

## Лабораторная работа №8

### Web API, Blazor (6 часов)

#### 1. Цель работы.

Знакомство с API контроллерами.

Изучение проекта Blazor.

Получение навыков в создании компонентов Razor.

**Время выполнения работы: 6 часов**

#### 2. Общие сведения.

#### 3. Выполнение работы

##### 3.1. Исходные данные

Используйте проект из лабораторной работы №7.

##### 3.2. Задание №1

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

При создании проекта в диалоге «Additional information» отметьте пункт «ASP.NET Core hosted»

В созданном проекте **XXX.Blazor.Client** найдите в файле Program.cs регистрацию компонента «app» и сервиса HttpClient.

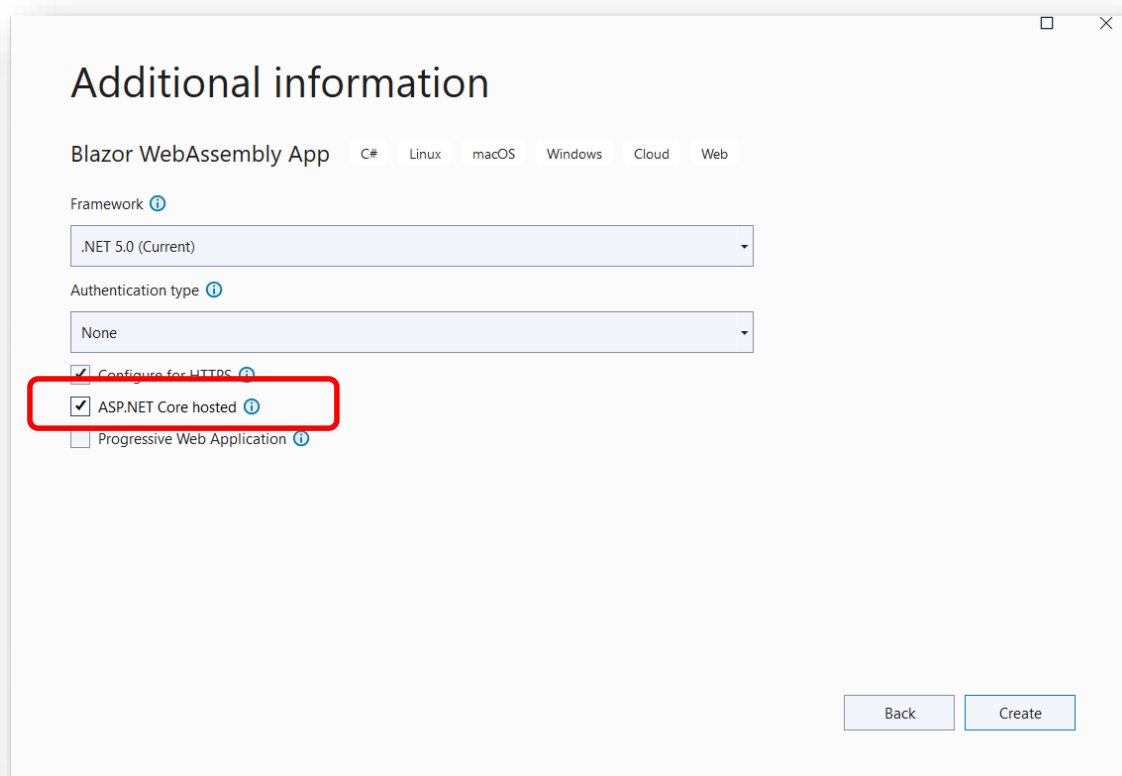
Найдите корневой компонент приложения app.razor. Откройте его. Познакомьтесь с использованием компонента Router.

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.



