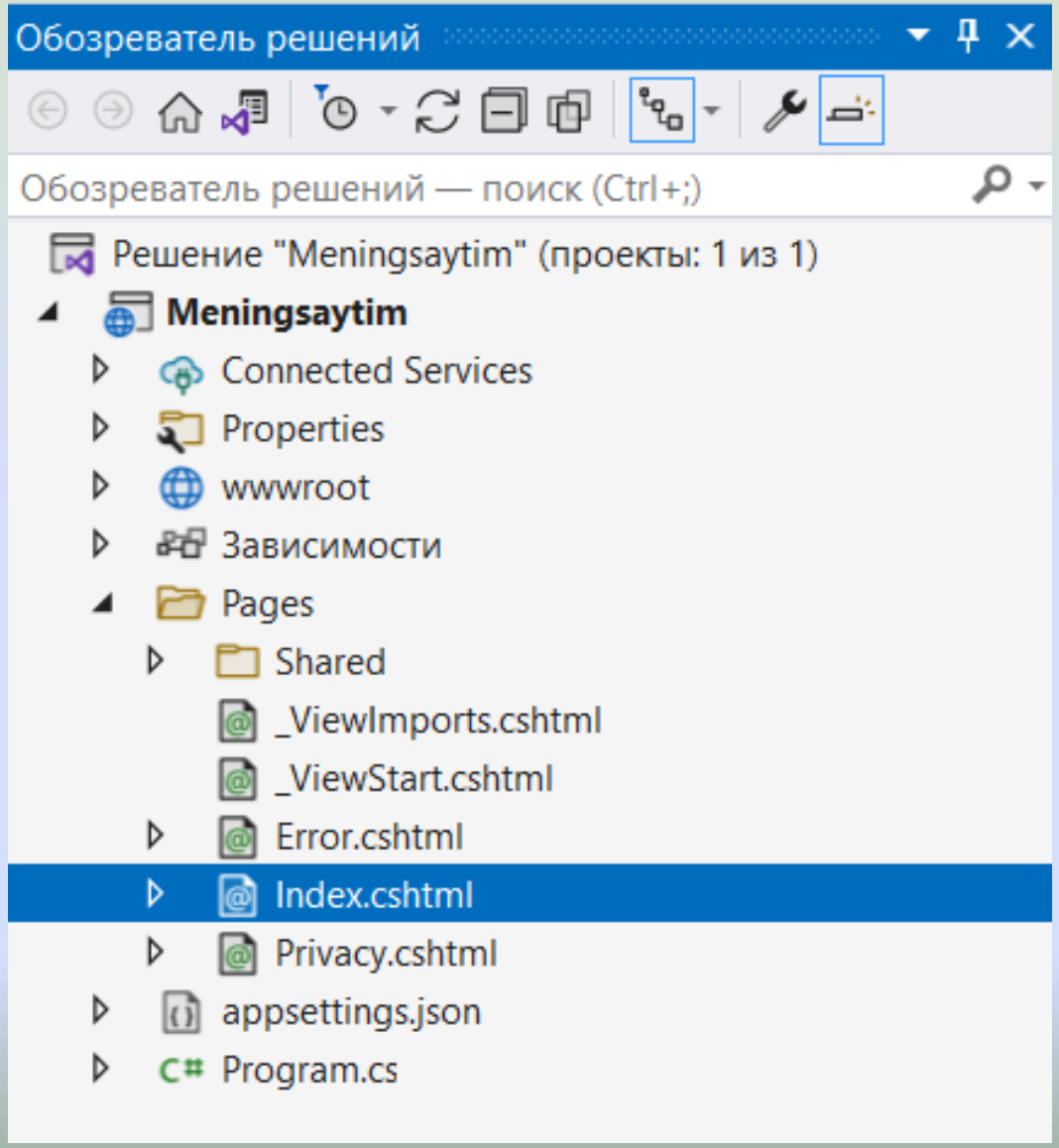


Yana Далее tugmasini bosib quyidagi oynaga o’tamiz:



Создать tugmasini bosib ASP.NET Core ishga tushiramiz:

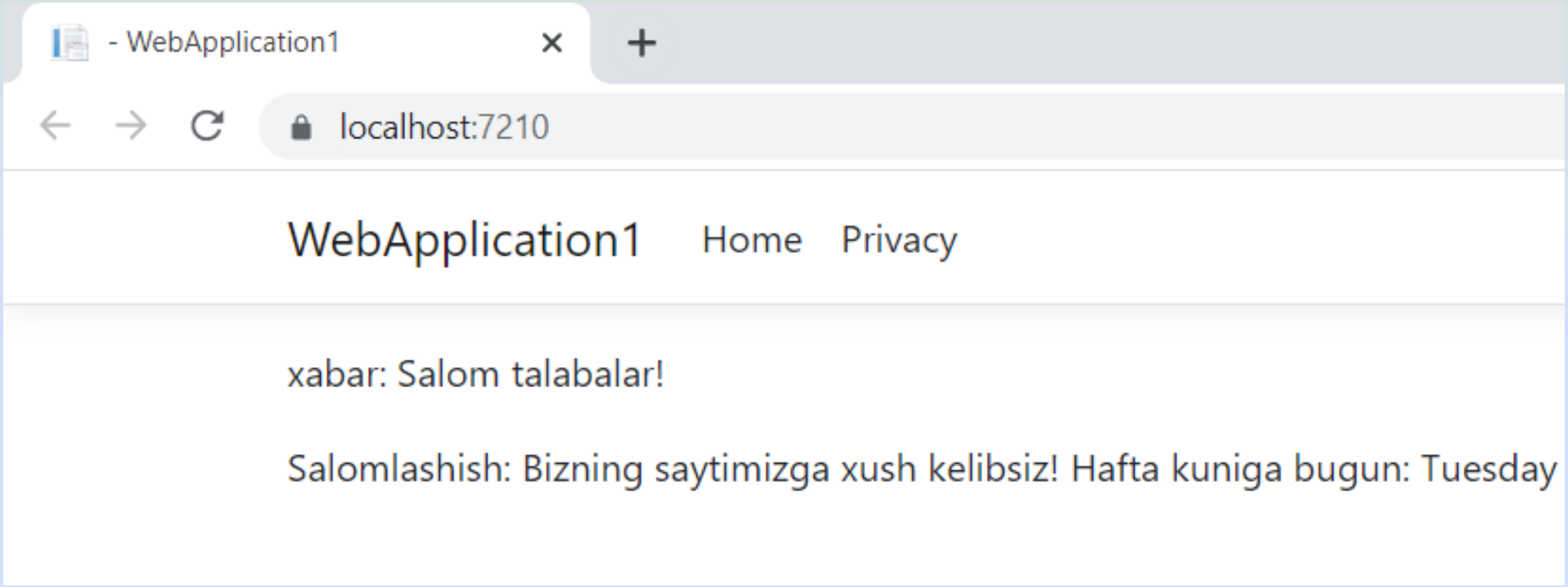
Ishchi sohadan Solution Explorer(Обозреватель решений bo’limini ko’ramiz va u yerdan **index.cshtml** faylini ochamiz:



C# misol

```
@page
<!-- Yagona bayonot bloki -->
@{ var xabar = "Salom talabalar!"; }

<!-- Inline ifoda yoki o'zgaruvchi -->
<p> xabar: @xabar</p>

<!-- Ko'p bayonotli blok -->
@{
var salomlashish = "Bizning saytimizga xush
kelibsiz!";
var haftakuni = DateTime.Now.DayOfWeek;
var salom_xabari = salomlashish + " Hafta kuniga
bugun: " + haftakuni;
}
<p>Salomlashish: @salom_xabari</p>
```

Yuqoridagi slayddagi C# misolning web brauzerdagi natijasi quyidagicha bo’ladi:



Ob'ektlar bilan ishlash

Serverni kodlash ko'pincha ob'ektlarni o'z ichiga oladi.

"DateTime" ob'ekti odatiy o'rnatilgan ASP.NET ob'ektidir, lekin ob'ektlar ham o'z-o'zidan aniqlanishi mumkin, veb-sahifa(web page), matn qutisi(text box), fayl(file), ma'lumotlar bazasi yozuvi(database record) va boshqalar.

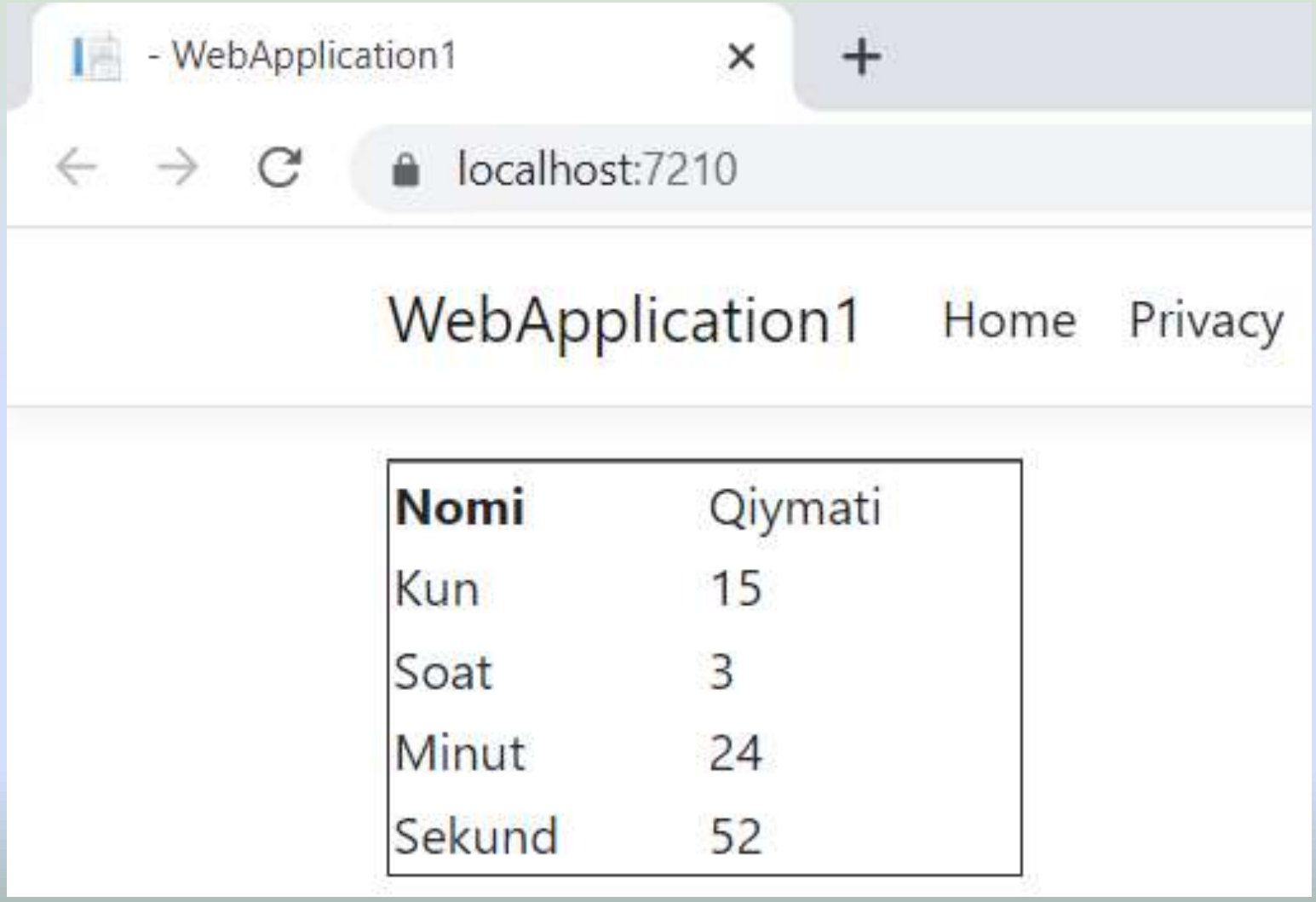
Ob'ektlar bajarishi mumkin bo'lgan usullarga ega bo'lishi mumkin. Ma'lumotlar bazasi yozuvida "Saqlash(Save)" usuli, tasvir ob'ektida "aylantirish(Rotate)" usuli, elektron pochta obyektida "Yuborish(Send)" usuli va hokazo bo'lishi mumkin.

Ob'ektlar, shuningdek, ularning xususiyatlarini tavsiflovchi xususiyatlarga ega. Ma'lumotlar bazasi yozuvi FirstName va LastName xususiyatiga ega bo'lishi mumkin (boshqalar qatorida).

ASP.NET DateTime obyekti Now xususiyatiga ega (DateTime.Now deb yozilgan), Now xususiyati esa Day xususiyatiga ega (DateTime.Now.Day sifatida yozilgan). Quyidagi misol DateTime ob'ektining ba'zi xususiyatlariga qanday kirishni ko'rsatadi:

```
<table border="1">
<tr>
<th width="100px">Name</th>
<td width="100px">Value</td>
</tr>
<tr>
<td>Day</td><td>@DateTime.Now.Day</td>
</tr>
<tr>
<td>Hour</td><td>@DateTime.Now.Hour</td>
</tr>
<tr>
<td>Minute</td><td>@DateTime.Now.Minute</td>
</tr>
<tr>
<td>Second</td><td>@DateTime.Now.Second</td>
</tr>
</td>
</table>
```

Yuqoridagi slayddagi C# misolning web brauzerdagi natijasi quyidagicha bo’ladi:



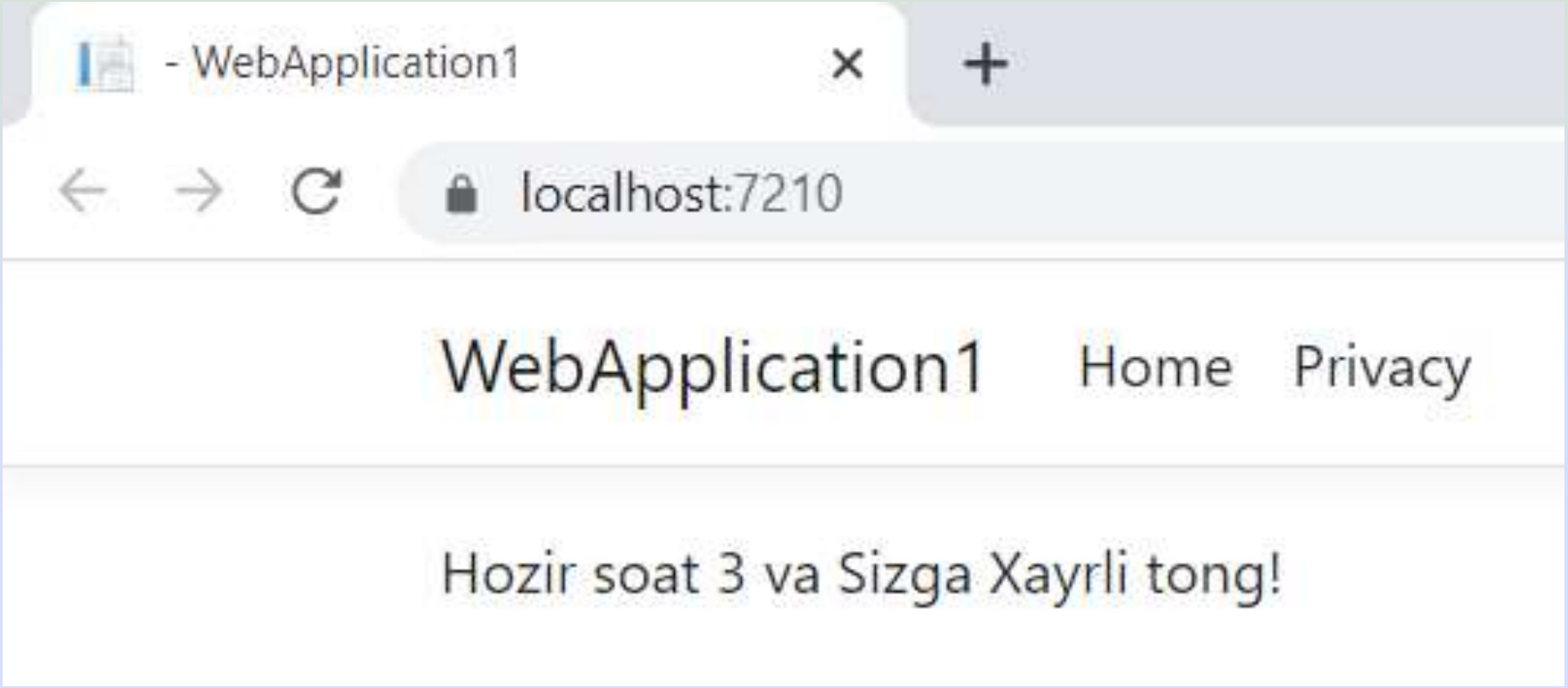
If va Else shartlari

Dinamik veb-sahifalarning muhim xususiyati shundaki, siz shartlarga qarab nima qilish kerakligini aniqlashingiz mumkin.

