Blazor

Course OutLine

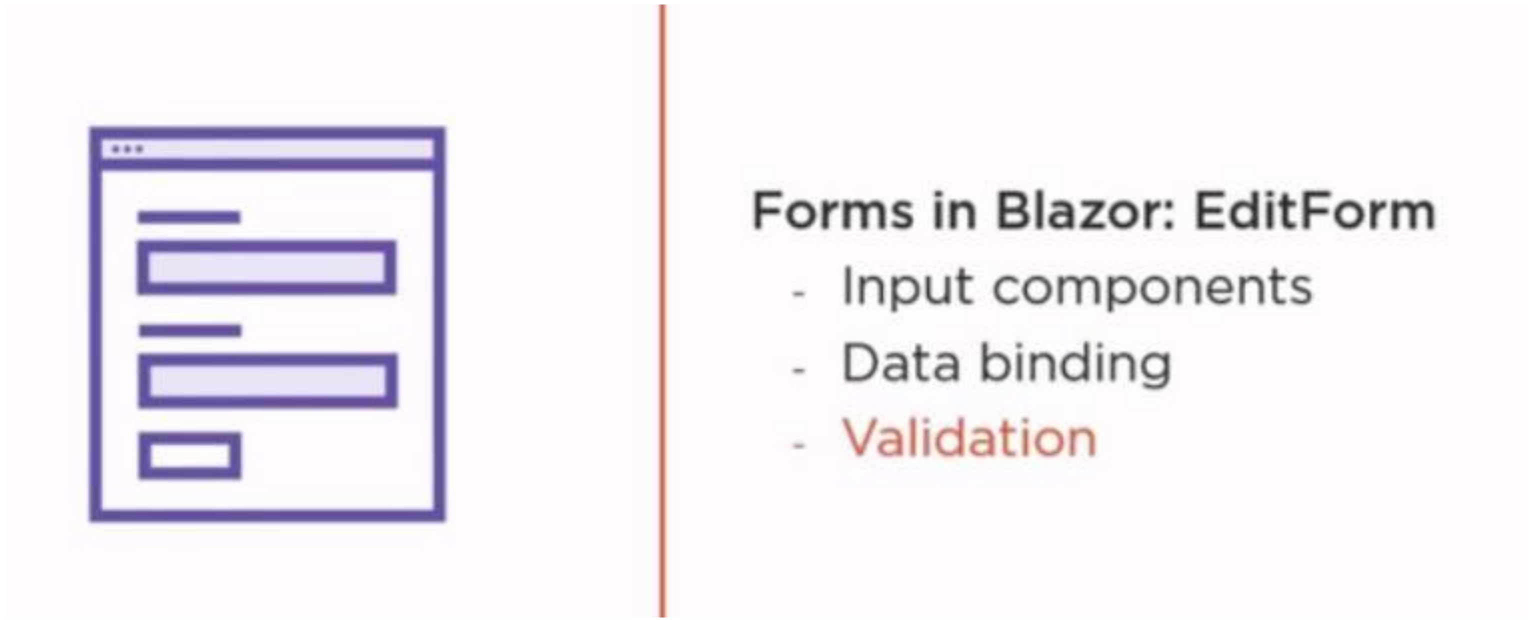
- Part1
  - What is Blazor?
  - What is WebAssembly?
  - Why blazor?
  - Hosting models
    - Advantage & disadva. Of each model
  - Development environment
  - Creating a Blazor App with Visual Studio 2019
  - Project structure
  - Reviewing the generated code
  - Creating a Blazor App using the .net core cli

- Part2 : Component
  - What is Component
  - Razor Directives
  - Component Code
  - Component Parameters
  - Rout Parameteres
  - attribute splatting
  - **Event handlers**
  - Assignment: Create product component
  - DataBinding
  - Adopt an animal by raising and handling an event and some JavaScript interop
  - Routing

- Part3 : Forms & Validation
    - Render Conditionally
    - Injecting and passing down object
    - LifeCycle Methods
    - Two-Way Data binding
    - Event callback
    - Form Component
    - Moving the animal data to a service
    - Assignment: Move the product data to a service
    - Create the animal form
    - Assignment: Create the product form
    - Add validation to the animal form
    - Assignment: Add validation to the product form

- Part4： Working With backend Services
    - Create tables and context
    - Assignment: Add product to the database
    - Create the animal and product webapi
    - Using the animal webapi in the animal component
    - Assignment: Use the Product WebAPI in the product components

