Add icon Add cover

Injects and Parameters in Blazor

`[Inject]` Dependency Injection

Injections got easier it seems, you can do them nicely in two ways now. Either mark it as `required`, or use a constructor.

```C#
// Option 1, mark required
[Inject]
public required IJSRuntime JSRuntime { get; init; }

// Option 2, use constructor
public MyComponent(IJSRuntime jsRuntime)
{
  // Either use in a primary constructor, or use in a normal constructor whatever works.
}
```

See here: ASP.NET Core Blazor dependency injection |...

`[Parameter]` Parameters

If the parameter is required, decorate it with the `[EditorRequired]` attribute so that the consumer of the component will get a warning at compile time if they don't provide the parameter.

Mark it as `required` to let the compiler know that it won't be null. Or use a question mark to mark it as nullable.

```C#
[Parameter, EditorRequired]
public required string RequiredParam { get; set; }

[Parameter]
[EditorRequired]
```

```
6  public string? RequiredNullableParam { get; set; }
```

If the parameter is optional, don't use the `[EditorRequired]` attribute. If it's optional, you can provide a default value.

C#

```
1  [Parameter]
2  public string? OptionalParam { get; set; }
3
4  [Parameter]
5  public string OptionalParamWithDefaultValue { get; set; }
   = "default value";
```

Optional Event Callback Parameter

The `EventCallback` in Blazor is a struct, so it can't be null in the first place. You can check it using `HasDelegate`, but it will not throw an error if `HasDelegate` is false, and you call `InvokeAsync()` on it.

C#

```
1  [Parameter]
2  public EventCallback OptionalCallback { get; set; }
3
4  protected async Task SomeFunction()
5  {
6      // Check HasDelegate, which is false when
   OptionalCallback hasn't been assigned a meaningful value.
7      if (OptionalCallback.HasDelegate)
8          await OptionalCallback.InvokeAsync();
9
10     // Calling it anyway when HasDelegate is false does
   not throw an error.
11     await OptionalCallback.InvokeAsync();
12 }
```

It may be clearer to explicitly mark it as nullable when it is optional, and not when it is required. This has the drawback of allowing for "two zeros" as I would call it: a null value, and a default value that's not null but `HasDelegate` is false. I think the pros outweigh the cons though, so I prefer this style.

C#

```
[Parameter]
public EventCallback? OptionalCallback { get; set; }

[Parameter, EditorRequired]
public required EventCallback RequiredCallback { get; set; }

protected async Task SomeFunction()
{
    // Compiler warns if you don't check for null
    if (OptionalCallback.HasValue)
        await OptionalCallback.Value.InvokeAsync();

    // No need to check the required one
    await RequiredCallback.InvokeAsync();
}
```