Kentico

MVC basics

Petr Svirák

2015

NuGets

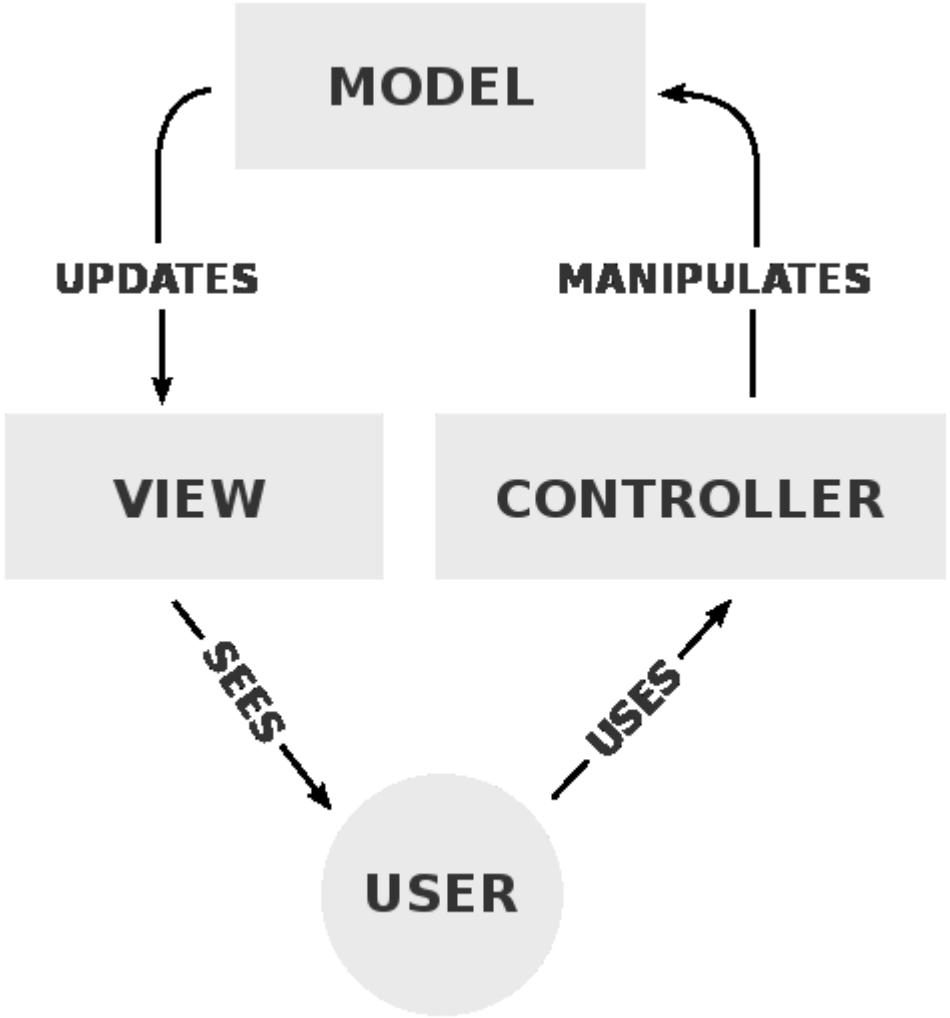
- Package manager in Visual Studio
  - 3rd-party DLL dependencies in a project
  - Easy to add/update/remove
- Use either UI (Right click on project → Manager NuGet packages) or console (Tools → NuGet Package Manager → Package Management Console)
- Always check the license prior usage

https://en.wikipedia.org/wiki/NuGet

MVC Pattern

- **M**odel-**V**iew-**C**ontroller architectural pattern for user-interface design (1970 – 1980)
- Ruby on Rails, ASP.NET MVC, Spring MVC, …

Model – state of an aspect of the application
View – displays user interface using provided model
Controller – handles user interaction by amending model and passing it to a view



https://en.wikipedia.org/wiki/ASP.NET_MVC_Framework

MVC web application

- Application run in IIS (express)
- *Global.asax(.cs)* – entry point for (any) ASP.NET application.
  - application-level and session-level events (Application_Start, Application_End, Session_Start, Session_End, …)
  - By default: routes, areas, filters and bundles registration
- *Startup.cs* – entry point for the Katana (Owin implementation) run in IIS
  - Allows Owin modules configuration
  - By default: authentication configuration
- web.config – main settings and configuration file for web application
  - Should contain DB connection string(s), application setting keys, assembly bindings, …

http://stackoverflow.com/a/20062253

Routing

- *~/AppStart/RouteConfig.cs*
- Defines how user request translated to a controller and its action
  - sufix Controller is removed (for routing)
  - additional route parameters might be optional
- Default route: "{controller}/{action}/{id}"
  - {controllor} is placeholder for a class in Controllers folder (inheriting from Controller class)
  - {action} is placeholder for the controller's public method
  - {id} is placeholder for optional parameter carying object identifier
  - Example: "/Home/Detail/3"

http://www.asp.net/mvc/overview/older-versions-1/controllers-and-routing/asp-net-mvc-routing-overview-cs