Forms-Validation

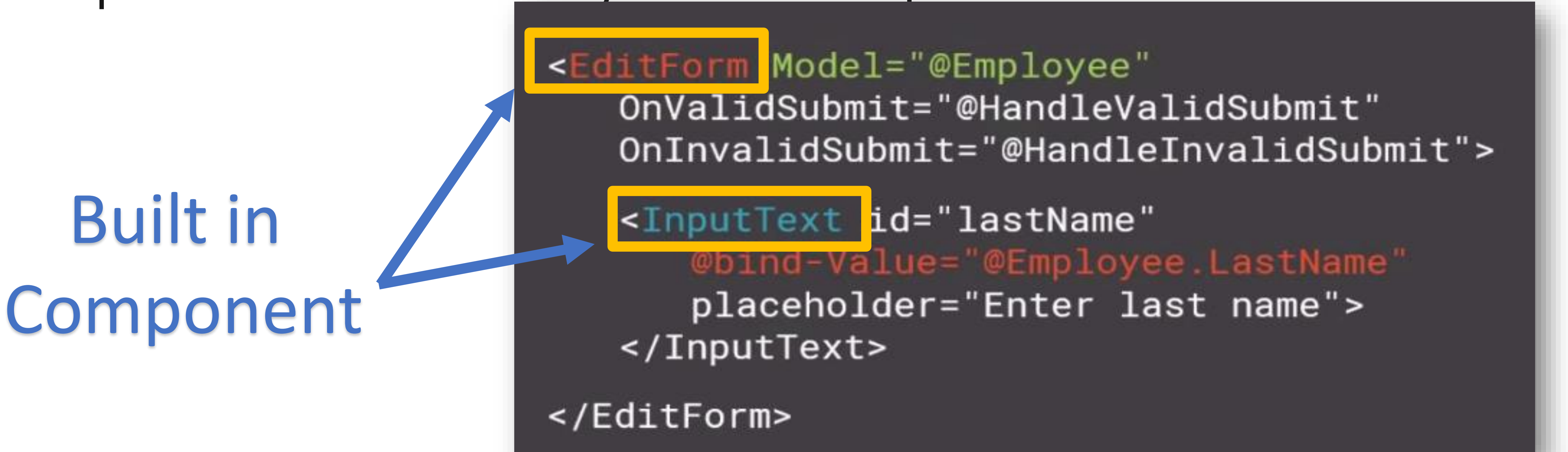


Forms in Blazor: EditForm

- Input components
- Data binding
- Validation

Forms

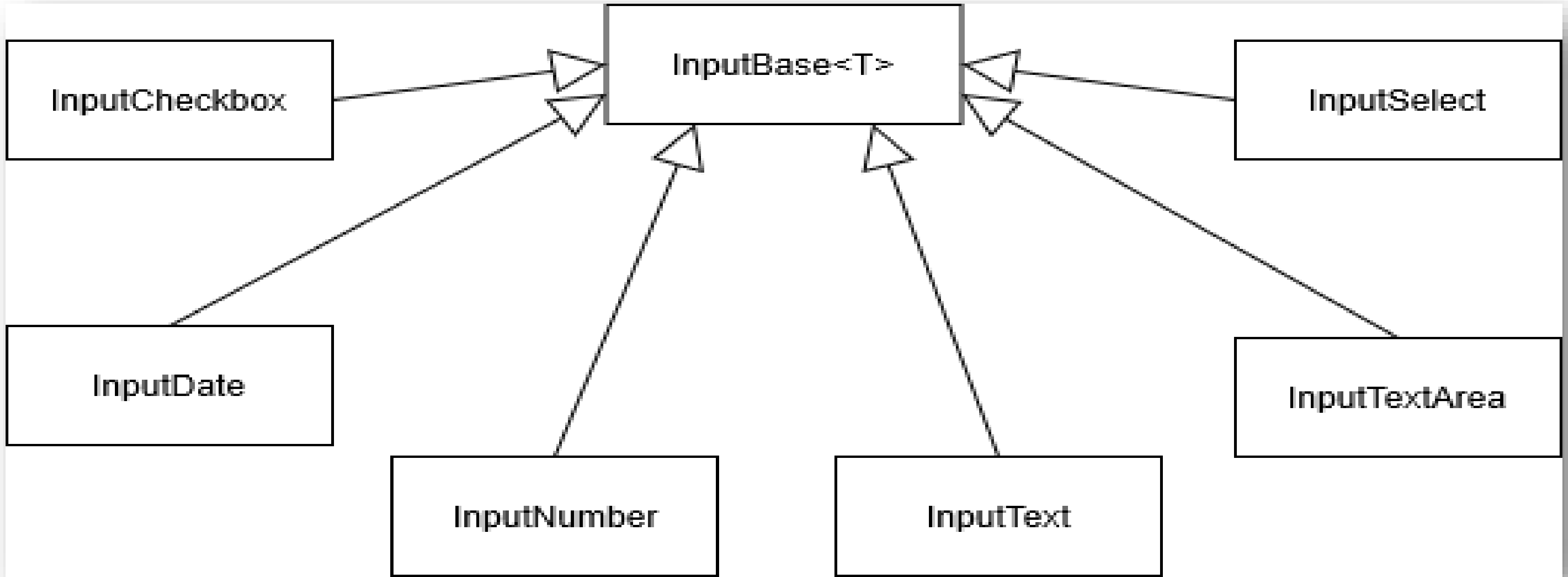
- A form is defined using the EditForm component.
- The EditForm component is Blazor's approach to managing user-input in a way that makes it easy to perform validation against user input.



```
<EditForm Model="@Employee"
    OnValidSubmit="@HandleValidSubmit"
    OnInvalidSubmit="@HandleInvalidSubmit">

    <InputText id="lastName"
        @bind-Value="@Employee.LastName"
        placeholder="Enter last name">
    </InputText>

</EditForm>
```

Built in input Component

- A set of built-in components are available to receive and validate user input.
- Inputs are validated when they're changed and when a form is submitted.

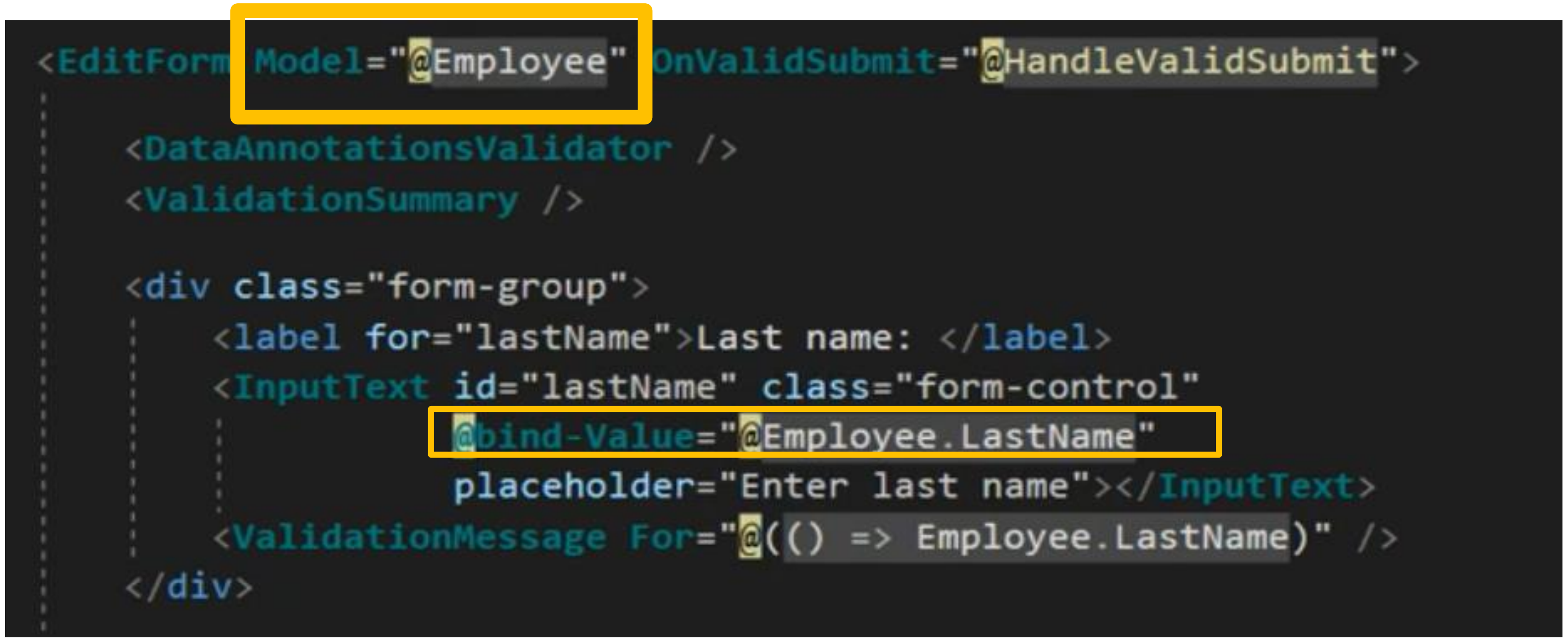


Built in input Component

Input component	Rendered as…
InputCheckbox	<input type="checkbox">
InputDate<TValue>	<input type="date">
InputFile	<input type="file">
InputNumber<TValue>	<input type="number">
InputRadio	<input type="radio">
InputRadioGroup	<input type="radio">
InputSelect<TValue>	<select>
InputText	<input>
InputTextArea	<textarea>

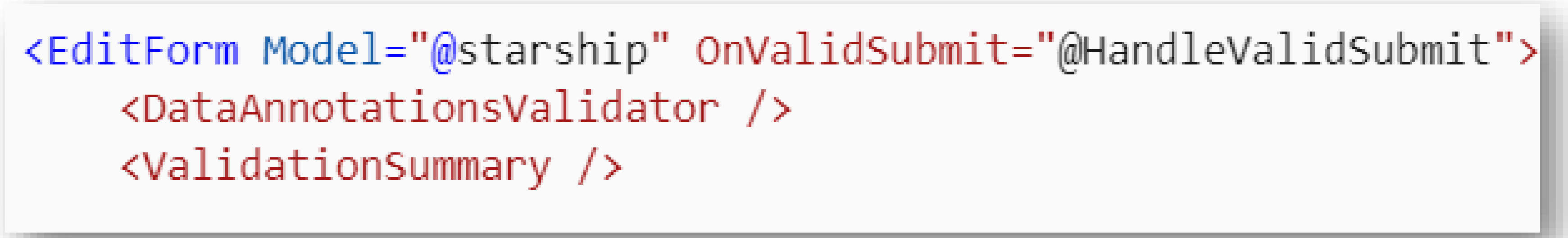
Form Binding

- There are built in component response to create form
- **Model** parameter provides the component with a context it can work with to enable user-interface binding and determine whether or not the user’s input is valid.



```
<EditForm Model="@Employee" OnValidSubmit="@HandleValidSubmit">

    <DataAnnotationsValidator />
    <ValidationSummary />

    <div class="form-group">
        <label for="lastName">Last name: </label>
        <InputText id="lastName" class="form-control"
                   @bind-Value="@Employee.LastName"
                   placeholder="Enter last name"></InputText>
        <ValidationMessage For="@(() => Employee.LastName)" />
    </div>
```

Validator components

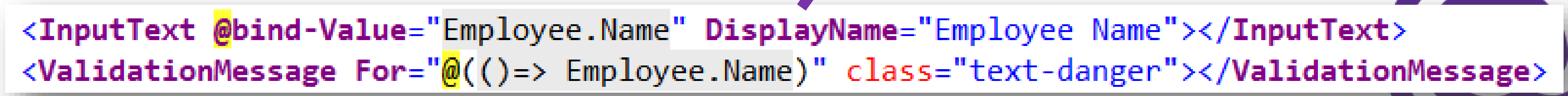


```
<EditForm Model="@starship" OnValidSubmit="@HandleValidSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />
```

- The Blazor framework provides the **DataAnnotationsValidator** component to attach validation support to forms based on validation attributes (data annotations).

- For each property in Model

This value is used when generating error messages when the input value fails to parse correctly.

