04 | Customizing Controllers

Taking Control of Controllers

- Adding Actions
- Model Binding
- Filters
- Vanity URLs
- Controller Best Practices

Adding Actions

Adding Actions

- Controllers are classes
- Actions are methods

- Creating an action involves adding a method to a class

Action Signature

- Return Types
  - ActionResult
    - FileResult
    - JsonResult
    - ViewResult
- Parameters
  - Normal parameters
  - MVC model binding

Get and Post

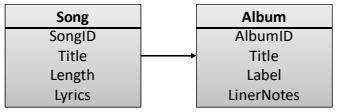
- Create/Update/Delete are typically two step operations
  1. Present the form
  2. Accept the input
- Create two actions
  1. Form presentation via HttpGet (default)
  2. Accept data via HttpPost

Model Binding

Default Model Binder

- Uses the name attribute of input elements
  - Automatically matches parameter names for simple data types
  - Complex objects are mapped by property name
    - Complex properties use dotted notation

```
<input type="text" name="Album.LinerNotes" />
```



Controlling Model Binding

- Imagine the following model

- Need
  - Create a form to edit everything but the lyrics
- Challenge
  - Default model binder automatically binds all inbound properties

Solutions

- Simplest
  - Use the bind attribute to indicate which properties to bind

```
Edit([Bind(Include = "SongID,Title,Length")] Song song)
```

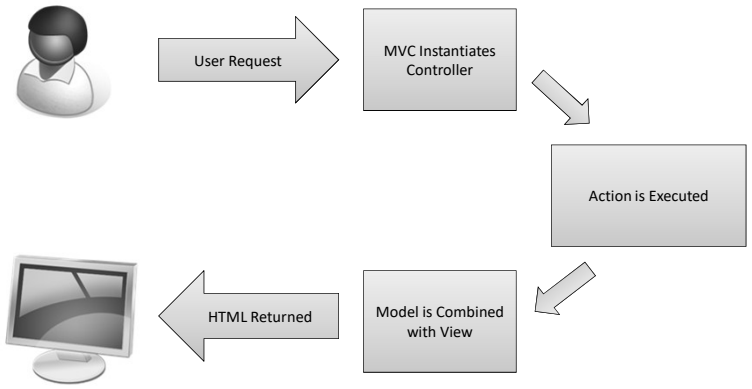
- Other solutions
  - Create a view model
  - Create a custom model binder

Filters

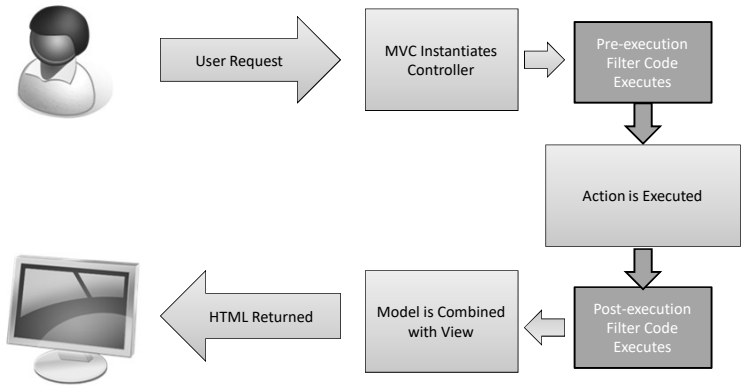
Filters

- Filters are attributes
  - Decorate controllers and actions
- Alter execution
- MVC contains several built-in filters
- Often used in lieu of updating web.config

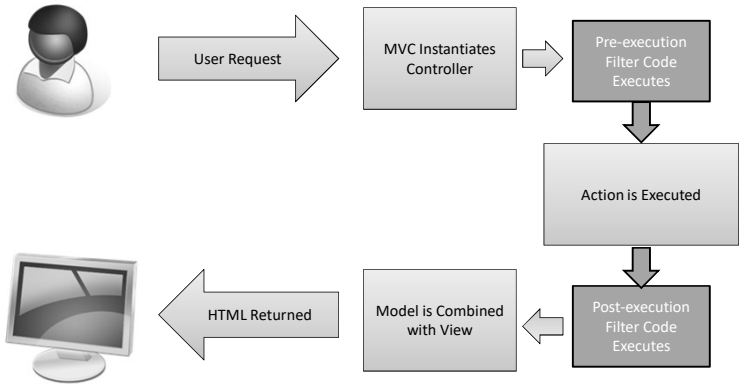
Normal Action Execution



Actions with Filters



Actions with Filters



Adding Filters

- Action
- Controller
- Global
  - FilterConfig.cs

Security Filters

- Authorize
  - Control who can access a controller/action
  - Properties
    - Users
    - Roles
- ValidateAntiForgeryToken
  - Defends against cross-site request forgery
  - Requires anti-forgery token to be added to view
- RequireHttps
  - Requries SSL

SSL

- Encrypts traffic and prevents tampering
- Authenticates server

- When to use SSL
  - Asking for sensitive information
  - After authentication

  - http://blog.codinghorror.com/should-all-web-traffic-be-encrypted/

HandleError & OutputCache

- HandleError
  - Redirect user to a view when an unhandled exception is thrown
  - Requires custom errors to be enabled in web.config file
- OutputCache
  - Instructs ASP.NET to cache HTML resulting from execution of action
  - Improves performance
  - Properties
    - VaryByParam
    - VaryByHeader
    - Duration (seconds)

Vanity URLs

Standard URL

www.mymusicstore.com/App/Album/Details/Display.aspx?ID=42&BandID=64

- Users have no idea what that URL refers to
- Search engines have no idea what that URL refers to
- It's just plain ugly

Vanity URL

www.mymusicstore.com/Album/Cure/Wish

- User knows information provided by the page
- Search engines know information provided by page
- Don't underestimate the importance of vanity URLs

MVC Routing

- Vanity URLs are handled by routing
- Routing in MVC controls what controller/action is called based on the URL provided
- Methods for updating routing
  - RouteConfig.cs
  - AttributeRouting

Attribute Routing

- Attributes control routing/URL
- RouteAttribute

```
[Route("Album/Edit/{id:int}")]
public ActionResult Edit(int id)
```

  - www.mymusicstore.com/Album/Edit/42
  - Calls the Edit action
  - Passes in the ID parameter
  - ID must be an integer

RoutePrefix

- Added to controller
- Adds prefix to all routes

```
[RoutePrefix("Album")]
public class AlbumsController : Controller
{
   [Route("Album/Edit/{id:int}")]
   public ActionResult Edit(int id)
   {
      // code
   }
}
```

Controller Best Practices

Controller Design Guidelines

- High Cohesion
  - Make sure all actions are closely related
- Low Coupling
  - Controllers should know as little about the rest of the system as possible
  - Simplifies testing and changes
  - Repository pattern
    - Wrap data context calls into another object