Views

Views

- View is responsible for providing the UI to the user
- View transforms the Model data to the format in which it needs to be presented to the user
- The views are added to the Views Folder under a subfolder with a Controller name.
- The view name is mapped to the Controller Action Name

Razor Views

- The views are Razor Views
- The Layout template + Razor View forms the output.

Strongly Typed Views

- ViewData is a ViewDataDictionary
- In the Controller method use the overload of View method to specify the model
- This sets the ViewData.Model property to the Model
- Then specify the @model in View

Example

```
@model IEnumerable<MvcRestaurants.Models.RestaurantReview>
```

Default Views and Views in Other Locations

- If no view is specified the Index View is considered
- Views can be picked up from different directories by specifying the path with a ~

Razor Expressions

- Implicit

  @item.Rating

- Explicit

  @(item.Rating /10.0)

Razor Code Blocks

- Razor is smart enough to distinguish between C# and Razor code.

Razor Comments and Escape

- @* *@ is for Comments
- @@ - To escape
- @: - To output a string

Layout Views

- Provides the skeleton
- It is similar to master pages
- Contains methods RenderBody() and RenderSection
- RenderBody is responsible for rendering the action specific views

Layout View

```
<!DOCTYPE html>
 <html>
   <head>
     <title>@ViewBag.Title</title>
 <script src="@Url.Content("~/Scripts/jquery-1.4.4.min.js")"
type="text/javascript"></script>
  </head>
 <body> @RenderBody() </body>
</html>
```

_ViewStart

- This is where the layout view for the application is defined
- This can be overridden by specifying a layout inside the folders. The one in the folders take precedence
- Layout can also be null

Partial Views

- Partial Views are reusable Views within the application
- Partial Views can be called using @Html.Partial
- They can also be called using @Html.Action which calls a sub-request to render a partial view
- ChildActionOnly restricts calling it from HTML.Action

Alternate View Engines

- **Spark -** Spark is a open source view engine for ASP.NET MVC projects as well as Castle MonoRail framework projects. This view engine is from the popular MVCContrib library.
- NHaml - NHaml is another View Engine and works like an replacement of inline page templating .

Alternate View Engines

- NDjango - NDjango is an implementation of Django Template Language using F#.
- **Hasic -**Hasic is a ASP.NET View Engine that uses the VB.NET XML kind of literals.
- **Bellevue -** Bellevue is another cool ASP.NET MVC View Engine that has the design goal which respects the HTML as first class

How to use Alternate View Engines

- Ref: http://stephenwalther.com/archive/2008/08/20/asp-net-mvc-tip-35-use-the-nhaml-view-engine

Areas

- Areas are an ASP.NET MVC feature used to organize related functionality into a group as a separate namespace (for routing) and folder structure
- (for views). Using areas creates a hierarchy for the purpose of routing by adding another route parameter, area, to controller and action.

Areas

- Your application is made of multiple high-level functional components that should be logically separated
- You want to partition your MVC project so that each functional area can be worked on independently

Areas

- An ASP.NET Core MVC app can have any number of areas
- Each area has its own controllers, models, and views
- Allows you to organize large MVC projects into multiple high-level components that can be worked on independently
- Supports multiple controllers with the same name - as long as they have different *areas*

Areas

- /Areas/<Area-Name>/Views/<Controller-Name>/<Action-Name>.cshtml
- /Areas/<Area-Name>/Views/Shared/<Action-Name>.cshtml
- /Views/Shared/<Action-Name>.cshtml

Areas

```
...
namespace MyStore.Areas.Products.Controllers
{
    [Area("Products")]
    public class HomeController : Controller
    {
        // GET: /Products/Home/Index
        public IActionResult Index()
        {
            return View();
        }

        // GET: /Products/Home/Create
        public IActionResult Create()
        {
            return View();
        }
    }
}
```

Areas

```
...
app.UseMvc(routes =>
{
  routes.MapRoute(name: "areaRoute",
    template: "{area:exists}/{controller=Home}/{action=Index}");

  routes.MapRoute(
      name: "default",
      template: "{controller=Home}/{action=Index}");
});
```