Buning umumiy usuli if ... else iboralari bilan amalga oshiriladi:

```
@page
@{
    var matn = "";
    var soat = DateTime.Now.Hour;
    if(DateTime.Now.Hour > 12)
  {matn = "Xayrli kech!";}
else
  {matn = "Xayrli tong!";}
}
<html>
<body>
<p>Hozir soat @soat va Sizga @matn</p>
</body>
</html>
```

Yuqoridagi slayddagi C# misolning web brauzerdagi natijasi quyidagicha bo’ladi:



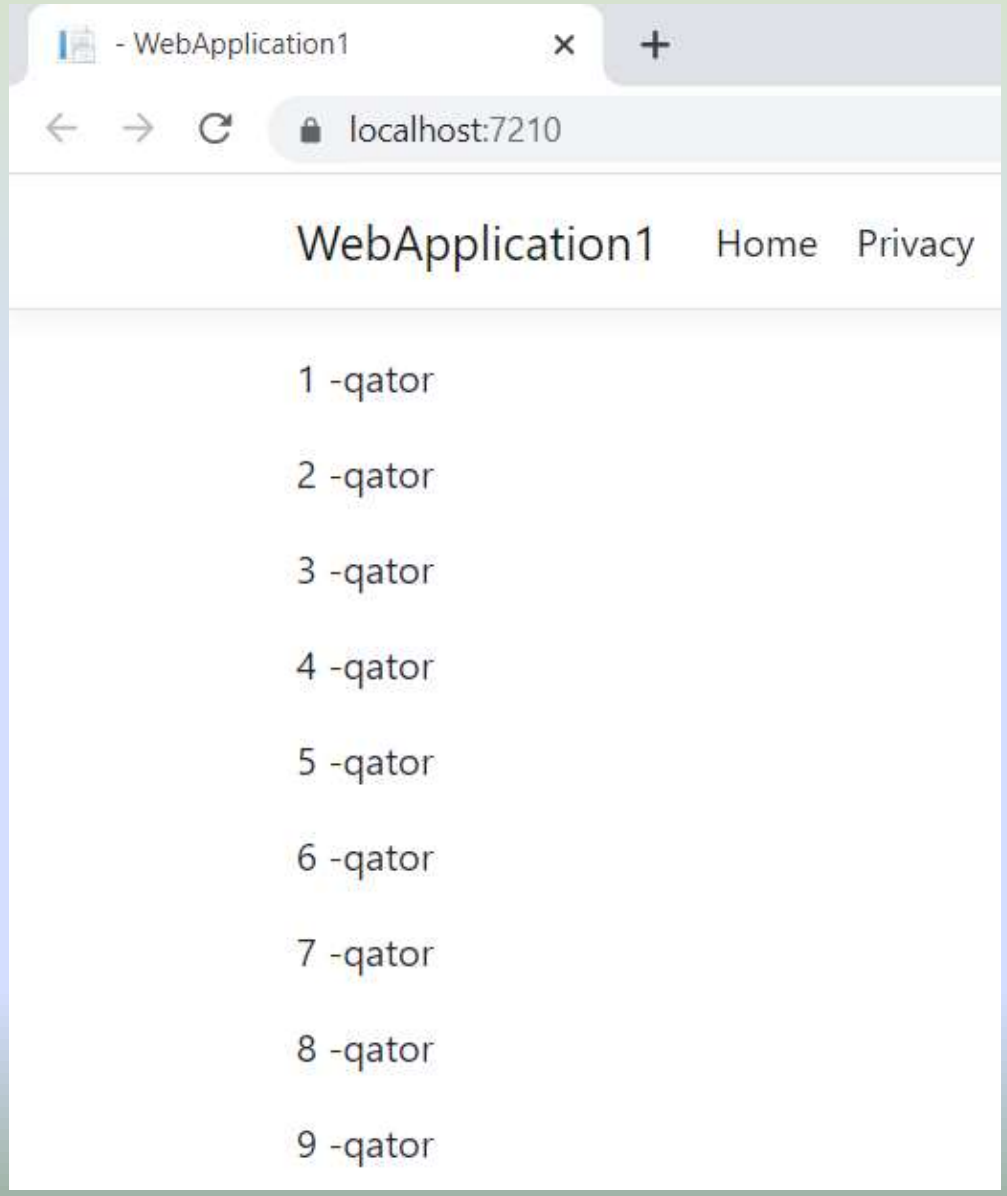
For sikl operatori

Agar siz bir xil iboralarni qayta-qayta bajarishingiz kerak bo'lsa, siz sikl operatoridan foydalanib dasturlashingiz mumkin.

Agar siz necha marta aylanishni xohlayotganingizni bilsangiz, for tsiklidan foydalanishingiz mumkin. Ushbu turdagi sikl, ayniqsa, yuqoriga yoki pastga sanash uchun foydalidir:

```
@page
<html>
<body>
@for(var i = 1; i < 10; i++)
    {<p>@i -qator </p>}
</body>
</html>
```

Yuqoridagi slayddagi C# misolning web brauzerdagi natijasi quyidagicha bo’ladi:



While sikl operatori

while sikli umumiy maqsadli sikldir.

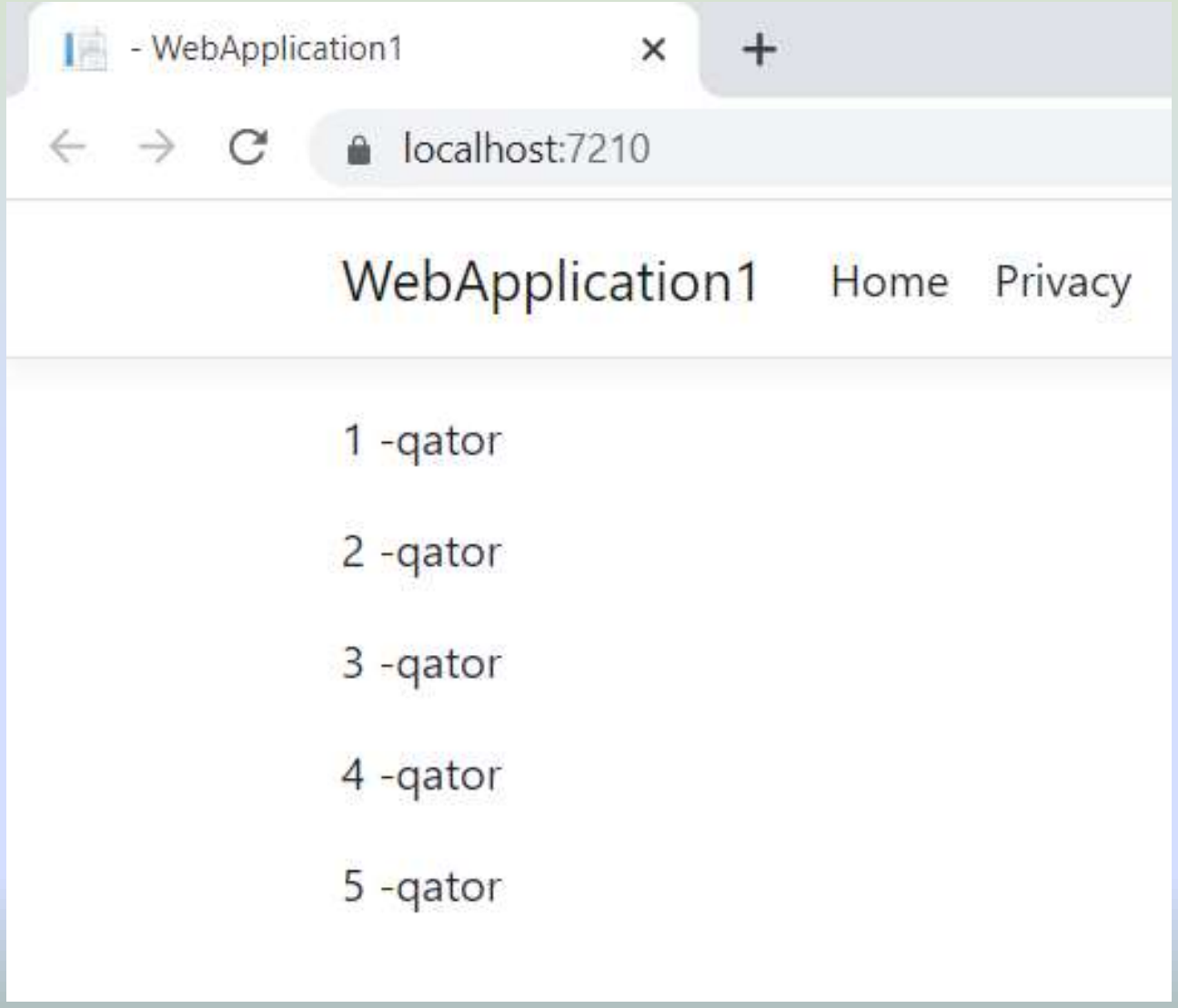
while sikli **while** kalit so'zidan boshlanadi, undan keyin qavslar olinadi, bu yerda sikl qancha davom etishini ko'rsatasiz, keyin esa takrorlanadigan blok.

While sikllari odatda hisoblash uchun ishlatiladigan o'zgaruvchiga qo'shadi yoki undan ayiradi.

Quyidagi misolda += operatori sikl har safar ishga tushganda i o‘zgaruvchisiga 1 qo‘shadi.

```
@page
<html>
<body>
@{
var i = 0;
while (i < 5)
    {
    i += 1;
    <p>@i -qator</p>
    }
}
</body>
</html>
```

Yuqoridagi slayddagi C# misolning web brauzerdagi natijasi quyidagicha bo’ladi:



ETIBORINGGIZ UCHUN RAHMAT!

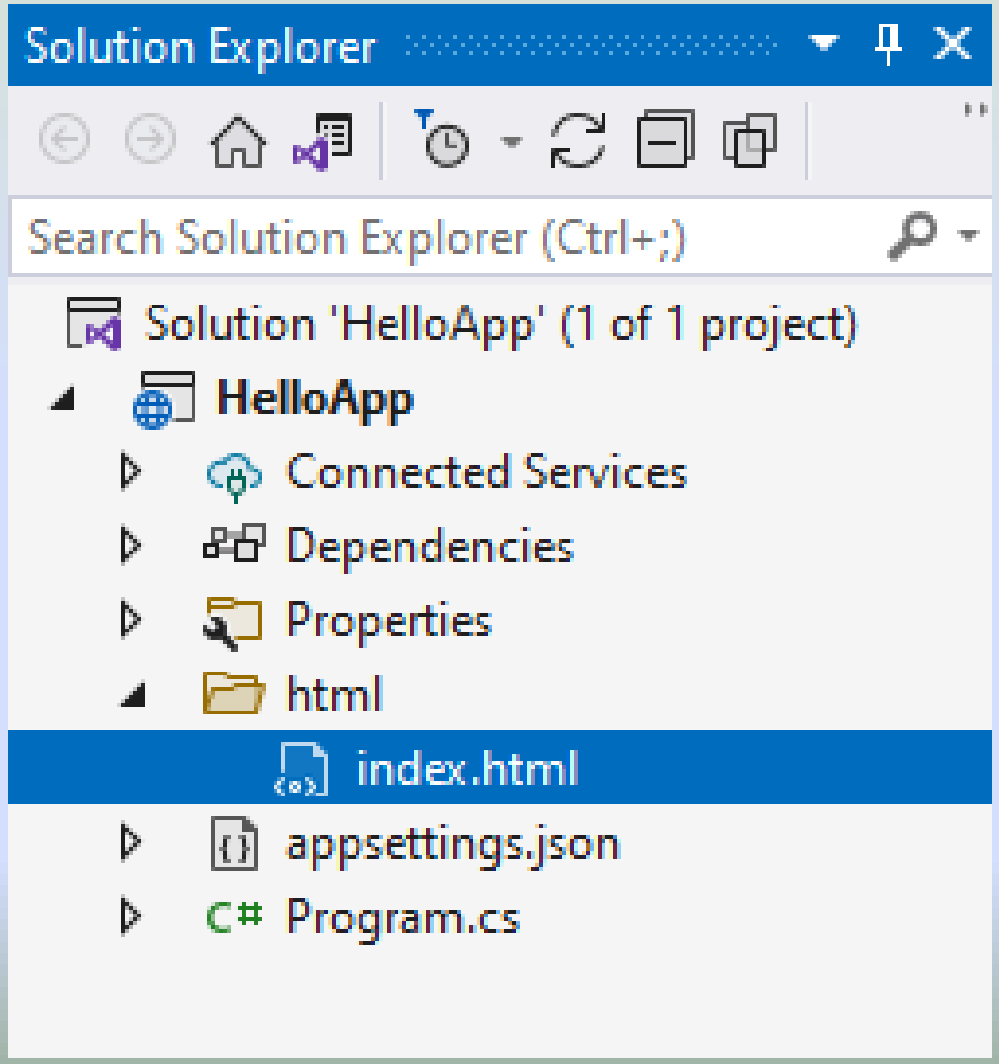
MAVZU: Hodisalar bilan ishlash

Maqsad: Talabalar Hodisalar bilan ishlash haqida tasavvurga ega bo’lish va Web sahifalar hosil qilishni o’rganish.

Formani yuborish

Odatda, **POST** so'rovida **html** formalari yordamida serverga ma'lumotlar yuborilishining odatiy holi emas. Bunday ma'lumotlarni olish uchun **Form** xususiyati **HttpRequest** sinfida aniqlanadi. bunday ma'lumotlarni qanday olishimiz mumkinligini ko'rib chiqaylik.

Avvalo, loyihadagi **html** papkasidagi index.html faylini aniqlaymiz.



index.html dagi quyidagi tarkibni aniqlaymiz:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
</head>
<body>
    <h2>User form</h2>
    <form method="post" action="postuser">
        <p>Name: <input name="name" /></p>
        <p>Age: <input name="age" type="number" /></p>
        <input type="submit" value="Send" />
    </form>
</body>
</html>
```

Bu yerda foydalanuvchi ma'lumotlarini kiritish uchun forma shartli ravishda aniqlanadi va u **POST** tipidagi so'rovda (method= "post“ atribut) ma'lumotlarni "postuser" (atribut action="postuser") manziliga yuboradi.

Forma ikkita kiritish maydoniga ega. Birinchi maydon foydalanuvchi nomini kiritish uchun. Ikkinchi maydon foydalanuvchining yoshini kiritish uchun.

Ushbu ma'lumotlarni olish uchun **Program.cs** faylida quyidagi kodni aniqlaymiz:

```
var builder = WebApplication.CreateBuilder();
var app = builder.Build();
app.Run(async (context) =>
{
    context.Response.ContentType = "text/html; charset=utf-8";
    // agar so'rov "/postuser" manziliga kirsa, biz forma ma'lumotlarini olamiz
    if (context.Request.Path == "/postuser")
    {
        var form = context.Request.Form;
        string name = form["name"];
        string age = form["age"];
        await context.Response.WriteAsync($"<div><p>Name:
{name}</p><p>Age: {age}</p></div>");
    }
    else
    {       await context.Response.SendFileAsync("html/index.html");
    }
});
app.Run();
```

Bu yerda, agar "/postuser" manzili so'ralsa, qandaydir forma topshirilgan deb taxmin qilinadi. Birinchidan, biz yuborilgan formani **form** o'zgaruvchisiga olamiz:

var form = context.Request.Form;

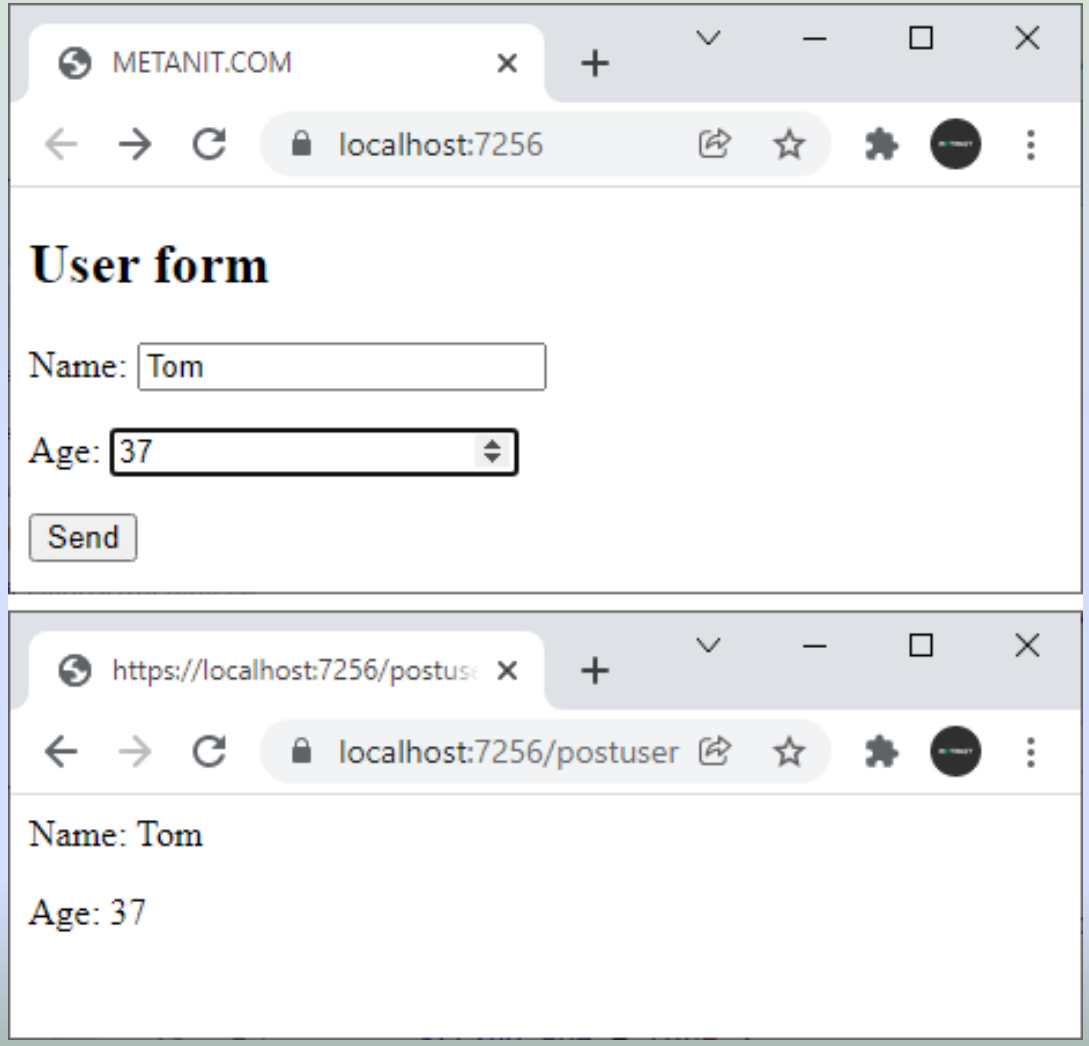
Request.Form xossasi **IFormCollection** ob'ektini qaytaradi - bu elementning qiymatini kalit bo'yicha olishingiz mumkin bo'lgan lug'at turidir. Bunday holda, forma maydonlarining nomlari (forma elementlarining **name** atributlarining qiymatlari) kalit sifatida ishlaydi:

```
<input name="age" type="number" />
```

Shunday qilib, bu holda, maydon nomi (**name** atributining) qiymati “age“ ga tengdir. Shunga ko'ra, **Request.Form da** ushbu nom bilan biz uning qiymatini olishimiz mumkin:

```
string age = form["age"];
```

Forma ma'lumotlari olingandan so'ng, u mijozga qaytariladi:



Massivlarni olish

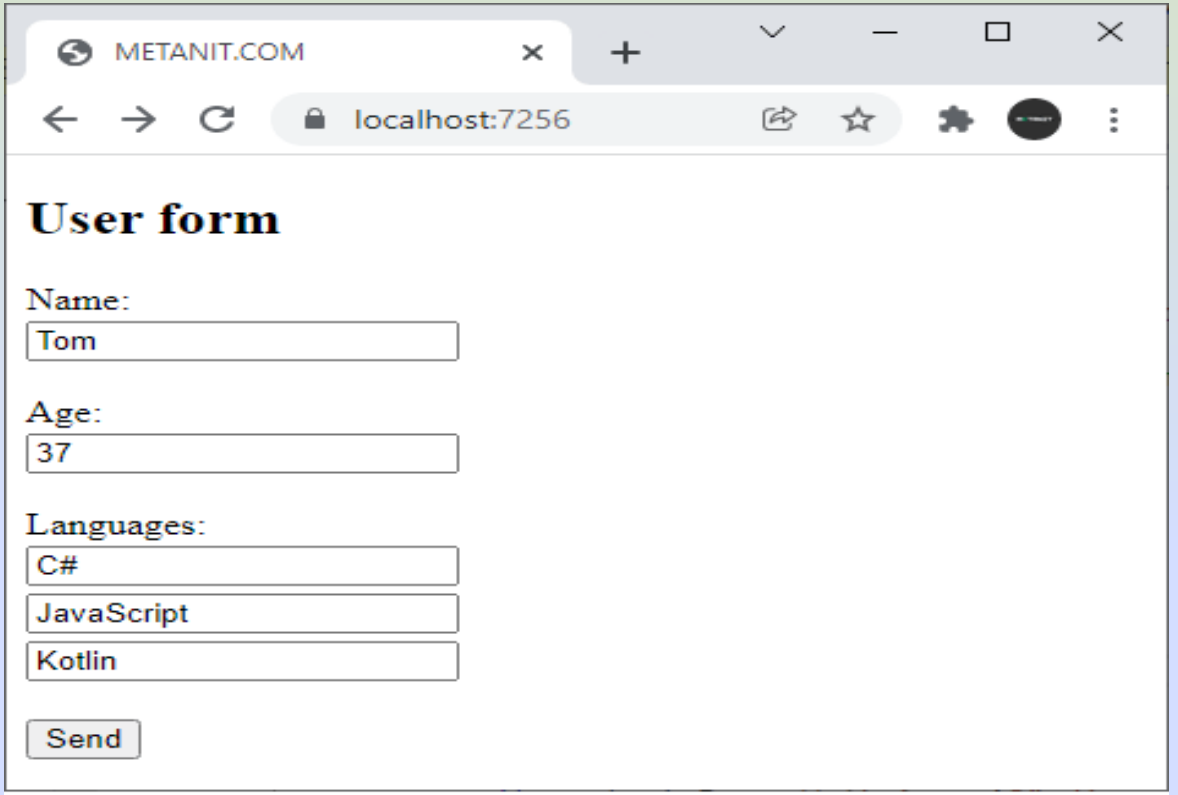
Keling, vazifani murakkablashtiramiz va **index.html** sahifasidagi formaga massivni ifodalovchi bir nechta maydonlarni qo'shamiz:

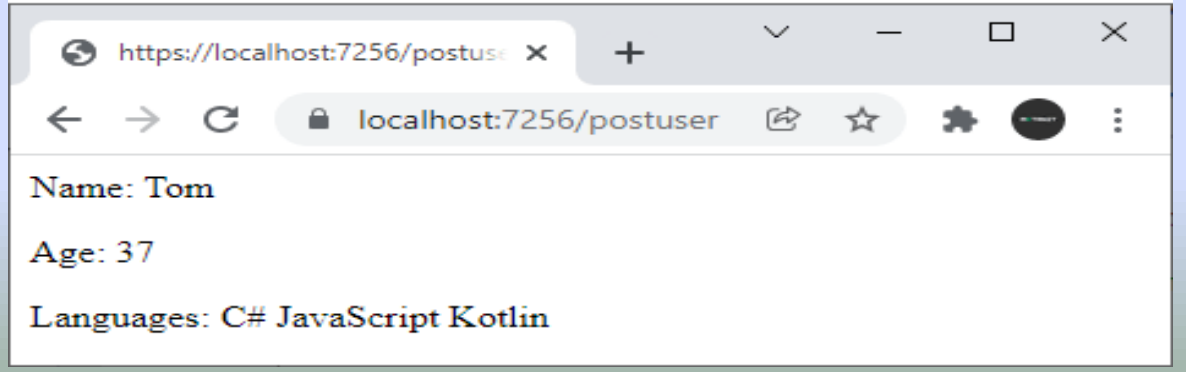
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
</head>
<body>
    <h2>User form</h2>
    <form method="post" action="postuser">
        <p>
            Name: <br />           <input name="name" />
        </p>
        <p>      Age: <br />           <input name="age" type="number" />
        </p>
        <p>     Languages:<br />
            <input name="languages" /><br />
            <input name="languages" /><br />
            <input name="languages" /><br />
        </p>
        <input type="submit" value="Send" />
    </form>
</body>


</html>
```

Bu yerda bir xil nomga ega uchta kiritish maydoni qo'shilgan. Shuning uchun, ular yuborilganda, uchta qiymatdan iborat massiv hosil bo'ladi. Endi biz ushbu qiymatlarni C# kodida olamiz:

```
app.Run(async (context) =>
{
    context.Response.ContentType = "text/html; charset=utf-8";
    // agar so'rov "/postuser" manziliga kirsa, biz forma ma'lumotlarini
olamiz
    if (context.Request.Path == "/postuser")
    {
        var form = context.Request.Form;
        string name = form["name"];
        string age = form["age"];
        string[] languages = form["languages"];
        // tillar massividan bitta satr yaratish
        string langList = "";
        foreach (var lang in languages)
        {
            langList += $" {lang}";
        }
        await context.Response.WriteAsync($"<div><p>Name: {name}</p>" +
            $"<p>Age: {age}</p>" +
            $"<ul>Languages:{langList}</ul></div>");
    }
    else
    {
        await context.Response.SendFileAsync("html/index.html");
    }
});
```

Veb-sahifaga chiqarish uchun html kodi ushbu massivdan satr sifatida shakllantiriladi:





MAVZU: Foydalanuvchi kiritgan ma’lumotlarni tekshirish

Maqsad: Talabalar foydalanuvchi kiritgan ma’lumotlarni tekshirishni o’rganish.

Massivlarni olish

Xuddi shunday, siz boshqa turdagi maydonlarning massiv qiymatlarini yoki elementlar to'plamini ifodalovchi maydonlarni, masalan, bir nechta tanlovni qo'llab-quvvatlaydigan **select** elementni tanlashingiz mumkin:

MAVZU: Klient tomondan holatni boshqarish

Maqsad: Talabalar Klient tomondan holatni boshqarishni o’rganish.

REJA:

1. ASP.NET Core MVC ga kirish
2. ASP.NET Core MVC birinchi loyiha
3. ASP.NET Core Empty loyihasi

ASP.NET Core MVC framework(struktura) i

ASP.NET Core MVC strukturasi ASP.NET Core platformasining bir qismidir va uning ajralib turadigan xususiyati MVC modelidan foydalanishdir. "Sof" ASP.NET Core bilan solishtirganda ASP.NET Core MVC strukturasidan foydalanishning afzalligi shundaki, u bir qator vaziyatlar va stsenariylarda, ayniqsa, katta ilovalar uchun ilovalarni tashkil etish va yaratishni osonlashtiradi.

MVC strukturasi kontseptsiyasi dasturni uchta komponentga bo'lishni o'z ichiga oladi:

1. Model (Model): ilovada ishlatiladigan ma'lumotlarni, shuningdek ma'lumotlarga bevosita bog'liq bo'lgan mantiqni, masalan, ma'lumotlarni tekshirish mantiqini tavsiflaydi. Odatda, model ob'ektlari ma'lumotlar bazasida saqlanadi.

MVC da modellar **ikkita asosiy tur** bilan ifodalanadi:

- **ko'rinish modellari:** ko'rinishlar tomonidan ma'lumotlarni ko'rsatish va uzatish uchun foydalaniladigan model;
- **domen modellari:** ma'lumotlarni boshqarish mantiqini tavsiflovchi model.

Model ma'lumotlarni o'z ichiga olishi, ushbu ma'lumotlarni boshqarish mantiqini saqlashi mumkin. Shu bilan birga, model foydalanuvchilarning o'zaro ta'siri mantig'ini o'z ichiga olmaydi va so'rovni qayta ishlash mexanizmini aniqlamasligi kerak. Bundan tashqari, modelda ko'rinishda ma'lumotlarni ko'rsatish uchun mantiq bo'lmasligi kerak.

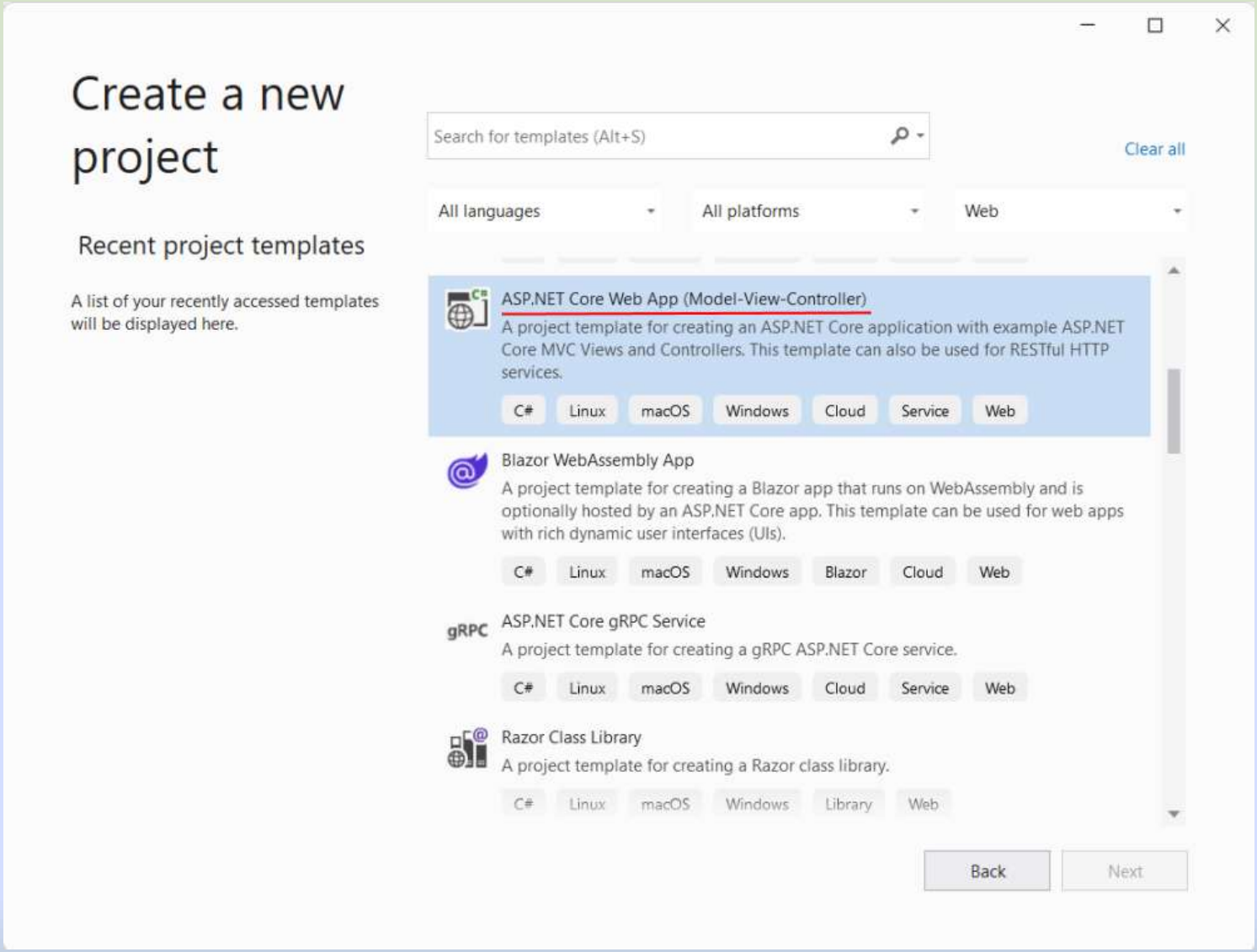
Ko'rinish (View): vizual qism yoki foydalanuvchi interfeysi uchun javob beradi, ko'pincha, foydalanuvchi ilova bilan o'zaro aloqada bo'lgan HTML sahifasidir. Shuningdek, ko'rinishda ma'lumotlarni ko'rsatish bilan bog'liq mantiq bo'lishi mumkin. Shu bilan birga, ko'rinishda foydalanuvchi so'rovini qayta ishlash yoki ma'lumotlarni boshqarish mantig'i bo'lmasligi kerak.

Nazoratchi (Controller): MVC ning markaziy komponentini ifodalaydi, u foydalanuvchi va ilova, ko'rinish va ma'lumotlar ombori o'rtasidagi aloqani ta'minlaydi. Unda **foydalanuvchi so'rovini qayta ishlash mantig'i mavjud**. Tekshirish moslamasi foydalanuvchi ma'lumotlarini qabul qiladi va uni qayta ishlaydi. Va qayta ishlash natijalariga qarab, u foydalanuvchiga ma'lum bir chiqishni yuboradi, masalan, model ma'lumotlari bilan to'ldirilgan ko'rinish shaklida bo’lishi mumkin.

ASP.NET Core MVC birinchi loyiha

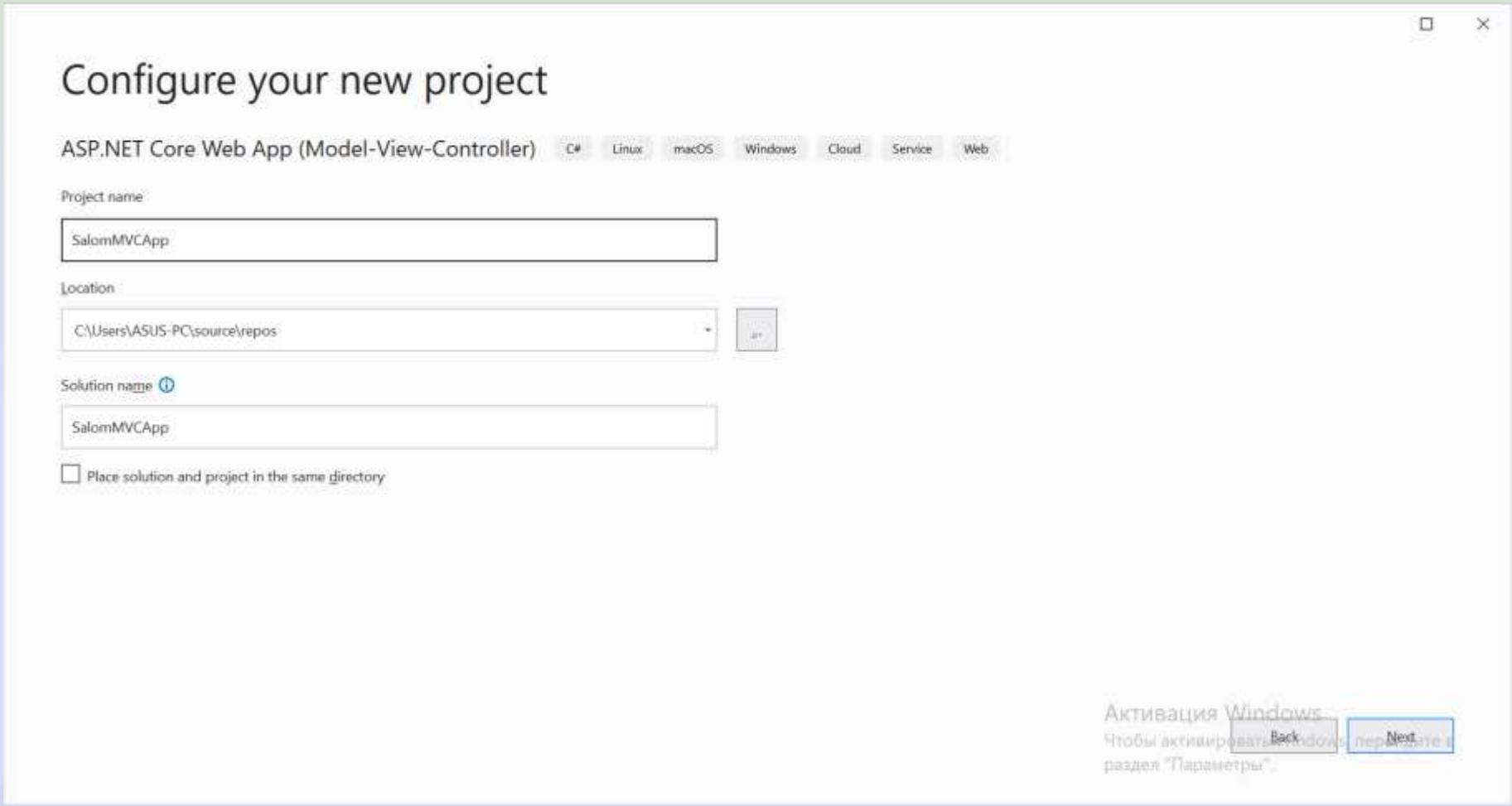
ASP.NET Core MVC da loyiha yaratish uchun biz ASP.NET Core da istalgan turdagi loyihani tanlashimiz va unga kerakli komponentlarni qo'shishimiz mumkin. Biroq, soddalashtirish uchun Visual Studio standart bo'yicha **ASP.NET Core Web App (Model-View-Controller)** shablonini taqdim etadi:

ASP.NET Core MVC birinchi loyiha

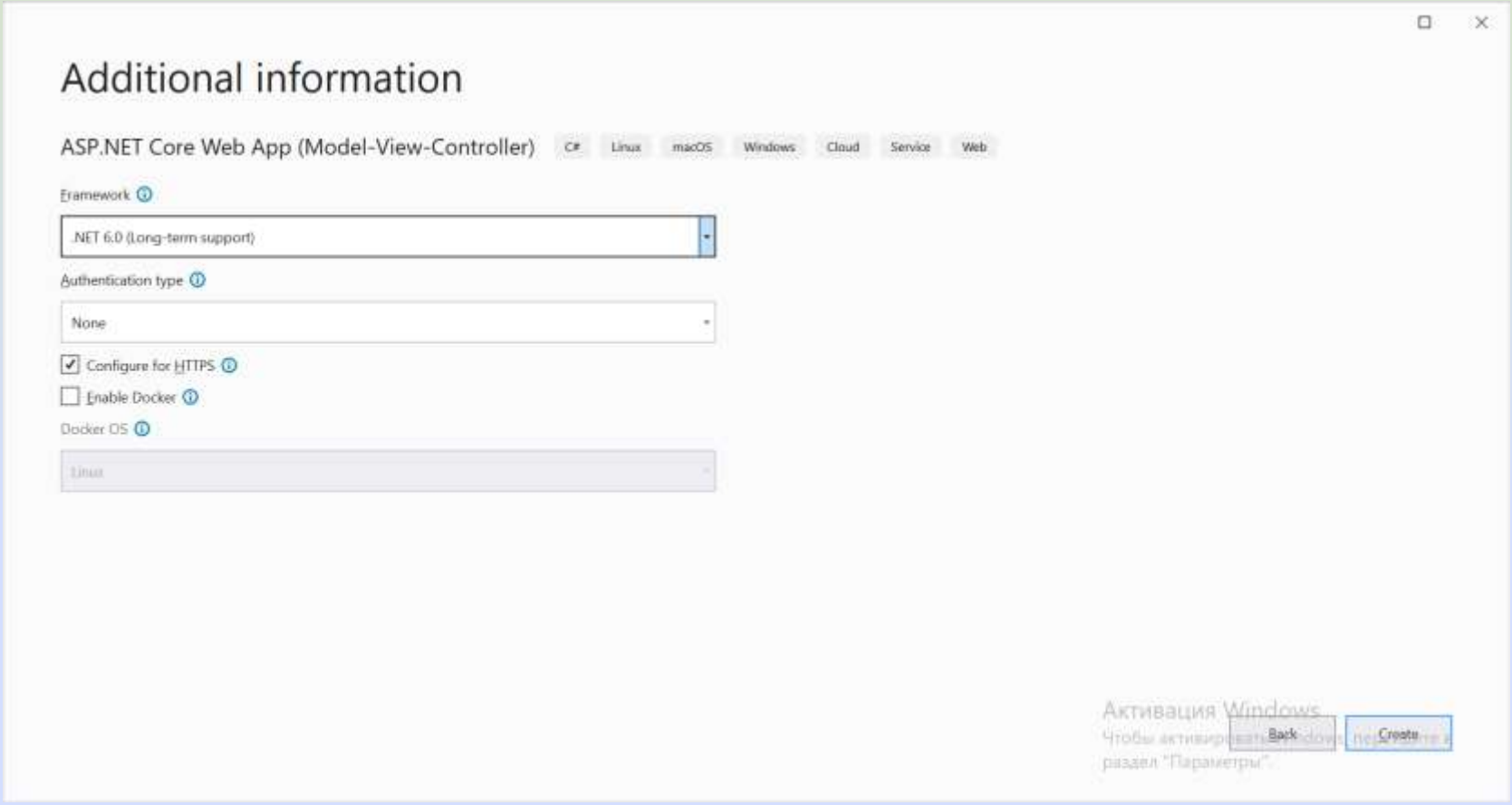


Loyiha yaratish uchun ushbu shablonni tanlaymiz.

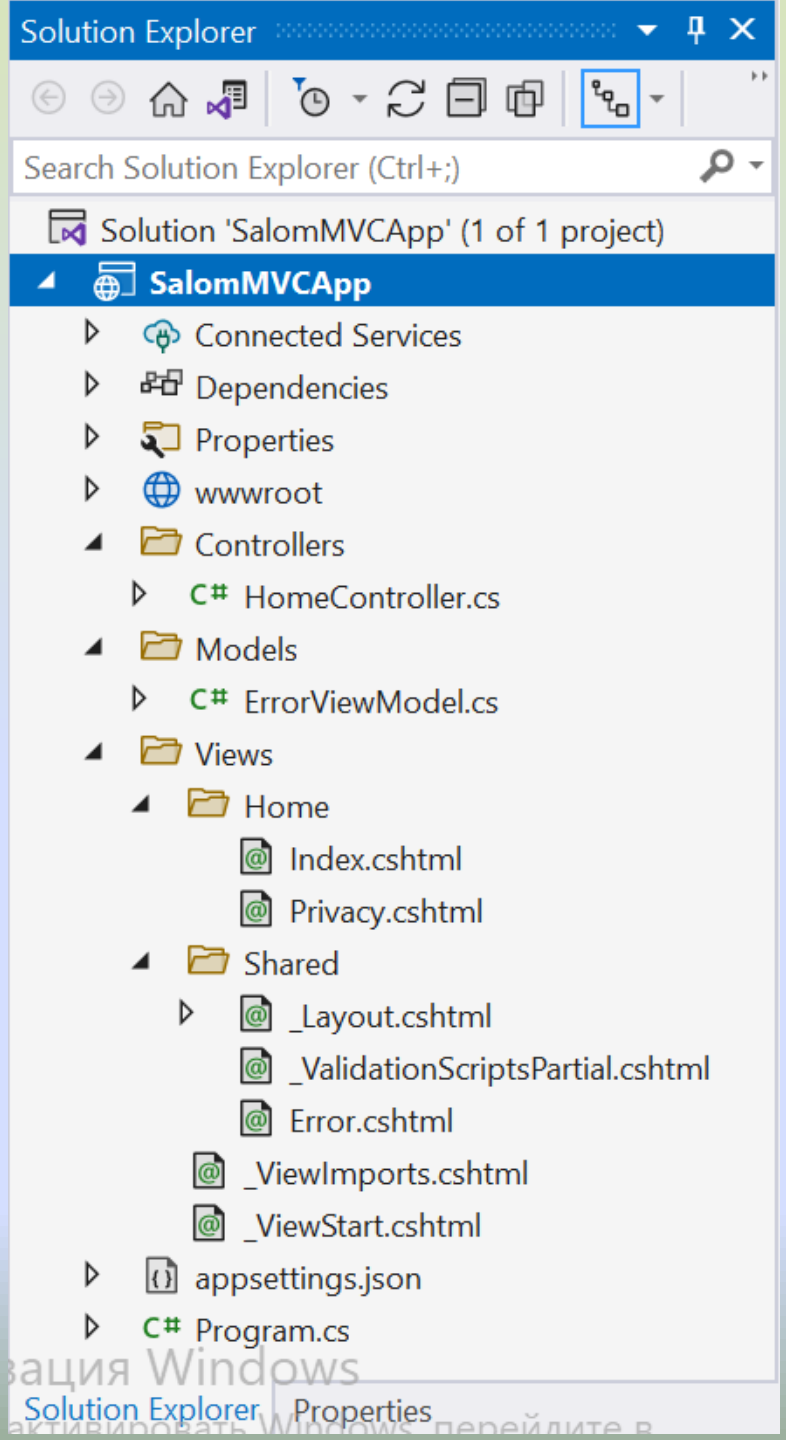
Keyinchalik, loyiha nomini o'rnatish uchun oyna ochiladi. Aytaylik, loyiha SalomMVCApp deb nomlanadi:



Keyinchalik, ASP.NET Core uchun standart sozlamalarni sozlashimiz kerak bo'ladi:



Barcha sozlamalarni standart holatida qoldirib, **Create** tugmasini bosamiz. Natijada, Visual Studio yangi MVC loyihasini yaratadi.



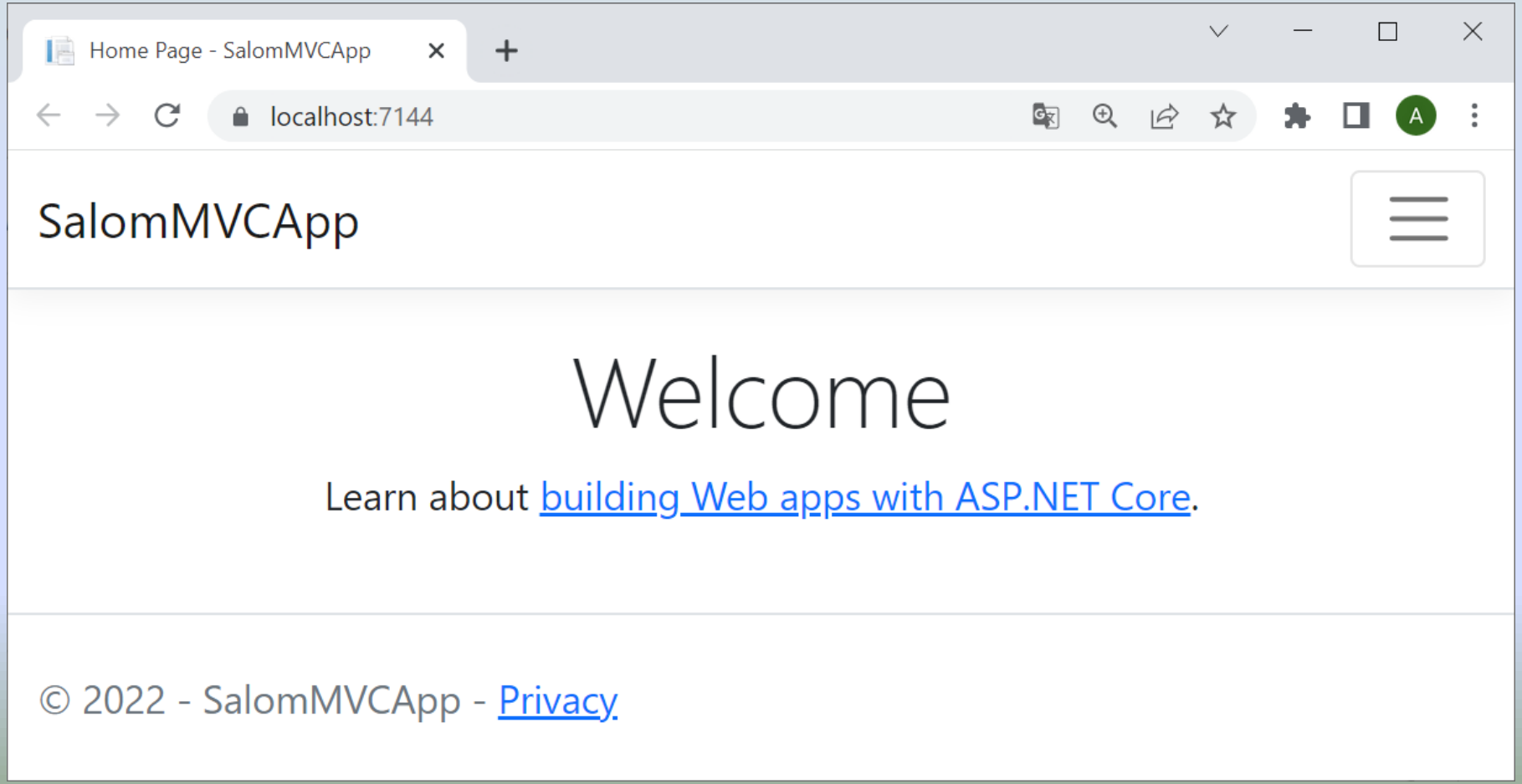
Yaratilayotgan loyihaning strukturasi **Empty** tipidagi loyiha tuzilishidan farq qiladi. Xususan, biz bir qator yangi papkalar va fayllarni ko'ramiz:

- **Bog'liqliklar(Dependencies):** loyihaga qo'shilgan barcha paketlar va kutubxonalar;
- **wwwroot:** bu tugun (qattiq diskda u xuddi shu nomdagi papkaga mos keladi) ilova tomonidan ishlatiladigan statik fayllar - tasvirlar, javascript skriptlari, CSS fayllari va boshqalarni saqlash uchun mo‘ljallangan;
- **Nazoratchilar(Controllers):** ilova tomonidan ishlatiladigan kontrollerlarni saqlash uchun papka. Odatiy bo'lib, bu yerda allaqachon bitta kontroller mavjud – **Homecontroller;**

- **Modellar(Models)**: modellarni saqlash uchun katalog. Odatiy bo'lib, **ErrorviewModel** bu yerda yaratilgan;
- **Ko'rinishlar(Views):** ko'rinishlarni saqlash uchun katalog. Bir qator fayllar - ko'rinishlar ham odatiy bo'yicha bu yerga qo'shiladi;
- **appsettings.json:** ilova konfiguratsiyasini saqlaydi;
- **Program.cs:** ASP.NET Core ilovasiga kirish nuqtasini belgilaydigan fayl.

Aslida, bu Empty loyihasi bilan bir xil tuzilma, bundan tashqari MVC strukturasining asosiy komponentlari uchun standart papkalar ham mavjud: **kontrollerlar** va **ko'rinishlar**. Shuningdek, dasturning mijoz tomonining bog'liqliklarini boshqarish uchun qo'shimcha tugunlar va fayllar mavjud.

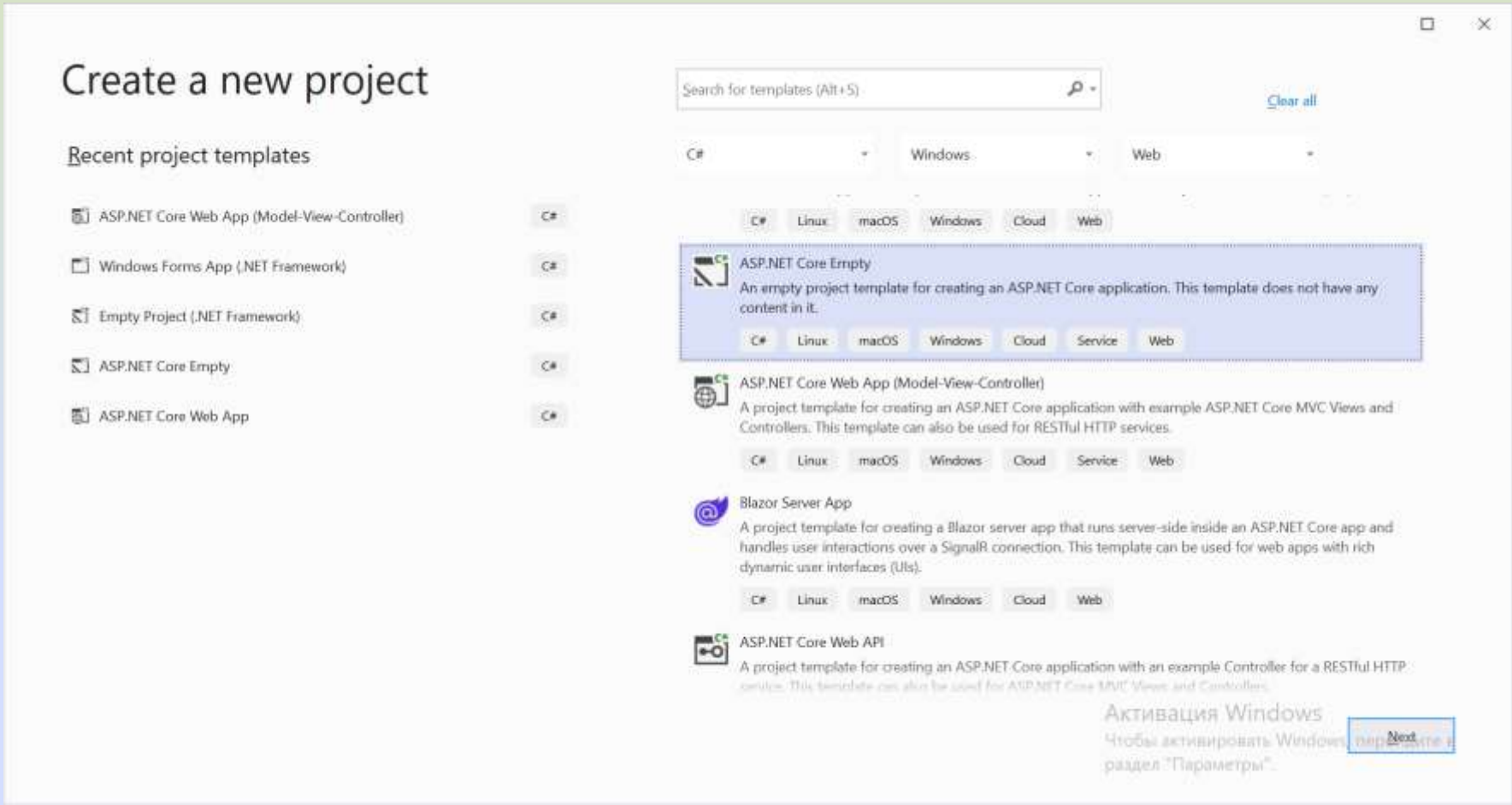
Va, agar biz loyihani bajarish uchun ishga tushiradigan bo'lsak, u holda standart kontrollerga - **HomeController** sinfiga so'rov yuboriladi, u javobni yaratish uchun kerakli ko'rinishni tanlaydi. Natijada, veb-brauzerimizda ko'radigan ko'rinishdan **html sahifasi** yaratiladi:



ASP.NET Core Empty loyihasi

Yuqorida biz ASP.NET Core loyihasi yaratdik, u standart bo'yicha MVC modelini ishlatadi, lekin sahna ortida qoldirilgan, MVC modelini ASP.NET Core loyihasida aynan nima qilinadi. Hozir ushbu holatni ko'rib chiqaylik va buni yaxshiroq tushunish uchun **ASP.NET Core Empty** tipidagi standart bo'sh loyihani olaylik, u standart bo'yicha loyihaga hech qanday MVC funksiyasini ulamaydi.

