

Лабораторная работа № 12

Введение в Blazor WebAssembly

1. Цель работы.

Изучение проекта Blazor.

2. Задача работы

Создание проекта Blazor, использующего аутентификацию удаленного сервера и получающий данные с API.

3. Общие сведения.

4. Выполнение работы

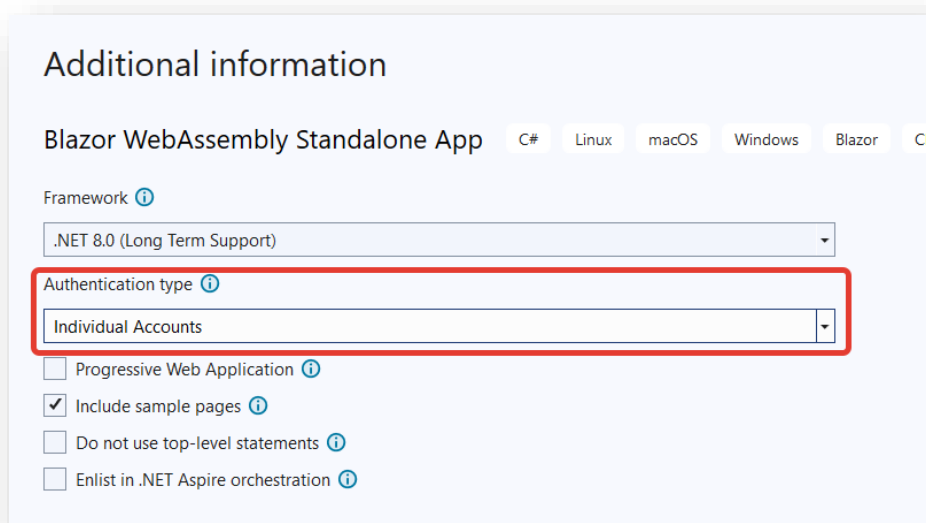
4.1. Исходные данные

Используйте проект из лабораторной работы №11.

4.2. Подготовка проекта

Добавьте в решение новый проект – приложение Blazor WebAssembly (Blazor WebAssembly Standalone App). Назначьте проекту имя XXX.BlazorWasm, где XXX – имя вашего решения.

Выберите аутентификацию с помощью индивидуальных учетных записей.



The screenshot shows the 'Additional information' dialog in Visual Studio. At the top, the project name is 'Blazor WebAssembly Standalone App'. Below it, there are tabs for different frameworks: C#, Linux, macOS, Windows, Blazor, and Clojure. The 'Blazor' tab is selected. Under the 'Framework' section, a dropdown menu shows '.NET 8.0 (Long Term Support)'. Below that, the 'Authentication type' dropdown is highlighted with a red rectangle and shows 'Individual Accounts'. At the bottom, there are four checkboxes: 'Progressive Web Application' (unchecked), 'Include sample pages' (checked), 'Do not use top-level statements' (unchecked), and 'Enlist in .NET Aspire orchestration' (unchecked).

В созданном проекте найдите в файле Program.cs регистрацию компонента «app» и сервиса HttpClient. Найдите регистрацию настроек для oidc: `builder.Services.AddOidcAuthentication`. Файл `appsettings.json` находится в папке `wwwroot`.

Найдите корневой компонент приложения `app.razor`. Откройте его. Познакомьтесь с использованием компонента Router.

Найдите файл `_Imports.razor` и познакомьтесь с его содержимым.

В папке `wwwroot` найдите корневую страницу приложения `index.html`. Откройте файл `index.html` и познакомьтесь с его содержимым. Найдите, где размещен главный компонент приложения («app») и где подключается скрипт `_framework/blazor.webassembly.js`.

Найдите файл `_Imports.razor` и познакомьтесь с его содержимым.

Найдите страницу макета (`MainLayout.razor`) и познакомьтесь с ее содержимым. Найдите использование на макете компонента `NavMenu`. Найдите выражение `@Body`. Сюда будет размещена разметка страницы, использующей макет.