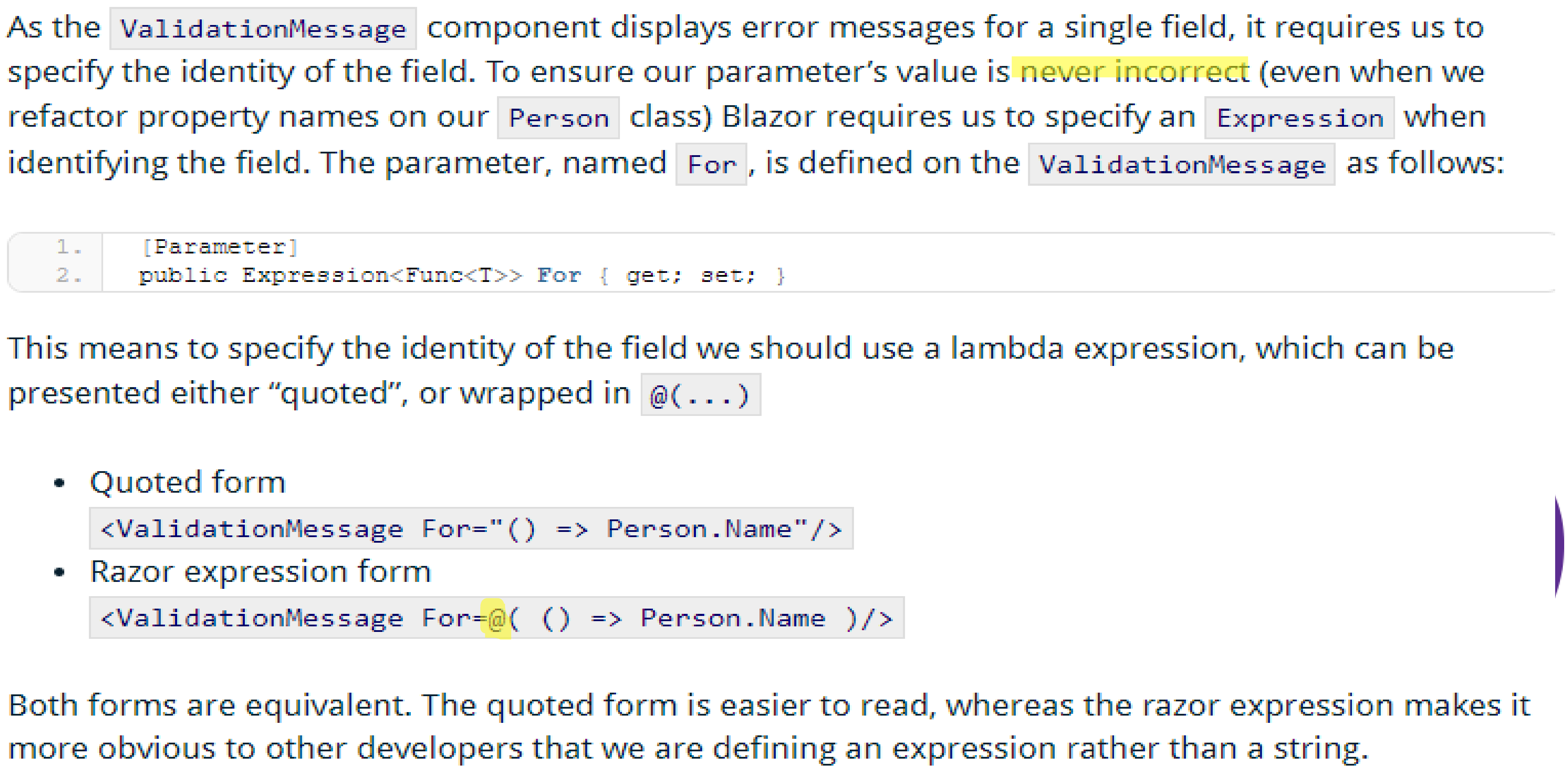


```
<InputText @bind-Value="Employee.Name" DisplayName="Employee Name"></InputText>
<ValidationMessage For="@(()=> Employee.Name)" class="text-danger"></ValidationMessage>
```

Validation Component (Con.)

- **DataAnnotationsValidator** validator component attaches validation support using data annotations (validation mechanism)
- **Displaying validation error message**
  - **ValidationSummary :**to show a comprehensive list of all errors in the form .
  - V**alidationMessage** component to display error messages for a specific input on the form.
- **HandleValidSubmit** is triggered when the form successfully submits (passes validation) to execute some code.

ValidationMessage Component



As the `ValidationMessage` component displays error messages for a single field, it requires us to specify the identity of the field. To ensure our parameter's value is never incorrect (even when we refactor property names on our `Person` class) Blazor requires us to specify an `Expression` when identifying the field. The parameter, named `For`, is defined on the `ValidationMessage` as follows:

```
[Parameter]
public Expression<Func<T>> For { get; set; }
```

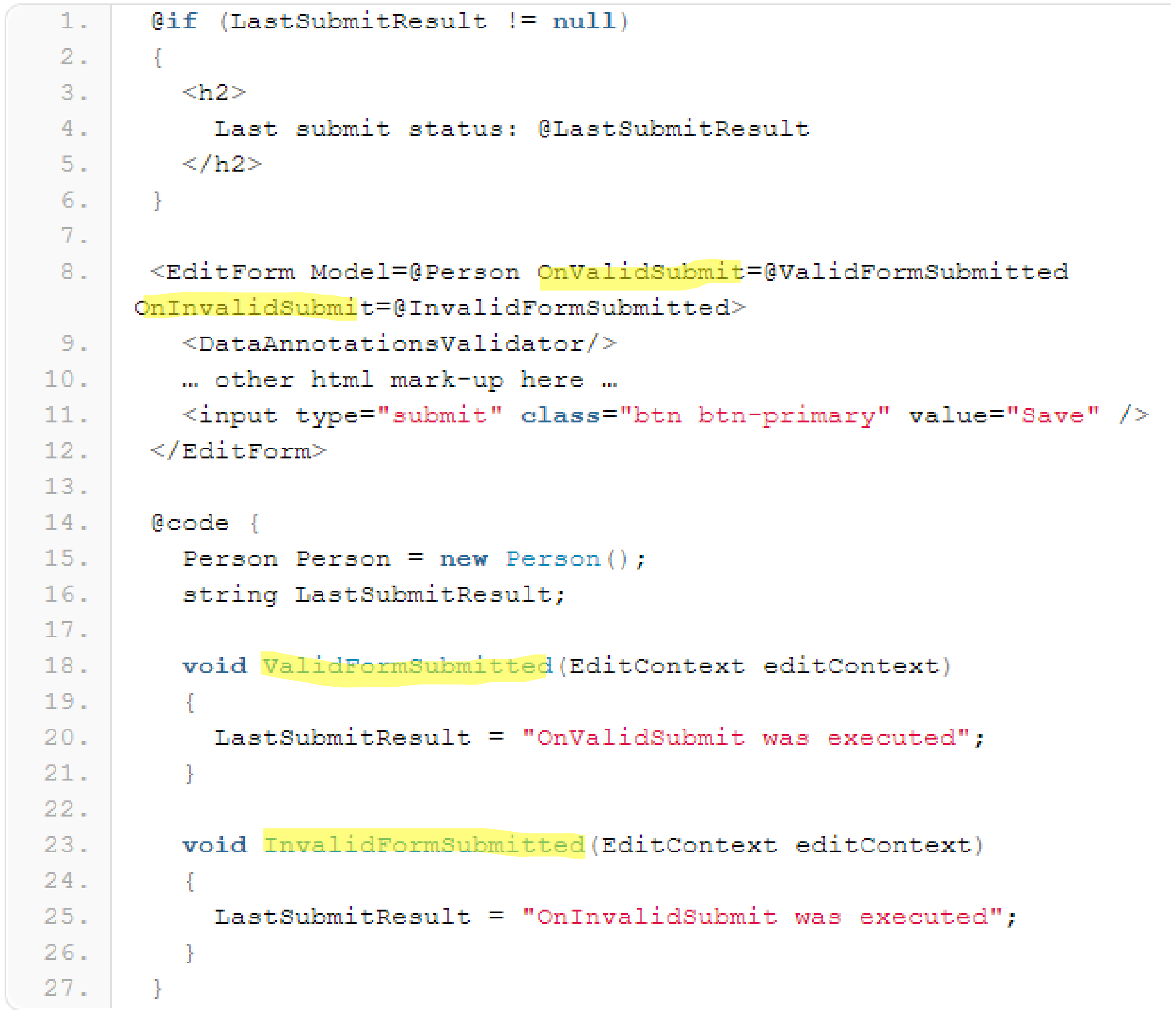
This means to specify the identity of the field we should use a lambda expression, which can be presented either "quoted", or wrapped in `@(...)`

- Quoted form
  `<ValidationMessage For="() => Person.Name"/>`
- Razor expression form
  `<ValidationMessage For=@( () => Person.Name )/>`

Both forms are equivalent. The quoted form is easier to read, whereas the razor expression makes it more obvious to other developers that we are defining an expression rather than a string.