Additional information

Blazor WebAssembly App C# Linux macOS Windows Cloud Web

Framework ⓘ

.NET 5.0 (Current)

Authentication type ⓘ

None

☒ Configure for HTTPS ⓘ

☒ ASP.NET Core hosted ⓘ

☐ Progressive Web Application ⓘ

Back Create

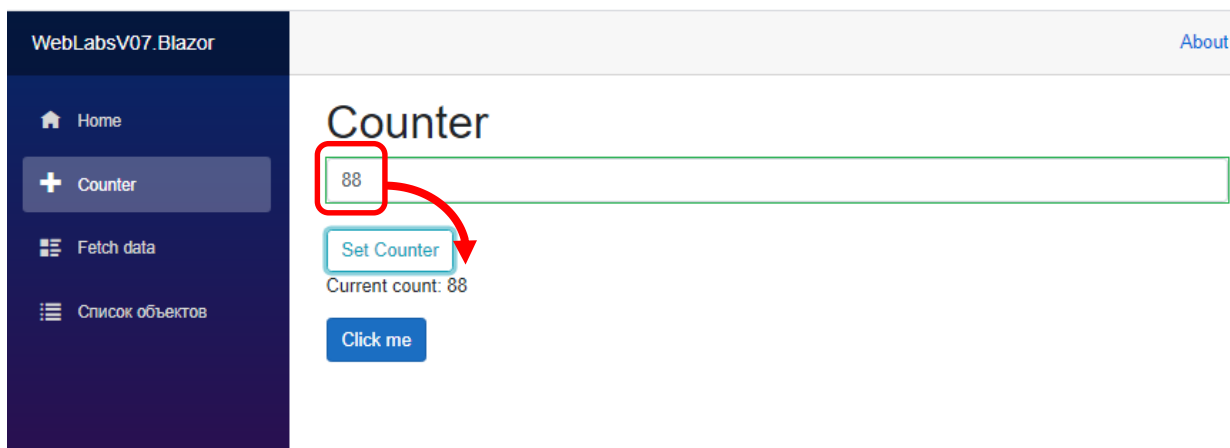
### 3.3. Задание №2

Сделайте проект **XXX.Blazor.Server** стартовым проектом.

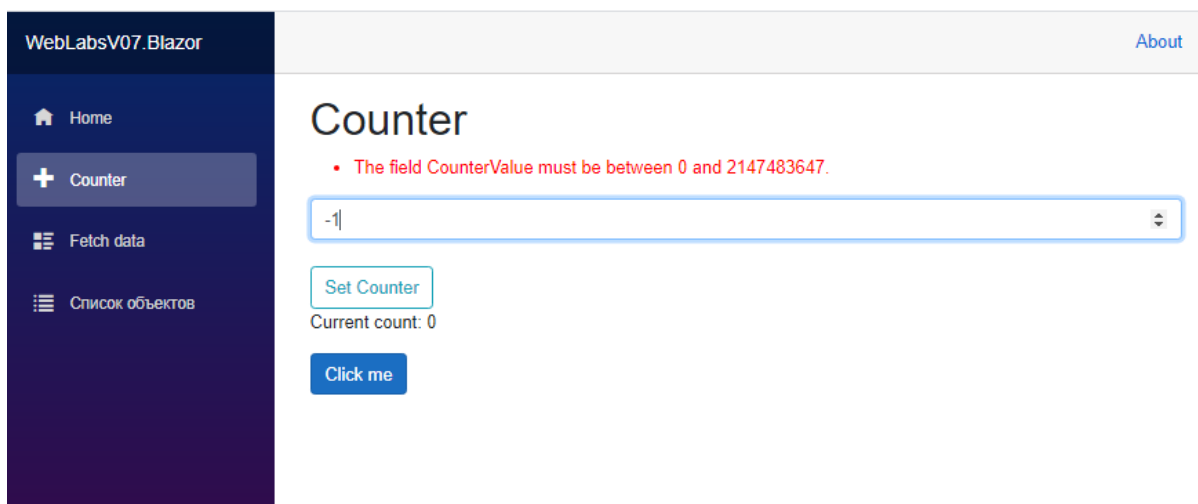
Запустите проект Blazor. В окне браузера запустите режим разработчика (клавиша F12). Перейдите к вкладке Network. Убедитесь, что при переключении страниц на сервер отправляются запросы Http только для чтения данных `WeatherForecast`.

### 3.4. Задание №3

На странице Counter поместите поле ввода и кнопку «Установить». При нажатии на кнопку счетчику должно присваиваться значение, введенное в поле ввода.



