# Toggly Blazor

SSR · Interactive Server · WebAssembly · Auto

**Cheat Sheet**
`Toggly.FeatureManagement.Blazor`
.NET 8+


## Install

```
dotnet add package
  Toggly.FeatureManagement.Blazor
# Server / SSR / Interactive Server:
dotnet add package
  Toggly.FeatureManagement.Blazor.Server
```

Server package reuses trusted `Toggly.FeatureManagement`. Browser package reuses portable `Toggly.FeatureManagement.Client`.

## Trusted server

```
builder.Services.AddToggly(options ⇒
{
  options.AppKey =
    builder.Configuration["TOGGLY_APP_KEY"] ?? "";
  options.Environment = "Production";
  options.UseSignedDefinitions = true;
});
builder.Services.AddTogglyBlazorServer();
```

Backend App Key stays on the server. `RefreshAsync` re-evaluates cached defs — it does not force a network fetch.

## WebAssembly

```
builder.Services.AddScoped(_ ⇒ new HttpClient());
builder.Services.AddTogglyBlazorWebAssembly(_ ⇒
  new TogglyClientOptions {
    AppKey = builder.Configuration
      ["Toggly:FrontendAppKey"],
    Environment = "Production",
    Defaults = new() { ["new-dashboard"] = false }
  });
```

Front-end App Key + **Available to Client SDK**. Never a backend or management credential.

## Feature + negate

**_IMPORTS.RAZOR**

```
@using Toggly.FeatureManagement.Blazor
@using Toggly.FeatureManagement.Client
```

```
<FeatureProvider>
  <Feature Key="new-dashboard">
    <ChildContent><p>New dashboard</p></ChildContent>
    <Loading><p>Loading…</p></Loading>
  </Feature>
  <Feature Key="new-dashboard" Negate="true">
    <p>Classic dashboard</p>
  </Feature>
</FeatureProvider>
```

Pair a second `Feature` with the same keys plus `Negate="true"`. Put `Loading` on one of the pair. Empty `Keys` evaluates true, then Negate applies. `Keys` wins over `Key`.

## Programmatic

```
@inject IFeatureSession Features

private Task<bool> UseApiV2() ⇒
  Features.EvaluateAsync(["api-v2"]);
private Task Refresh() ⇒ Features.RefreshAsync();
private Task ChangeUser() ⇒ Features.SetContextAsync
  new EvaluationContext(
    Identity: "alice", Groups: ["vip"],
    Claims: new() { ["role"] = "admin" }));
```

Session is scoped — never a singleton. `SetContextAsync` is not for a shared user client.

# Toggly Blazor

SSR · Interactive Server · WebAssembly · Auto



## Entity

```
<Feature Key="ExpressCheckout" Entity="order">
  <button>Express checkout</button>
</Feature>
<Feature Key="ExpressCheckout" Entity="order"
         Negate="true">
  <p>Standard checkout</p>
</Feature>
@code {
  private readonly EntityContext order = new(
    "Order", "ord-vip",
    new() { ["Vip"] = true });
}
```

Browser entity gates fail closed without the required entity.

## Hydration (Auto)

```
<FeatureHydration
  PublicKeys="@(new[] { "new-dashboard" })"
  RenderMode="@(new InteractiveAutoRenderMode())">
  <Feature Key="new-dashboard"><p>New</p></Feature>
</FeatureHydration>
```

Payload is allowlisted booleans only. Entity evaluations are never satisfied by hydration.

## Pitfalls

- Server uses a **backend** App Key; WASM uses a **Front-end** App Key.
- Presentation gates do not authorize backend actions.
- Interactive Server does not register HTTP UserClaims / country / UA filters for long-lived circuits. Use `BlazorTargetingContext.Current` for custom filters.
- Browser has no telemetry or variant-assignment API.
- Auto does not transfer a live server DI scope into the browser.

## Resources

- Docs: `docs.toggly.io/sdks/blazor`
- Sample: `Toggly.Samples/blazor-sdk`
- .NET server sheet: `/cheatsheets/dotnet`
- .NET client sheet: `/cheatsheets/dotnet-client`