Внедрение авторизации и аутентификации в Blazor Server.

Overview

Каждому ресурсу которым пользуются огромное количество пользователей необходим механизм авторизации и аутентификации. Данный механизм дает возможность идентифицировать пользователя и предоставлять именно тот контент на сайте, который ему положен.

Начнём с определений, чтобы не путать понятия Авторизация и Аутентификация.

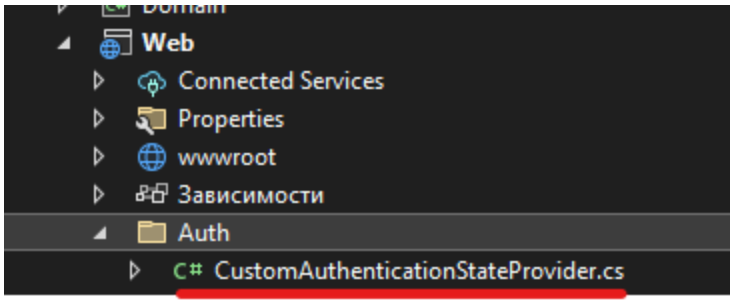
> 💡 **Аутентификация** — предоставление доказательств, что вы на самом деле есть тот, кем идентифицировались (от слова “authentic” — истинный, подлинный). Например при помощи комбинации Логин + Пароль.

> 💡 **Авториза́ция** (англ. authorization «разрешение; уполномочивание») — предоставление определенному лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Как раз для того, чтобы мы могли фиксировать заказы, которые совершают различные пользователи и идентифицировать каждого - нам необходимо реализовать данный механизм.

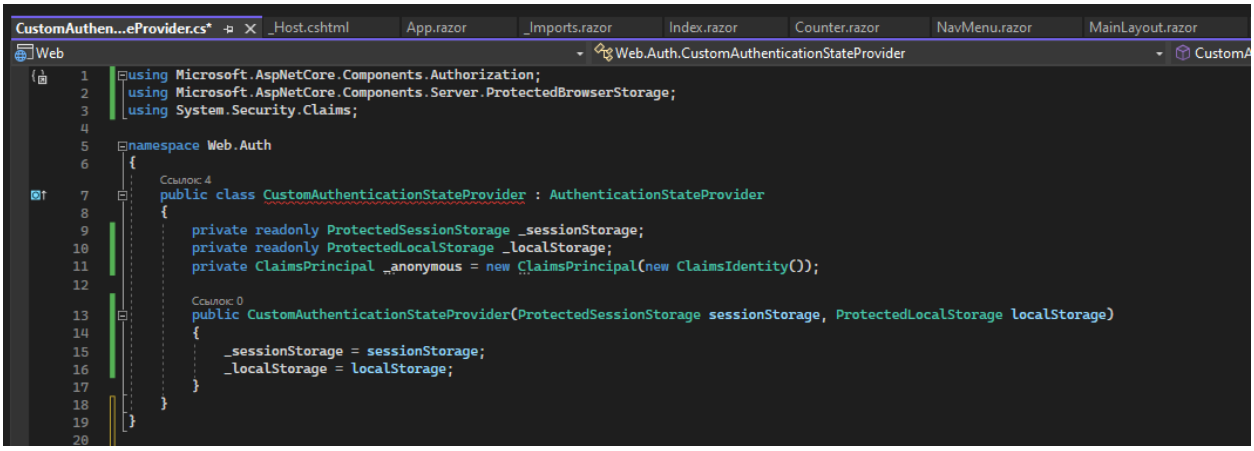
Создание механизма аутентификации

Откройте проект, который имеет проект Blazor Server, создайте папку Auth и внутри нее создайте класс CustomAuthenticationStateProvider

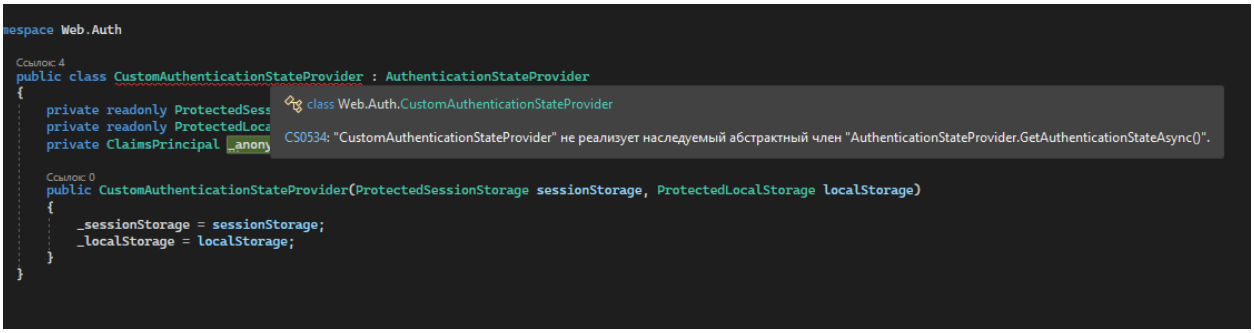


Перейдите в данный класс. Наследуйте его от класса AuthenticationStateProvider и создайте конструктор со следующими полями:

- ProtectedSessionStorage для того, чтобы работать с хранилищем данных сессии браузера
- ProtectedLocalStorage для того, чтобы работать с локальным хранилищем браузера



Теперь реализуем метод GetAuthenticationStateAsync



Данный метод реализуйте следующим образом:

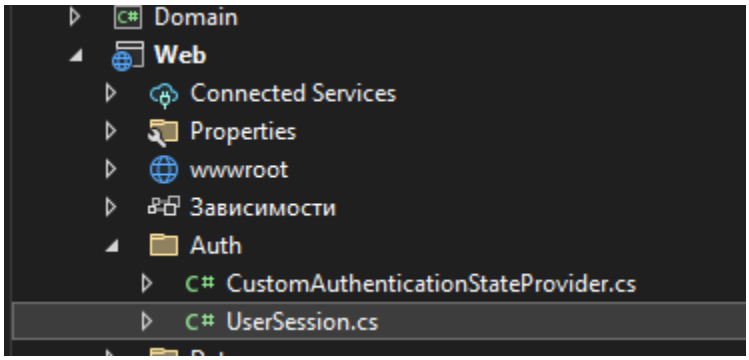
> Данные сессии извлекаем из браузера и возвращаем в качестве состояния аутентификации. В случае, если данных в браузере нет - возвращаем пустое состояние.

```
Ссылок: 0
public override async Task<AuthenticationState> GetAuthenticationStateAsync()
{
    try
    {
        var userSessionStorageResult = await _sessionStorage.GetAsync<UserSession>("UserSession");
        var userSession = userSessionStorageResult.Success ? userSessionStorageResult.Value : null;

        if (userSession == null)
        {
            return await Task.FromResult(new AuthenticationState(_anonymous));
        }

        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new List<Claim>
        {
            new Claim(ClaimTypes.Sid, userSession.Id),
            new Claim(ClaimTypes.Name, userSession.FirstName),
            new Claim(ClaimTypes.Email, userSession.Email),
            new Claim(ClaimTypes.Role, userSession.Role)
        }, "CustomAuth"));

        return await Task.FromResult(new AuthenticationState(claimsPrincipal));
    }
    catch
    {
        return await Task.FromResult(new AuthenticationState(_anonymous));
    }
}
```

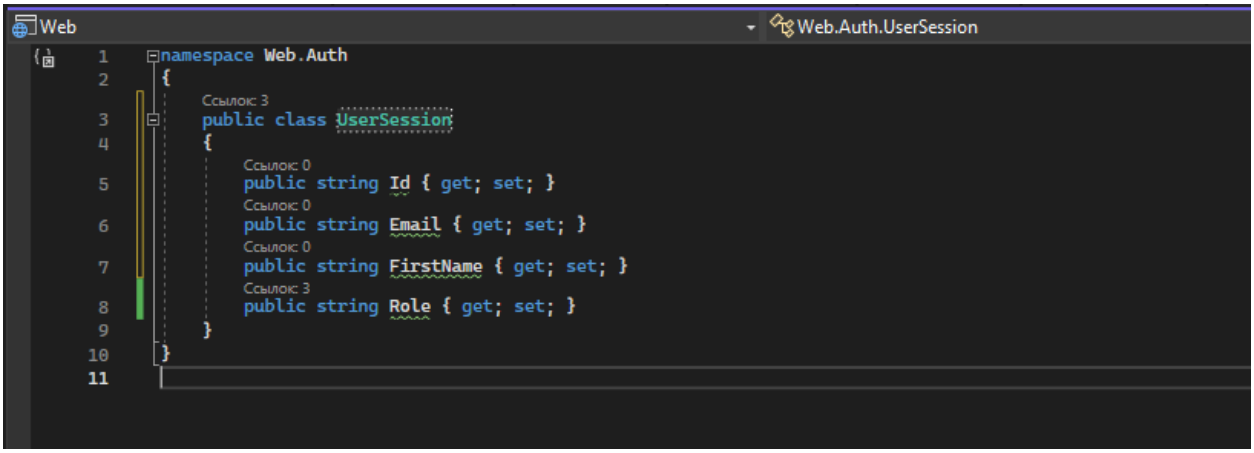
Создание класса, который будет описывать хранимую информацию о текущей сессии

Как вы уже могли заметить в коде имеется ссылка на несуществующий класс UserSession.

Данный класс будет отвечать за хранимую информацию о сессии, поэтому создадим данный класс в папке Auth



Пусть данный класс будет описан следующим образом:



Теперь создадим метод, который будет обновлять состояние аутентификации. Например при авторизации или выхода из системы.