Введенное значение должно быть целым положительным числом. Предусмотреть валидацию с выводом соответствующего сообщения об ошибке.



#### 3.4.1. Рекомендации к заданию №3

Используйте компонент `EditForm`. Инициализацию счетчика выполните в обработчике события формы `OnValidSubmit`.

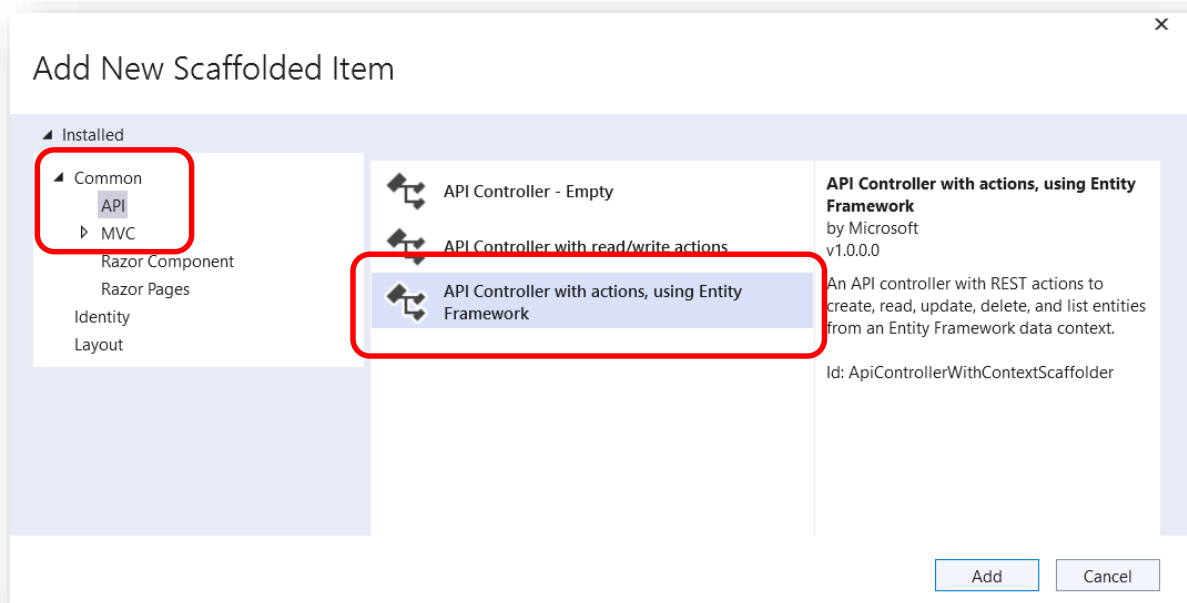
Для ввода значения счетчика используйте компонент `<InputNumber>`

Для задания правил валидации опишите вспомогательный класс модели, содержащий свойство типа `int`. Используйте атрибут `[Range]`.

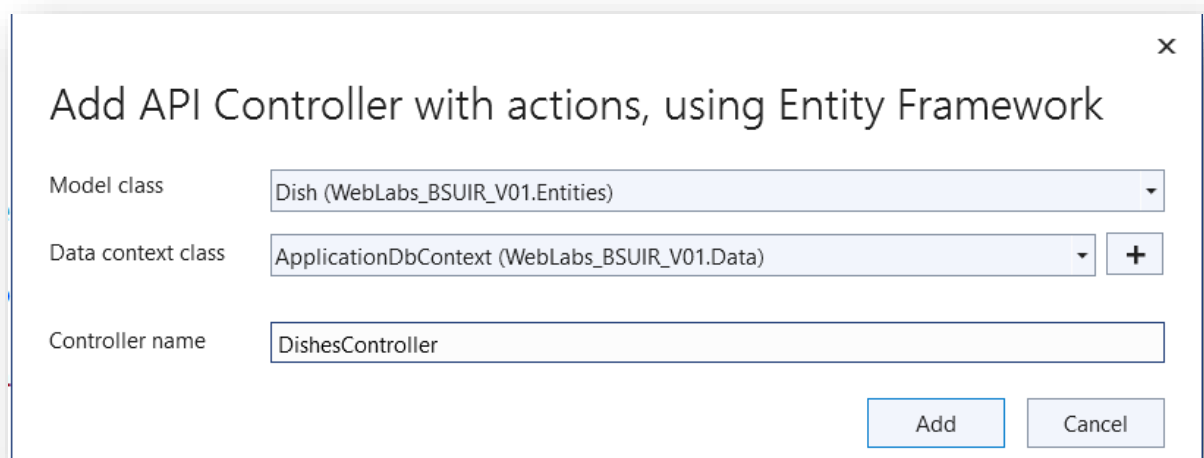
Для валидации введенного значения счетчика и вывода сообщения об ошибке используйте компоненты `<DataAnnotationsValidator/>` и `<ValidationSummary/>`.