Handling form submission

- There are three events on an EditForm related to form submission.
  - OnValidSubmit
  - OnInvalidSubmit
  - OnSubmit
    - is executed when the form is submitted, regardless of whether it the form passes validation or not.
    - It is possible to check the validity status of the form by executing editContext.Validate() which returns true if the form is valid or false if it is invalid (has validation errors).



```
@if (LastSubmitResult != null)
{
  <h2>
    Last submit status: @LastSubmitResult
  </h2>
}

<EditForm Model=@Person OnValidSubmit=@ValidFormSubmitted OnInvalidSubmit=@InvalidFormSubmitted>
  <DataAnnotationsValidator/>
  … other html mark-up here …
  <input type="submit" class="btn btn-primary" value="Save" />
</EditForm>

@code {
  Person Person = new Person();
  string LastSubmitResult;

  void ValidFormSubmitted(EditContext editContext)
  {
    LastSubmitResult = "OnValidSubmit was executed";
  }

  void InvalidFormSubmitted(EditContext editContext)
  {
    LastSubmitResult = "OnInvalidSubmit was executed";
  }
}
```

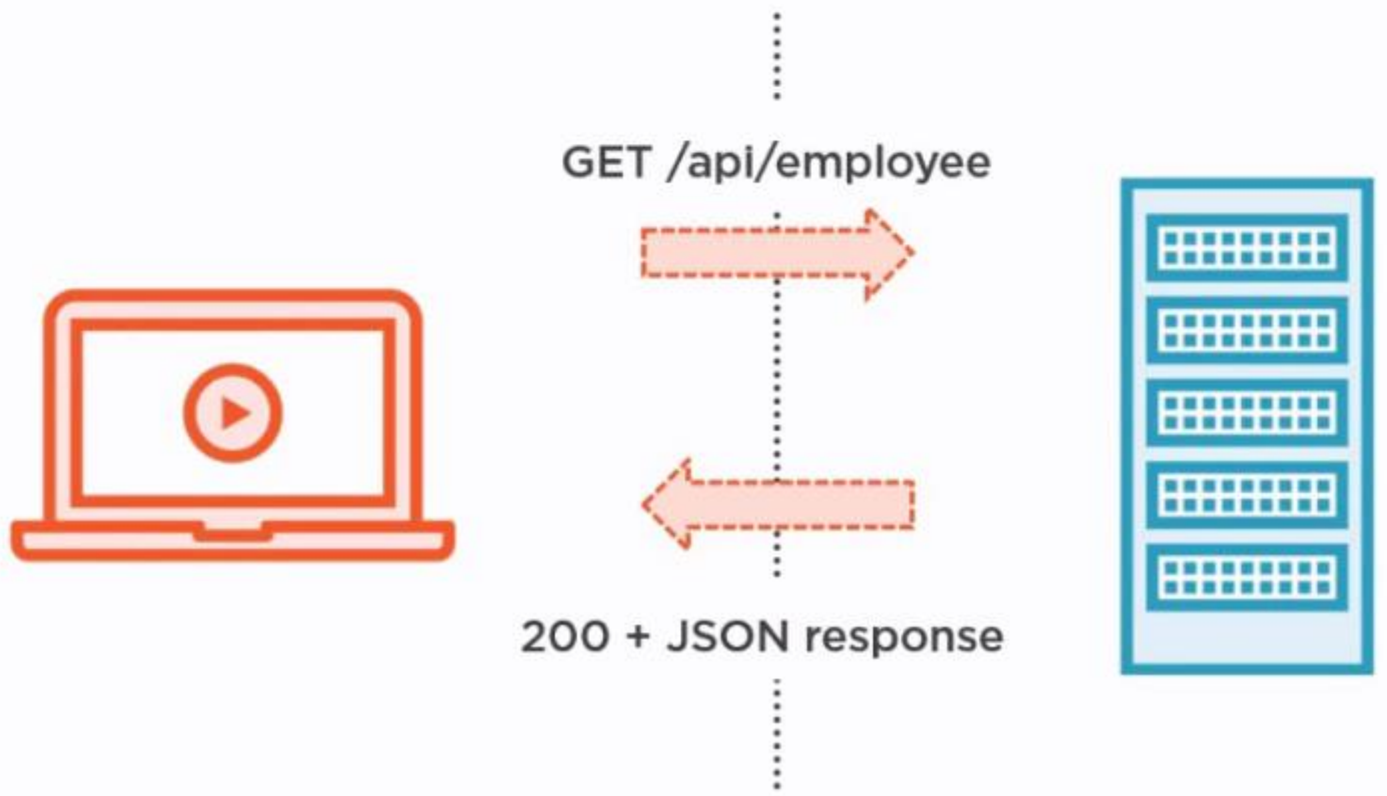
Accessing Real Data from a REST API

Data in Our Blazor App

REST API

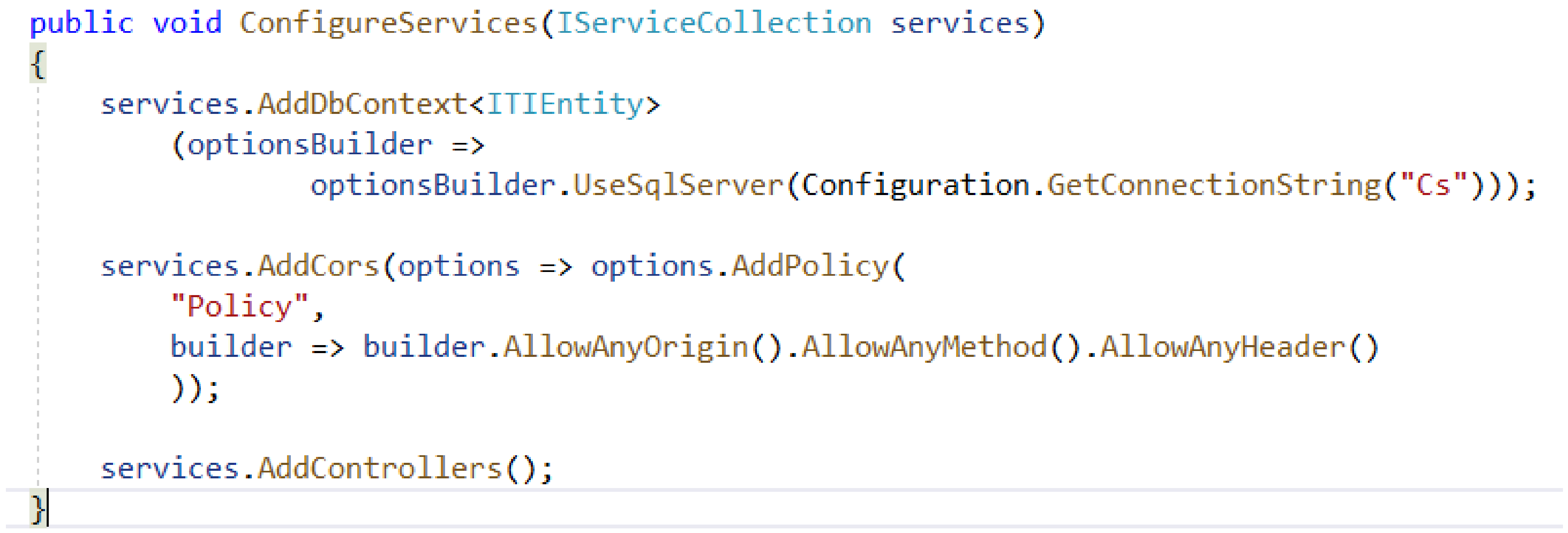
Local storage

Accessing a REST API



Web services startup class

- In configure method write
  - app.UseCors(″Policy″);



