Use of Feature Flags

Feature Flags, also known as Feature Toggles, are a modern software development practice that decouples feature release from code deployment. This technique allows you to control the availability of features in your application without having to redeploy or change the code.

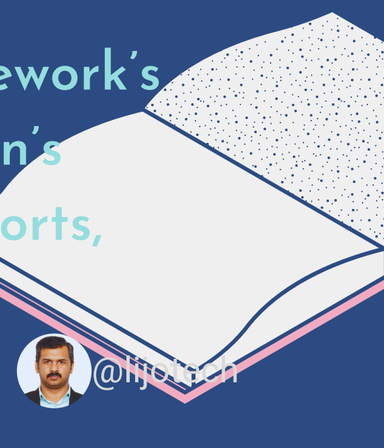
The Microsoft.FeatureManagement library provides idiomatic support for implementing feature flags in a .NET or ASP.NET Core application.

Let's dive in!

@lijotech

Key points about FeatureManagement library

- **Declarative Feature Flags:** The library allows you to declaratively add feature flags to your code, eliminating the need to manually write code to enable or disable features with if statements.
- **Feature Flag Lifecycle Management:** The Feature Management libraries manage feature flag lifecycles behind the scenes. For example, they refresh and cache flag states, or guarantee a flag state to be immutable during a request call.
- **Integration with ASP.NET Core:** The ASP.NET Core library offers out-of-the-box integrations, including MVC controller actions, views, routes, and middleware.
- **Configuration:** The .NET feature manager is configured from the framework's native configuration system. As a result, you can define your application's feature flag settings by using any configuration source that .NET supports, including the local appsettings.json file or environment variables.



@iijoykkn

Advantages of using feature flag

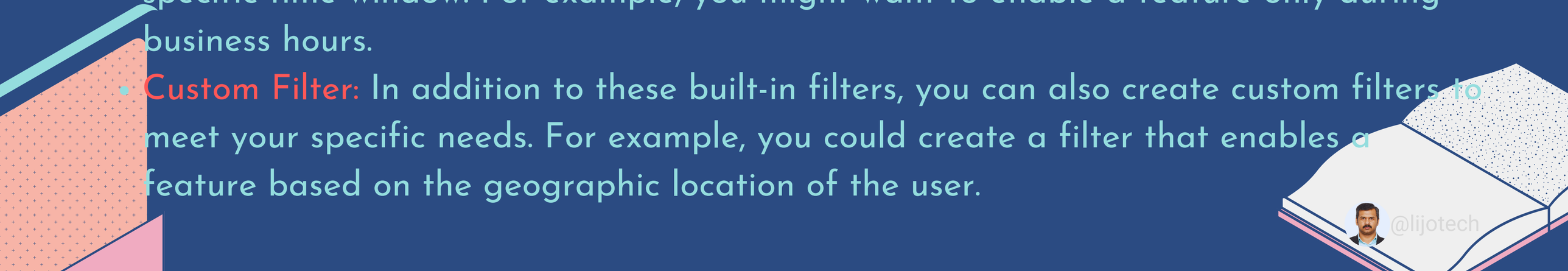
1. Incremental Rollouts: Feature flags allow you to gradually roll out a new feature to a small subset of users before making it available to everyone. This can help you test the feature's performance and user feedback.
2. Risk Mitigation: If a new feature has issues, feature flags allow you to turn off the feature without needing to redeploy your application. This can significantly reduce the risk associated with releasing new features.
3. A/B Testing: Feature flags can be used to implement A/B testing. This allows you to test different versions of a feature with different segments of your users.
4. Separation of Deployment and Release: Feature flags allow you to deploy your code at any time, even if the feature is not ready to be released. You can release the feature when it's ready by simply turning on the feature flag.
5. Targeted User Experiences: You can use feature flags to enable features for specific users or groups. This can be useful for beta testing, canary releases, or to provide premium features to certain users.
6. Simplifies Code Testing: Feature flags can simplify your code testing process. You can enable or disable features in your test environments using flags, making it easier to test different scenarios.




@lijotech

Types of filters in Microsoft.FeatureManagement library

- Boolean Filter: This is the simplest type of filter. It's used when a feature is either enabled or disabled without any conditions.
- Percentage Filter: This filter is used when you want to enable a feature for a certain percentage of requests. For example, you might want to enable a new feature for 10% of your users to gather feedback and catch potential issues before rolling it out to everyone.
- Targeting Filter: This filter is used when you want to enable a feature for a specific group of users. You can define the group based on user properties like user id, email, role, etc.
- Time Window Filter: This filter is used when you want to enable a feature during a specific time window. For example, you might want to enable a feature only during business hours.
- Custom Filter: In addition to these built-in filters, you can also create custom filters to meet your specific needs. For example, you could create a filter that enables a feature based on the geographic location of the user.



Example- TimeWindow Filter Flag

```
[Route("api/[controller]")]
[ApiController]
public class FeaturesController : ControllerBase
{
    private readonly IFeatureManager _featureManager;
    public FeaturesController(IFeatureManager featureManager)
    {
        _featureManager = featureManager;
    }
    [HttpGet("TimeWindowFeature")]
    public async Task<IActionResult> GetTimeWindowFilter()
    {
        if (await _featureManager.IsEnabledAsync("TimeWindowFeature"))
        {
            return Ok("Time Window feature enabled");
        }
        else
        {
            return BadRequest("Time Window feature is disabled");
        }
    }
}
```

In this example, we use a **TimeWindow** filter.This filter enables or disables the feature based on the current time and a specified time window. In the **FeaturesController**, the **GetTimeWindowFilter** method checks if the TimeWindowFeature is enabled. It does this by calling the **IsEnabledAsync** method of the **IFeatureManager** interface with the name of the feature flag.
This allows you to control the availability of the **TimeWindowFeature** based on the current time. For example, you might use this to enable a feature only during certain hours of the day, or to roll out a feature gradually over a period of time.

```
"FeatureManagement": {
  "TimeWindowFeature": {
    "EnabledFor": [
      {
        "Name": "TimeWindow",
        "Parameters": {
          "Start": "2024-05-01T10:00:00+00:00"
          "End": "2024-05-30T11:00:00+00:00"
        }
      }
    ]
  }
}
```

@lijotech

Stay Focused & Happy Coding

Follow Lijo Sebastian for more coding tips

@lijotech

Repost