### 3.5. Задание №4

В проекте **XXX.Blazor.Server** создайте контроллер API, реализующий CRUD функции для объектов вашей предметной области (в предлагаемых примерах – Dish).



Укажите класс модели вашей предметной области и класс контекста базы данных:



Ознакомьтесь с содержимым с созданного контроллера.

В методе Get реализуйте возможность фильтрации по группе. Номер группы должен передаваться в строке запроса, например, «?group=3».

Обратитесь к созданному контроллеру из браузера (пример запроса для проверки: <https://localhost:44318/api/dishes> ). В браузере должны отобразиться данные в формате Json.

Проверьте фильтрацию объектов по группе (пример запроса для проверки: <https://localhost:44318/api/dishes?group=4> )

#### 3.5.1. Рекомендации к заданию №4

Класс контекста базы данных и классы сущностей используйте из проекта предыдущих лабораторных работ. Для этого достаточно в новом проекте сделать ссылку на проект из предыдущих работ.

Строку подключения используйте также из проекта предыдущих работ.

Зарегистрируйте контекст базы данных в качестве сервиса в классе Startup.

Скопируйте папку Images из предыдущих работ в проект XXX.Blazor.Server

### 3.6. Задание №5

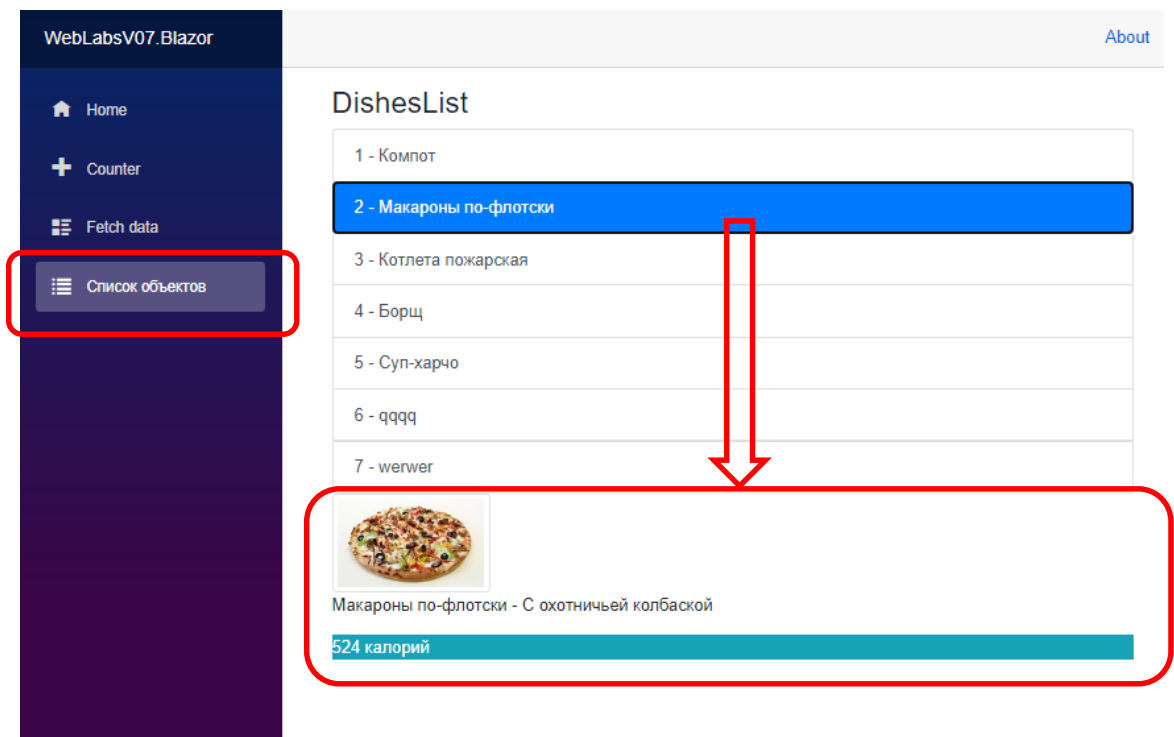
Создайте страницу (ApiDemo.razor), выводящую список объектов, полученных с помощью контроллера API, описанного в задании №4.