Найдите компонент `NavMenu`. Изучите его содержимое. Обратите внимание на использование компонента `NavLink` для переключения между страницами.

4.3. Подключение к Keycloak

Добавьте в ваш Realm сервера Keycloak нового клиента для созданного приложения WebAssembly (см. добавление клиента в лабораторной работе 6).

В проекте XXX.BlazorWasm, в файле `appsettings.json` и `appsettings.development.json` (в папке `wwwroot`) добавьте секцию настроек для подключения к Keycloak:

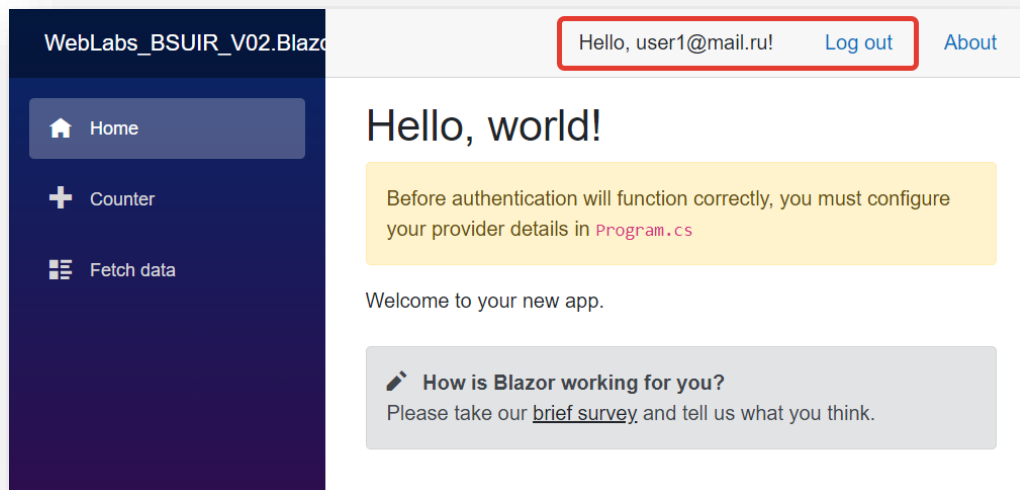
```
"Keycloak": {
  "Authority": "http://localhost:8080/auth/realms/[Ваш Realm]",
  "ClientId": "[Id созданного клиента]",
  "MetadataUrl": "http://localhost:8080/realms/[Ваш Realm]/.well-known/openid-configuration",
  "ResponseType": "id_token token"
}
```

В классе Program при подключении Oidc вместо «Local» укажите использование настроек Keycloak

```
builder.Services.AddOidcAuthentication(options =>
{
    // Configure your authentication provider options here.
    // For more information, see https://aka.ms/blazor-standalone-auth
    builder.Configuration.Bind("Keycloak", options.ProviderOptions);
});
```

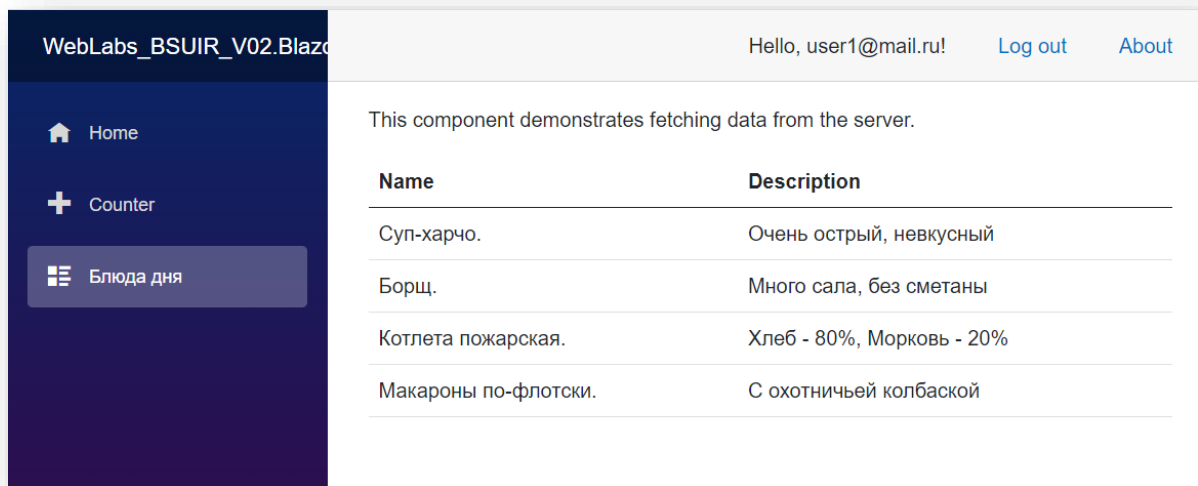
Запустите проект. При нажатии пункта меню Login должна появиться страница входа в систему.

