FatModel

Quick Start Guide

Best Practices Framework
for
ASP.Net MVC

By
Loma Consulting

February 5, 2016

Table of Contents

1. What is FatModel

The LomaCons FatModel framework can be added to any ASP.Net MVC based applications to enhance coding best practices and notifications to MVC models. These best practices are based on these well-known tenets of the MVC design pattern.

- Models should be Fat
- Controllers should be Skinny
- Views should be Dumb

Additionally the FatModel framework helps to enforce these best practices and long term code consistency, by providing consistent, well named over-ridable functions and events on model objects.

FatModel can be used and added to any Model and Controller, and can be used with any action result or view type, including cshtml, vbhtml, partial views, redirect, json and more.

2. Prerequisites

These prerequisites are a minimum guideline for installation and are a place to start. Confirm that the following lists of prerequisites are met on the application developer systems that will be running the components of FatModel. Additional detailed installation information can be found in the FatModel Developer Guide.

FatModel requires the installation and configuration of Visual Studio 2012 or newer and ASP.Net MVC 5 or newer components and features. If you are already building applications with ASP.Net MVC, then you are ready to get started.

Visual Studio and Frameworks

The features below are typically installed along with Visual Studio or can be pulled in via NuGet packages. The versions listed are the current stable and minimum supported versions by Microsoft at the time that the provided assemblies and documentation was last updated.

- Visual Studio 2012, 2013 or 2015
- .Net Framework 4.5.2 or later
- ASP.Net MVC 5.2.3 or later

In general, if you are already developing applications with Visual Studio and the current release of ASP.Net MVC, these components are likely already installed.

Slightly earlier versions are also supported with the source code edition of FatModel. The code for FatModel will work unchanged with any version of .Net Framework 4.x.x and ASP.Net MVC 5.x.x. To support these versions, you will need to rebuild the FatModel source code targeting the appropriate framework and referencing the proper assemblies.

Please note that support for .Net4.5.2 or later is not included with Visual Studio 2012, but it can be added. Please contact Microsoft for more information about how to install support for version of the .Net framework you are targeting.

Domain Knowledge

The biggest prerequisite for FatModel is a good understanding of Visual Studio, C# or VB.Net, ASP.Net MVC, HTML and other web applications in general.

FatModel is easy to work with and in depth knowledge of ASP.Net MVC is not required. However, if you are just getting started with web development, and have already worked though some of the basic samples provided by Microsoft on getting started with ASP.Net MVC, you should be ready for FatModel.

3. Adding FatModel to a Project

Follow the steps below to add FatModel to any ASP.Net MVC application. FatModel can be added to an existing project or a new project. The derived class objects and over-ridable methods and events need only be added to your code where you want to utilize FatModel.

Simple Example

The following is an example of how to use FatModel with a data entry form. The controller, model and view handles both the initial Get request of the page as well as all Post requests of the form content.

This example shows how to add FatModel to your projects, and does not show all the methods and functionality in FatModel. For complete information and full API documentation, please see the Developer Guide documentation.

1. Add a reference to the LomaCons.FatModel.Mvc.dll assembly to your web application project.
2. Initialize the FatModel binder in the ASP.Net application startup code.

```
    FatModelConfig.RegisterBinder();
```

3. Derive a controller from SkinnyControllerBase.

Normally, you would derive from Controller. However, SkinnyControllerBase is derived from the Controller as well, so is easy to insert it into your controller class hierarchy.

```
public class PersonController : SkinnyControllerBase {
    public PersonController()
        : base() {
        return;
    }
. . .
}
```

4. Derive a model from FatModelBase.

You may already have a base class for your models, if so, it is easy to insert it into your controller class hierarchy. Your code will not directly create this model. Rather, the FatModel framework will look for a Model that matches this signature and FatModel will create the model, passing to the base class a reference to the controller.

```
public class PersonCreateModel : FatModelBase {
    public PersonCreateModel(ISkinnyController controller)
        : base(controller) {
        return;
    }
. . .
}
```