ASP.NET Core Empty loyihasi



Aytaylik, mening loyiham **MvcApp** deb ataladi. Odatiy holatdagi, dastlabki ASP.NET Core loyihasi yaratiladi, unda **Program.cs** faylida standart kod mavjud:

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

app.MapGet("/", () => “Salom
Dunyo!");

app.Run();
```

Endi biz ilovamizda MVC modelining funksiyasidan foydalanamiz va **Program.cs** faylini quyidagicha o'zgartiramiz:

```
var builder = WebApplication.CreateBuilder(args);

//MVC xizmatlarini qo'shish
builder.Services.AddControllersWithViews();
var app = builder.Build();

//nazoratchilarga marshrutlarni o`rnatish
app.MapControllerRoute(
    name: "default",
    pattern:
"{controller=Home}/{action=Index}/{id?}");

app.Run();
```

Avvalo shuni ta'kidlash kerakki, MVC funksionalligi, xususan, kontrollerlar va ko'rinishlarni qo'llab-quvvatlash ilovaga xizmat sifatida ulanadi - bu holda **services.AddControllersWithViews**() ga murojaat qilish orqali. Shundan so'ng, biz MVC modelining funksionalligidan foydalanishimiz mumkin.

Bundan tashqari, **MapControllerRoute**() usuli kiruvchi foydalanuvchi so'rovlarini kontrollerlarga joylashtirish uchun ishlatiladi. Marshrut nomi usulga birinchi parametr - **name** orqali uzatiladi - bu holda “**default**". Ikkinchi parametr – **pattern** parametri orqali so'rov mos kelishi kerak bo'lgan marshrut shabloni o'tkaziladi. Marshrut namunasi **"{controller=Home}/{action=Index}/{id?}",** bu uch segmentli so'rovni ifodalaydi.

Unda birinchi segment **nazoratchi**ni, ikkinchi segment **nazoratchi usuli**ni, uchinchisi esa **ixtiyoriy parametr**ni ifodalaydi. Bunday holda, agar so'rovda segmentlar ko'rsatilmagan bo'lsa (masalan, murojaat veb-ilovaning ildiziga o'tadi), u holda **HomeController** standart boshqaruvchi sifatida ishlatiladi va **Index** usuli uning usuli sifatida ishlatiladi.

Nazoratchini(Controller) qo'shish

Endi kontrollerlarni saqlash uchun loyihaga **Controllers** papkasini qo'shamiz. Va keyin biz unga yangi sinf qo'shamiz, uni biz **HomeController** deb nomlaymiz va quyidagi kodga ega bo'lamiz:

```
using Microsoft.AspNetCore.Mvc;

namespace MvcApp.Controllers
{
    public class HomeController: Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Nazoratchi Controller sinfidan meros bo'lib, bu holda u bitta **Index** usuliga ega bo'lib, u **View** usuli yordamida ko'rinishga kiradi.

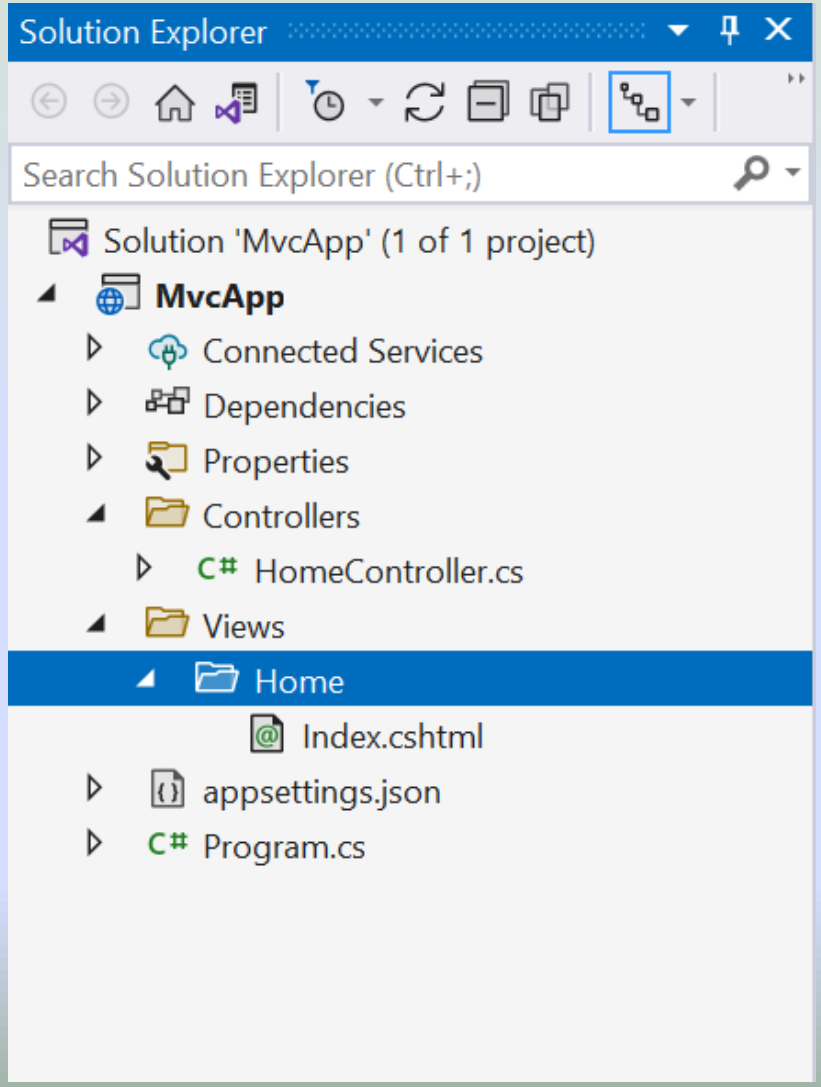
Ko'rinish(View) qo'shish

Endi loyihaga ko'rinishlarni saqlaydigan **Views** papkasini qo'shamiz. Ushbu papkada biz birinchi navbatda **Home** katalogini qo'shamiz - bu to'g'ridan-to'g'ri **Home** Nazoratchisining ko'rinishlari uchun mo'ljallangan katalog. Nihoyat, **Views/Home** katalogiga yangi **Razor View (Empty)** element qo'shamiz, biz uni **Index.cshtml** deb nomlaymiz.

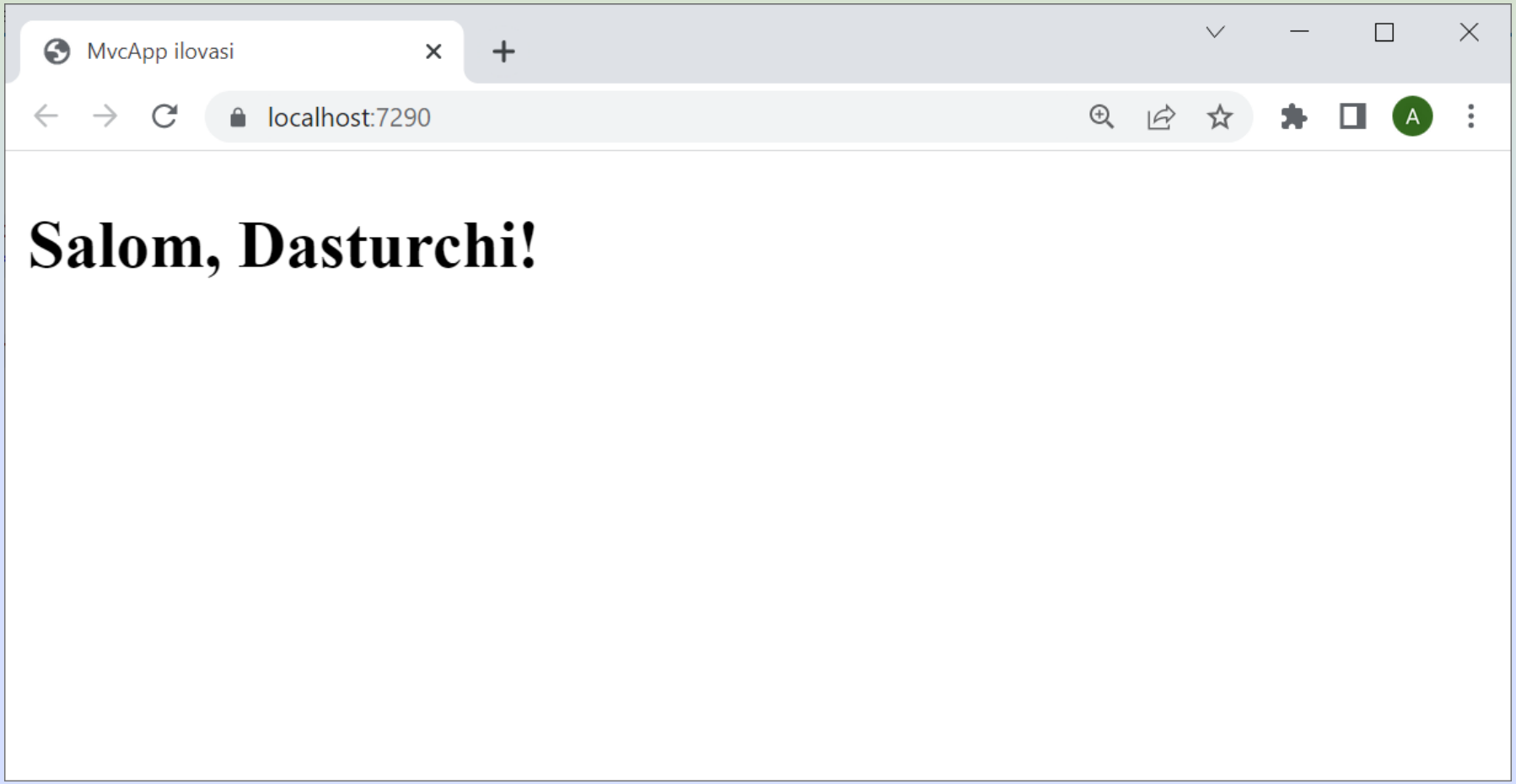
Ushbu faylda quyidagi kodni kiritamiz:

```
@{
    Layout = null;
}
<!doctype html>
<html>
<head>
    <title>MvcApp ilovasi</title>
    <meta charset="utf-8" />
</head>
<body>
    <h2>Salom, Dasturchi!</h2>
</body>
</html>
```

Bu HomeController ning Index usuli tomonidan qo'llaniladigan eng oddiy ko'rinishdir. Ya'ni, natijada biz quyidagi loyiha tuzilmasini olamiz:



Endi dasturni ishga tushiramiz va brauzer **Index.cshtml** ko'rinishida belgilangan xabarni ko'rsatadi:



Shunday qilib, biz loyihamizga MVC funksiyasini qo'shishimiz mumkin. Biroq, MVC odatda standart bo'yicha ba'zi asosiy funktsiyalarni o'z ichiga olgan **ASP.NET Core Web App (Model-View-Controller)** modelidan foydalanadi.

MVC xizmatlari

MVC funksionalligi va uning ilovada qanday ishlashi siz qo'shadigan xizmatlarga bog'liq. Yuqoridagi misolda marshrutlash tizimi so'rovni kontroller bilan bog'lashi uchun MVC xizmatlarini qo'shish uchun **AddMvc**() usulidan foydalanganmiz. Biroq, bu holda, bizda kerak bo'lganda foydalanishimiz mumkin bo'lgan joylashtirish xizmatlarining bir qator variantlari mavjud:

- **AddMvc**(): Barcha MVC strukturasi xizmatlarini (shu jumladan autentifikatsiya va avtorizatsiya, tekshirish va h.k. bilan ishlash xizmatlarini) qo'shadi.
- **AddMvcCore**(): faqat MVC strukturasining asosiy xizmatlarini qo'shadi va autentifikatsiya va avtorizatsiya, tekshirish va boshqalar kabi barcha qo'shimcha funktsiyalar mustaqil ravishda qo'shilishi kerak.
- **AddControllersWithViews**(): Faqat kontrollerlar, ko'rinishlar va tegishli funksiyalarga ruxsat beruvchi MVC struktura xizmatlarini qo'shadi. ASP.NET Core Web App (Model-View-Controller) turidagi loyihani yaratishda ushbu usuldan foydalaniladi.
- **AddControllers**(): kontrollerlardan foydalanishga imkon beradi, lekin ko'rinishlarsiz.

MAVZU: Server tomonidan holatni boshqarish. MVC texnologiyasi

Maqsad: Talabalar Server tomonidan holatni boshqarishni va MVC texnologiyasini o’rganish.

REJA:

1. Nazoratchilar va ularning harakatlari
2. Fayllarni yuborish
3. Ko'rinishlarga kirish

ASP.NET Core MVC arxitekturasining asosiy elementi **controller(nazoratchi)** hisoblanadi. So'rov qabul qilinganda, marshrutlash tizimi so'rovni qayta ishlash uchun mos boshqaruvchini tanlaydi va so'rov ma'lumotlarini unga uzatadi. Tekshiruvchi bu ma'lumotlarni qayta ishlaydi va qayta ishlash natijasini qaytarib yuboradi.

Masalan, ASP.NET Core Web App (Model-View-Controller) shablonidan foydalangan holda yaratilgan loyiha odatiy bo'yicha kamida bitta kontrollerni o'z ichiga oladi, **HomeController** sinfi quyidagi kodga ega:

```
using Microsoft.AspNetCore.Mvc;
using SalomMVCApp.Models;
using System.Diagnostics;
namespace SalomMVCApp.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }
        public IActionResult Index()
        {
            return View();
        }
        public IActionResult Privacy()
        {
            return View();
        }
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
        {
return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier
});
        }
    }
}
```

Bunday holda, biz tekshirgichda konstruktorga ega ekanligini ko'ramiz, u orqali **logging** uchun ishlatiladigan **ILogger** xizmati bog'liqlik kiritish mexanizmi orqali uzatiladi. Nazoratchi shuningdek, uchta usulni belgilaydi - Index, Privacy va Error.

Nazoratchini qo'shish

Tekshirgichlardan foydalanishda ba'zi konvensiyalar mavjud. Birinchidan, nazoratchilar, odatda, loyihadagi **Controllers** katalogiga joylashtiriladi. Biroq, bu printsipial jihatdan ixtiyoriy - siz boshqa papkalarga yoki hatto loyiha ildiziga nazoratchilarni qo'shishingiz mumkin. Biroq, avval loyihaga yangi **Controllers** papkasini qo'shamiz.

ASP.NET Core MVC da kontroller oddiy C# sinf bo'lib, u odatda Microsoft.AspNetCore.Mvc.Controller mavhum tayanch sinfidan meros bo'lib, har qanday C# sinfi kabi maydonlar, xususiyatlar va usullarga ega bo'lishi mumkin.

Nomlash qoidalariga ko'ra, nazoratchi nomlari odatda "Controller" qo'shimchasi bilan tugaydi va shu qo'shimchaga qadar qolganlari **HomeController** kabi boshqaruvchining nomi hisoblanadi. Lekin printsipial jihatdan, bu konvensiyalar ham ixtiyoriydir.