При удачном входе появится приветственное сообщение:



4.4. Задание №1. Вывод списка объектов

Создайте страницу, выводящую список названий объектов вашей предметной области. Данные получить с Арі. Добавьте пункт меню для перехода на эту страницу.



4.4.1. Рекомендации к заданию №1

Запрос к API для получения списка отправляйте в методе `OnInitializedAsync`.

4.5. Задание №2. Описание сервиса доступа к данным

Добавьте в проект `XXX.BlazorWASM` папку `Services`. В созданной папке создайте интерфейс, описывающий методы и свойства, необходимые для обращения к API и для хранения полученных данных :

```
public interface IDataService
{
    // Событие, генерируемое при изменении данных
    event Action DataLoaded;

    // Список категорий объектов
    List<Category> Categories { get; set; }
    //Список объектов
    List<Dish> Dishes { get; set; }

    // Признак успешного ответа на запрос к Api
    bool Success { get; set; }
    // Сообщение об ошибке
    string ErrorMessage { get; set; }

    // Количество страниц списка
    int TotalPages { get; set; }
    // Номер текущей страницы
    int CurrentPage { get; set; }
    // Фильтр по категории
    Category SelectedCategory { get; set; }
}
```

```

    /// <summary>
    /// Получение списка всех объектов
    /// </summary>
    /// <param name="pageNo">номер страницы списка</param>
    /// <returns></returns>
    public Task GetProductListAsync(int pageNo = 1);

    /// <summary>
    /// Получение списка категорий
    /// </summary>
    /// <returns></returns>
    public Task GetCategoryListAsync();
}

```

Создайте класс DataService, реализующий интерфейс IDataService.

Обязательно: адрес сервера API и размер страницы должны быть получены из файла appsettings.json

Зарегистрируйте DataService как **scoped** сервис.

Используйте созданный сервис на странице списка объектов.

Запустите проект. Проверьте результат.

4.5.1. Рекомендации к заданию №2

Установите адрес сервера Api в качестве базового адреса HttpClient в классе Program.cs.

Добавьте пространство имен XXX.BlazorWASM.Services в файле _imports.razor.

Для передачи в запросе к Api номера страницы и размера страницы используйте класс QueryString:

```

var route = new StringBuilder("dishes/");
// добавить категорию в маршрут
if (SelectedCategory is not null)
{
    route.Append($"{SelectedCategory.NormalizedName}/");
};

List<KeyValuePair<string, string>> queryData = new();

```

```
// добавить номер страницы в маршрут
if (pageNo > 1)
{
    queryData.Add(KeyValuePair.Create("pageNo", pageNo.ToString()));
};
// добавить размер страницы
if (!_pageSize.Equals("3"))
{
    queryData.Add(KeyValuePair.Create("pageSize", _pageSize));
}
// добавить строку запроса в Url
if (queryData.Count > 0)
{
    route.Append(QueryString.Create(queryData));
}
}
```

4.6. Задание №3. Авторизация с помощью JWT

Запретите неавторизованный доступ к методу получения списка объектов в контроллере Api, а также неавторизованный доступ к странице списка объектов в проекте XXX.BlazorWASM.

