Toggly

.NET / C#

ASP.NET Core, MVC, Hangfire, EF, Dapper, MongoDB, RavenDB, Health Checks, NSwag

Install

CORE

```
dotnet add package Toggly.FeatureManagement.Web
```

OPTIONAL ADD-ONS

```
Toggly.FeatureManagement.HealthChecks
Toggly.FeatureManagement.NSwag
Toggly.FeatureManagement.Hangfire
Toggly.FeatureManagement.Storage.EntityFramework
Toggly.FeatureManagement.Storage.Dapper
Toggly.FeatureManagement.Storage.MongoDB
Toggly.FeatureManagement.Storage.RavenDB
```

Register

```
using Toggly.FeatureManagement.Configuration;
using Toggly.FeatureManagement.Web;
using Toggly.FeatureManagement.Web.Configuration;

builder.Services.AddTogglyWeb(o ⇒
{
  o.AppKey      =
builder.Configuration["Toggly:AppKey"]!;
  o.Environment =
builder.Configuration["Toggly:Environment"]!;
  o.UndefinedEnabledOnDevelopment = false;
}).WithTogglyTargeting
  <HttpContextTargetingContextAccessor>();
```

Authenticate before gates; identity and group claims come from the current request.

Evaluate a flag

```
public class Home(IFeatureManagerSnapshot fm)
  : Controller
{
  public async Task<IActionResult> Index() ⇒
    await fm.IsEnabledAsync(FeatureFlags.NewUI)
      ? View("NewIndex")
      : View("Index");
}
```

`IFeatureManagerSnapshot` caches each flag per request. Use `IFeatureManager` for different entities.

FeatureGate & Usage

```
[FeatureGate(FeatureFlags.Users)]
[FeatureUsage(FeatureFlags.Users)] // after Gate
[Route("users")] [Controller]
public class UsersController : Controller { … }
```

Disabled actions return 404 — customise via `IDisabledFeaturesHandler`.

Razor views

_VIEWIMPORTS.CSHTML

```
@addTagHelper *, Toggly.FeatureManagement.Web
@removeTagHelper
Microsoft.FeatureManagement.Mvc.TagHelpers.FeatureTa
gHelper, Microsoft.FeatureManagement.AspNetCore
```

```
<feature name="NewUI">…new content…</feature>
<feature name="Cart,Checkout"
         requirement="Any">…</feature>
<feature name="OldUI" negate="true">…</feature>
```

Custom context (filters)

```
var ctx = new MyAppContext { AccountId = id };
if (await fm.IsEnabledAsync("PremiumOnly", ctx))
{ … }

public class AccountFilter
  : IContextualFeatureFilter<MyAppContext>
{
  public Task<bool> EvaluateAsync(
    FeatureFilterEvaluationContext c,
    MyAppContext app) ⇒ …;
}
```

.NET / C#

ASP.NET Core, MVC, Hangfire, EF, Dapper, MongoDB, RavenDB, Health Checks, NSwag

Named variants

```
// Resolve IVariantFeatureManager and
// ITargetingContextAccessor from DI.
var context = await targeting.GetContextAsync();
var variant = await variants.GetVariantAsync(
  "new-dashboard", context);
// Use variant?.Name in ordinary Razor conditionals.
```

The Toggly feature tag has no variant attribute.

Conditional DI & middleware

CONDITIONAL SERVICE

```
services.AddTransientForFeature
  <ILinkService, GlobalLinkService>("GlobalLink");
services.DecorateForFeature
  <ITaxService, AvalaraTaxService>("Avalara");
```

MIDDLEWARE / MVC FILTER

```
app.UseMiddlewareForFeature
  <ReferralTrackingMiddleware>("ReferralTracking");
services.AddMvc(o ⇒
  o.Filters.AddForFeature<AuditFilter>("Audit"));
```

Snapshot providers

Local cache of definitions for fast startup & outage tolerance.

ENTITY FRAMEWORK

```
services.AddTogglyEntityFrameworkSnapshotProvider(
  o ⇒ o.UseSqlServer(connStr));
```

DAPPER

```
services.AddTogglyDapperSnapshotProvider(
  () ⇒ new SqlConnection(connStr),
  o ⇒ o.Provider = DatabaseProvider.SqlServer);
```

RAVENDB / MONGO

```
services.AddTogglyRavenDbSnapshotProvider();
services.AddTogglyMongoDBSnapshotProvider(mongoConnS
tr);
```

Health checks

```
builder.Services.AddHealthChecks()
  .AddTogglyHealthCheck(o ⇒
  {
    o.StalenessThreshold =
      TimeSpan.FromMinutes(15);
    o.RequiredFeatures =
      new[] { "payment-gateway" };
  });
app.MapHealthChecks("/health");
```

Hangfire

```
featureStateService.AddOrUpdateJob<IDailyJob>(
  app.Services,
  FeatureFlags.NightlyReport,
  "Nightly report",
  j ⇒ j.RunAsync(),
  Cron.Daily());
```

NSwag & OpenAPI

```
services.AddOpenApiDocument((config, services) ⇒
{
  config.AddFeatureGateFiltering(services);
});
// pipeline: app.UseFeatureAwareOpenApi();

[FeatureGate("Beta")]
[ApiController, Route("v1/beta")]
public class BetaController
  : ControllerBase { … }
```

.NET / C#

ASP.NET Core, MVC, Hangfire, EF, Dapper, MongoDB, RavenDB, Health Checks, NSwag

Bridge to front-end

```
[Route("flags.json")]
public async Task<IActionResult> Flags()
{
  var d = new Dictionary<string,bool>();
  await foreach (var n in
    fm.GetFeatureNamesAsync())
    d[n] = await fm.IsEnabledAsync(n);
  return Ok(d);
}
```

Pitfalls

- `[FeatureUsage]` must be after `[FeatureGate]` — else usage is double-counted on blocked requests.
- Use `IFeatureManagerSnapshot` for stable user-only checks. Evaluate each Order with `IFeatureManager`.
- `UndefinedEnabledOnDevelopment` only works when `app.Environment.IsDevelopment()` is true.
- `RequiredFeatures` and Hangfire callbacks follow environment AlwaysOn state, not per-user targeting.

Resources

- Docs: `docs.toggly.io/sdks/dotnet`
- Client / desktop sheet: `docs.toggly.io/cheatsheets/dotnet-client`
- Runnable .NET SDK Sample
- NuGet: `Toggly.FeatureManagement.*`
- GitHub: `github.com/ops-ai/Toggly.FeatureManagement`
- Perf: ~360 ns/eval · 2.7 M evals/s/core