Для перехода на страницу добавьте в меню сайта пункт «Список объектов».

Список должен содержать порядковый номер и название объекта. При клике на объект необходимо вывести информацию о выбранном объекте: название, описание и количественную характеристику объекта.

Предусмотреть выделение стилем текущего выбранного объекта в списке.

Пример вида страницы:



Список объектов и подробная информация о выбранном объекте должны быть реализованы **в виде отдельных компонентов**.

Получение списка объектов и поиск объекта должны осуществляться **на родительской** странице (ApiDemo.razor).

Пример разметки страницы ApiDemo:

```
<div class="container">
  <DishesList @bind-Dishes="@dishes"
              SelectedObjectChanged="ShowDetails"></DishesList>
  <DisDetails @bind-Dish="@SelectedDish"></DisDetails>
</div>
```

В приведенном примере:

<DishesList> - компонент вывода списка объектов

<DishDetails> - компонент вывода подробной информации об объекте

### 3.6.1. Рекомендации к заданию №7

Для отправки запросов к API внедрите на страницу объект **HttpClient**.

Для десериализации данных Json, получаемых от API, используйте метод **GetFromJsonAsync** объекта HttpClient.

Для вывода информации добавьте в проект папку Models, содержащую модели представления:

- класс `ListViewModel` – содержит данные для отображения списка объектов
- класс `DetailsViewModel` – содержит подробную информацию о выбранном объекте

Формат `Json` использует `camel-style` для именования свойств, т.е. все названия начинаются со строчной (маленькой) буквы. Свойства объектов `C#` именуются с прописной (заглавной) буквы. Для разрешения конфликта при десериализации воспользуйтесь одним из следующих методов:

- добавьте в метод `DeserializeAsync` параметр `new JsonSerializerOptions { PropertyNameCaseInsensitive=true }`.
- или в классе модели перед свойствами поставьте атрибуты `JsonPropertyName`, например `[JsonPropertyName("dishId")]`

Второй способ более производительный.

Для оформления списка объектов можно воспользоваться компонентом `bootstrap list-group` (см. <https://getbootstrap.com/docs/4.5/components/list-group/#links-and-buttons>).

Получение списка объектов и поиск объекта должны осуществляться на родительской странице (`ApiDemo.razor`). Выполните обмен данными между страницей `ApiDemo.razor` и дочерними компонентами (компонент вывода списка и компонент вывода подробной информации об объекте). Также в компоненте вывода списка объектов обработайте событие клика для передачи `id` выбранного объекта родительской странице.

В базе данных хранится имя файла изображения, а не путь к нему. Поэтому в компоненте вывода подробной информации предусмотрите приватное свойство, предоставляющее путь к файлу, например:

```
string imageSrc
{
    get
    {
        return $"images/{Dish.Image}";
    }
}
```

### Примечание:

Если при обращении к API сервер откажет в доступе, то в классе Startup, в методе ConfigureServices, нужно добавить политику, разрешающую любой доступ:

```
// CORS - политика "Разрешен любой доступ"
services.AddCors(opt =>
    opt.AddPolicy("AllowAny",
        builder => {
            builder.AllowAnyMethod()
                .AllowAnyHeader()
                .AllowAnyOrigin();
        }));
```

и в методе Configure добавить компонент CORS с созданной политикой:

```
app.UseCors("AllowAny");
```