Ammo nazoratchilarga nisbatan qo'llaniladigan majburiy konvensiyalar ham mavjud. Xususan, boshqaruvchi sinfi quyidagi shartlardan kamida bittasiga javob berishi kerak:

- Controller sinfida "Controller" qo'shimchasi mavjud:

```
public class HomeController
  {
     //.............
  }
```

- Controller klassi "Controller" qo'shimchasiga ega bo'lgan sinfdan meros bo'lib o'tadi:

```
public class Home : Controller
  {
     //.............
  }
```

- [Controller] atributi kontroller sinfiga qo'llaniladi:

```
[Controller]
public class Home
  {
     //.................
  }
```

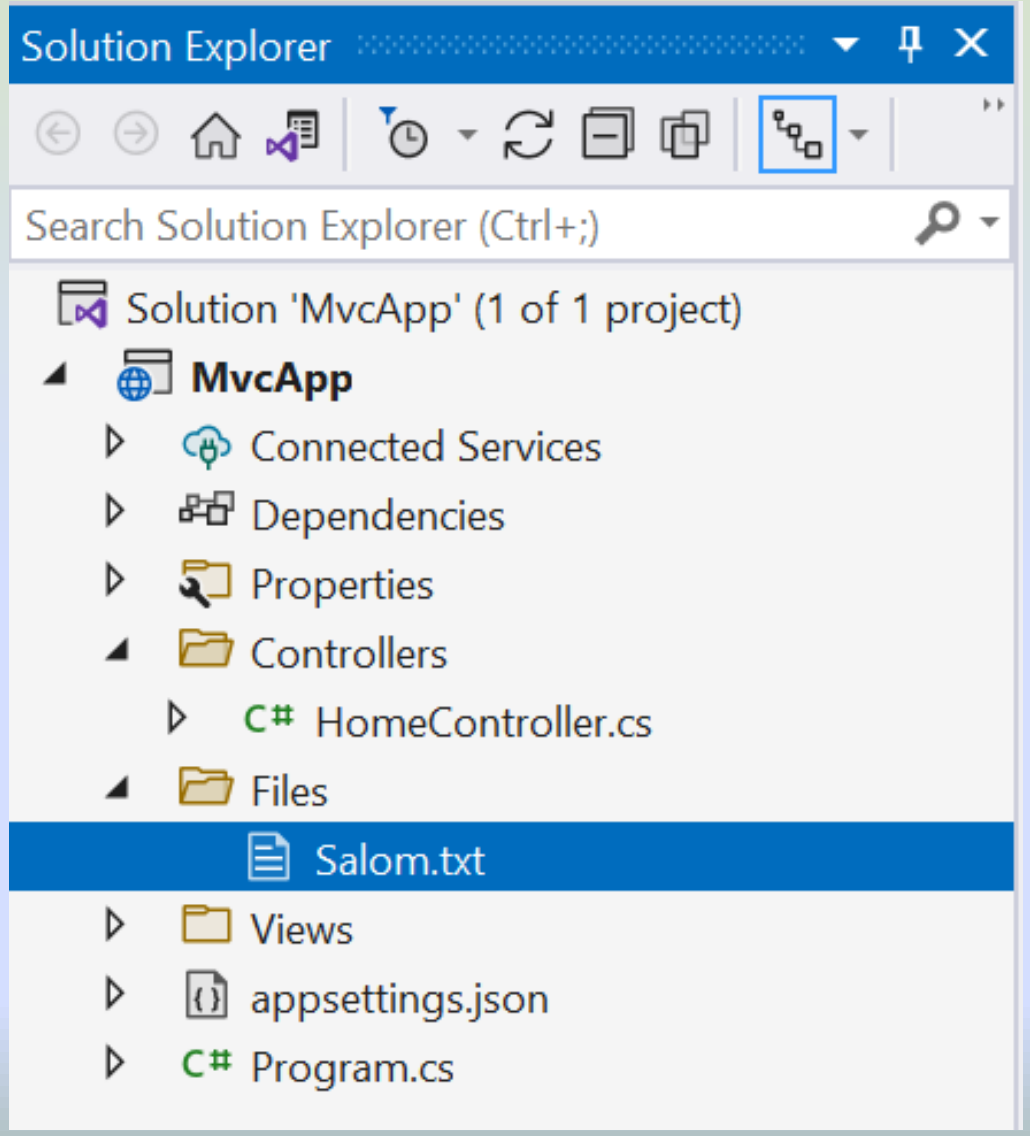
Fayllarni yuborish

Mijozga fayllarni yuborish uchun mavhum **FileResult** klassi mo'ljallangan, uning funksionalligi avlod sinflarida amalga oshiriladi:

- **FileContentResult**: fayldan o'qilgan baytlar massivini mijozga yuboradi
- **VirtualFileResult**: to'g'ridan-to'g'ri serverdan virtual yo'l orqali oddiy fayl yuklanishini ifodalaydi
- **FileStreamResult**: oqim yaratadi - System.IO.Stream ob'ekti, u yordamida faylni o'qiydi va mijozga yuboradi.
- **PhysicalFileResult**: shuningdek, serverdan fayl yuboradi, lekin jo'natish uchun haqiqiy jismoniy yo'l ishlatiladi.

Dastlabki uchta holatda fayllarni jo'natish uchun **File()** usuli, PhysicalFileResult obyektini yaratish uchun **PhysicalFile()** usuli qo'llaniladi.

Masalan, **Salom.txt** faylini o'z ichiga olgan loyiha ildiziga **Files** jildini qo'shamiz:



Salom.txt faylini mijozga yuborish uchun **PhysicalFileResult** dan foydalanamiz:

```
public IActionResult GetFile()
{
// Faylga yo'l
string file_path =
Path.Combine(AppDomain.CurrentDomain.BaseDi
rectory, "Files/Salom.txt");
// Fayl tipi - content-type
string file_type = "text/plain";
// Fayl nomi - ixtiyoriy
string file_name = "Salom.txt";
return PhysicalFile(file_path, file_type,
file_name);
}
```

Loyihaga nisbatan katalogning to‘liq jismoniy yo‘lini olish uchun joriy ilova joylashgan papkaga yo‘lni qaytaruvchi **AppDomain.CurrentDomain.BaseDirectory** xususiyatidan foydalaniladi.

Va, masalan, **Home/GetFile** yo'liga kirganingizda, bizdan ushbu faylni shaxsiy kompyuterda saqlashimiz so'raladi.

Program.cs fayliga esa quyidagi mashrutni kiritamiz:

```
app.MapControllerRoute(
    name: "GetFile",
    pattern:
"{controller=Home}/{action=GetFile}/
{id?}"
    );
```

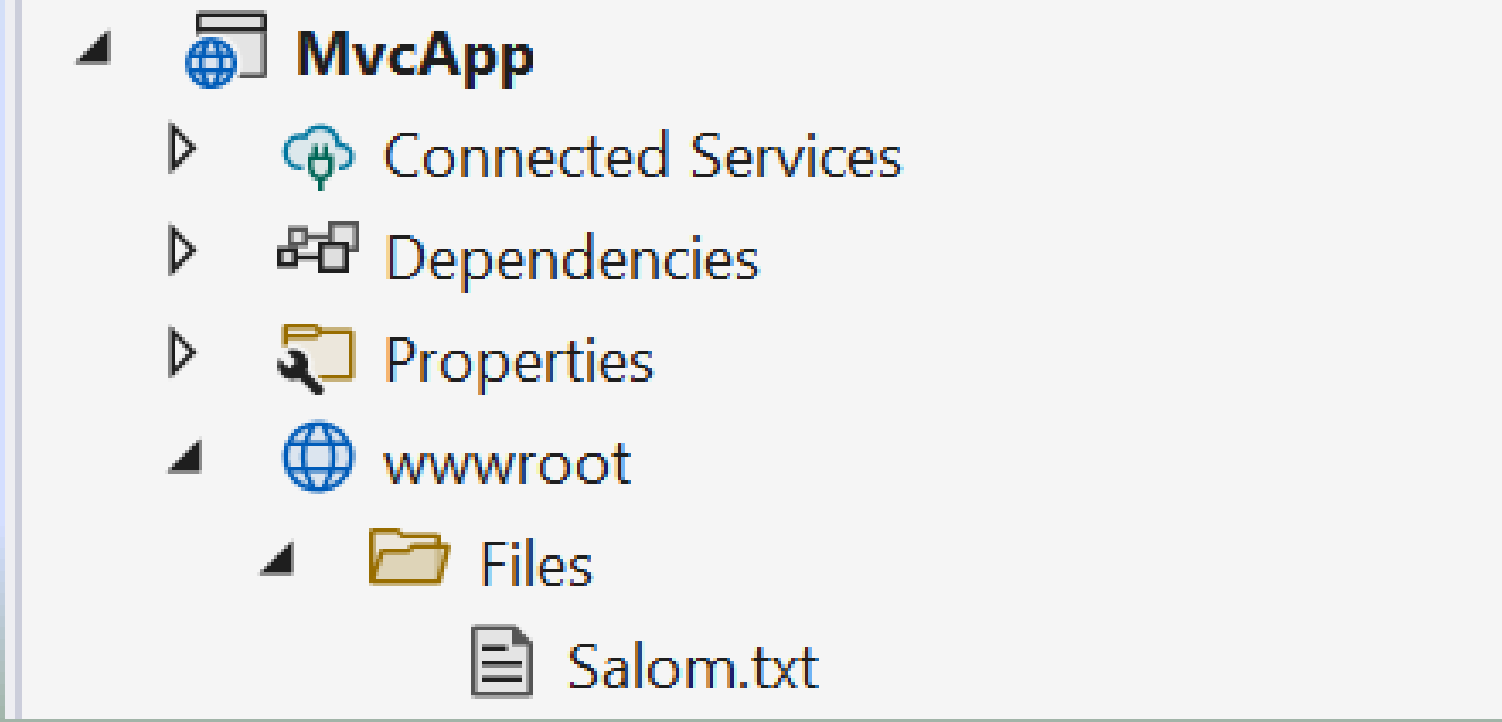
VirtualFileResult yordamida fayl jo’natish

VirtualFileResult ham xuddi yuqoridek ishlaydi, faqat virtual yo'ldan faylni qaytaradi. Bu yerda shuni hisobga olish kerakki, odatiy bo'yicha bu holda fayllarning barcha yo'llari **wwwroot** papkasi bilan taqqoslanadi. Ya'ni, biz **wwwroot** katalogiga fayllar yoki alohida fayllar bilan papkalarni joylashtirishimiz kerak:

```
public IActionResult GetVirtualFile() =>
File("Files/Salom.txt", "text/plain", “Salom4.txt");
    }
```

Bunday holda, "**Salom.txt**" fayli "**wwwroot/Files/**"

jildida joylashgan deb taxmin qilinadi:



Bog'liqliklarni nazoratchiga o'tkazish

Har qanday sinf singari, boshqaruvchi ham bog'liqlikni kiritish mexanizmi orqali dastur xizmatlarini olishi mumkin. Tekshirish moslamasida buni quyidagi usullar bilan amalga oshirish mumkin:

- Konstruktor orqali
- **FromServices** atributi qo'llaniladigan usul parametri orqali
- **HttpContext.RequestServices** xususiyati orqali

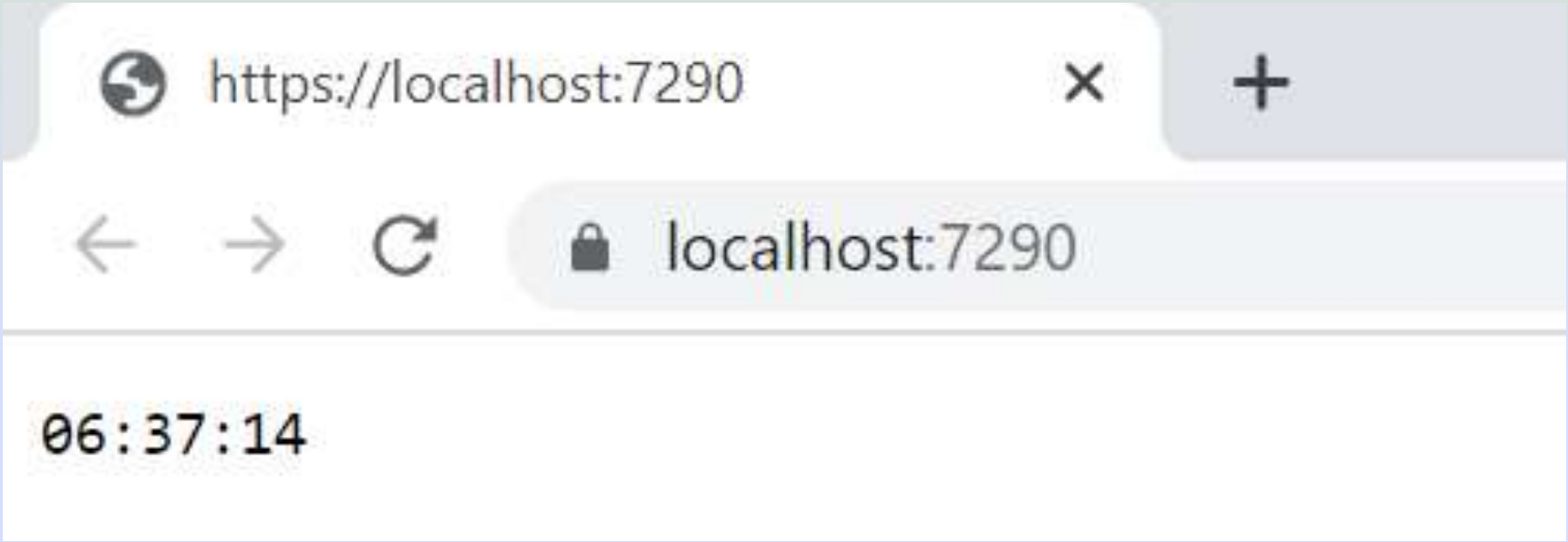
Masalan, bizda quyidagi **Program.cs** fayli bor deylik:

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();  // nazoratchi xizmatini qo'shish
builder.Services.AddTransient<ITimeService,
SimpleTimeService>(); // ITimeService xizmatini qo'shish
var app = builder.Build();
// nazoratchilarga marshrutlarni yuklash
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.Run();
public interface ITimeService
{
    string Time { get; }
}
public class SimpleTimeService : ITimeService
{
    public string Time => DateTime.Now.ToString("hh:mm:ss");
}
```

Bunday holda, **ITimeService** interfeysi va uning **SimpleTimeService** amalga oshirilishi aniqlanadi. Va ilovada **ITimeService** xizmati ro'yxatdan o'tgan bo’ladi: Masalan, nazoratchi konstruktoridagi bog’liqlikni olamiz:

```
public class HomeController : Controller
{
    public string Index([FromServices]
ITimeService timeService)
    {
    return timeService.Time;
    }
}
```

Ilovani ishga tushursak, natija quyidagicha bo’ladi:



Ko'rinishlarga kirish
ViewResult va ko’rinish yo’li

Ko'rinishni veb-sahifaga aylantiradigan va uni mijozga javob sifatida qaytaradigan **ViewResult** ob'ektini qaytarish uchun **View**() usulidan foydalaniladi.

Misol uchun, yuqoridagi misolda ushbu usulning hech qanday parametrga ega bo'lmagan versiyasi ishlatilgan:

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
```

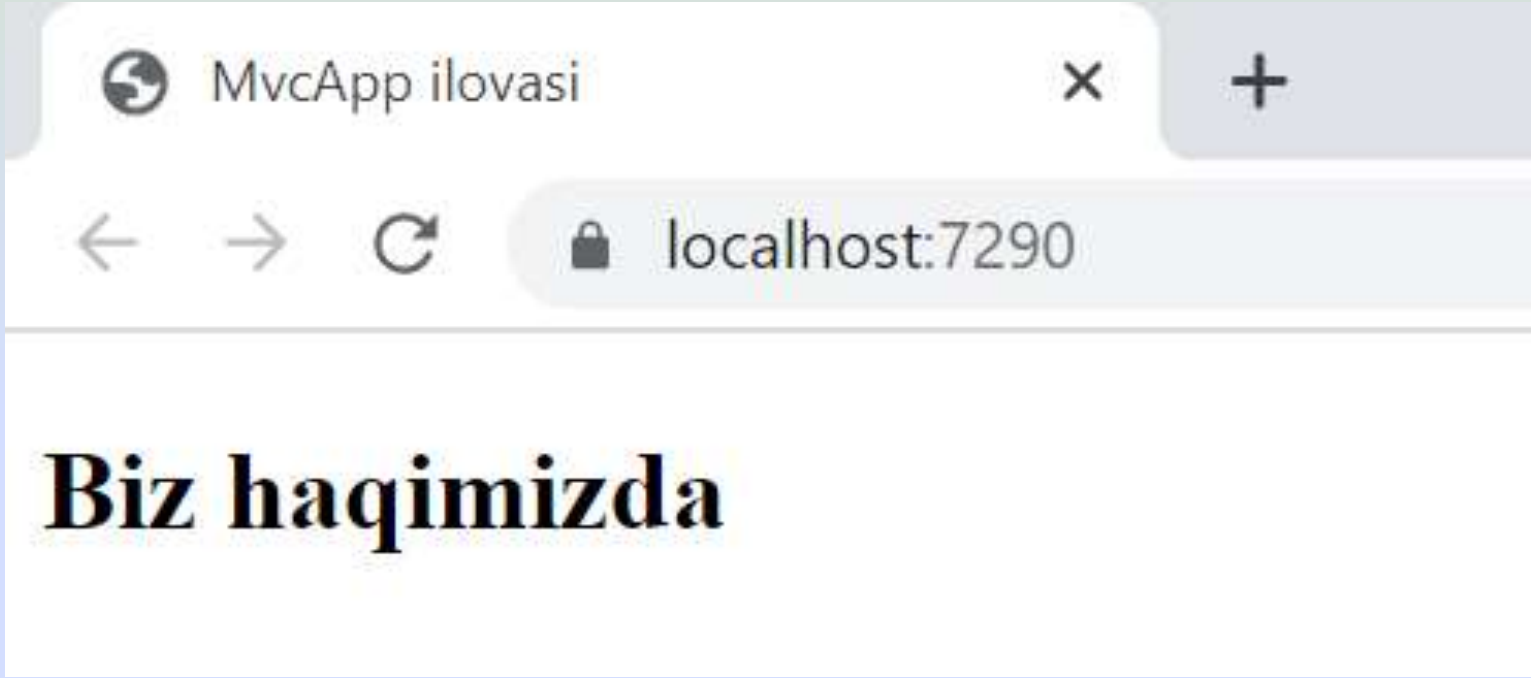
Umuman olganda, View() usuli to'rtta haddan tashqari yuklangan versiyaga ega:

- **View():** ko'rinish chaqiruv usuli bilan bir xil nomga ega bo'lgan javobni yaratish uchun ishlatiladi
- **View(string? viewName):** Koʻrinish nomi standart koʻrinishni bekor qilish imkonini beruvchi usulga oʻtkaziladi.
- **View(object? model):** model ob'ekti sifatida ko'rinishga ma'lumotlarni uzatadi
- **View(string? viewName, object? model):** ko'rinish nomini bekor qiladi va unga model ob'ekti sifatida ma'lumotlarni uzatadi

Usulning ikkinchi versiyasi ishlatilgan ko'rinishni bekor qilishga imkon beradi. Agar ko'rinish ushbu kontroller uchun mo'ljallangan papkada joylashgan bo'lsa, u holda ko'rinish nomini **View()** usuliga kengaytmasiz o'tkazish kifoya:

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View("About");
        }

    }
```

Ilovani ishga tushursak, natija quyidagicha bo’ladi:



Bunday holda, Index usuli Views/Home/About.cshtml ko'rinishidan foydalanadi. Agar ko'rinish boshqa papkada joylashgan bo'lsa, biz ko'rinishga to'liq yo'lni o'tkazishimiz kerak:

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
return View("~/Views/Home/About.cshtml");
        }

    }
```

Bosh sahifa Layout

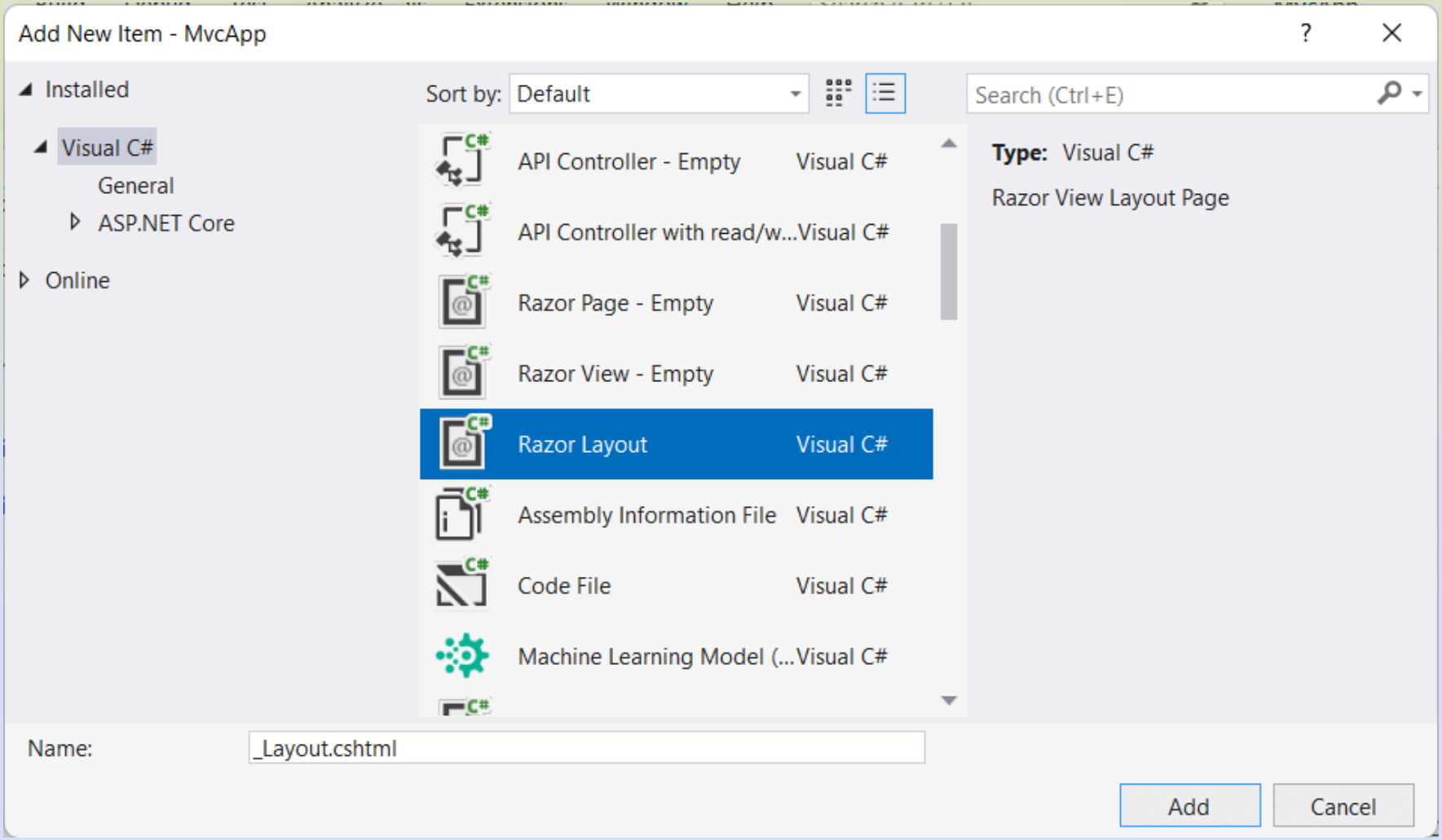
Asosiy sahifalar yoki Layoutlar ko'rinishlar uchun yagona shablonni o'rnatish imkonini beradi va sayt uchun yagona ko'rinishni yaratish uchun ishlatiladi.

Asosan, asosiy sahifalar bir xil ko'rinishlar bo'lib, ular boshqa ko'rinishlarni ham o'z ichiga olishi mumkin.

Misol uchun, siz asosiy sahifadagi barcha boshqa ko'rinishlar uchun umumiy bo'lgan menyularni belgilashingiz mumkin, shuningdek, u umumiy uslublar va skriptlarni o'z ichiga oladi.

Natijada, biz har bir alohida ko'rinishda uslublar fayllariga yo'lni yozishimiz shart emas va agar kerak bo'lsa, uni o'zgartiramiz. Va maxsus teglar sizga asosiy sahifalarning ma'lum bir joyiga boshqa ko'rinishlarni kiritish imkonini beradi.

Birinchi navbatda, layout-asosiy sahifasini aniqlaymiz. Keling, loyihada **Views** papkasini yarataylik va unda **Shared** papkasini yarataylik. Keyinchalik, **Views/Shared** katalogiga yangi ko'rinish qo'shamiz, uni biz **_Layout.cshtml** deb nomlaymiz. **Visual Studio'da** asosiy sahifalarni qo'shish uchun **Razor Layout** fayl shablonidan foydalanishingiz mumkin:



_Layout.cshtml faylini qo'shgandan so'ng, uning kodini quyidagicha o'zgartiramiz:

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-
width" />
    <title>Bizning saytimiz |
@ViewBag.Title</title>
</head>
<body>
     <div><a href="/Home/Index">Home</a> | <a
href="/Home/About">About</a></div>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

Index.cshtml fayliga quyidagi kodni kiritamiz:

```
@{
    ViewBag.Title = "Index";
    Layout =
"/Views/Shared/_Layout.cshtml";
}
```

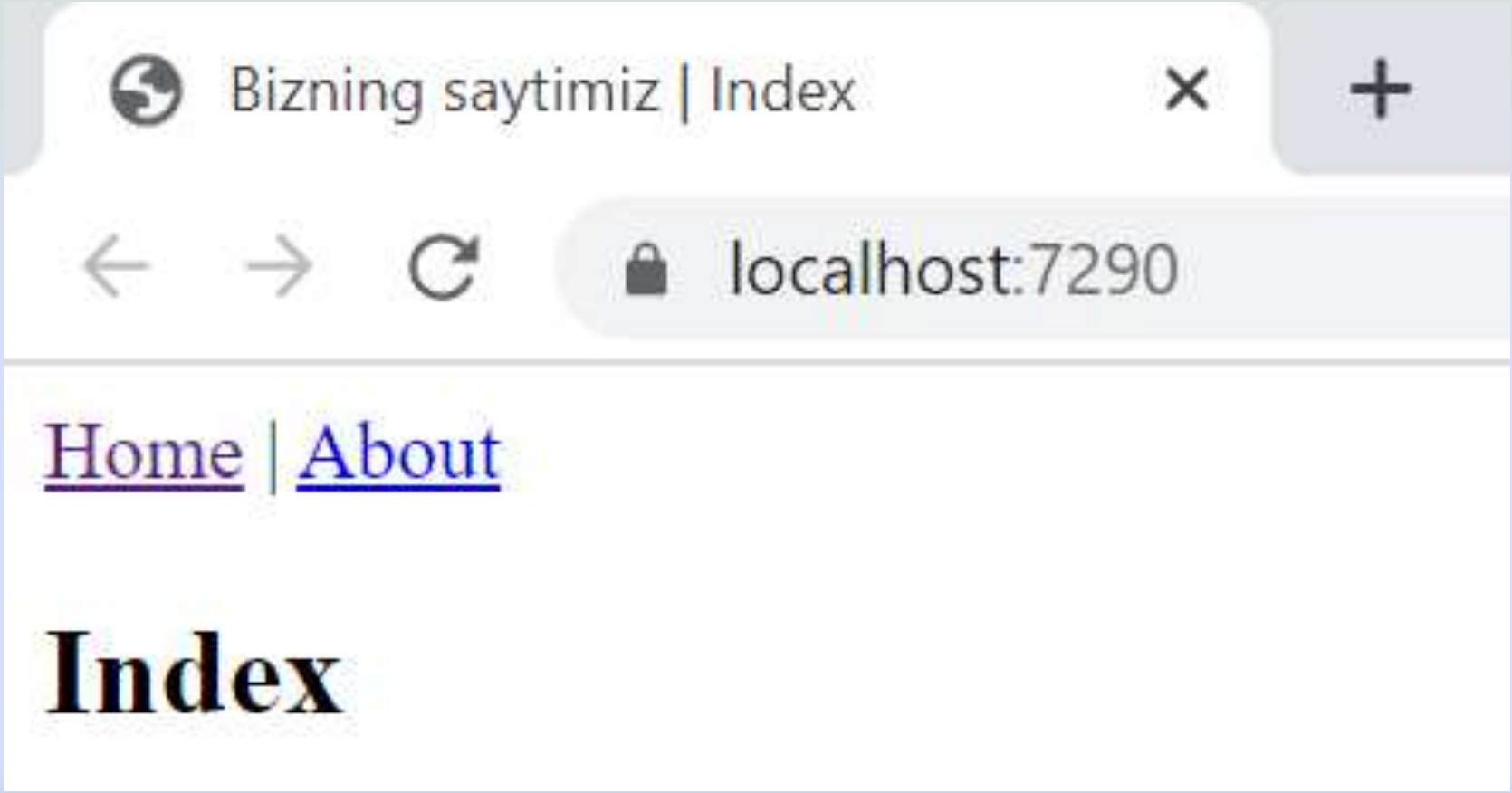
About.cshtml fayliga ham quyidagi kodni kiritamiz:

```
@{
    ViewBag.Title = “About";
    Layout =
"/Views/Shared/_Layout.cshtml";
}
```

HomeController.cs faylini quyidagicha o’zgartiramiz:

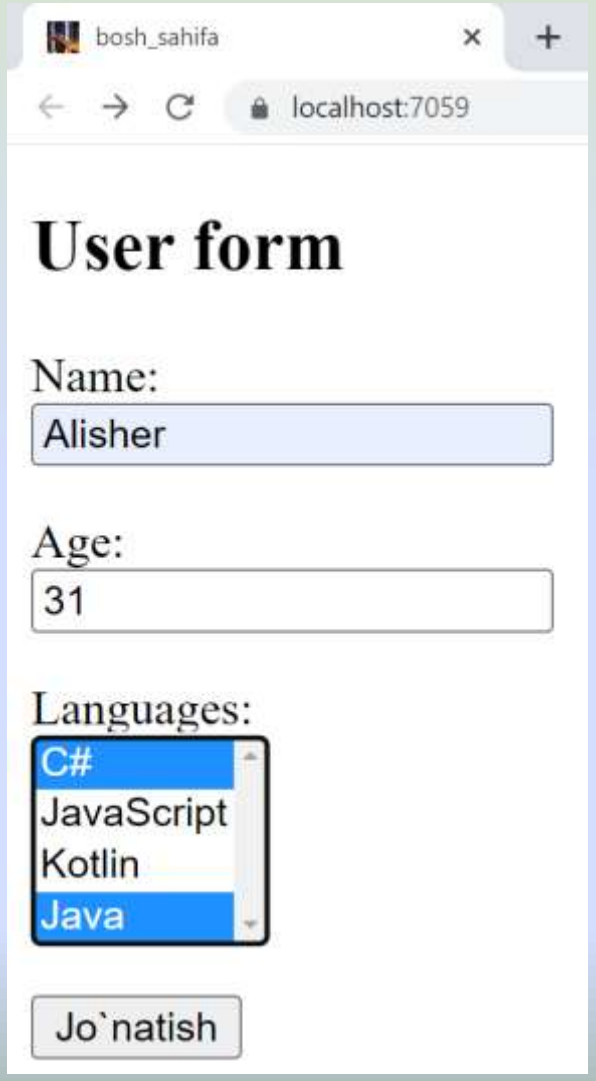
```
public class HomeController : Controller
    {
public IActionResult Index() => View();

public IActionResult About() => View();
    }
```

Ilovani ishga tushursak, natija quyidagicha bo’ladi:



```
<form method="post" action="postuser">
    <p>   Name: <br />
        <input name="name" />
    </p>
    <p>   Age: <br />
        <input name="age" type="number" />
    </p>
    <p>   Languages:<br />
        <select multiple name="languages">
            <option>C#</option>
            <option>JavaScript</option>
            <option>Kotlin</option>
            <option>Java</option>
        </select>
    </p>
    <input type="submit" value="Jo'natish'" />
</form>
```

Dasturni ishga tushirganimizda natija quyidagicha bo’ladi:



json faylni yuborish va qabul qilish

JSON ma'lumotlarni uzatish uchun keng tarqalgan formatdir. json ma'lumotlarini qanday yuborishimiz va qabul qilishimiz mumkinligini ko'rib chiqaylik.

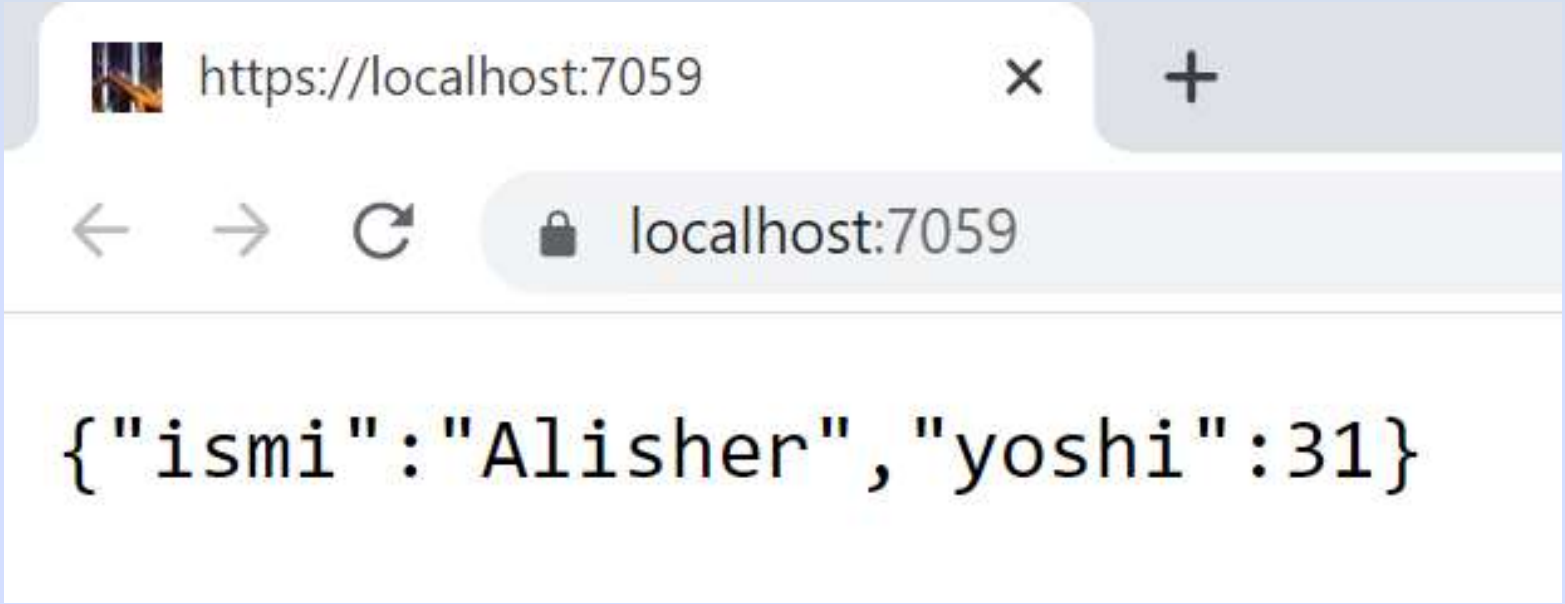
JSON yuborish

WriteAsJsonAsync usuli Json yuborish uchun **HttpResponse** obyektining **WriteAsJson**()/**WriteAsJsonAsync**() usulidan foydalanishingiz mumkin.