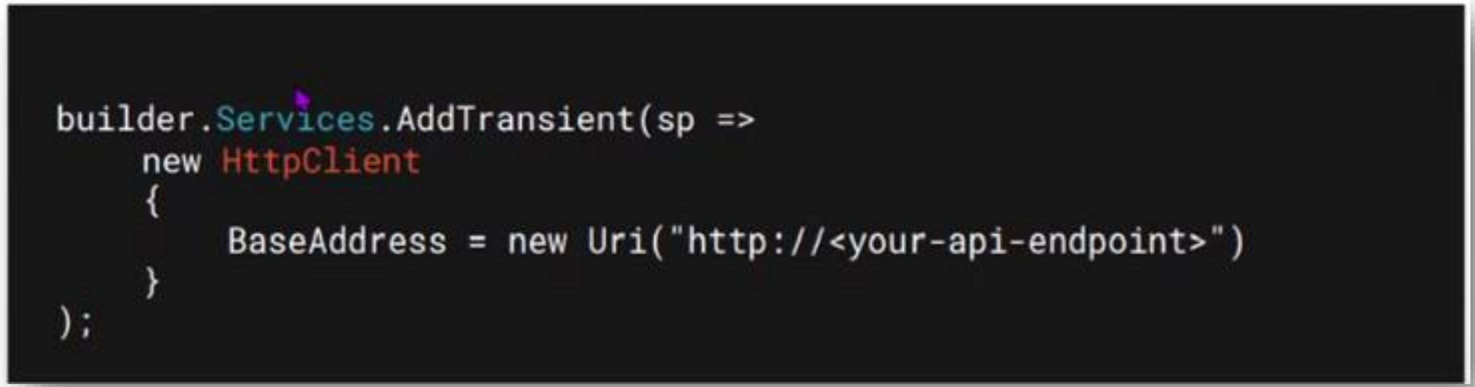
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ITIEntity>
        (optionsBuilder =>
            optionsBuilder.UseSqlServer(Configuration.GetConnectionString("Cs")));

    services.AddCors(options => options.AddPolicy(
        "Policy",
        builder => builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader()
        ));

    services.AddControllers();
}
```

HttpClient

- in a Blazor WebAssembly app, HttpClient is available as a preconfigured service for making requests back to the origin server

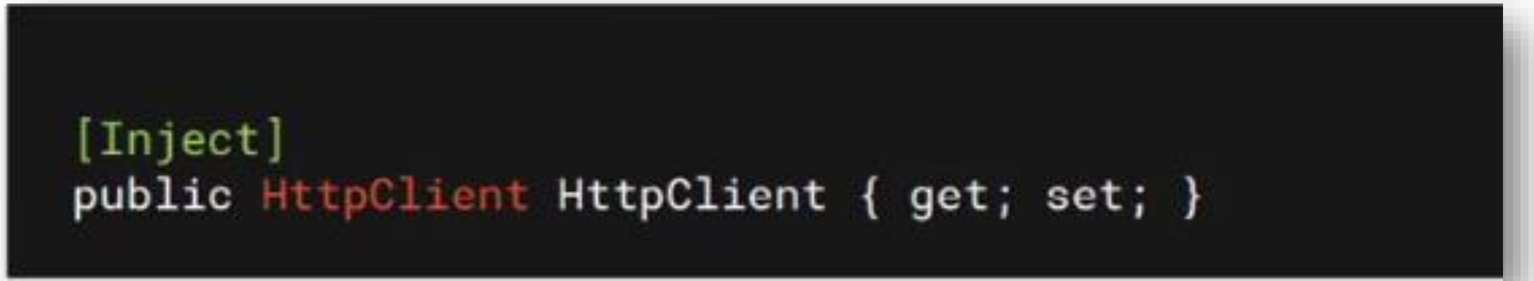
Access Rest Api

- Inject HttpClient Services : ”In Man Function”



```
builder.Services.AddTransient(sp =>
    new HttpClient
    {
        BaseAddress = new Uri("http://<your-api-endpoint>")
    }
);
```

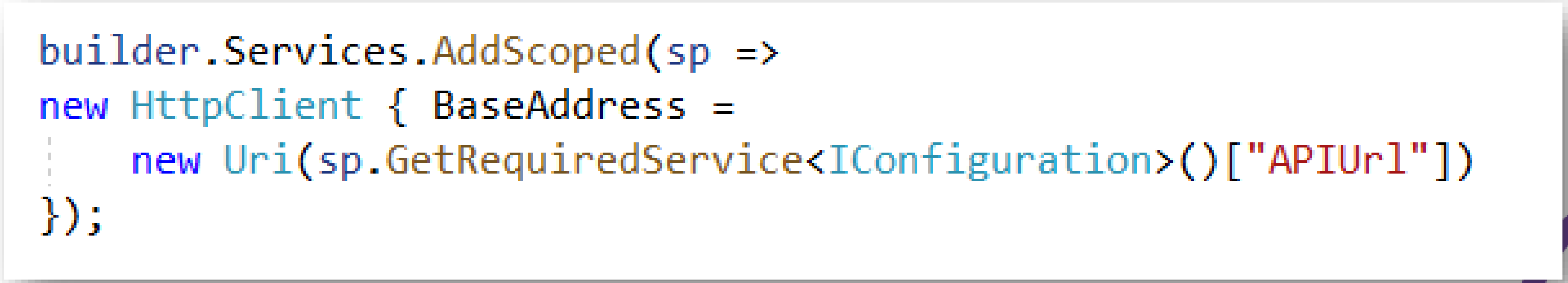
- To Access HttpClient from Any component “in component”



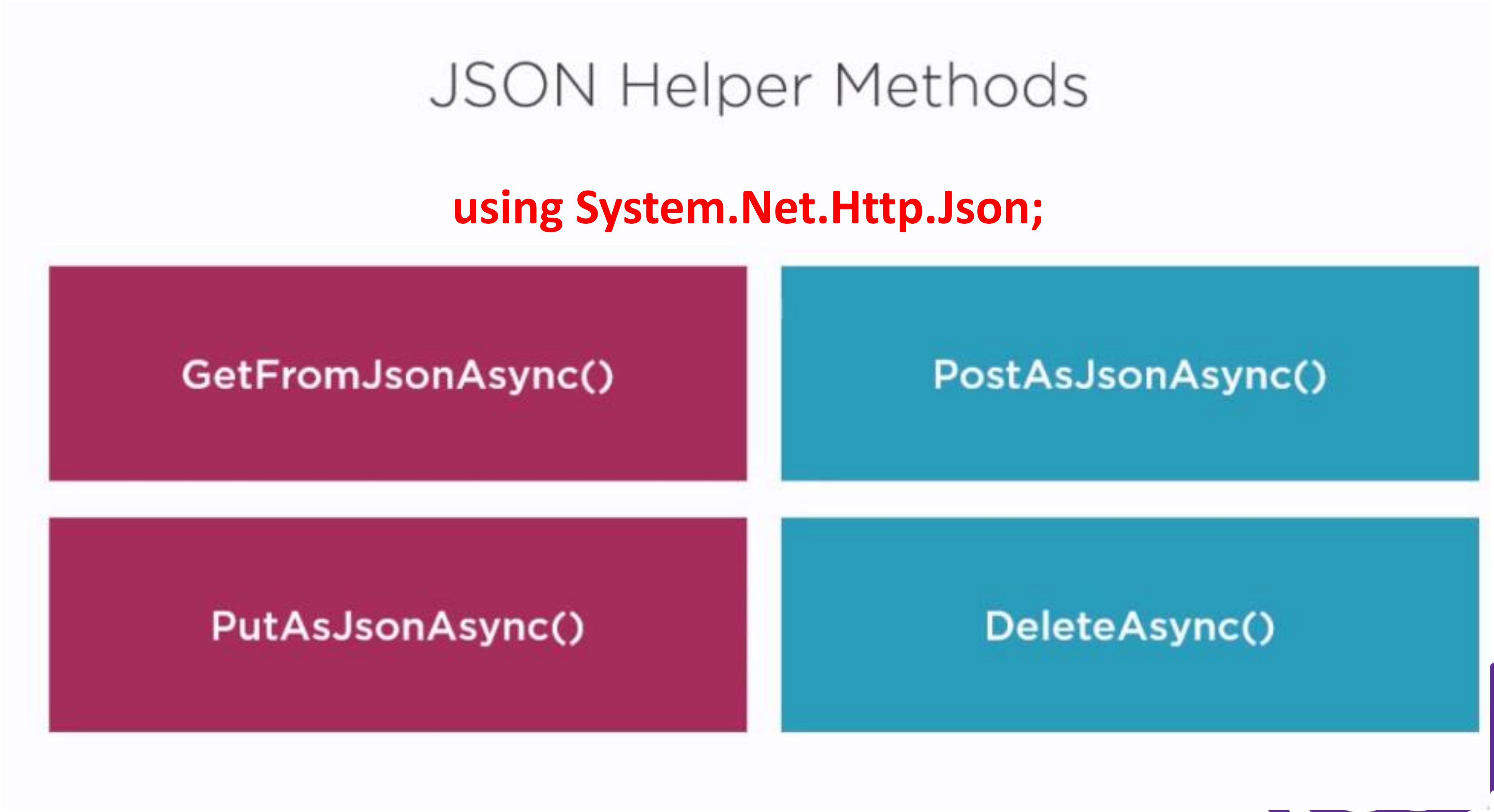
```
[Inject]
public HttpClient HttpClient { get; set; }
```

HttpClient

- in a Blazor WebAssembly app, HttpClient is available as a preconfigured service for making requests back to the origin server
- To Read rest API url from appsetting.json file



