19

Lightweight Controllers

This chapter covers

How lightweight controllers can make programming easier

Decorating action results to apply common behavior

How to manage common view data without filter attributes

Using a hub and spoke architecture

Controllers are dangerous. Because, snugly nestled between the model and view, they are an easy place to put decision-making code, controllers are often mistaken to be a good place to put decision-making code. And it is quite convenient, at first. Building a select list? Do it in the action - it's two lines of code! Harnessing global data for a master page? Put that in an action filter attribute, it's right there! Orchestrating a process to find the specified order, authorize it, transimit it to the shipping service and email a receipt to the user, before redirecting the client to the confirmation page? In the controller! Wait... what?

19.2 Why lightweight controllers

It's important to focus on keeping controllers lightweight because without intentional, continuous refactoring they will become bloated. In many contexts, and especially in software design, bloat is bad. One symptom of a bloated class is that it's hard to understand exactly, precisely, only what it's doing. It's hard to understand because it's doing many things.

19.2.1 Maintainability

As code becomes hard to understand it becomes hard to change. As code becomes hard to change it becomes a minefield of errors and rework and headaches. Deep technical analysis must be rendered for each seemingly simple enhancement or bug fix, because the developer is unsure what the ramifications of her change will be.

The Single Responsibility Principle

The guiding principle behind this is the SRP. Basically, SRP states that a class should have one and only one responsibility. Another way to look at it is that a class should only have one reason to change. If you find that a class has potential to be changed for nonrelated reasons, the class is probably doing too much. A common violation of SRP is mixing data access with business logic. For example, a Customer probably shouldn’t have a Save() method.

SRP is a core concept of good object-oriented design, and its application can help your code become more maintainable. SRP is sometimes referred to as Separation of Concerns (SoC). You can read more about SRP/SoC from Bob Martin’s excellent article on the subject:

<http://www.objectmentor.com/resources/articles/srp.pdf>

Not only that, but understanding how to make a change becomes difficult. Without clear responsibilities, a change could potentially happen anywhere. We don't want building software to be a guessing game, where we blindly slap logic into action methods. We want to create a system where software design exists apart from controllers so that we don't struggle when working with our source code.

19.2.1 Testing

The best way to ensure it's easy to work with our source code is to apply test driven development. When we TDD, we work with our source code before it exists. And hard to test classes, including controllers, are immediately suspect as flawed.

19.2.2 Maintainability

19.2 Decorating action results

19.2.1 AutoMapActionResult

19.3 Managing common view data

19.4 Leveraging an application bus for a simple hub and spoke architecture

19.5 Summary