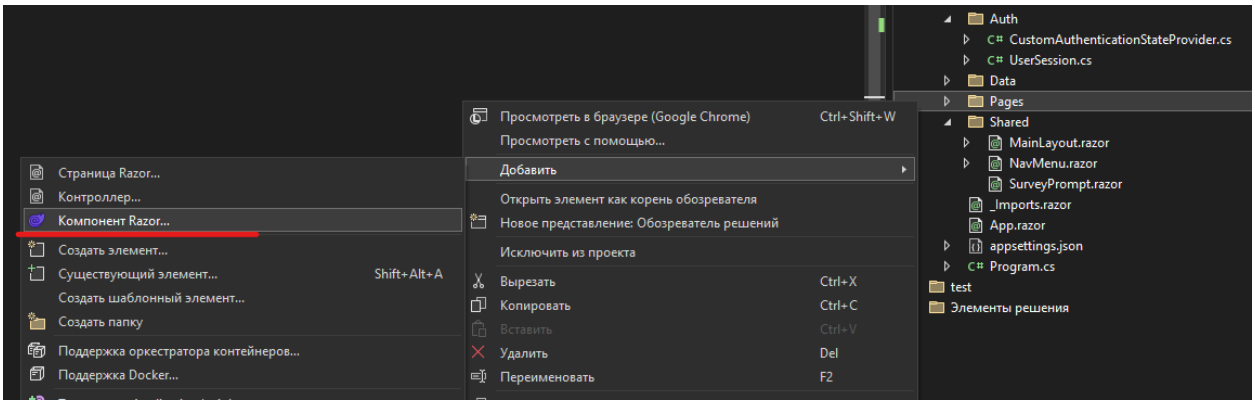
```
Ссылок: 2
public async Task UpdateAuthenticationStateAsync(UserSession userSession)
{
    ClaimsPrincipal claimsPrincipal;

    if (userSession != null)
    {
        await _sessionStorage.SetAsync("UserSession", userSession);
        claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new List<Claim>
        {
            new Claim(ClaimTypes.Sid, userSession.Id),
            new Claim(ClaimTypes.Name, userSession.FirstName),
            new Claim(ClaimTypes.Email, userSession.Email),
            new Claim(ClaimTypes.Role, userSession.Role)
        }));
    }
    else
    {
        await _sessionStorage.DeleteAsync("UserSession");
        claimsPrincipal = _anonymous;
    }

    NotifyAuthenticationStateChanged(Task.FromResult(new AuthenticationState(claimsPrincipal)));
}
```

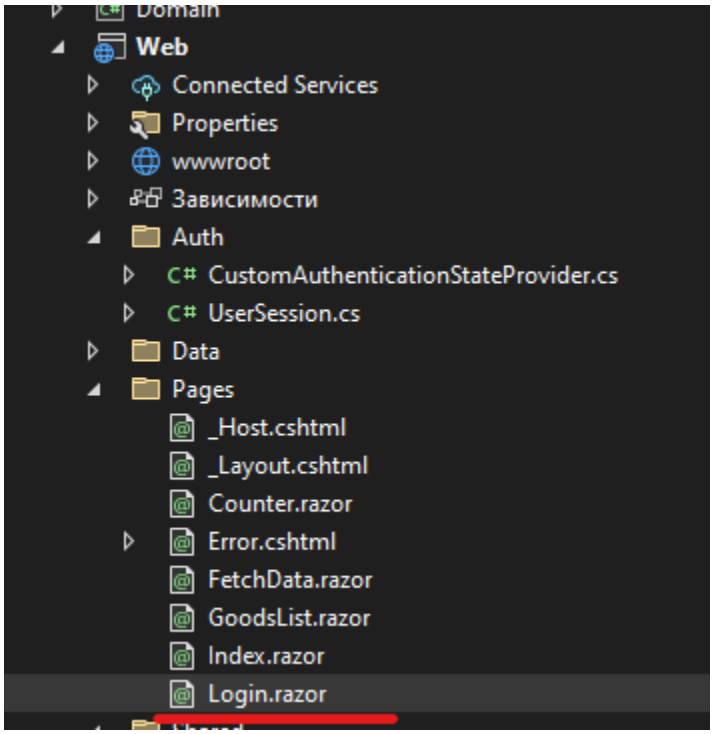
Теперь перейдите в файл Program.cs и добавьте данные строчки кода, которые будут включены в контейнер зависимостей.

```
// Add services to the container.
builder.Services.AddAuthenticationCore();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();

builder.Services.AddScoped<AuthenticationStateProvider, CustomAuthenticationStateProvider>();
builder.Services.AddScoped<ProtectedSessionStorage>();
builder.Services.AddScoped<ProtectedLocalStorage>();

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

Создание страницы для входа

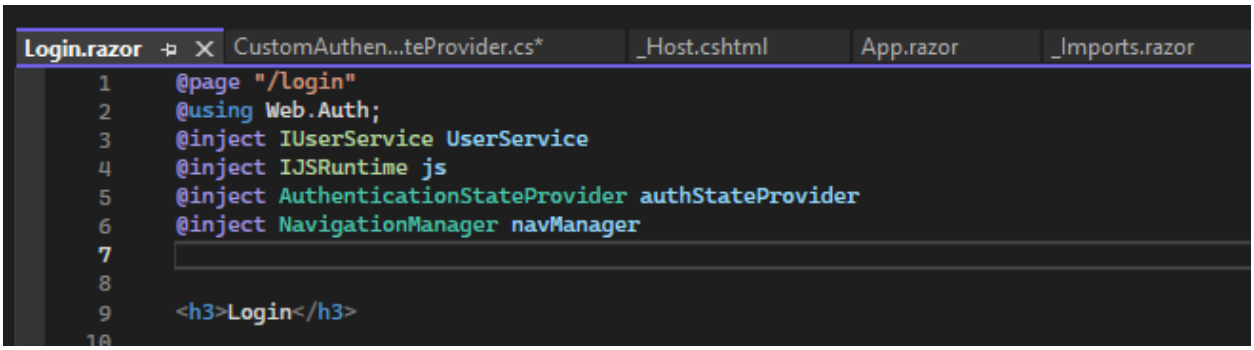
Создайте страницу Login, которая позволит пользователю вводить данные для входа



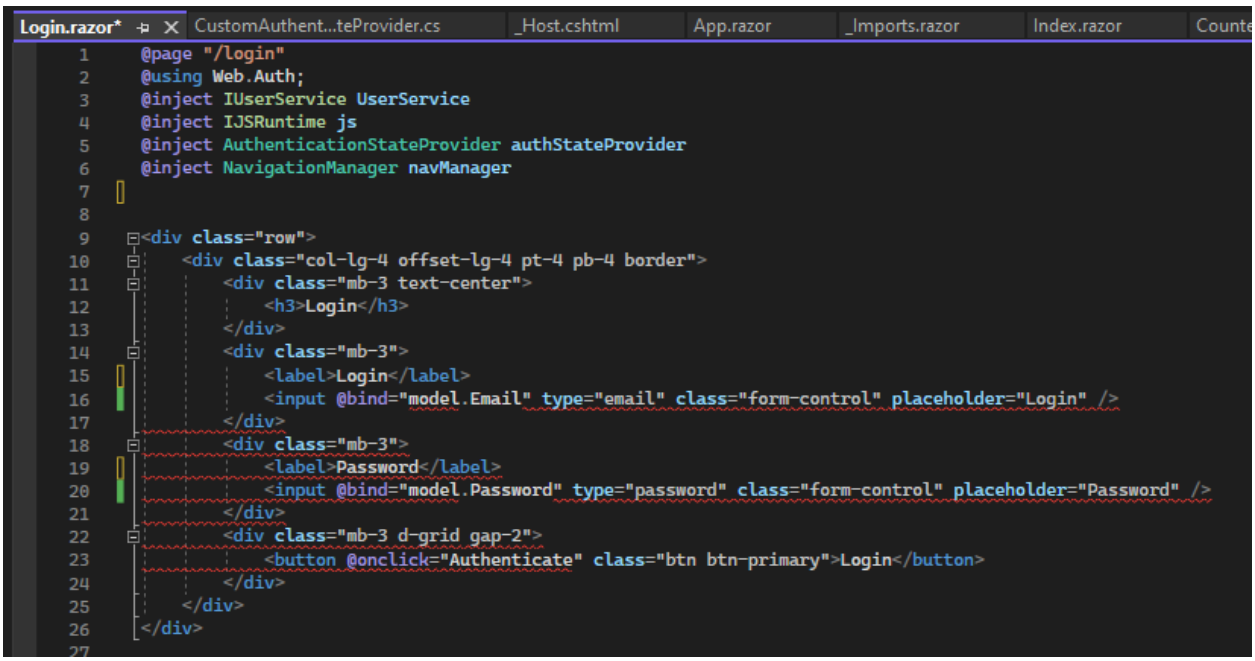


В странице Login.razor включите следующие директивы:

- @inject IUserService - подключаем сервис, который позволит работать с пользователями
- @inject IJSRuntime js - подключаем инструмент, который позволит выполнять код JavaScript в браузере
- @inject AuthenticationStateProvider - подключаем провайдер для аутентификации пользователя
- @inject NavigationManager - подключаем менеджер для навигации по страницам



Добавим форму для ввода данных



Красные строчки кода обозначают то, что компилятор не знает про переменную model и метод Authenticate. Создадим их в раделе @code.

```
@code {
    private class Model
    {
        public string Email { get; set; }
        public string Password { get; set; }
    }

    private Model model = new Model();

    private async Task Authenticate()
    {
        var userAccount = await UserService.Login(model.Email, model.Password);
        if (userAccount == null)
        {
            await js.InvokeVoidAsync("alert", "Invalid Colyaska");
            return;
        }

        var customAuthStateProvider = (CustomAuthenticationStateProvider)authStateProvider;

        await customAuthStateProvider.UpdateAuthenticationStateAsync(new UserSession
            {
                Id = userAccount.Id.ToString(),
                Email = userAccount.Email,
                UserName = userAccount.Firstname,
                Role = "User"
            });
        navManager.NavigateTo("/", true);
    }
}
```

В моем сервисе (UserService) есть метод Login, который возвращает запись о пользователе, если логин и пароль совпадают.
Реализован он следующим образом:

- Интерфейс IUserRepository имеет метод для поиска пользователя по почте и паролю

```
Ссылок: 6
public interface IUserRepository : IRepositoryBase<User>
{
    Ссылок: 2
    public Task<User?> GetByEmailAndPassword(string email, string password);
}
```

- Интерфейс IUserService имеет метод Login для авторизации пользователя по почте и паролю

```
Ссылок: 9
public interface IUserService
{
    Ссылок: 2
    Task<List<User>> GetAll();
    Ссылок: 2
    Task<User> GetById(int id);
    Ссылок: 5
    Task Create(User model);
    Ссылок: 2
    Task Update(User model);
    Ссылок: 2
    Task Delete(int id);
    Ссылок: 2
    Task<User> Login(string email, string password);
}
```

- Класс UserRepository, который реализует данный метод.