Запустите проект. Убедитесь, что данные не получаются с сервера Api.

Измените код класса DataService для получения данных с API с использованием JWT-токена.

4.6.1. Рекомендации к заданию №3

JWT токен можно получить из сервиса IAccessTokenProvider :

```
var tokenRequest = await _tokenProvider.RequestAccessToken();
if (tokenRequest.TryGetToken(out var token))
{
    . . .
}
```

4.7. Задание №4. Компонент списка объектов

Оформите вывод списка в виде отдельного компонента. В результате страница списка объектов должна выглядеть примерно так:

```
@page "/menu"
```

```
<PageTitle>Menu</PageTitle>
<DishesList />
```

```
@code {
    [Inject]
    public IDataService DataService { get; set; }
    protected override async Task OnInitializedAsync()
    {
        await DataService.GetProductListAsync(null);
    }
}
```

4.7.1. Рекомендации к заданию №4

В качестве источника данных использовать свойство `ObjectsList` сервиса `DataService`.

Для того, чтобы компонент «увидел», что данные получены, в классе `DataService` вызовите событие `DataLoaded` после получения данных из сервиса `Api`

В компоненте подпишитесь на событие:

```
@inject IDataService DataService
@implements IDisposable
```

<< разметка компонента >>

```
@code {

    protected override void OnInitialized()
    {
        DataService.DataLoaded += StateHasChanged;
    }

    public void Dispose()
    {
        DataService.DataLoaded -= StateHasChanged;
    }
}
```

4.8. Задание №5. Фильтрация по категории

Создайте компонент для выбора категории. При выборе категории в классе `DataService` должно устанавливаться свойство `SelectedCategory` и выполняться вызов метода `GetProductListAsync`. Компонент должен также показывать текущую категорию.

Разместите компонент на странице.

Пример разметки страницы:

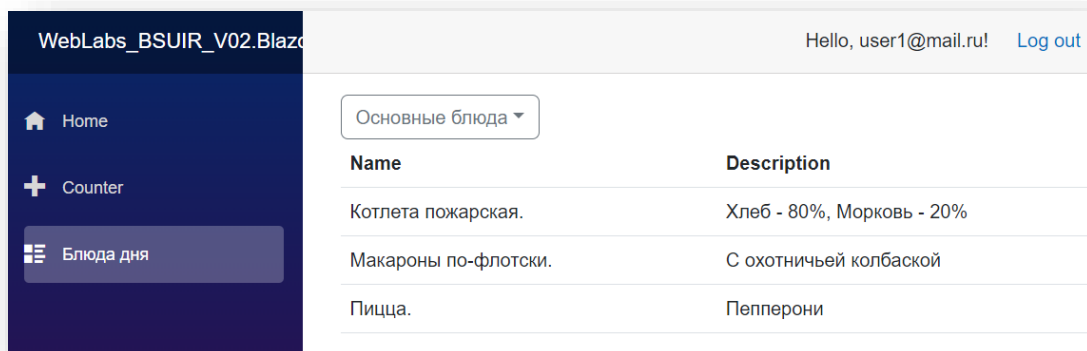
```
@page "/menu"
@using Microsoft.AspNetCore.Authorization
@attribute [Authorize]

<h3>Меню дня</h3>
<CategorySelector/>
<DishesList />

@code {
    [Inject] IDataService DataService { get; set; }

    protected override async Task OnInitializedAsync()
    {
        DataService.GetProductListAsync(null);
    }
}
```

Пример вида страницы:



4.8.1. Рекомендации к заданию №5

Можно воспользоваться компонентом Dropdown из библиотеки Bootstrap.

В проекте присутствуют только таблицы стилей Bootstrap. Но для работы Dropdown необходимо наличие скриптов bootstrap.bundle.min.js. Добавьте в проект недостающие скрипты и поместите их на страницу Index.html.