5. Create a GET and POST controller action in your controller.

The Get creates an instance of the model, using the CreateFatModel method, and returns the model to the view as a View Result. The Post will redisplay the View if there are any server side validation errors; if there are no errors then the Post will use a Redirect Action to redirect to another page.

```
    [HttpGet]
    public ActionResult Create() {
        PersonCreateModel m;
        m = base.CreateFatModel<PersonCreateModel>();
        return (View(m));
    }

    [HttpPost]
    public ActionResult Create(PersonCreateModel model) {
        if (ModelState.IsValid == false)
            return (View(model));
        return (RedirectToAction("Samples", "Home"));
    }
```

6. Override the FatModel notification methods in your model.

In the example below, The Load override is responsible for loading initial model values on the Get, and will persist the changes on the Post if the model is valid. The Validate override will perform server side validation, and utilizes the MVC ModelState object to add model errors to be displayed in the view.

```
protected override void OnLoad() {
    base.OnLoad();

    if (IsPostBack == true){
        if (ModelState.IsValid){
            _business.CreatePerson(FirstName, LastName, Age);
        }
        return;
    }
    FirstName = string.Empty;
    LastName = string.Empty;
    return;
    }

protected override void OnValidate() {
    base.OnValidate();

    if (_business.GetPerson(FirstName, LastName) != null)
        ModelState.AddModelError("Failure",
        "A Person with this First Name and Last Name already exisits.");
    return;
}
```

7. Create an MVC View with a form, data entry controls, validation and a submit button. The code sample below uses C# Razor, however any view engine can be used.

Note that there is nothing revolutionary or different about the view. FatModel extends and enhances only the Controllers and Models, so everything you know already about creating and manipulating views and action results does not change.

```
@model PersonCreateModel
@using (Html.BeginForm()) {
    <fieldset>
    <div class="row">
    @Html.ValidationSummary(false, null, new { @class = "text-danger" })
    </div>
    <div class="form-group col-md-6">
        @Html.LabelFor(model => model.FirstName,
                htmlAttributes: new { @class = "control-label" })
        @Html.EditorFor(model => model.FirstName,
                new { htmlAttributes = new { @class = "form-control" } })
    </div>
    <div class="form-group col-md-6">
            @Html.LabelFor(model => model.LastName,
                        htmlAttributes: new { @class = "control-label" })
            @Html.EditorFor(model => model.LastName,
                new { htmlAttributes = new { @class = "form-control" } })
    </div>
    <div class="form-group">
        <div class="col-md-12">
            <div class="pull-right">
                @Html.ActionLink("Cancel", "Samples", "Home",
                                null, new { @class = "btn btn-link" })
                <input type="submit" value="Create Person"
                                                class="btn btn-success" />
                </div>
            </div>
        </div>
    </fieldset>
}
```

A complete working example of this code and additional examples of other FatModel methods and functions can be found in the included FatModel Web Sample.

8. Build and run your application.

Conclusion

From a code perspective, compare this to some of the other controllers and models that you have seen or coded. You should see this.

- The Controller has become skinny, where it handles only the incoming request and quickly returns the proper action result. Little to no business layer interaction is needed.
- The Model has become fat, where it handles the interaction with the business layer, server side validation and data properties to be displayed by the view.
- The View remains dumb, where it only reads and display the data properties from the view. Note that the view, does not change data in the model and it does not interact with the business layer.

Business Layer Interaction

In the example above, you might notice that all interaction with the business layer (the layer that reads and writes content to/from the data storage repository) occurs within the Model and no business layer calls are made from within the Controller.

It is neither right nor wrong to have all business layer calls in the model, all in the controller or in both. Where you call your business layer is entirely up to you and what you feel should be your best practice. Therefore, FatModel does not enforce where your business layer calls occur.

See the included WebSample project for examples of both methodologies.

4. More Information and Help

More detailed and complete information can be found in the FatModelDevGuide.pdf documentation. Other answers to frequently asked questions can be found on the FAQ page on the www.lomacons.com website.