```
Ссылок: 2
public async Task<User?> GetByEmailAndPassword(string email, string password)
{
    var result = await base.FindByCondition(x => x.Email == email && x.Password == password);
    if (result == null || result.Count == 0)
    {
        return null;
    }

    return result[0];
}
```

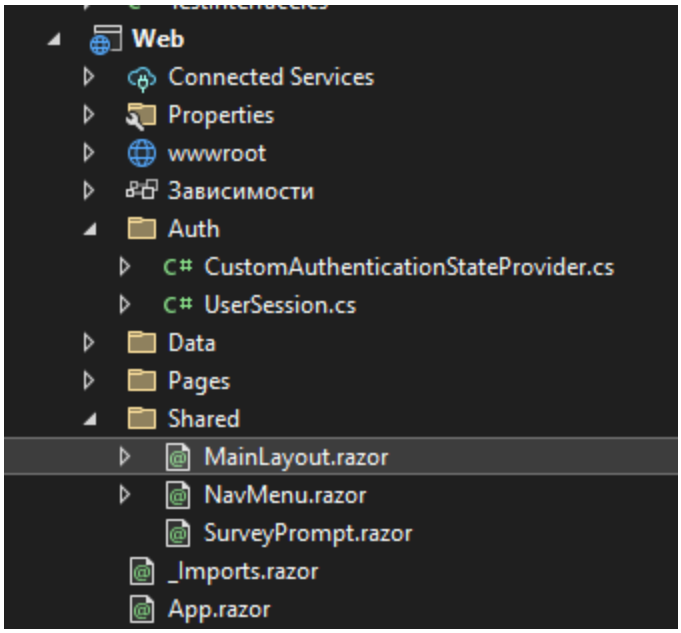
- Класс UserService, который реализован максимально просто

```
Ссылок: 2
public async Task<User> Login(string email, string password)
{
    var user = await _repositoryWrapper.User.GetByEmailAndPassword(email, password);
    return user;
}
```

Реализация функции выхода из системы

Так как пользователь может войти в систему - также создадим функционал для выхода.

Перейдите в файл MainLayout.razor

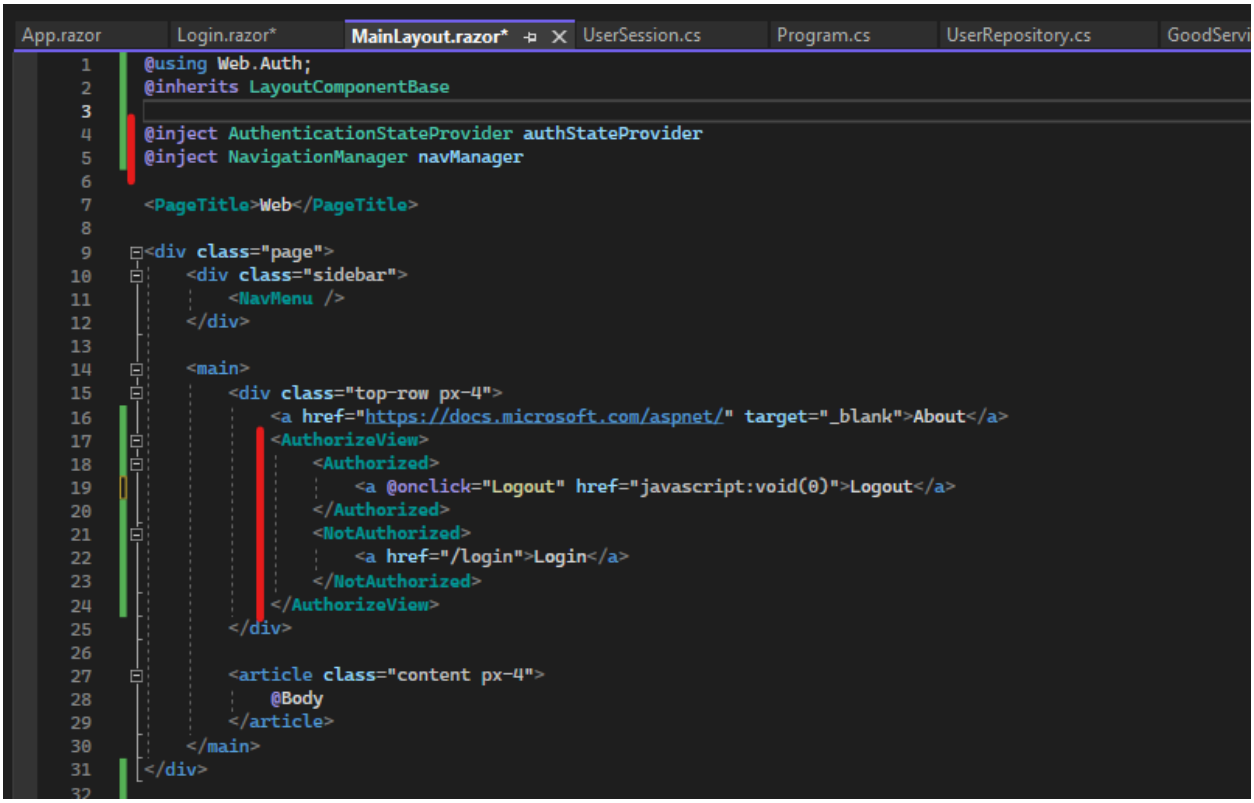


Добавьте отображаемые элементы (кнопка входа и выхода) в данном компоненте.

AuthorizeView - это такой компонент, который позволяет выводить элементы в случае различных ситуациях:

- В случае если пользователь вошел в систему
- В случае если пользователь не входил в систему
- В случае если проходит процесс входа (аутентификации)

```
<AuthorizeView>
    <Authorized>
        <a @onclick="Logout" href="javascript:void(0)">Logout</a>
    </Authorized>
    <NotAuthorized>
        <a href="/login">Login</a>
    </NotAuthorized>
</AuthorizeView>
```



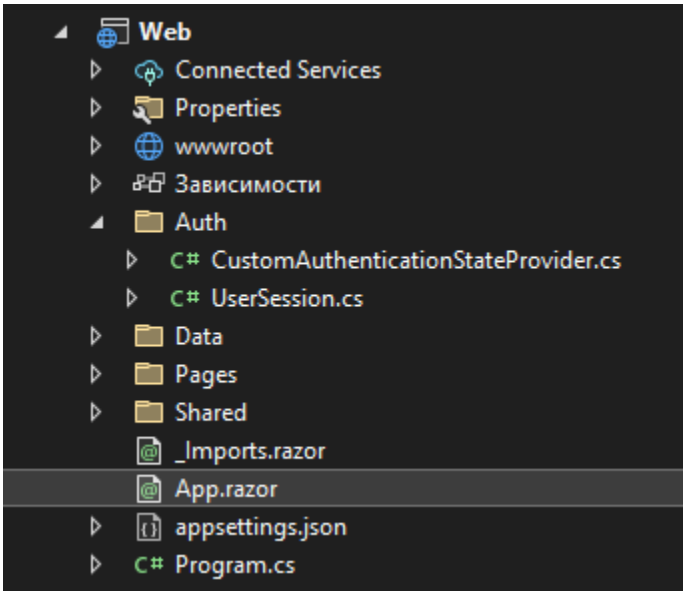
В разделе @code создайте метод Logout, который будет менять состояния входа и будет перенаправлять пользователя на главную страницу

```
@code {
    private async Task Logout()
    {
        var customAuthStateProvider = (CustomAuthenticationStateProvider)authStateProvider;
        await customAuthStateProvider.UpdateAuthenticationStateAsync(null);
        navManager.NavigateTo("/", true);
    }
}
```

Включение механизма авторизации

Для того, чтобы наше веб-приложение могло показывать именно тот контент, который ему будет разрешен - добавим механизм авторизации.

Перейдите в класс App.razor

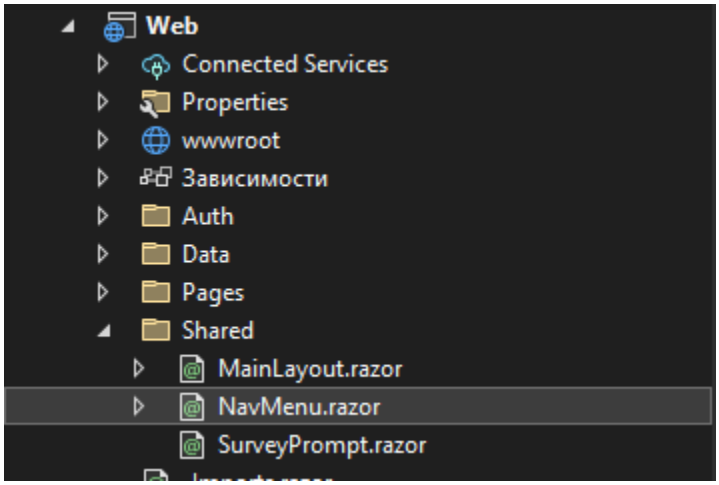


Переделайте класс в следующий вид.