4.9. Задание №6. Переключение страниц списка

Создайте компонент, выводящий кнопки переключения страниц списка.

Кнопка текущей страницы должна быть выделена.

Если общее количество страниц равно 1, то кнопки не отображаются.

Поместите компонент на страницу вывода списка объектов. Передайте в компонент имя текущей категории.

Пример разметки страницы:

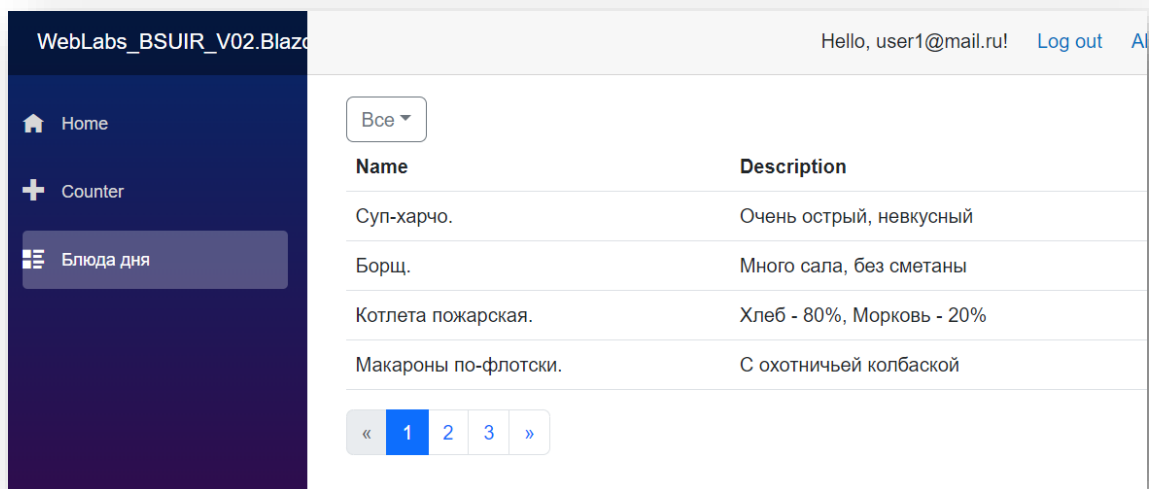
```
page "/menu"
@using Microsoft.AspNetCore.Authorization
@attribute [Authorize]

<h3>Меню дня</h3>
<CategorySelector/>
<DishesList />
<Pager/>

@code {
    [Inject] IDataService DataService { get; set; }

    protected override async Task OnInitializedAsync()
    {
        DataService.GetProductListAsync(null);
    }
}
```

Пример пейджера:



4.9.1. Рекомендации к заданию №6

Исходные данные (номер текущей страницы и общее количество страниц) есть в сервисе DataService.

Можно воспользоваться компонентом **Pagination** библиотеки **Bootstrap**. Замените тэги `<a>` на тэги `<button>`. По событию кнопки **Click** с помощью сервиса DataService получите список объектов для выбранной страницы