Bu usul sizga JSON formatiga o'tkazilgan ob'ektlarni ketma-ketlashtirish imkonini beradi va "**content-type**" sarlavhasini avtomatik ravishda "**application/json**; **charset=utf-8**" ga o'rnatadi:

```
var builder =
WebApplication.CreateBuilder();
var app = builder.Build();
app.Run(async (context) =>
{
    Person inson = new("Alisher", 31);
    await
context.Response.WriteAsJsonAsync(inson);
});

app.Run();

public record Person(string Ismi, int
yoshi);
```

Bunday holda, mijozga yozuvlar sinfini ifodalovchi **Person** tipidagi ob'ekt yuboriladi, lekin u oddiy sinf ham bo'lishi mumkin:



Standart **WriteAsync**() usulidan ham foydalanishingiz mumkin:

```
var builder =
WebApplication.CreateBuilder();
var app = builder.Build();
app.Run(async (context) =>
{
    var response = context.Response;
    response.Headers.ContentType =
"application/json; charset=utf-8";
    await
response.WriteAsync("{'ismi':'Alisher',
'yoshi':37}");
});
```

JSON qabul qilish. ReadFromJsonAsync usuli

So'rovdan JSON ob'ektini olish uchun **HttpRequest** klassi **ReadFromJsonAsync**() usulini belgilaydi. Bu sizga ma'lumotlarni ma'lum turdagi ob'ektga ketma-ketlashtirish imkonini beradi.

Masalan, loyihada **html** papkasini yaratamiz, unda biz yangi **bosh_sahifa.html** faylini aniqlaymiz.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>bosh_sahifa</title>
</head>
<body>
    <h2>User form</h2>
    <div id="message"></div>
    <form>
<p> Name: <br />
<input name="userName" id="userName" />
</p>
<p> Age: <br />
<input name="userAge" id="userAge" type="number" />
</p>
<button id="sendBtn">Jo`natish</button>
</form>
```

```
<script>
document.getElementById("sendBtn").addEventListener("click"
, send);
async function send(e) {
e.preventDefault();
const response = await fetch("/api/user", {
method: "POST",
headers: { "Accept": "application/json", "Content-Type":
"application/json" },
body: JSON.stringify({
name: document.getElementById("userName").value,
age: document.getElementById("userAge").value
})
});
const message = await response.json();
document.getElementById("message").innerText =
message.text;
}
</script>
</body>
</html>
```

Bu yerda **fetch**() funksiyasidan foydalangan holda tugmani bosgandan so'ng, nomi va yoshiga ega ob'ekt **"/api/user"** manziliga yuboriladi, uning qiymatlari forma maydonlaridan olinadi. Serverdan javob sifatida veb-sahifa json formatidagi ob'ektni ham oladi, u matn xususiyatiga ega - serverdan xabarni saqlaydigan xususiyatga ega.

Endi, **Program.cs** faylida veb-sahifa tomonidan yuborilgan ma'lumotlarni olish uchun kodni aniqlaymiz:

```
var builder = WebApplication.CreateBuilder();
var app = builder.Build();
app.Run(async (context) =>
{
    var response = context.Response;
    var request = context.Request;
    if (request.Path == "/api/user")
    {
var responseText = “Aniqlanmagan ma'lumotlar";   // standart xabar mazmuni
        if (request.HasJsonContentType())
        {
            var person = await request.ReadFromJsonAsync<Person>();
            if (person != null)
                responseText = $"Name: {person.Name}  Age: {person.Age}";
        }
        await response.WriteAsJsonAsync(new { text = responseText });
    }
    else
    {
        response.ContentType = "text/html; charset=utf-8";
        await response.SendFileAsync("html/bosh_sahifa.html");
    }
});
app.Run();
public record Person(string Name, int Age);
```

Dastur natijasi quyidagicha:

Dasturning ismi va yoshini qayta o’zgartirsak va Jo’natish tugmasini bossak, ma’lumot qayta yuklanadi:

MAVZU: MVC texnologiyasi

Maqsad: Talabalar MVC texnologiyasini o’rganish.

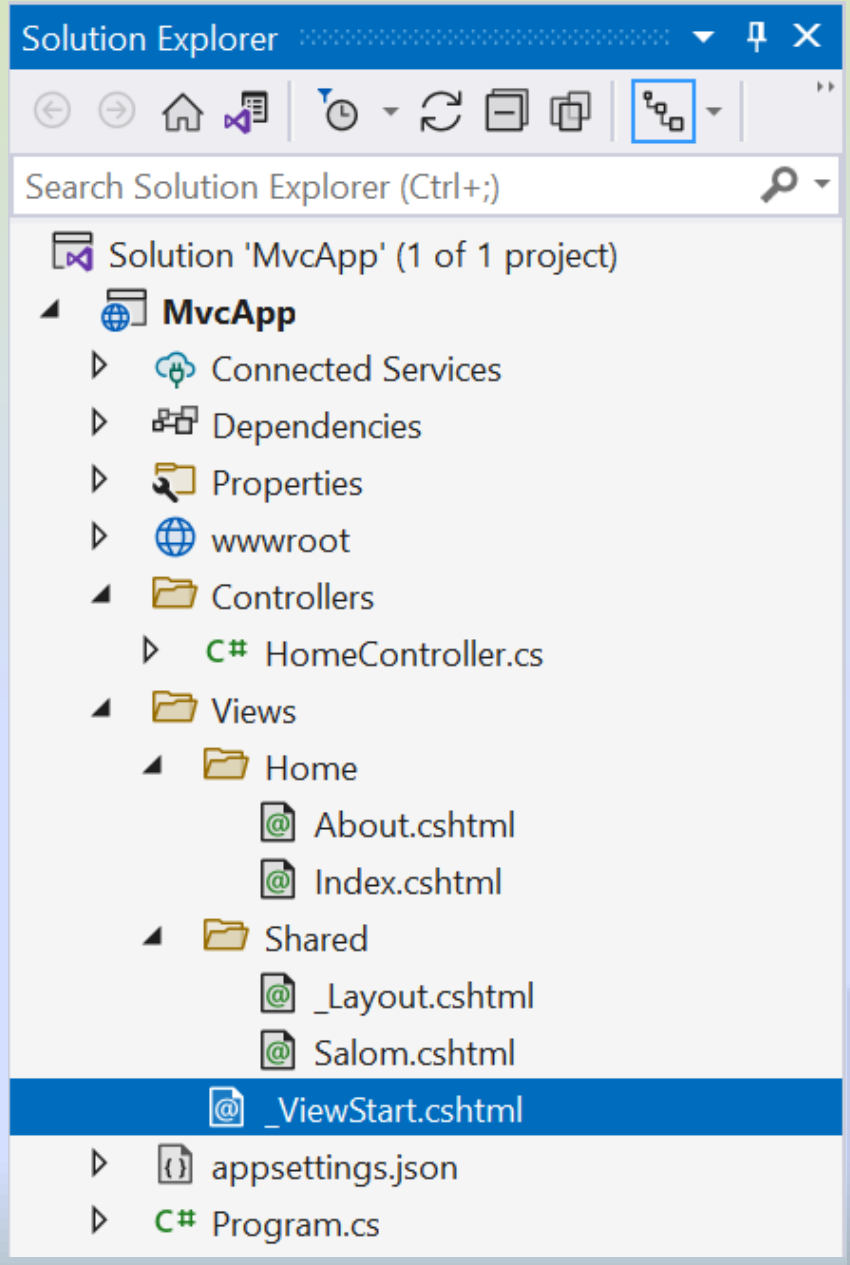
REJA:

1. ViewStart faylidan foydalanish
2. Asosiy sahifani bekor qilish
3. Footer bo'limi

ViewStart

Oldingi darsimizdagi **_Layout.cshtml** sahifasini bog’lash kod juda yaxshi ishlayotgan bo'lsa-da, bizda bitta muammo bor - biz har bir ko'rinishda qaysi bosh sahifa ko'rinishi qo'llanilishini aniq ko'rsatish zarurligiga duch kelamiz. Ushbu amalni soddalashtirish uchun **_ViewStart.cshtml** fayllaridan foydalanishingiz mumkin.

Shunday qilib, **Views** papkasiga **_ViewStart.cshtml** deb nomlangan yangi ko'rinish qo'shamiz.



Ushbu faylning kodi ishga tushirilganda ko'rishlar kodining eng boshiga qo'shiladi. Shu bilan birga, **_ViewStart.cshtml** qo'llaniladigan ko'rish fayllari ushbu fayl bilan bir xil katalogda joylashgan bo'lishi kerak.

_ViewStart.cshtml faylida quyidagi kodni aniqlaymiz:

```
@{
    Layout = "_Layout";
}
```

Har bir ko'rinishda asosiy sahifaga murojaatni saqlaydigan Razor sintaksisidan foydalanilgan Layout xususiyati mavjud. Bu yerda **_Layout.cshtml** fayli bosh sahifa sifatida o'rnatiladi. Bunday holda, **kengaytmadan foydalanish mumkin emas**.

Ko'rinish berilganda, tizim **_Layout** asosiy sahifasini quyidagi yo'llar bilan qidiradi:

/Views/[Controller_nomi]/_Layout.cshtml

/Views/Shared/_Layout.cshtml

Agar ikkala yo’lda: /Views/[Controller_nomi] va /Views/Shared/ da bir xil nomdagi fayl bo'lsa, masalan, **_Layout.cshtml**, u holda u bilan bir xil papkada joylashgan fayl yuqoriroq ko'rinishga ustuvorlik qo'llaniladi.

Ya'ni, shu tarzda biz har bir alohida kontrollerning ko'rinishlari yoki bitta papkada joylashgan ko'rinishlar uchun alohida bosh sahifani belgilashimiz mumkin.

Ushbu fayl aniqlanganda, biz **Index.cshtml** va **About.cshtml** ko'rinishlaridan asosiy sahifa ulanishini olib tashlashimiz mumkin. Masalan, **Index.cshtml** ko'rinishi:

Ushbu fayl aniqlanganda, biz **Index.cshtml** va **About.cshtml** ko'rinishlaridan asosiy sahifa ulanishini olib tashlashimiz mumkin. Masalan, **Index.cshtml** ko'rinishini:

```
@{
ViewBag.Title = "Index";
//Layout = "/Views/Shared/_Layout.cshtml";
}
<h2>@ViewData["Title"]</h2>
```

Asosiy sahifani bekor qilish

Agar biz to'satdan butun loyiha uchun bosh sahifani global miqyosda boshqa papkada, masalan, Views katalogining ildizida joylashgan boshqa faylga o'zgartirmoqchi bo'lsak, _ViewStart.cshtml-da faylning to'liq yo'lidan foydalanishimiz kerak:

```
@{
Layout = "~/Views/_Layout.cshtml";
}
```

_ViewStart.cshtml-dagi kod ko'rinishdagi har qanday koddan oldin bajariladi. Va asosiy sahifani qayta belgilash uchun ko'rinishda **Layout** xususiyatini o'rnatish kifoya.

Tabiiyki, biz Layout xususiyatidan foydalanib, har bir alohida ko'rinishdagi bosh sahifani ham bekor qilishimiz mumkin.

```
@{
ViewBag.Title = “Bosh sahifa";
Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2> Index.cshtml ko'rinishi</h2>
```

Ko'rinishda **Layout** xususiyatiga **null** qiymatini o'rnatilish orqali biz asosiy sahifadan umuman foydalana olmaymiz:

```
@{
    Layout = null;
}
```

Bo'limlar

Ko'rinishlarning asosiy mazmunini kirituvchi **RenderBody**() usulidan tashqari, asosiy sahifalar bo'limlarni kiritish uchun maxsus **RenderSection()** usulidan ham foydalanishi mumkin. Asosiy sahifada ko'rishlar o'z mazmunini joylashtirishi mumkin bo'lgan bir nechta bo'limlar bo'lishi mumkin.

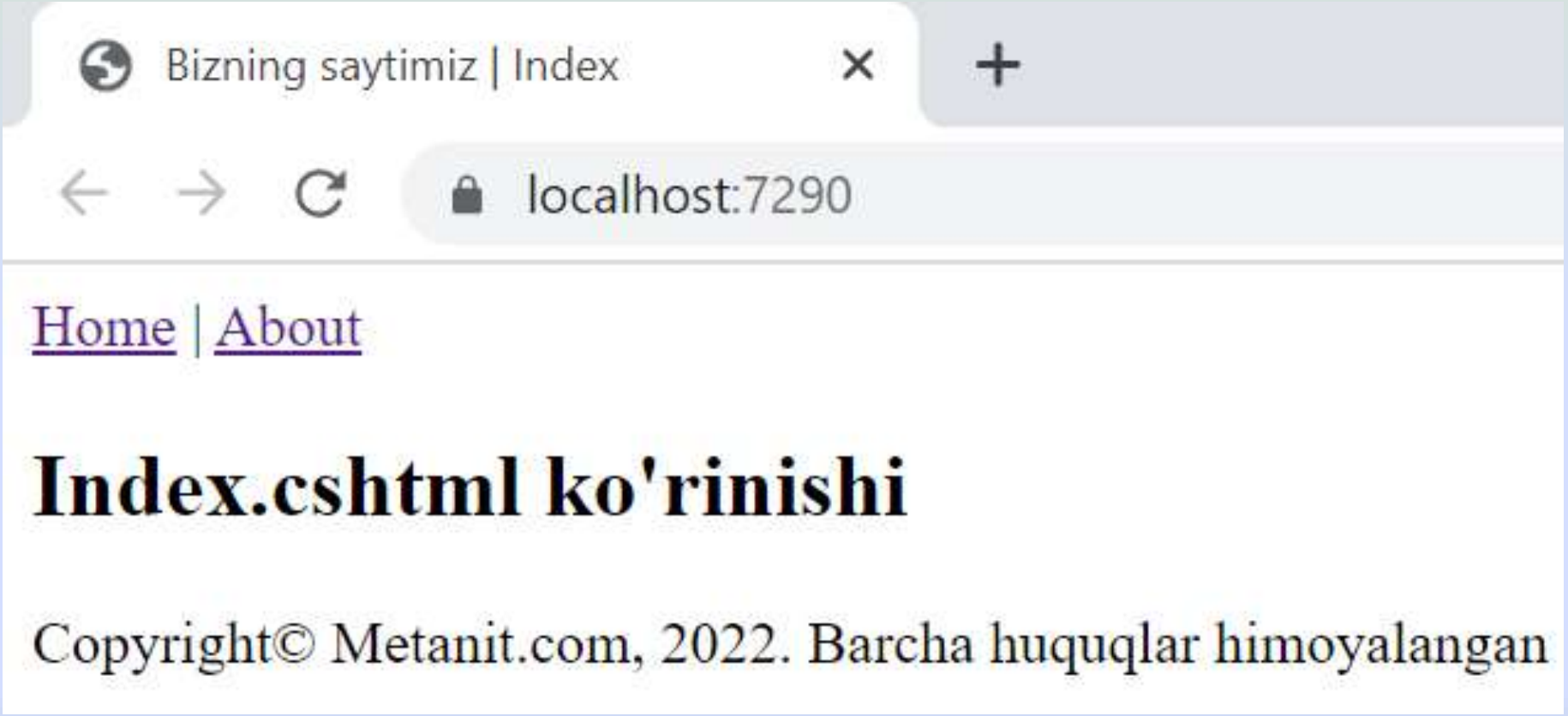
Misol uchun, **_Layout.cshtml** bosh sahifasiga “**footer**" deb nomlangan bo'limni qo'shamiz:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-
width" />
<title>Bizning saytimiz | @ViewBag.Title</title>
</head>
<body>
<div><a href="/Home/Index">Home</a> | <a
href="/Home/About">About</a></div>
    <div>
        @RenderBody()
    </div>
<footer>@RenderSection("Footer")</footer>
</body>
</html>
```

Endi, oldingi **Index** ko'rinishini ishga tushirganda, biz xatolikka duch kelamiz, chunki pastki **footer** bo'limi aniqlanmagan. Shunday qilib, **Index** ko'rinishining pastki qismiga **footer** bo'limini qo'shamiz. Buni @**section** ifodasi bilan amalga oshirishimiz mumkin:

```
@{
    ViewBag.Title = "Index";
}
@section Footer {
   Copyright© Metanit.com,
@DateTime.Now.Year. Barcha huquqlar
himoyalangan
}
```

Web sahifani ishga tushirsak, natijada quyidagicha ko’rinishga ega bo’lamiz:



Ammo bu yondashuv bilan, agar bizda ko'plab ko'rinishlar mavjud bo'lsa va biz to'satdan asosiy sahifada yangi bo'limni aniqlamoqchi bo'lsak, biz barcha mavjud ko'rinishlarni o'zgartirishimiz kerak bo'ladi, bu juda qulay emas. Bunday holda, biz moslashuvchan bo'lim sozlamalari uchun variantlardan birini ishlatishimiz mumkin.

Birinchi variant - RenderSection usulining haddan tashqari yuklangan versiyasidan foydalanish, bu sizga ushbu bo'limni ko'rinishda aniqlash kerak emasligini belgilash imkonini beradi. **Footer** bo'limini ixtiyoriy deb belgilash uchun usulga ikkinchi parametr sifatida "**false**" ni ko’rsatish:

```
<footer>@RenderSection("Footer", false)</footer>
```

Ikkinchi variant, agar ushbu bo'lim ko'rinishda aniqlanmagan bo'lsa, bo'limning standart tarkibini o'rnatishga imkon beradi:

```
<footer>
@if(IsSectionDefined("Footer"))
    {
        @RenderSection("Footer")
    }
    else
    {
<span>Footer elementining standart
tarkibi.</span>
    }
</footer>
```