**VSLive 2018 Redmond**

**Demo Script for Feature Flags Talk**

**DEMO: POOR MAN’S FEATURE FLAGS v1 – Using MVC Web App**

Goal: Use feature flags to hide text on the About and Contact pages when flag is turned off.

* Run app and show default text on those two pages
* Open web.config and add a key to drive the feature:

**<add key="MyFeature" value="true"/>**

* Open the **Controllers/HomeController.cs** file. We are going to use the ViewBag to tell status of the feature from the web.config

**Using System.Configuration**

//Add this to the About() method and the Contact() Method

**ViewBag.MyFeature = bool.Parse(ConfigurationManager.AppSettings["MyFeature"]);**

* Open About View and Contact Views and add the following code

**@if (ViewBag.MyFeature)**

**{}**

* Run the app. You should be able to see the strings
* Modify the web.config to turn the feature off
* Run the app. You should not be able to see the strings
* **Now let’s look at why this isn’t a great way to do feature flags**
* Go to HomeController.cs and change the “About” AppSetting to **MyFeatures.** Run the app, You will get a null exception when you navigate to the page.
* Fix the code and enable the feature in the web.config to set things back to working
* **LET'S REMOVE THE FEATURE FLAG CODE AND SEE WHAT HAPPENS**
* Delete the key from the web.config
* Remove code from HomeController.cs
* Remove code from the About view
* Run application. Go to the Contact page and it will fail
* Remove code from Contact page
* The issue with the poor man approach is nothing is strongly typed, and you don’t get compile time checks.

**DEMO: USING FEATURETOGGLE FRAMEWORK**

Web Forms

* Create a windows form app and add two buttons. Set framework to 4.6.2
* Install FeatureToggle from nuget
* Start with the AlwaysOffFeatureToggle flag
* Create Flags folder
* Create new class called ButtonTwoFeatureToggle
* Inherit from FeatureToggle.AlwaysOffFeatureToggle
  + Instantiate the class: Flags.ButtonTwoFeatureToggle bt = new Flags.ButtonTwoFeatureToggle();
  + Wrap code in If Statement: if (bt.FeatureEnabled)
* Add code to Form1.Designer.cs
* Show feature toggle off and feature toggle on
* Change to be AlwaysOnFeatureToggle and run
* Change to be RandomFeatureToggle and run
* Change to be simple feature toggle, add code to app.config, and run
* Make some errors and show how the code is strongly typed

**Old code from using MVC app, which we can still use if we feel like we have time**

Goal: Show how the FeatureToggle Framework works

* First, let’s install the FeatureToggle Library using Nuget
* Open Manage Nuget Packages in Visual Studio
* Search for **FeatureToggle** and install it
* Let’s start with the AlwaysOnFeatureToggle flag. Explain how this flag works, and how it is compiled into the code directly.
* **CREATE THE MYALWAYSONFEATURETOGGLE CLASS**
* Only way to turn this on or off is to re-release a new assembly
* Create a new folder called **Flags**
* Create a new class called **MyFeatureToggle**
* Inherit from **FeatureToggle.AlwaysOnFeatureToggle**
* Open **Views/Shared/\_Layout.cshtml**
* Wrap the **About** link in the following code

@if (new GenericMVCApp.Flags.MyFeatureToggle().FeatureEnabled)

{

}

* Run the app and show that About is there.
* Change inheritence to AlwaysOffFeatureToggle, recompile, and rerun
* **CREATE A SIMPLE FEATURE TOGGLE**
* Enable/Disable based off a configuration setting
* Change the Inheritance on the class to be SimpleFeatureToggle
* Run the app and it will error out because we haven't updated the app.config. That strong typing coming in handy
* Add the following key to web.config

**<add key ="FeatureToggle.MyFeatureToggle" value="true"/>**

* Run and should see About
* Change to false
* Run and should not see about
* **CREATE A RANDOM FEATURE TOGGLE**
* Change the Inheritance on the class to be RandomFeatureToggle
* Remove web.config values
* Run multiple times to see changes

