##  Introduce an Asp.net MVC Design Pattern:

##  Server Response Display

 Changde Wu

This is a pattern that is unique to Asp.net MVC. We will see how it can help developers to 1) Communication information with users in a consistent way; 2) Reduce the code duplication; 3) Avoid the intermediate parameters to controller action methods.

I will show it in action with real working example in using Asp.net Identity for securing web site.

 **One. Introduction**

Pattern: Something can be generalized so it can be applied repeatedly

 Best knows is by Gang of Four

 As For web application, the server present some kind of visual for client, provide some means to let clients to interact(send request), then the server collect the client request, and process the request, and send back the result. Frequently, the server will also send message to inform the user of request processing status in addition to the result.

 As a web developer, to achieve this not an easy job. Different developers handles this job differently,for good or for bad. This author, also as a web developer, find out a way that can achieve the job in a consistent way. The way is so consistent, itself well fits in a pattern.

**Two. Motivation**

Let’s start with a sample project. Launch your Visual Studio 2013 (at this time, it is Update 4), create a MVC project, I give its name as ServerResponsePattern, and use the **Individual User Accounts** as Authentication type. You can use SQL local DB,or SQL Sever express. I always use SQL Express DB, so I created a Database with name ServerResp\_DB, and modified the web config DefaultConnection string to point to my new database. After project is created, use the Nuget package manager to update all packages to the latest. Compile the project, and make sure it is running. Try to register as a new user and login to the new web site and play with it.

A complete working project that has none user code can be downloaded <a href=”<http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/ServerResponsePattern_0.zip>>

http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/ServerResponsePattern\_0.zip </a>.

Take a moment to get familiar with the project you created using Asp.net Identity API. It is good enough to get started with your web application, and fufill your mission of securing your web site.

 Two apparent patterns that we can identify:

1. Manager pattern: To get an employee or ask an employee a favor, get the manager first

 ApplicationUserManager

 ApplicationUserManager

 (RoleManager)

1. Factory Method

 Give midware a factory method, it can get an instance of the manager on demand

You can also see these patterns (kind of):

1. await/async
2. Bubiness object vs Data Store

**One. Motivation**

The complete project for this section can be downloaded from:

<a href=”[http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/ServerResponsePattern\_1.zip](http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/ServerResponsePattern_0.zip)>

http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/ServerResponsePattern\_1.zip </a>.

Take a look at the ManagerController.cs, this class give us the way of managing user identity.

1. the Index action

 public async Task<ActionResult> Index(ManageMessageId? message)

 the parameter is optional and merely used to pass info, then used for message. - Why don’t pass message directly?

 then the message is passed to ViewBag for client display? - what happens if Index will redirect to another action?

 ManageLogins is very similar to Index

public async Task<ActionResult> ManageLogins(ManageMessageId? message)

2) The intermediate type

public enum ManageMessageId

is not necessary here

3) In the Views

 Such as Index.cshtml and ManageLogins.cshtml, you need to remember to fetch the message, and display it to the user.

 Question: How do you display? Is the message for info, for error, or for warning?

 To get this, you need to refactor the Action method to passing more view related info.

4).What if we need to pass more information to display to users when redirect an action call?

 Mes up the Action! who’s responsibility is not deliver the info from third party, but does its own core business.

**Two: Solution -- First try**

The complete project for this section can be downloaded from:

<a href=”[http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/](http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/ServerResponsePattern_0.zip)ServerResponsePattern\_final.zip>

http://www.jiacheng-laowu.com/Content/Articles/CodeCamp/Camp23/ServerResponsePattern\_final.zip </a>.

1. We will make life of passing message info and display it easier by creating a class: MessageInfo.

This simple class not only has message to display, but also the hint of the message type. If we use Bootstrapper, this info can be used as css class without change.

2) We will refactor the ManageController

1. Remove the ManageMessageId

3 We add a helper method to post the message

 It will post the message to both ViewBag and TempData

 So, the message will be passed to View whether we redirect to an action, or call action method, or run the View.

4) Cleanup the optional parameter for Index, and ManageLogins since they are no longer needed.

 Change the code that make redirect call to Index and ManageLogins accordingly.

5) Now let’s do some changes in View side

1. Add Models namespace to Web.config in Views folder so we don’t need a full path every time.
2. Modify the Index.cshtml and ManageLogins.cshtml under Views/Manage folder

 So the message info passed from controller actions will be used.

@{

 MessageInfo msg = ViewBag.StatusMessage;

 if (msg == null)

 {

 msg = TempData["StatusMessage"] as MessageInfo;

 }

}

Now, it is easy for us to tell users the message type

Change

 <p class="text-success">@ViewBag.StatusMessage</p>

To

 <p class="@msg.StyleText">@msg.Text</p>

This is much cleaner and disciplined.

**Three: Let this be a pattern**

The above work is great, we can go beyond. Let’s find a pattern and apply this application wide, and incorporated into the Framework.

If we apply the above strategy to other controller classes, it will work great as expected. However, the code repetition appeals a strategy, the Server Response Pattern in this presentation.

Strategy:

1. Make a base class that facilitate the message posting. So, every controller in the application can derive from it without repeating the same routines.

 Add a class in the controller folder that will serve as the base class for all of your controllers in your application.

 public class SRPBaseController : Controller

 {

 public virtual void PostErrorMessage(string msg)

 {

 PostMessage(msg, MessageType.Error);

 }

 public virtual void PostWarningMessage(string msg)

 {

 PostMessage(msg, MessageType.Warning);

 }

 public virtual void PostSuccessMessage(string msg)

 {

 PostMessage(msg, MessageType.Success);

 }

 public virtual void PostMessage(string msg, MessageType type = MessageType.Info)

 {

 MessageInfo msgInfo = new MessageInfo(type, msg);

 TempData["StatusMessage"] = msgInfo;

 ViewBag.StatusMessage = msgInfo;

 }

 }

 As you can see, you can also add Manager(s) so they are not repeated in Account and Manage controllers.

1. Refactor you controllers so they are derive from our new base controller.

 Remove the helper method PostMessage since it is no longer necessary.

 You can also refactor the calling of PostMessage to correct call such as PostSuccessMessage().

 Also, remove the AddErrors() helper, since we favor the PostMessage().

1. Lets handle the message display in master page (\_Layout.cshtml)

 So the message has consistent look and feel across the application.

 The concrete View will be much cleaner - no longer need code to handle message display

1. Add partial view for \_InfoSection in Views/Shared folder for displaying the messages in the web page,, and add styles to Content/Site.css file for top-banner styling.
2. Clean up the Index.cshtml and ManageLogins.cshtml in Views/Manage folder by removing our previous message display code since our framework automatically take care of the server message.

Benefit is apparent: All you controllers and actions have the message publish power tool for free.

And all your view are automatically wired up with server response messages for free, yet in a consistent way.

Going further: Visual studio project template can take advantage of this pattern to provide a very powerful and helpful way.