```razor
<CascadingAuthenticationState>
    <Router AppAssembly="@typeof(App).Assembly">
        <Found Context="routeData">
            <AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
            <FocusOnNavigate RouteData="@routeData" Selector="h1" />
        </Found>
        <NotFound>
            <PageTitle>Not found</PageTitle>
            <LayoutView Layout="@typeof(MainLayout)">
                <p role="alert">Sorry, there's nothing at this address.</p>
            </LayoutView>
        </NotFound>
    </Router>
</CascadingAuthenticationState>
```

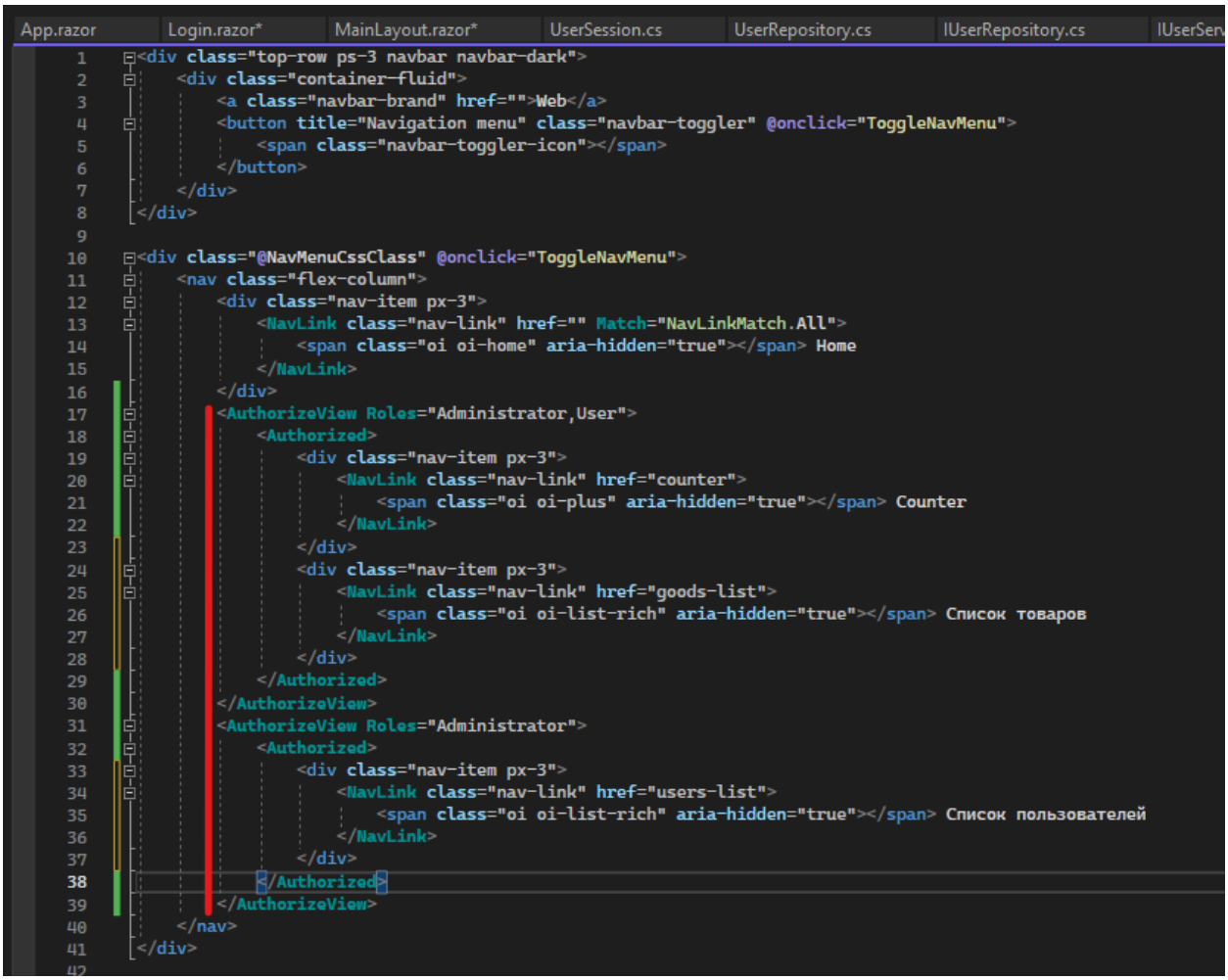
Таким образом, веб-приложение будет производить аутентификацию и авторизацию пользователей.

Теперь, перейдите в компонент, который представляет из себя навигационную панель

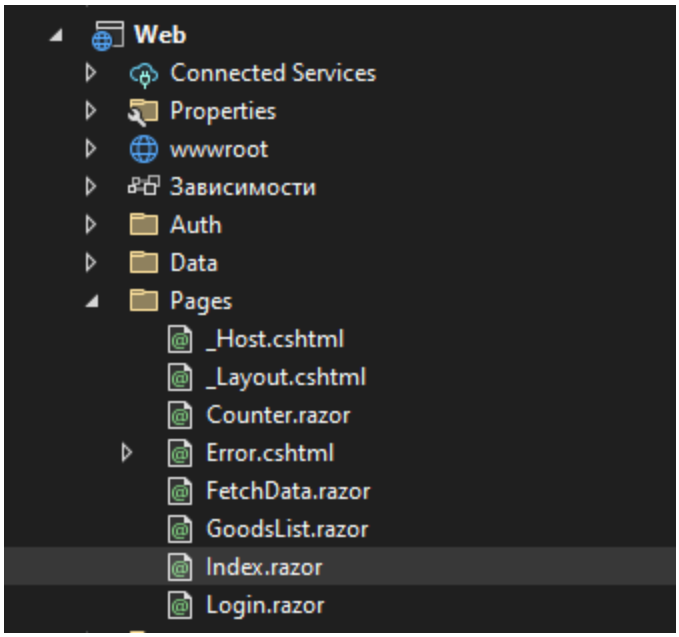


Переделайте его так, чтобы ссылки на страницы Counter и Goods List показывались в случае, если пользователь будет авторизован и иметь подходящую роль.

Аттрибут Roles в компоненте AuthorizeView указывает, что данные элементы будут показаны, если пользователь имеет подходящую роль.



Откройте код файла Index.razor, который представляет из себя главную страницу

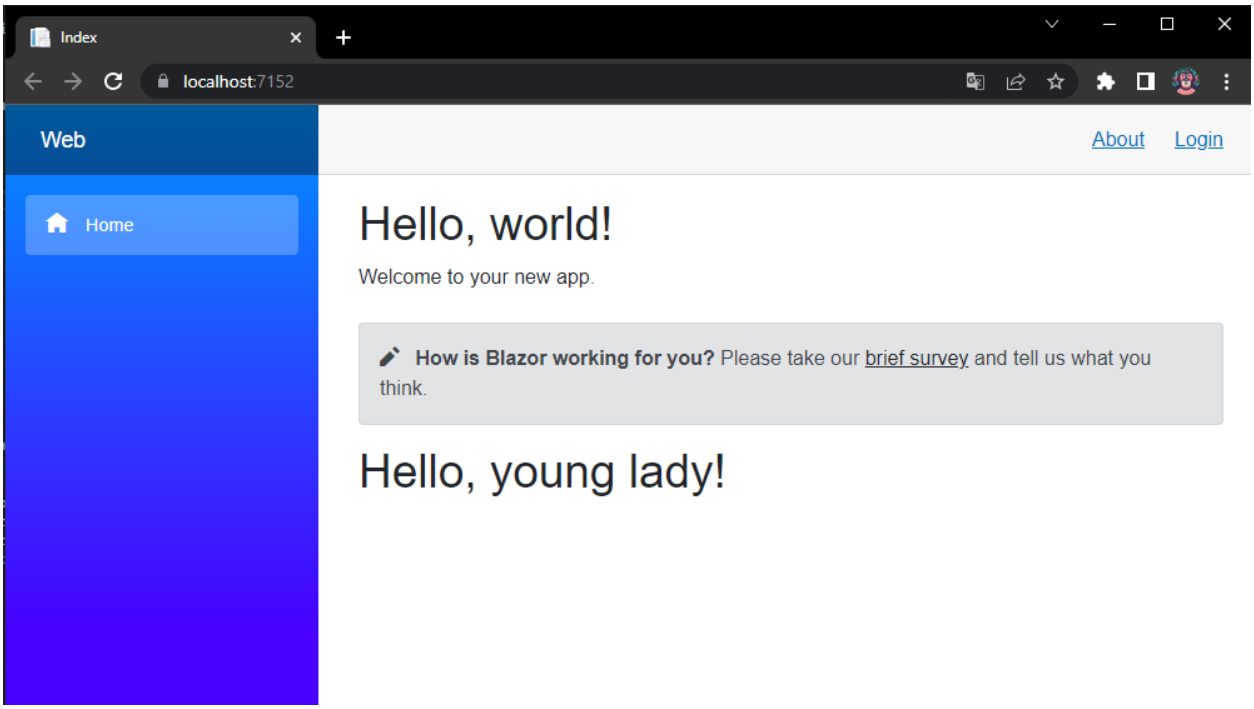


Теперь, сделаем так, чтобы на главной странице показывались элементы в различных ситуациях.

```razor
@page "/"
@using Web.Auth;
@inject IJSRuntime js
@inject IUserService UserService
<PageTitle>Index</PageTitle>

<h1>Hello, world!</h1>

Welcome to your new app.

<SurveyPrompt Title="How is Blazor working for you?" />

<AuthorizeView>
    <Authorized>
        <h1>Welcome, @context.User.Identity.Name!</h1>
    </Authorized>
    <NotAuthorized>
        <h1>Hello, young lady!</h1>
    </NotAuthorized>
</AuthorizeView>

<AuthorizeView>
    <Authorized>
        <br /><br />
       <button class="btn btn-outline-primary" @onclick="DisplayGreetingAlert">Greeting</button>
    </Authorized>
</AuthorizeView>