Controllers

- *Controllers* folder
- Scope
  - Each resource/entity is managed by a single controller **[up to middle-size applications]**
  - Each operation has a single controller **[complex application]**
- Should NOT contain business logic
- New instance for each HTTP request

https://en.wikipedia.org/wiki/Multilayered_architecture

Actions

- Public method of a controller
- Any result is sent back to the user
  - even void methods are called
- *ModelState* property – to validate model and/or report additional errors
- Use most strict **Result* available
  - *ViewResult* vs. *JsonResult* vs. *RedirectResult* vs. *ActionResult*...

http://forums.asp.net/t/1448398.aspx
https://msdn.microsoft.com/en-us/library/system.web.mvc.actionresult(v=vs.118).aspx

Selectors and Filters

- Attributes applied on *Controller* or *Action*
- Selectors influence which method is invoked
  - ActionName, AcceptVerb…
- Filters modify the way an action(s) are executed
  - Authorize, HandleError, OutputCache

http://www.asp.net/mvc/overview/older-versions-1/controllers-and-routing/understanding-action-filters-cs
http://www.codeproject.com/Articles/577776/Filters-and-Attributes-in-ASPNET-MVC
http://www.codeproject.com/Articles/291433/Custom-Action-Method-Selector-in-MVC

Models

- *Models* folder
- Represent state of a particuallar aspect of the application
- Each controller should have a sub-folder for its models
  - Models are usually named by corresponding Actions
- Always prefer strongly-typed model to a *ViewBag* or *ViewData*
- *Models should be POCO objects without any business logic*

https://en.wikipedia.org/wiki/Plain_Old_CLR_Object

Views

- *Views* folder
  - sub-folders per *controller*
  - *Shared* folder common for all controllers
- Razor syntax – combines (native) HTML with (server-based) code in C# *[*.cshtml]*
- Rendering is requested by a *Controller*'s *Action* that also provides a *Model* to render
- Should contain as few code as possible
- *ViewBag* for carring view-specific data to its partial views and layout

http://www.asp.net/mvc/overview/older-versions-1/views/asp-net-mvc-views-overview-cs

Layouts

- *Default: ~/Views/Shared/_Layout.cshtml (set in ~/Views/_ViewStart.cshtml)*
- Master page or template all views are rendered into
  - Contains HTML opening and closing tags, header and menu
- Changeble per view by adding: @{ Layout = "~/Views/Shared/_CustomLayout.cshtml"; }
- View in question is rendered by *RenderBody(…)*
- Evaluation order:
  - _ViewStart.cshtml
  - <an action view>.cshtml
    - <partial views in order of usage>.cshtml
  - _Layout.cshtml
    - <partial views in order of usege>.cshtml (e.g. _LoginPartial.cshtml)

http://weblogs.asp.net/scottgu/asp-net-mvc-3-layouts

Sections

- *RenderSection(…)* in layout (usually)
- Used for view-specific HTML that is not part of body (*RenderBody()*)
  - Side bars, adds, action-specific content
  - Scripts (~/Views/Shared/_Layout.cshtml)
- Not required to be defined in all views (usually, *required: false*)

http://weblogs.asp.net/scottgu/asp-net-mvc-3-layouts-and-sections-with-razor

Partial views

- *Html.Partial(…)* – displays view of given name (no controller/action is called)
- *Html.RenderAction(…)* – calls a controller's action, rendering its result
  - always use *PartialViewResult*
- Usually view name starts with underscore

http://www.codeproject.com/Articles/698246/ASP-NET-MVC-Special-Views-Partial-View-and-Layout

View helpers

- *Html* and *Url* helpers available
  - Methods to create application-specific HTML (e.g. action links/URLs, templated-display/edit, form elements for (model) properties)
  - Good points for extension methods (prefer over @helper syntax)
- Keep code under compiler controll, better maintanablity and readability of code
- Disposable pattern → tags with inner content (*Html.BeginForm*)

http://www.codeproject.com/Articles/228825/Razor-Helpers-Syntax
http://www.asp.net/mvc/overview/older-versions-1/views/creating-custom-html-helpers-cs

EntityFramework

- NuGet package @ nuget.org
- Object-relational mapper to work with relational data using domain-specific objects
- Eliminates the need for most of the data-access code
- Code-first or Database-first
- Quarable interface to work with DB

http://www.asp.net/entity-framework

Decorating models

- Applied on models
- *System.ComponentModel* and *System.ComponentModel.DataAnnotations* namespaces
- Model-specific constrains or additional meta-information
  - Display, DisplayName, DisplayFormat, HiddenInput – how is a property presented to a user
  - Required, Range, DataType, EnumDataType, Key – property's limitations and DB specifications

http://www.asp.net/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-6
https://msdn.microsoft.com/en-us/library/dd901590(VS.95).aspx

Voluntary homework

- Download *Games* web application from study materials
- Create async *Studios* controller by hand
- Create view model for each action
  - Decorate it with *DisplayName* and *DisplayFormat* (where applicable) attributes
  - Use *Range* attribute on FoundationYear property (in a view model, different range)
- Make a *Studio* dropdown list in *Create* view for the *Create* action of *GamesController*
  - Pass the existing studios to the model for *Edit* action
    - Preferably transform collection of studios to a collection of *SelectListItem* first
    - Solve out problem with user entering invalid data and re-entering *Edit* view (from the post action)
  - Make it so no new game can be created without an existing studio selected

Resources

MVC

- http://www.asp.net/mvc
- http://vswebessentials.com/
- http://www.pluralsight.com/courses/mvc4

EF

- http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application
- http://www.pluralsight.com/courses/entity-framework-6-getting-started