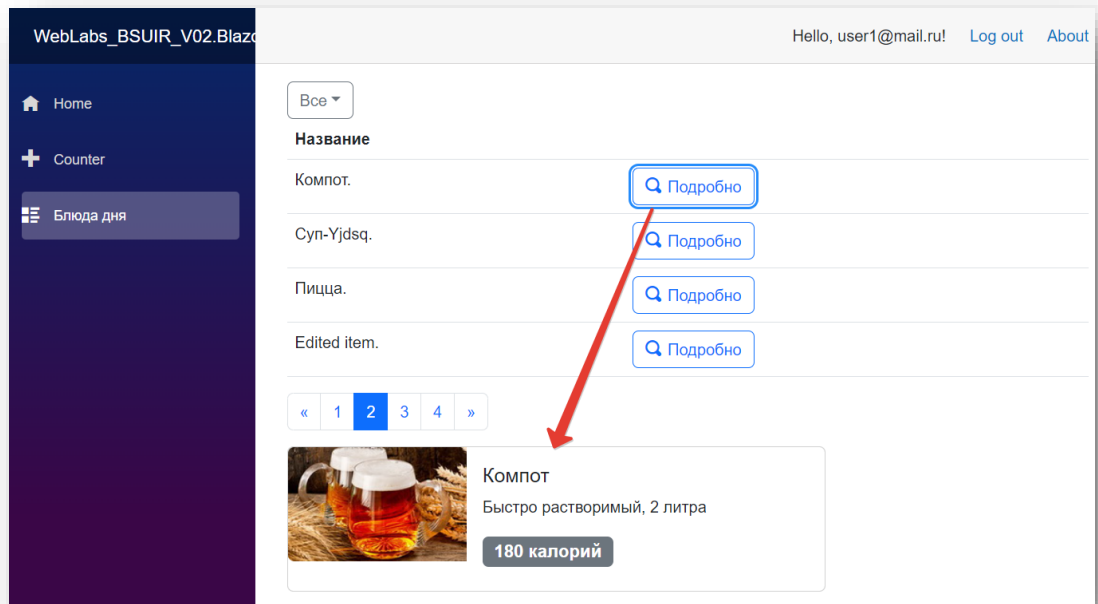
4.10. Задание № 7. Взаимодействие между компонентами

Опишите компонент, отображающий подробную информацию об объекте. Разместите компонент на странице вывода списка объектов.

В списке объектов, напротив названия объекта, поместите кнопку. По клику кнопки найти выбранный объект по Id и передать найденный объект в созданный компонент для получения подробной информации о выбранном объекте.

Обработка клика кнопки (поиск объекта) должна происходить на странице списка объектов, а не в компоненте списка.

Пример внешнего вида страницы:



Пример разметки страницы (в примере оставлен только код, относящийся к заданию):

```

@page "/menu"
@using Microsoft.AspNetCore.Authorization
@attribute [Authorize]

<h3>Меню дня</h3>
<CategorySelector/>
<DishesList DishSelected="OnDishSelected" />
<Pager/>
<DishDetails SelectedDish="SelectedDish" />

@code {
    [Inject] IDataService DataService { get; set; }
    Dish SelectedDish { get; set; }

    protected override async Task OnInitializedAsync()
    {
        DataService.GetProductListAsync();
    }

    void OnDishSelected(int id)
    {
        SelectedDish = DataService.Dishes
            .FirstOrDefault(d => d.Id == id);
    }
}

```

The diagram illustrates the data flow in the provided code. Red boxes highlight the `DishSelected` attribute in the `<DishesList>` tag, the `SelectedDish` property in the `@code` block, and the `SelectedDish` attribute in the `<DishDetails>` tag. Red arrows show the flow: one arrow points from the `SelectedDish` property to the `SelectedDish` attribute in `<DishDetails>`, and another arrow points from the `OnDishSelected` method to the `SelectedDish` property.

4.10.1. Рекомендации к заданию №7

В проекте для отображения иконок можно использовать любую библиотеку, *например*, open-iconic. Подробную информацию об использовании библиотеки – см. <https://www.nsbasic.com/app/OpenIconic.html>

5. Контрольные вопросы

1. В какой вид компилируется клиентский код приложения Blazor Webassembly?
2. Чем отличается одностраничное приложение (SPA –Single Page Application) от обычного Веб-приложения?
3. Что такое CascadingParameter?

4. На странице (родительский компонент) размещен компонент (дочерний компонент). Как в родительском компоненте обработать событие, произошедшее в дочернем компоненте?
5. Для чего используется метод `StateHasChanged`?
6. Как используется компонент `<AuthorizeView/>` ?
7. Для чего используется класс `AuthenticationStateProvider`?
8. Для чего используется параметр `Task<AuthenticationState>`?
9. Как ограничить доступ неавторизованных пользователей к странице Blazor?