**~~DEMO: USING LaunchDarkly~~**

~~Goal: Show how the LaunchDarkly Framework works~~

* ~~Create a C# Console application called LaunchDarklyDemo~~
* ~~Use Nuget to install the LaunchDarkly.Client~~
* ~~Open Program.cs and add using LaunchDarkly.Client~~
* ~~In web browser, go to Account Settings : Projects and create a new Project called VSLive. Get the SDK Key~~
* ~~Add the following code to the program.cs main method, adding your SDK key~~

~~LdClient client = new LdClient("sdk-63d1be49-c78d-44e0-adc6-c2d015342acc");~~

* ~~In the web browser, switch to the VSLive project and production environment~~
* ~~Create a feature flag called VSLiveRedmone2018~~
* ~~Create a user object~~

~~User user = User.WithKey("bob@vsliveexample.com") .AndFirstName("Bob") .AndLastName("VSLive") .AndCustomAttribute("groups", "beta\_testers");~~

* ~~Add LaunchDarkly Code~~

~~bool value = client.BoolVariation("vs-live-redmond-2018", user, false);~~

~~if (value)~~

~~{~~

~~Console.WriteLine("Showing feature for user " + user.Key);~~

~~}~~

~~else~~

~~{~~

~~Console.WriteLine("Not showing feature for user " + user.Key);~~

~~}~~

~~client.Flush();~~

~~Console.WriteLine("Press any key to exit");~~

~~Console.ReadKey();~~

* ~~In web browser open the Debugger~~
* ~~Run the application. It should tell you it is not showing feature~~
* ~~Go back into web browser and turn feature flag on~~
* ~~Run the application, it should tell you it is showing feature, and you should see things in the Debugger~~
* ~~Discuss some of the other things you can do with LaunchDarkly~~

~~Demo Extras (This to build if we have time)~~

* ~~Custom featuretoggle using the featuretoggle framework~~
* ~~Use the SQL featuretoggle~~
* ~~More in-depth LaunchDarkly demo~~
  + ~~Also need to read up on LaunchDarkly a little more~~

**DEMO: USING LaunchDarkly v2**

Goal: Show how the LaunchDarkly Framework works

* Have the app prebuilt
  + MVC App With Individual Accounts
  + Created the VSLive Project
  + Used Nuget to install the LaunchDarkly.Client packages
  + Added 4 user accounts
    - [Testuser1@bubba.com](mailto:Testuser1@bubba.com) – [Testuser4@bubba.com](mailto:Testuser4@bubba.com)
* Talk about how you can create projects and show it, but don’t use a new project, because we have already seeded the users
* Create a new feature flag for the talk in the Test Env. Set it where some users can get to it (like user 1 AND 2)
* Add the following code to the Controllers/HomeConsoller.cs to modify the value displayed on the about page
* Make sure you have the launch darkly debugger open in a separate browser window, then run the code. When you hit the about page, the debugger will also show some information.
  + Show how you can hit the page with different users and get different results, as expected
  + Remove users and show how the flag is updated in real-time
* Go through the LaunchDarkly interface discussing a couple of things that are there, like the ability to do A/B testing and integrations. – I need to read up on this

using LaunchDarkly.Client;

using Microsoft.AspNet.Identity;

LdClient ldClient = new LdClient("sdk-9c402321-46d5-480c-9bba-9f93578b2769");

User user = LaunchDarkly.Client.User.WithKey(User.Identity.GetUserName());

bool showFeature = ldClient.BoolVariation("about-page-feature-flage", user);

if (showFeature)

{

// application code to show the feature

ViewBag.Message = "Feature Flag Turned On";

}

else

{

// the code to run if the feature is off

ViewBag.Message = "Feature Flag Turned Off";

}