Introduction to MVC

Slavomír Moroz

Kentico

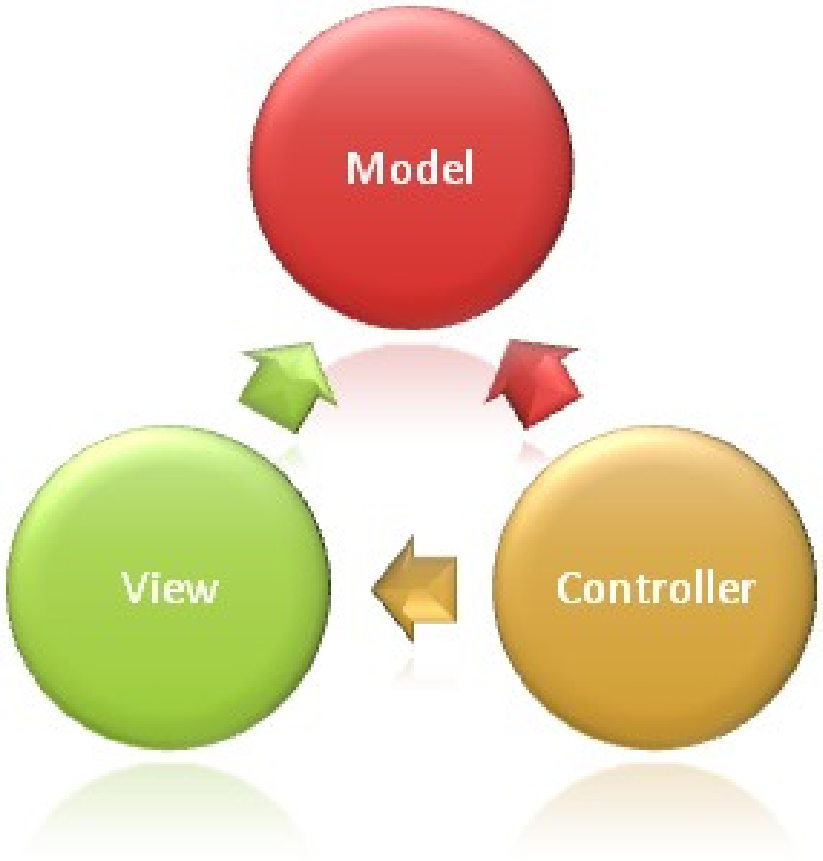
Revision from Previous Lesson

- ASP.NET WebForms applications
  - Abstract away HTTP (similar to desktop app development)
  - Object oriented and event-based
- Cons
  - UI logic coupled with code. Hard to separate.
  - Hard to unit test
  - Heavy page size due to viewstate management

ASP .NET MVC

- Design pattern
- Separates UI from logic through shared data
- Makes testing easier
- Has no viewstate

ASP .NET MVC



Kentico

ASP .NET MVC

- Controller can change the model
- Controller can send model to the view to get the representation of the model

- How does view recieves data from controller?
  - Viewbag object
  - Strongly and weakly typed model

Routing

- Routes are defined as patterns.
- Route must always lead to a controller and his action.
- Route may define additional parameters that will be passed to action handler.
- Order is important

Routes

Route Pattern	URL Example
`mysite/{username}/{action}`	~/mysite/jatten/login
`public/blog/{controller}-{action}/{postId}`	~/public/blog/posts-show/123
`{country}-{lang}/{controller}/{action}/{id}`	~/us-en/products/show/123
`products/buy/{productId}-{productName}`	~/products/but/2145-widgets

ActionResult

- Every action method returns object derrived from ActionResult class

Name	Framework Behavior	Producing Method
ContentResult	Returns a string literal	Content
EmptyResult	No response	
FileContentResult / FilePathResult / FileStreamResult	Return the contents of a file	File
HttpUnauthorizedResult	Returns an HTTP 403 status	
JavaScriptResult	Returns a script to execute	JavaScript
JsonResult	Returns data in JSON format	Json
RedirectResult	Redirects the client to a new URL.	Redirect
RedirectToRouteResult	Redirect to another action, or another controller's action	RedirectToRoute / RedirectToAction
ViewResult PartialViewResult	Response is the responsibility of a view engine	View / PartialView

Action selectors

- MVC provides attributes to tweak how the actions can be selected based on the name or the request verbs (post, get, put…).

```
[HttpPost]
[ActionName("PostData")]
public ActionResult Save(object o)
{
    return new EmptyResult();
}

[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Put | HttpVerbs.Delete)]
public ActionResult Save()
{
    return new EmptyResult();
}
```

Action filters

- An action filter is an attribute that you can apply to a controller action -- or an entire controller -- that modifies the way in which the action is executed.

- The ASP.NET MVC framework includes several action filters:
  - OutputCache – This action filter caches the output of a controller action for a specified amount of time.
  - HandleError – This action filter handles errors raised when a controller action executes.
  - Authorize – This action filter enables you to restrict access to a particular user or role.

Action selectors vs. Action filters

An ActionSelector dictates which action method is triggered. An ActionFilter provides some methods that are run before and after request and response processing.

Razor and Helpers

- Razor is a markup syntax that lets you embed server-based code (Visual Basic and C#) into web pages.

- ASP.NET helpers are components that can be accessed by single lines of Razor code.
- You can build your own helpers using Razor syntax, or use built-in ASP.NET helpers.

- Razor syntax

Razor helpers

HelperSample - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Tools Test Wi

ScottGu.cshtml | Index.cshtml | ProductsController.cs

```
@helper DisplayPrice(Decimal price) {

    if (price == 0) {
        <span>FREE!</span>
    }
    else{
        @String.Format("{0:C2}", price)
    }
}

@helper AnotherHelper(int value = 0) {
    @: My value is @value
}
```

100 %

Ready Ln 1

Index.cshtml | ProductsController.cs

```
@model IEnumerable<HelperSample.Models.Product>

@helper DisplayPrice(Decimal price) {

    if (price == 0) {
        <span>FREE!</span>
    }
    else{
        @String.Format("{0:C2}", price)
    }
}

<h2>Products</h2>

<ul>
   @foreach (var product in Model) {
      <li>
          <span class="producttitle">
            @product.Name
          </span>

          <span class="description">
            @product.Description
          </span>

          <span class="price">
            @DisplayPrice(product.UnitPrice)
          </span>
      </li>
   }
</ul>
```

Layouts

- You typically want to maintain a consistent look and feel across all of the pages within your web-site/application. Razor supports a feature called “layouts” – which allow you to define a common site template, and then inherit its look and feel across all the views/pages on your site.

Partial views

- If you want to reuse a view in your web application, you can go for the partial view concept.
- We can use partial views in a situation where we need a header, footer reused for an MVC web application. We can say that it’s like a user control concept in ASP.NET WebForms.

Model validation

- Attributes defined in System.ComponentModel.DataAnnotations
- IValidatableObject

```
[Required]
[DataType(DataType.Password)]
[Display(Name = "Password")]
public string Password { get; set; }

[DataType(DataType.Password)]
[Display(Name = "Confirm password")]
[Compare("Password")]
public string ConfirmPassword { get; set; }
```