@code {
    [CascadingParameter]
    private Task<AuthenticationState> authenticationState { get; set; }

    private async Task DisplayGreetingAlert()
    {
        var authState = await authenticationState;
        var message = $"Hello, {authState.User.Identity.Name}";
        await js.InvokeVoidAsync("alert", message);
    }
}
```

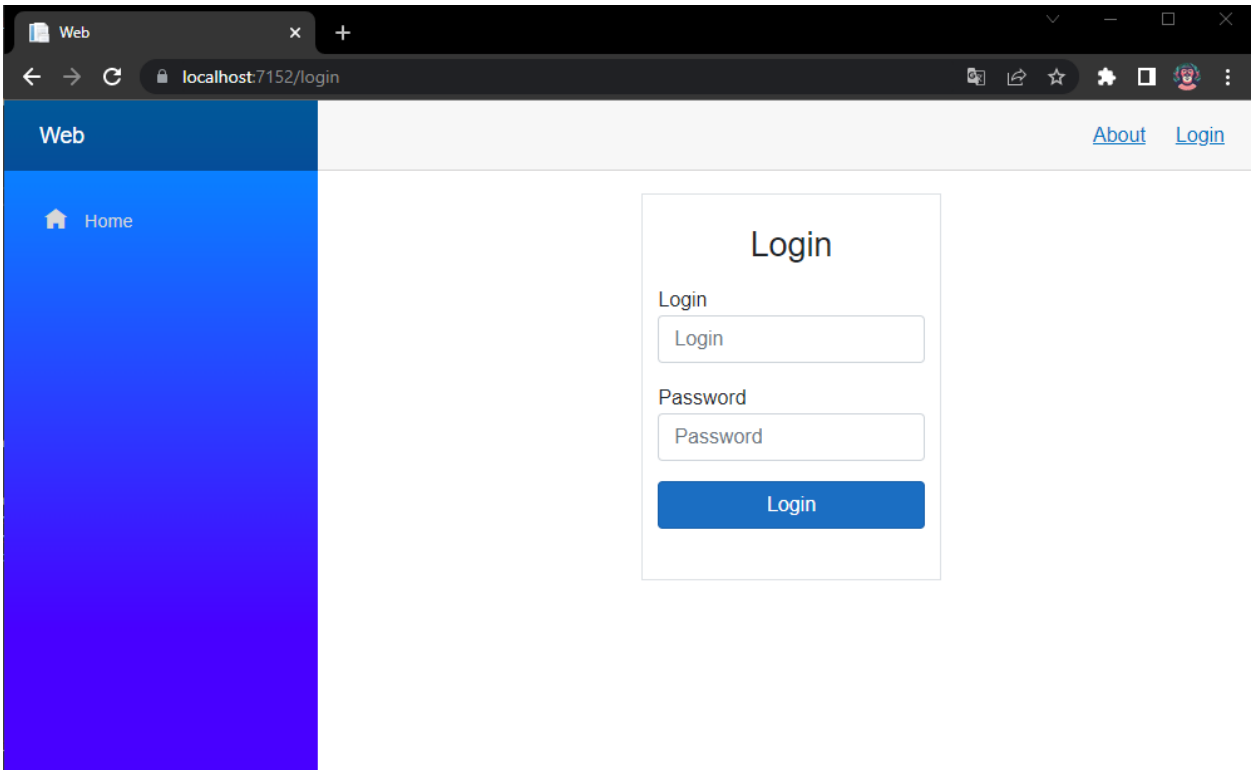
Проверка работоспособности веб-приложения

Теперь запустите проект Blazor Server и проверьте работу аутентификации и авторизации.

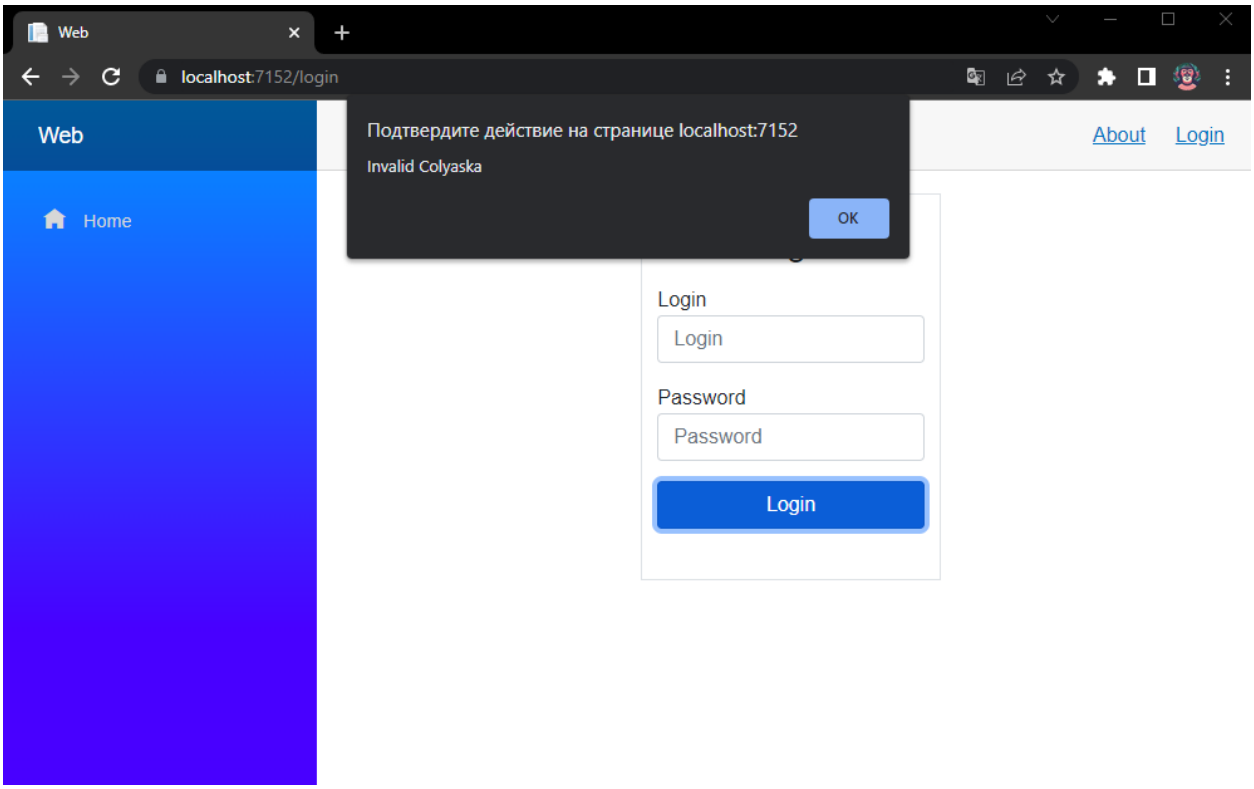


Как вы можете заметить, в навигационной панели нет ссылок для перехода на другие страницы - потому что пользователь не авторизован.

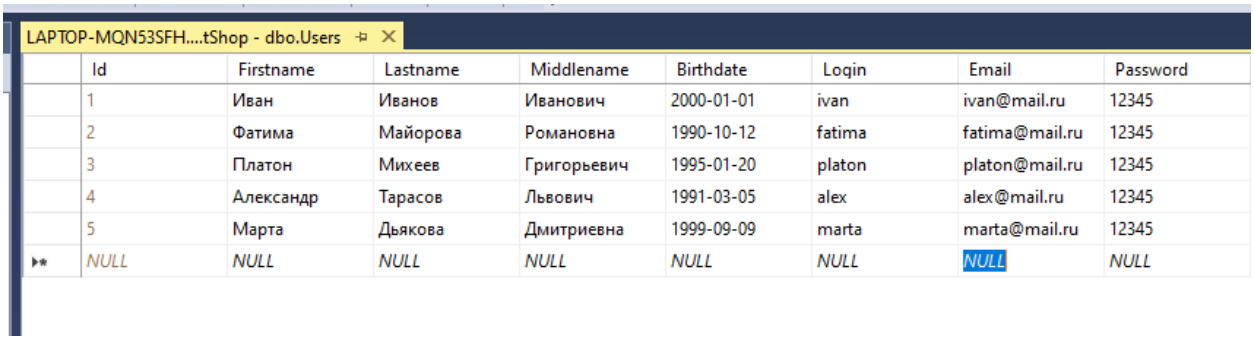
Перейдите на страницу для входа и нажмите на кнопку Login



В случае, если вылезло всплывающее сообщение - значит UserService и UserRepository работают корректно. Пользователь по пустым полям не обнаружен.

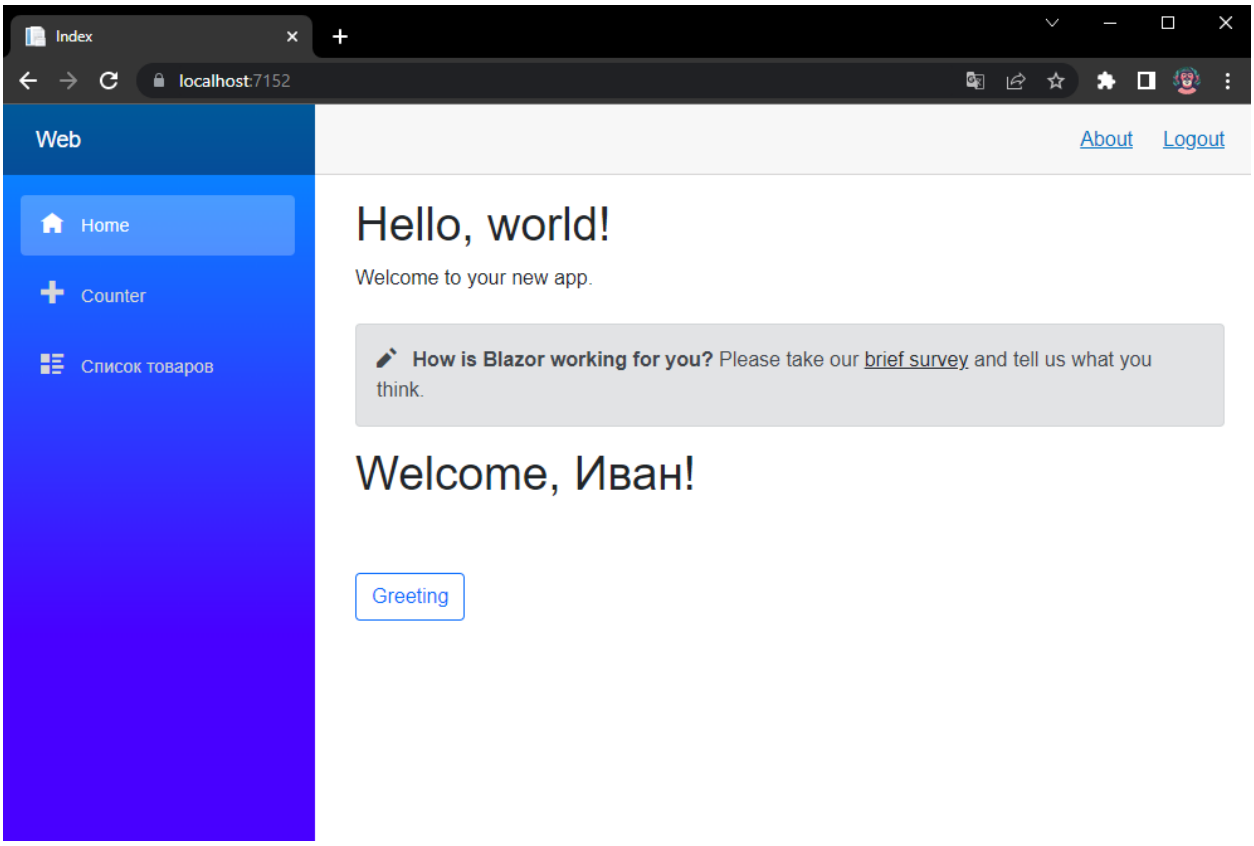


Проверьте, что в базе данных имеются тестовые записи в таблице Users. Например:

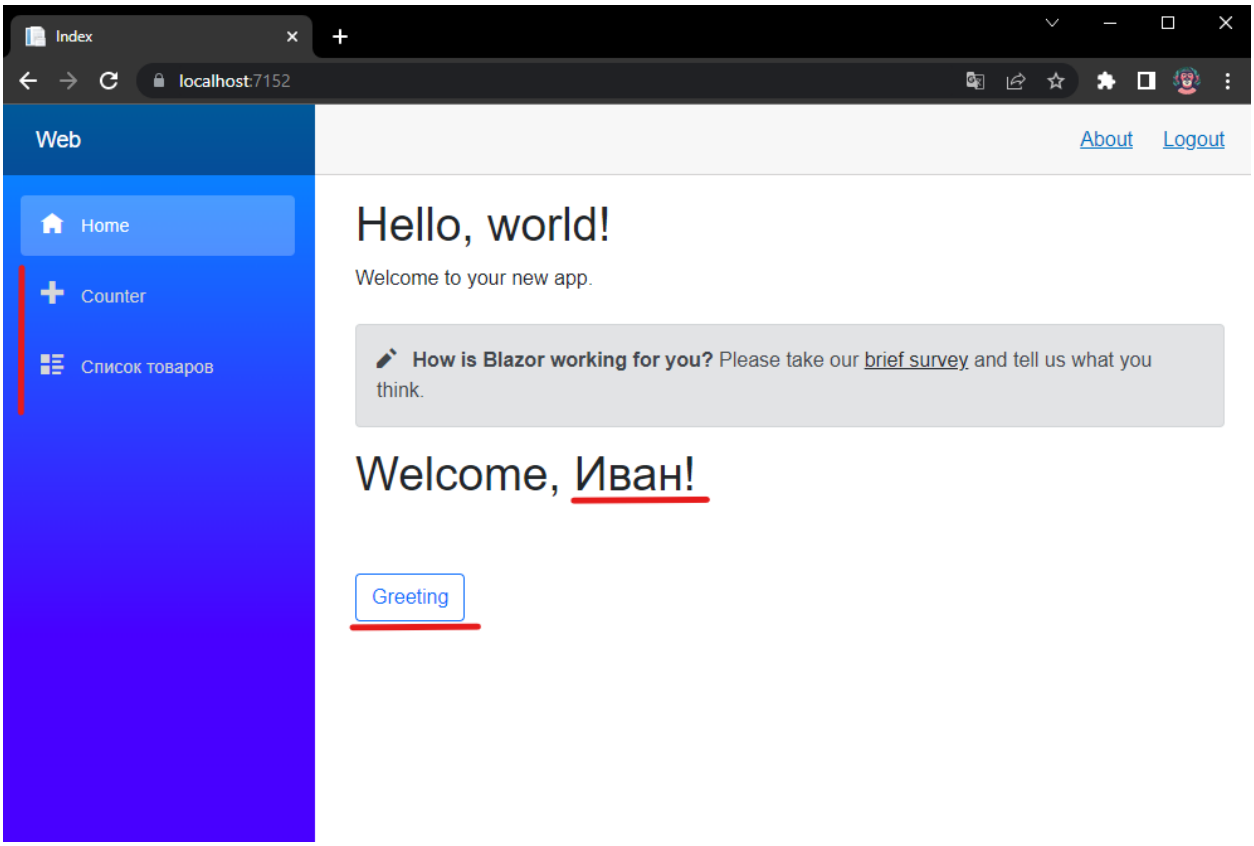


	Id	Firstname	Lastname	Middlename	Birthdate	Login	Email	Password
	1	Иван	Иванов	Иванович	2000-01-01	ivan	ivan@mail.ru	12345
	2	Фатима	Майорова	Романовна	1990-10-12	fatima	fatima@mail.ru	12345
	3	Платон	Михеев	Григорьевич	1995-01-20	platon	platon@mail.ru	12345
	4	Александр	Тарасов	Львович	1991-03-05	alex	alex@mail.ru	12345
	5	Марта	Дьякова	Дмитриевна	1999-09-09	marta	marta@mail.ru	12345
▶*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

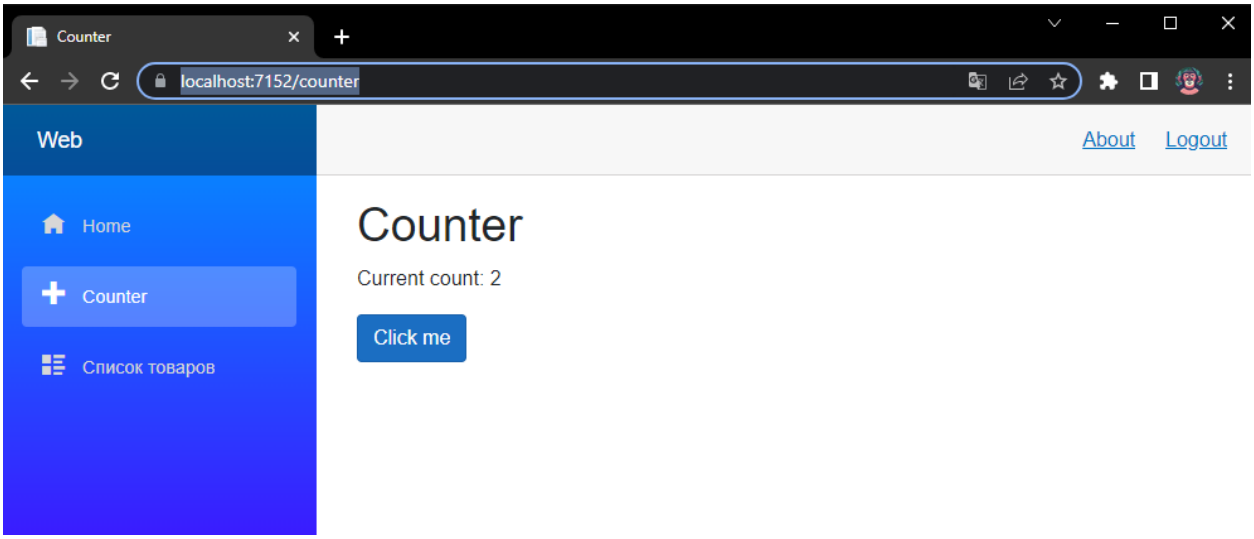
Теперь введите корректные данные и войдите в систему



В случае успеха - вы перенаправитесь на главную страницу и увидите изменения

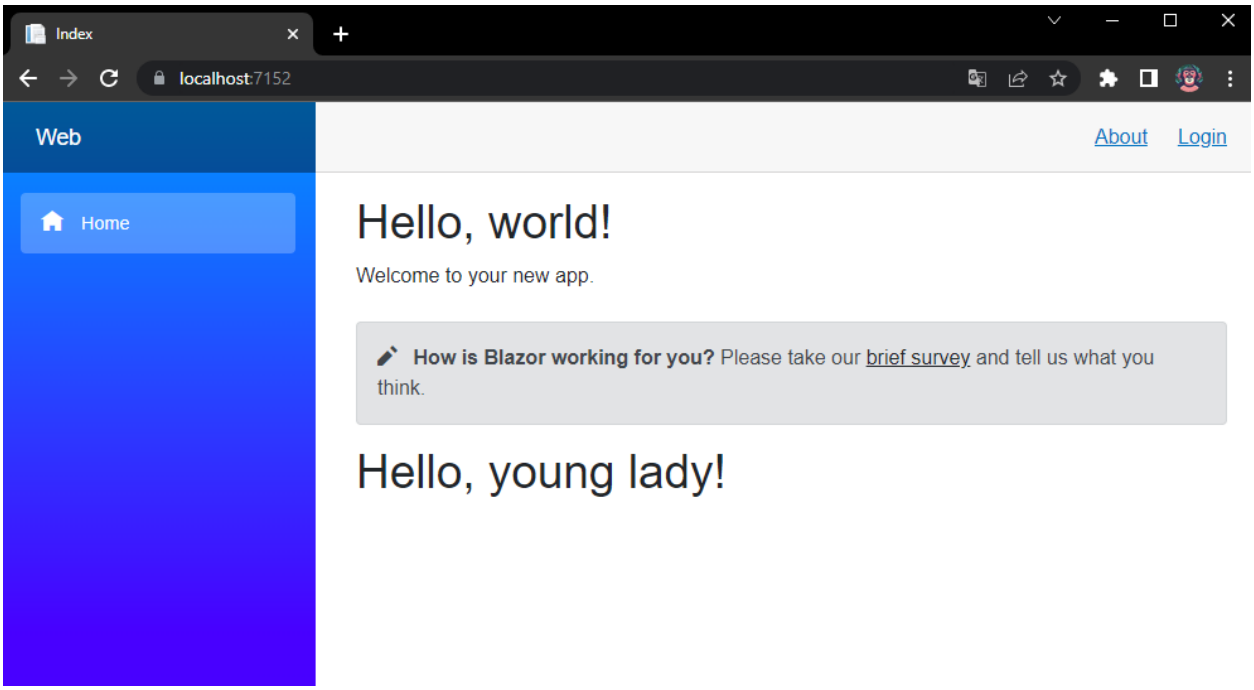


Перейдите на страницу counter и скопируйте ссылку. Ссылка понадобится чуть позже.



Нажмите на кнопку Logout и проверьте функцию выхода из системы.

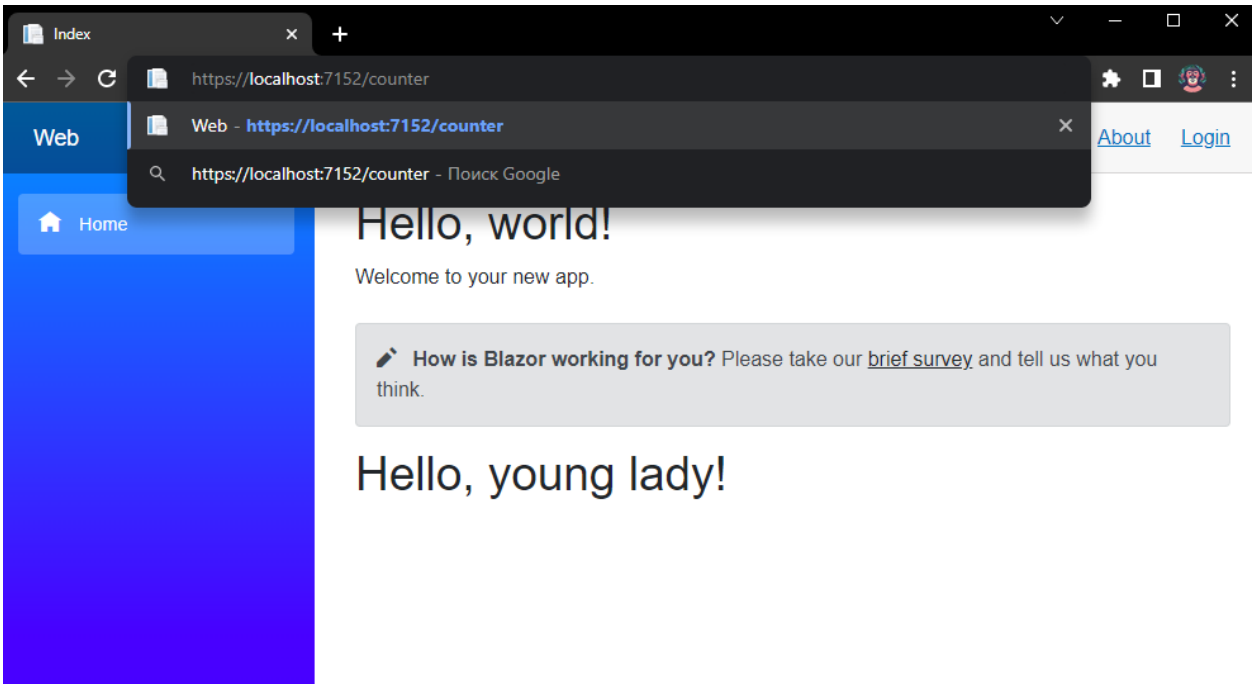
В случае, если выход будет успешным - вы вернетесь на главную страницу.



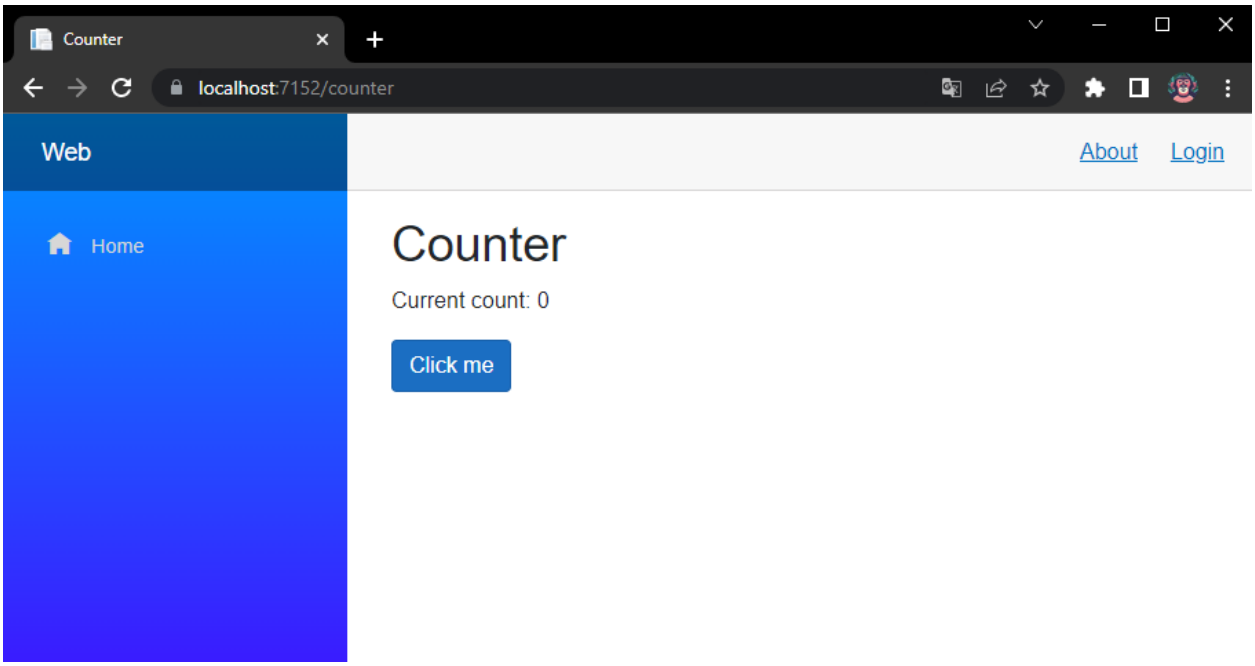
Безопасность веб-приложения

Проблема отображения страниц