```
builder.Services.AddScoped(sp =>
new HttpClient { BaseAddress =
    new Uri(sp.GetRequiredService<IConfiguration>()["APIUrl"])
});
```



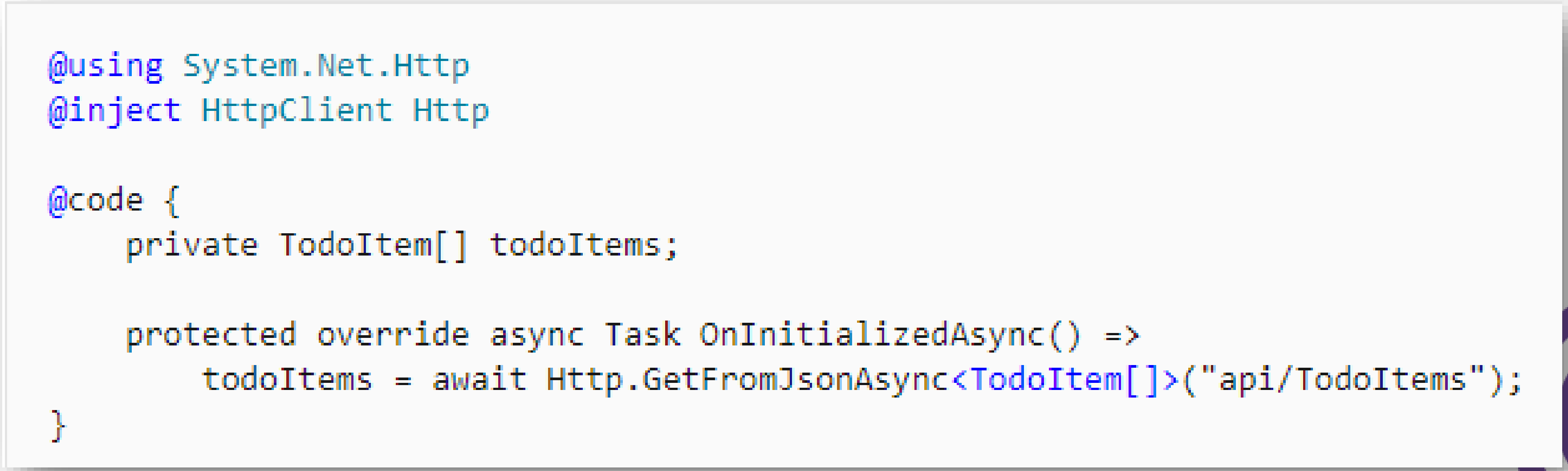
JSON Helper Methods

```
using System.Net.Http.Json;
```

- GetFromJsonAsync()
- PostAsJsonAsync()
- PutAsJsonAsync()
- DeleteAsync()

HttpClient Helper MEthod

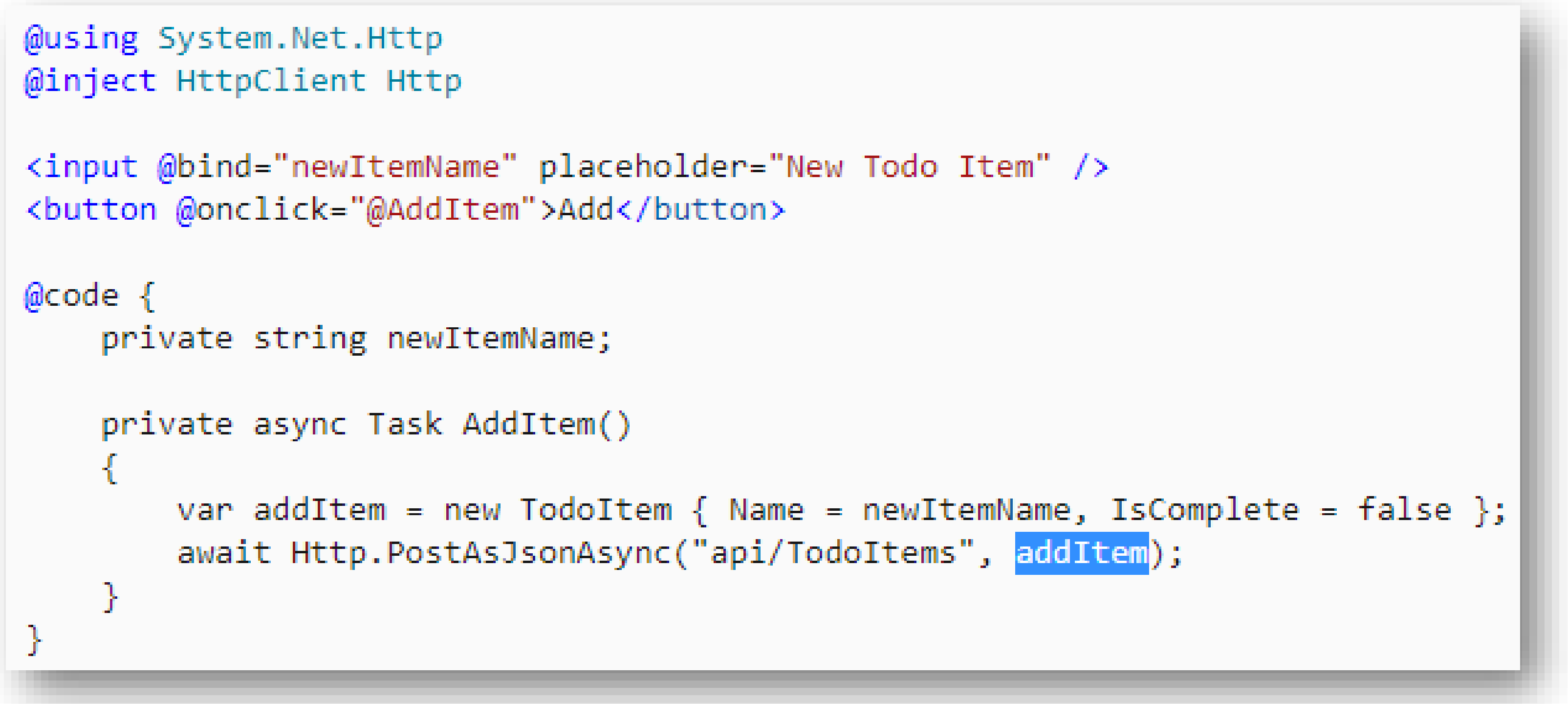
- **GetFromJsonAsync**:
  - Sends an HTTP GET request and parses the JSON response body to create an object



```
@using System.Net.Http
@inject HttpClient Http

@code {
    private TodoItem[] todoItems;

    protected override async Task OnInitializedAsync() =>
        todoItems = await Http.GetFromJsonAsync<TodoItem[]>("api/TodoItems");
}
```

HttpClient Helper Method(con.)

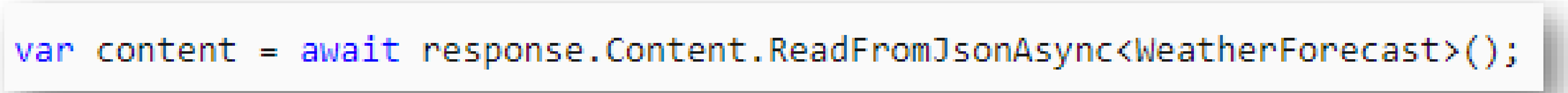
- **PostAsJsonAsync:**
  - Sends a POST request to the specified URI containing the value serialized as JSON in the request body.



