# Technical Design exercise

**Approach:**

In order to meet the requirements of enabling/disabling features on the fly as well as ensuring the solution works across multiple servers, feature flags will need to be configurable and persisted to a database. As feature flags may result in multiple request for the same flag, a fetching strategy that would ensure multiple calls to the database are prevented, this may include caching flags for a small amount of time. In order to turn off features for a given percentage of users, something as simple as preforming a modulo operation against the user’s Id would suffice in determining whether to turn on a given feature for a give user.

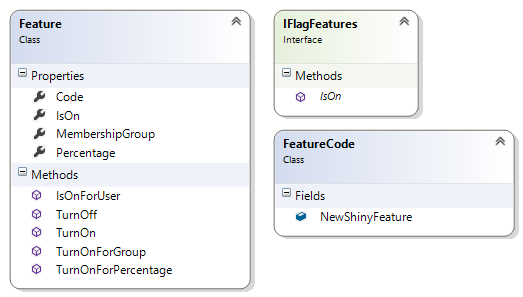
The creation of feature flags, into the database, would be done via data migrations, tools such as fluent migrator would assist. Finally, it’s important that stable features that do not require flipping should be removed from the feature flag system in order to ensure maintainance, complexity and the overall footprint of feature flags is kept low

**Guidance for implementing:**

It is recommended that feature flags, as much as possible, are reserved only for entry points to the system. Examples of ideal places for flags might be a particular UI component/control, MVC Controllers/Actions and in particular bootstrapping at application start.

For UI components, the flagging system would be exposed in the base view page, making this accessible during view rendering in Razor. Attributes would adorn controllers or actions preventing requests from invoking features that are off. Finally, entire components could be removed from bootstrap, in the case that they are not feature complete.

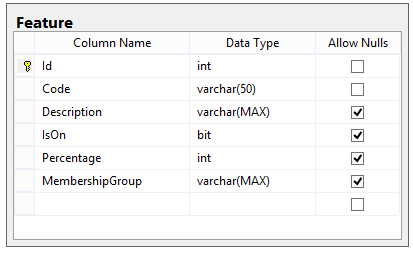
In some cases the use of decorator components that determine which inner components to invoke could be used. Inline flagging is highly discouraged as it can significantly raise the complexity of code. Other strategies should be considered prior to taking this route.

**Basic class diagrams** 

The IFlagFeatures interface is what will be used to test for feature status, it will take as parameters (visual studio class diagrams don’t display params it seems) the FeatureCode and the current UserProfile.

Feature on the top left defines the behaviour for managing a feature as well as the data that is required to determine if a feature is on.

Database



Example usage:

Assume here that the WebViewPage has been overrided and is available in Razor:

@if(Feature.IsOn(FeatureCode.ChangePassword, CurrentUser))

{

<fieldset>

<legend>Change Password Form</legend>

....

In the following case, entire controllers can be turned off.

[FeatureFlag(Feature.NewShinyFeature)]

public class NewShinyControllerController : Controller

{