Если вы попробуете перейти на страницу Counter как неавторизованный пользователь, то веб-приложение пустит без ошибок.

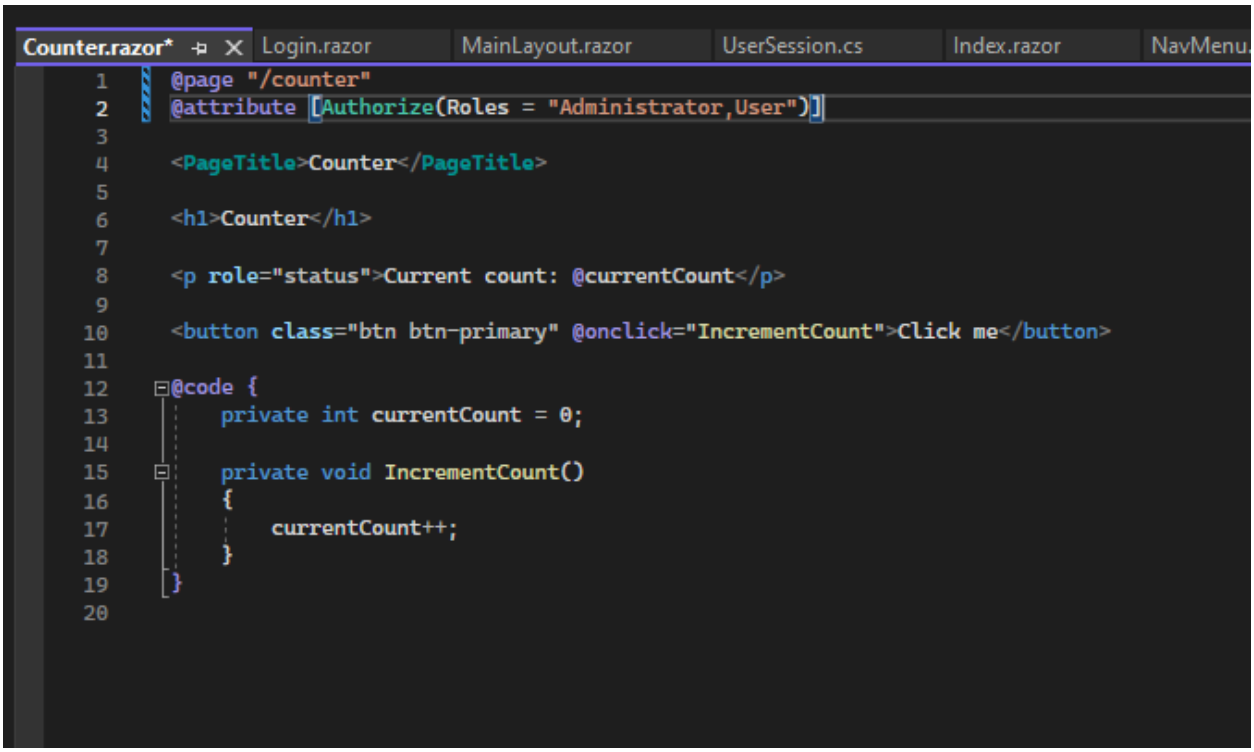


Хоть вы и не авторизовались, содержимое страницы всё равно показывает.

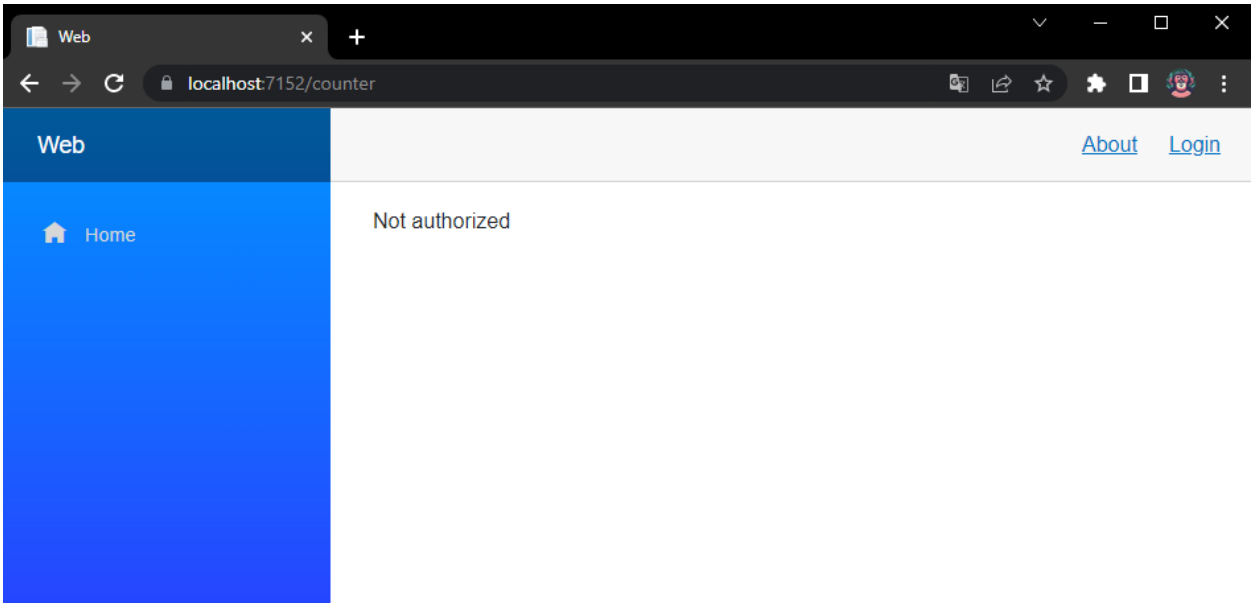


Для того, чтобы не было такой ошибки - для страниц нужно добавлять атрибут Authorize.

Если на данную страницу должны входить пользователи с определенными правами - указывайте внутри атрибута список ролей через запятую.

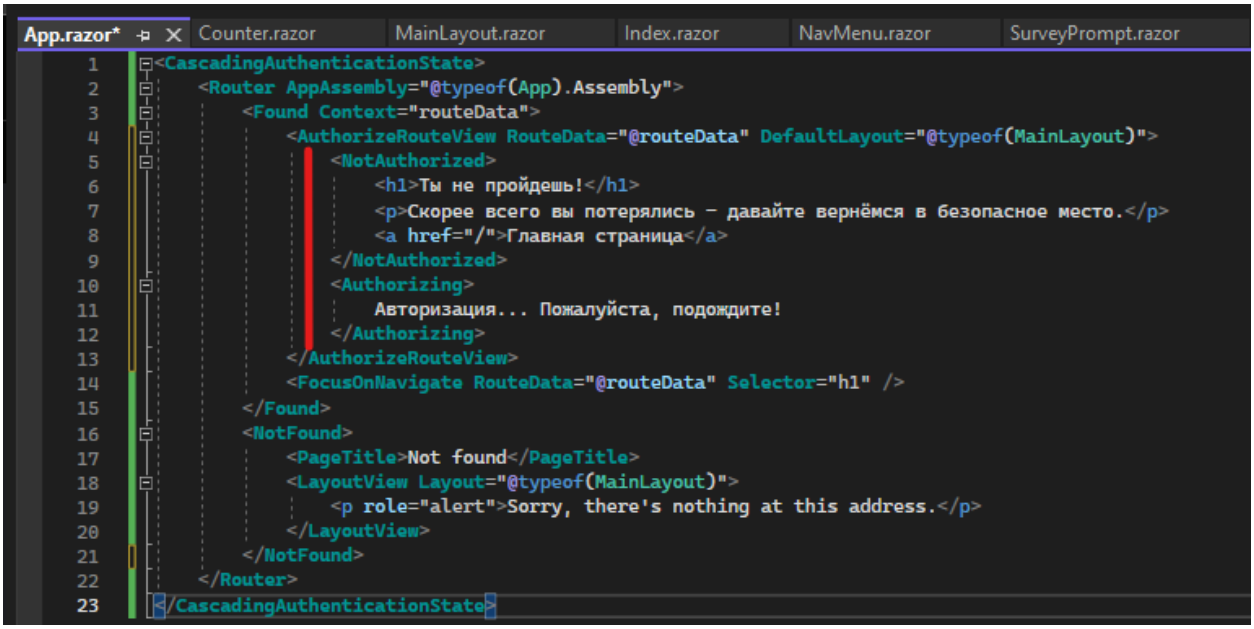


Таким образом страница будет говорить о том, что пользователь не авторизован

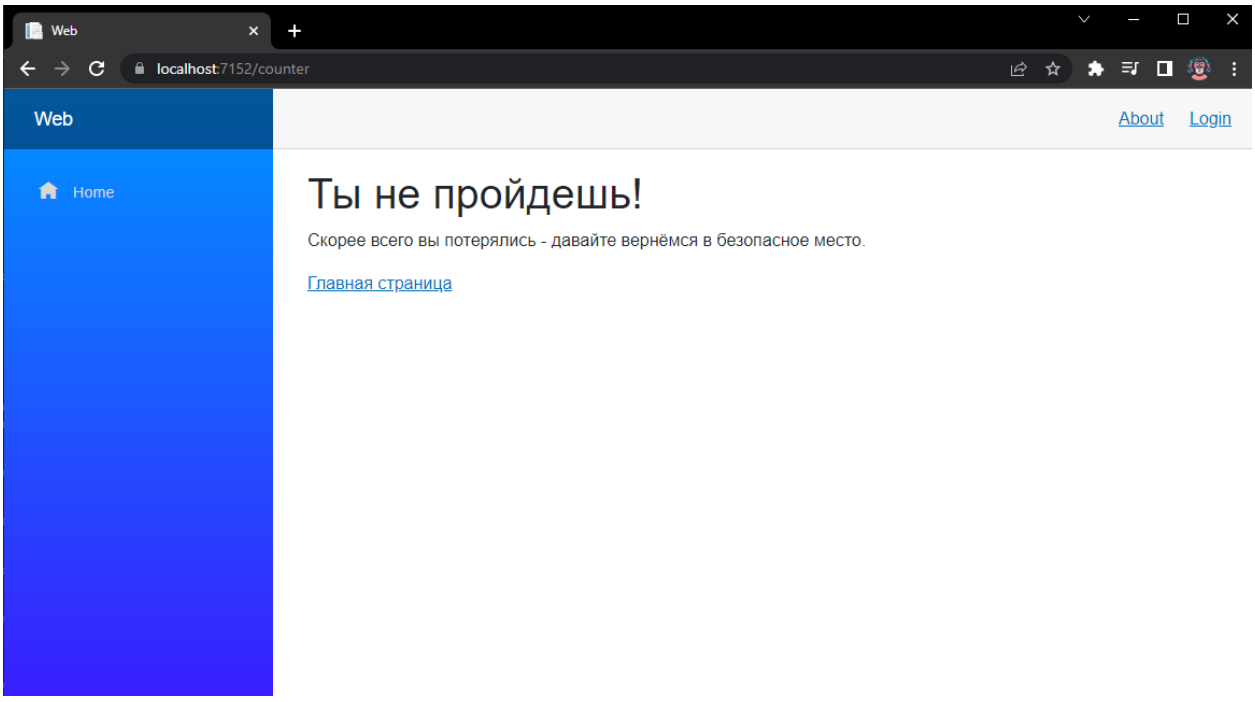


Чтобы пользователю было понятно, что произошло - вы можете изменять отображаемый текст.

Перейдите в файл App.razor и добавьте следующий код:



Теперь страница будет выглядеть так:



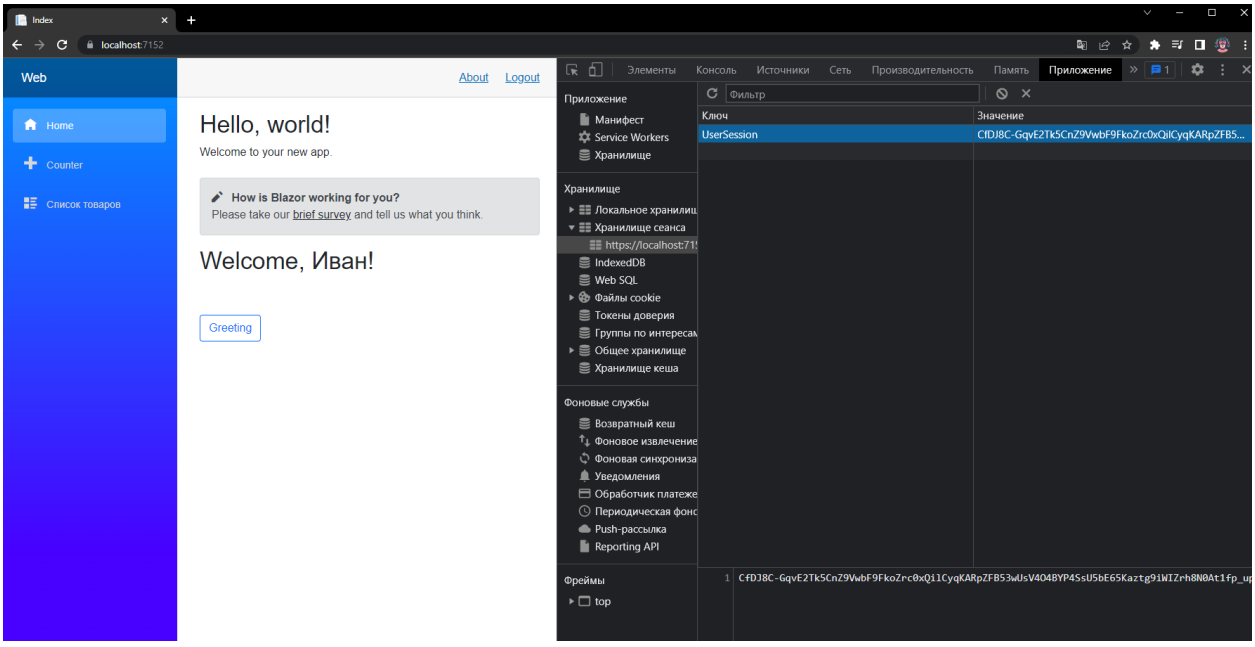
Проблема авторизации пользователей

На данный момент мы реализовали самую простейшую форму аутентификации пользователей. А именно использование сессии.

Данный способ имеет значительный недостаток и может быть с легкостью украден злоумышленниками. Для этого достаточно скопировать значение ключа и использовать его для авторизации.

> ℹ️ Чтобы защититься от такого существуют и другие способы для аутентификации пользователей, рассмотреть их можете тут: https://habr.com/ru/companies/dataart/articles/311376/

Откройте консоль разработчика и в разделе Приложение (Application) найдите ключ UserSession.



Необходимость авторизовываться после перезапуска браузера.

Если вы попробуете авторизоваться, затем закрыть браузер и после открыть снова сайт - вы будете не авторизованы, потому что ключ сессии не сохраняется в браузере.

Выполнение заданий

1) Внедрите сохранение ключа сессии в локальном хранилище браузера (Local Storage), чтобы не было необходимости пользователю вновь входить на сайт после перезапуска браузера.

> 💡 Подсказка: в классе CustomAuthenticationStateProvider используйте переменную _localStorage.

2) Добавьте страницу регистрации и обеспечьте возможность регистрироваться новым пользователям