```
@using System.Net.Http
@inject HttpClient Http

<input @bind="newItemName" placeholder="New Todo Item" />
<button @onclick="@AddItem">Add</button>

@code {
    private string newItemName;

    private async Task AddItem()
    {
        var addItem = new TodoItem { Name = newItemName, IsComplete = false };
        await Http.PostAsJsonAsync("api/TodoItems", addItem);
    }
}
```

HttpClient Helper Method(con.)

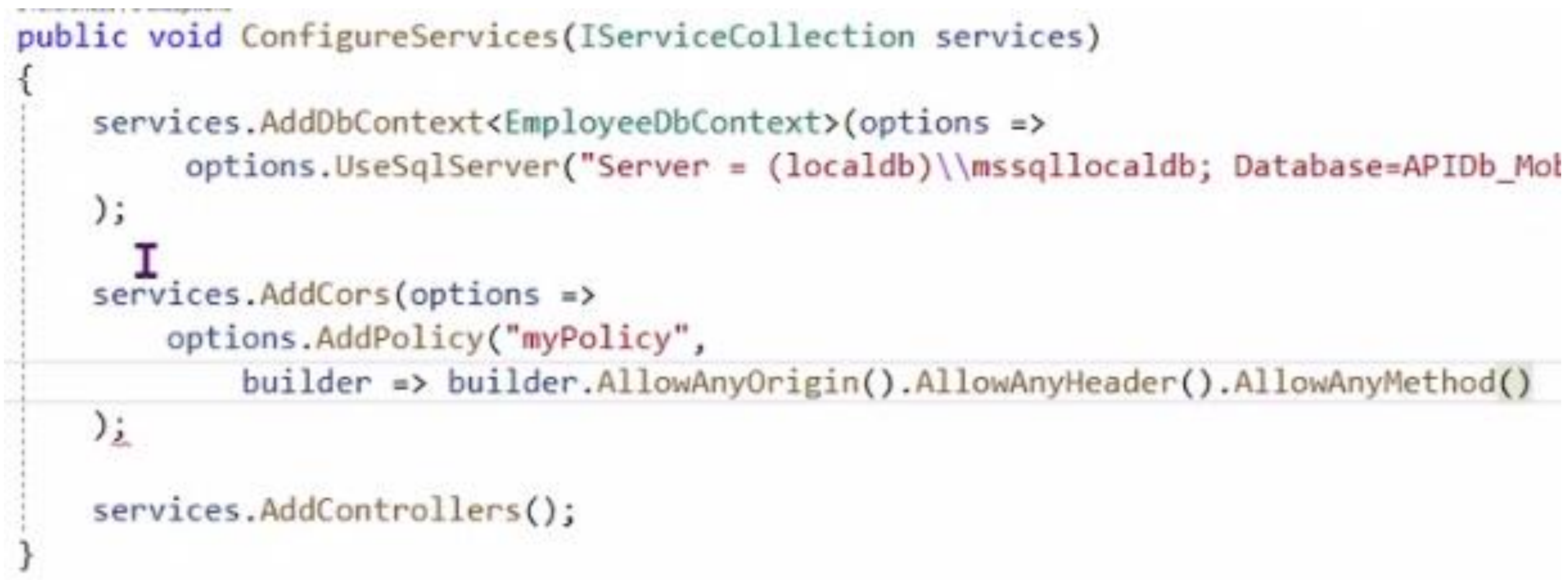
- **PostAsJsonAsync:**
  - PostAsJsonAsync return an HttpResponseMessage. To deserialize the JSON content from the response message, use the ReadFromJsonAsync<T> extension method:



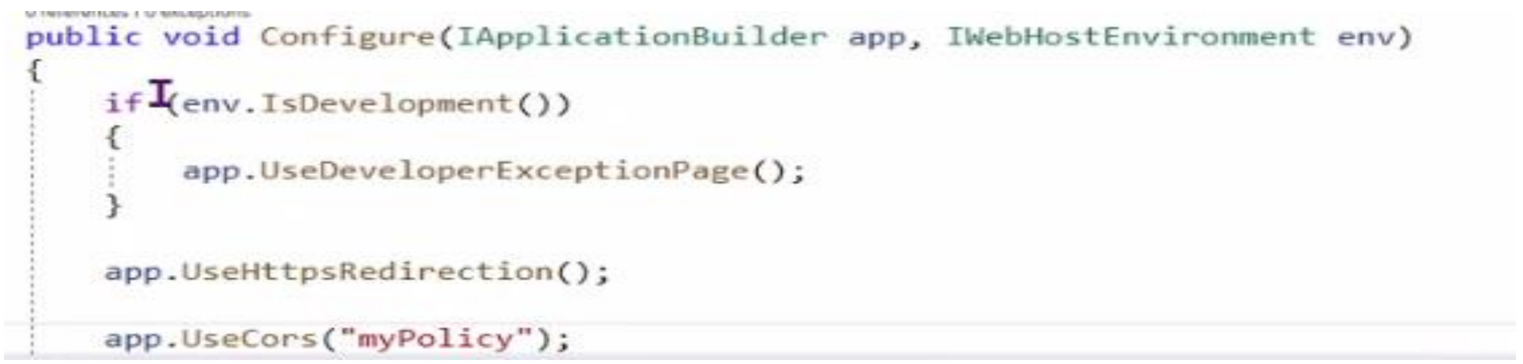
```
var content = await response.Content.ReadFromJsonAsync<WeatherForecast>();
```

- **PutAsJsonAsync:**
  - Sends an HTTP PUT request, including JSON-encoded content.
- **DeleteAsync**
  - is used to send an HTTP DELETE request to a web API.

Web Api Setting

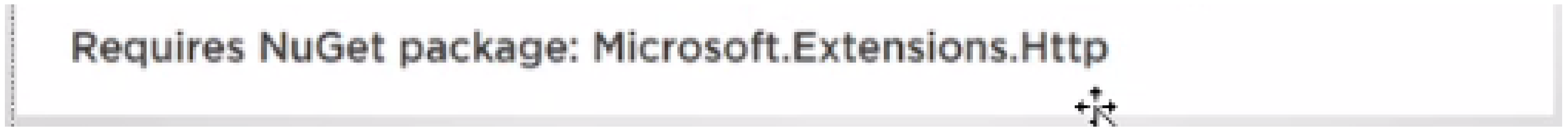


```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<EmployeeDbContext>(options =>
         options.UseSqlServer("Server = (localdb)\\mssqllocaldb; Database=APIDb_Mol
    );

    services.AddCors(options =>
        options.AddPolicy("myPolicy",
            builder => builder.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod()
    );

    services.AddControllers();
}
```

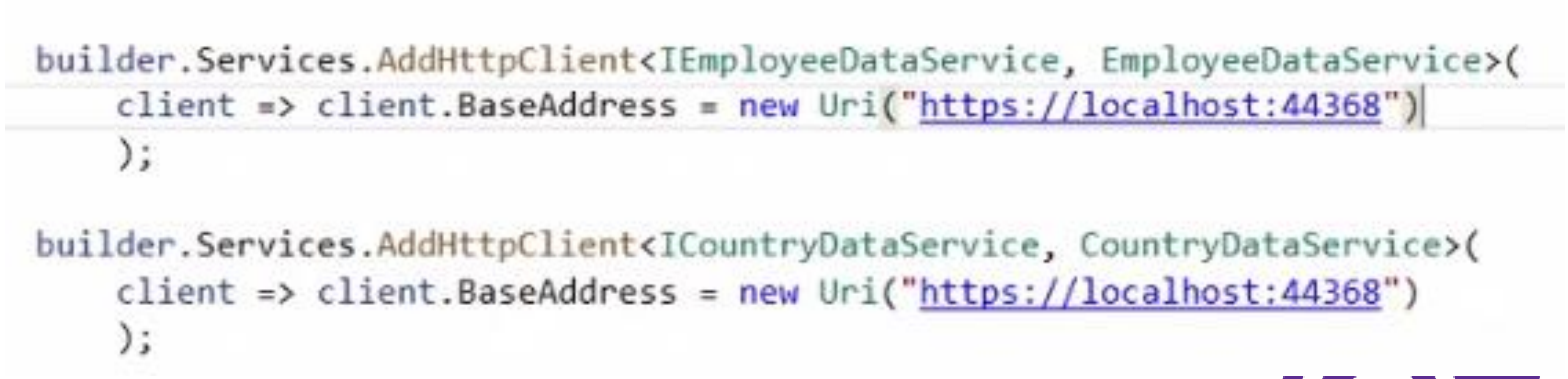


```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseHttpsRedirection();

