**Where we are:**

|  |  |
| --- | --- |
| Intro to MVC | Navigation, Searching and Filtering |
| Controllers and Views | Forms and Tag Helpers |
| Unit Testing, Publishing to Azure | Annotation and validation  Azure SQL Database |
| Dependency Injection, Models | Authentication |
| Entity Framework and Scaffolding | Authorization |

**Announcements**  
A beta version of your term project is due Tuesday of Week 10 (3/14).  
In addition to the feature requirements listed in the Term Project Requirements document, I expect you to use “best practices”, which means using all the things you’ve learned in this course.

**Validation**

* Validation expected in the browser (for Ux), required on the server (for security)
* Good Ux provides a way for users to easily fix entry errors
* In ASP.NET MVC, validation is done on the models
* Implemented using C# attributes – MVC calls them “data annotations”

Validation Data Annotations

Example:

[Required]­  
public string Name {get; set;}

Validation in the Controller

* Explicit Validation  
  if (string.IsNullOrEmpty(appt.ClientName)) {  
   ModelState.AddModelError(nameof(appt.ClientName), "Please enter your name");
* Model State Dictionary
  + AddModelError(property, message) This method is used to record a model validation error for the specified property.
  + GetValidationState(property) This method is used to determine whether there are model validation errors for a specific property, expressed as a value from the ModelValidationState enumeration.
    - Unvalidated This value means that no validation has been performed on the model property, usually because there was no value in the request that corresponded to the property name.
    - Valid This value means that the request value associated with the property is valid.
    - Invalid This value means that the request value associated with the property is invalid and should not be used.
    - Skipped This value means that the model property has not been processed, which usually means that there have been so many validation errors that there is no point continuing to
    - perform validation checks.
    - IsValid This property returns true if all the model properties are valid and returns false otherwise.

Displaying Validation Errors in the View

* Tag Helpers that display validation errors
  + Summary  
    <**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>
  + Individual field  
    <**label** **asp-for**="ClientName">Your name:</**label**>  
    <div><**span** **asp-validation-for**="ClientName" class="text-danger"></**span**></div>  
    <**input** **asp-for**="ClientName" class="form-control" />

Discussion

All non-nullable model fields are required by default. Make fields nullable by adding ? – this will also make them nullable in the database.

Hidden fields: look at the Edit scaffolding. They are used to hold the entity ID so that it is sent back with the post data. Models that are sent to the view aren’t kept in memory and can’t automatically be posted with all the original data in them. Why? (Statelessness of HTTP)

Updating edited fields. db.State (entity)

Partial views:

* Created like any other view. Put it in Views/Shared/ (or the same folder as the view that consumes it).
* Only contain a fragment of HTML, not a full HTML document
* Include a partial view with @Html.Partial(“PartialViewName”) // no extension on the name
* @Html.ActionLink helpers that don’t specify a controller will be directed to the controller for the view that consumes the partial view.
* Strongly typed partial views: The model can be passed to @Html.Partial as the second parameter. It could be the same model used in the parent view or it could be a new object created in the parent’s razor markup.