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

HttpClientFactory



Requires NuGet package: Microsoft.Extensions.Http

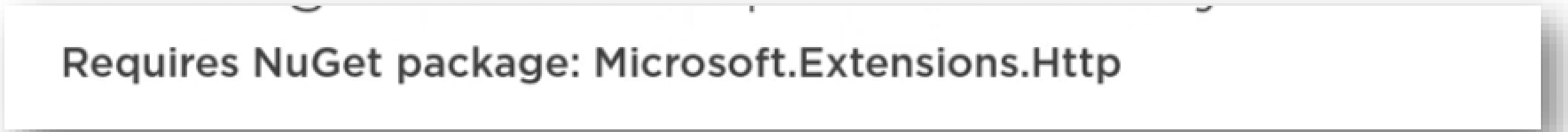


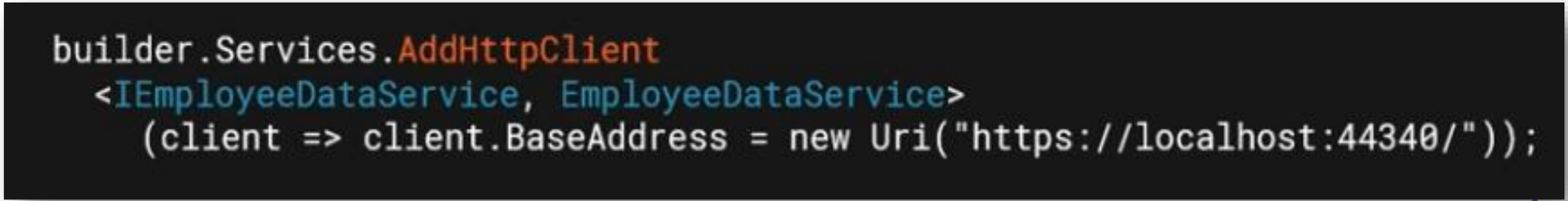
```
builder.Services.AddHttpClient<IEmployeeDataService, EmployeeDataService>(
    client => client.BaseAddress = new Uri("https://localhost:44368")
    );

builder.Services.AddHttpClient<ICountryDataService, CountryDataService>(
    client => client.BaseAddress = new Uri("https://localhost:44368")
    );
```

HttpClientFactory

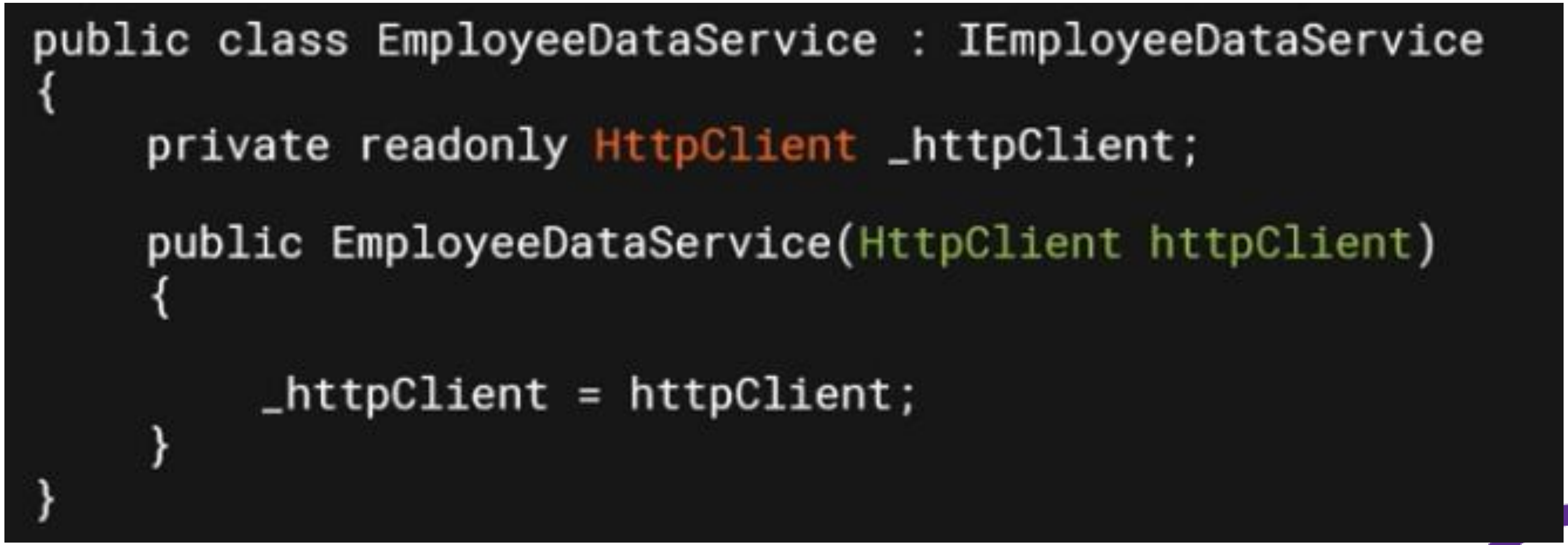
Used to configure and create HttpClient instances in a central location

HttpClientFactory


Requires NuGet package: Microsoft.Extensions.Http


```
builder.Services.AddHttpClient
  <IEmployeeDataService, EmployeeDataService>
    (client => client.BaseAddress = new Uri("https://localhost:44340/"));
```

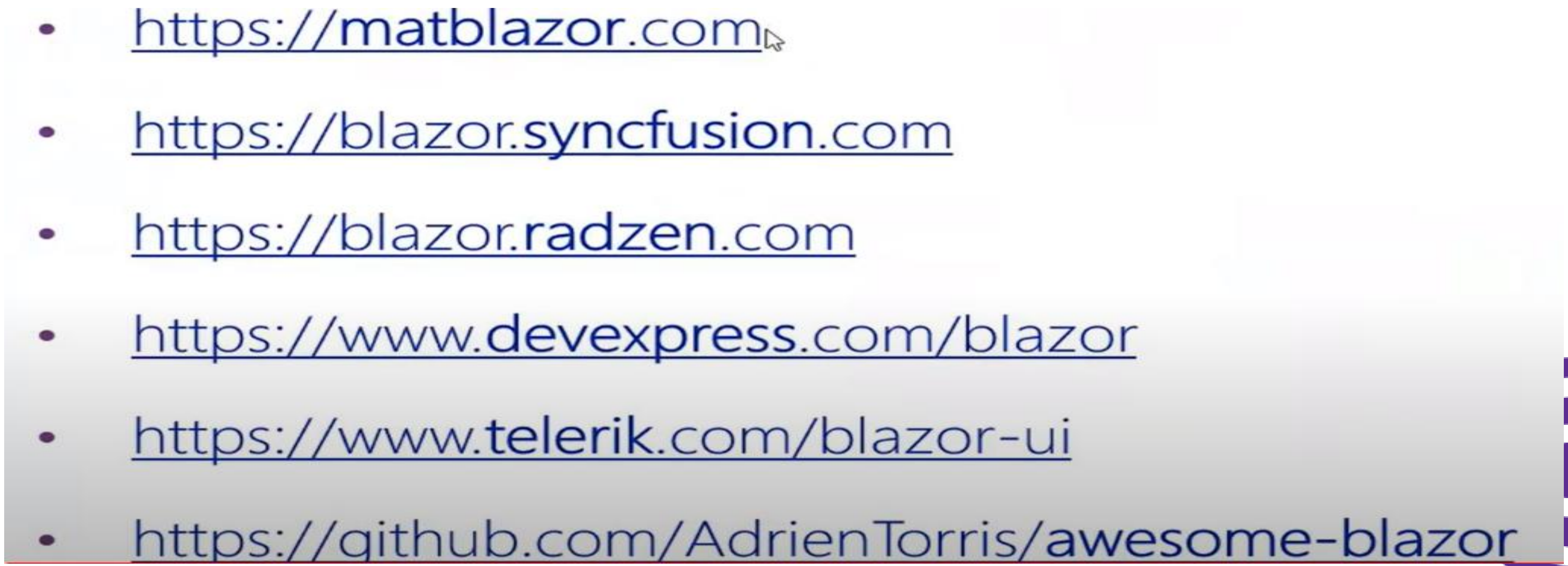
Constructor injection in the Services



```
public class EmployeeDataService : IEmployeeDataService
{
    private readonly HttpClient _httpClient;

    public EmployeeDataService(HttpClient httpClient)
    {

        _httpClient = httpClient;
    }
}
```

Composant graphices

- https://www.mudblazor.com/
- https://blazor-university.com/overview/what-is-blazor/



- https://matblazor.com
- https://blazor.syncfusion.com
- https://blazor.radzen.com
- https://www.devexpress.com/blazor
- https://www.telerik.com/blazor-ui
- https://github.com/AdrienTorris/awesome-blazor