WEB524

WEB PROGRAMMING ON WINDOWS

WEEK 3 – LECTURE 1

DATA ANNOTATIONS & LINQ

Project Template Document

- Last Week: Using a persistent store – get all, get one, add new
- This Week: Using a persistent store – edit existing, delete

Resources

- Review the “Data Annotations” code examples
  - Your professor has added “comment tokens” to the code examples. This is done to highlight areas of the code that need your attention.
  - Review the comment tokens.

- Optional – Textbook – Chapter 6
  - Skim “CUSTOM VALIDATION LOGIC” section

Data Annotations

- Descriptive text elements added to properties in a class.
- A data annotation is located before/above the property declaration.
- Data annotations help with:
  - Property constraints and limits;
  - Input data validation;
  - Property display and formatting in views;
  - Communicating error messages to users.
- Some data annotations are intended for *design model* classes and others for *view model* classes. Some annotations will work with both.
- Include the System.ComponentModel.DataAnnotations namespace.

Data Annotations

- Their overall benefit is to reduce the amount of code that must be written to handle typical data management scenarios.
- Data annotations DO NOT replace the need to validate data against business rules and/or application logic!
- Improve the quality of input data by inspecting it for its integrity and appropriateness.
- You can comma-separate multiple data annotations or simply place one annotation per line.
- See the “DataAnnotations” code example published on Blackboard.

Data Annotations – Design Model Classes

- **[Required]**
  - Add NOT NULL in the database.
  - Can use for data and for relations.
  - Value types (e.g. *int* and *double*) should never use the [Required] attribute since they will always have a non-null value.

- **[StringLength(n)]**
  - specify the length of a varchar/nvarchar in the database.

- **[Key]**
  - used if the primary key is named something other than “Id” or “<entity-name>Id”.

Data Annotations – View Model Validation

- **[Required]**
  - Property cannot be null (optionally for string types you may allow empty strings).
- **[StringLength(n)]** or **[StringLength(n, MinimumLength = m)]**
  - Specify the maximum (and possibly minimum) length of a string.
- **[Range(min, max)]**
  - Ensure a number data type is between two values.
- **[Compare("PropertyName")]**
  - Often used for password or email entry; it compares this field's value with the value in the PropertyName property.

Data Annotations – View Model Validation

- **[RegularExpression(“regex”)]** – A regular expression.
  - You may omit the ^ and $ delimiters since they’re assumed.
  - Regular expressions commonly require the use of the backslash character (e.g. \d) but in a C# string, the backslash is a quote character. To workaround this, you may either precede the entire string with an at sign (@) or use a double backslash (\\d). Some common scenarios may include:

    [0-9]+ – digits only
    [a-zA-Z]+ – letters only
    [0-9a-zA-Z]+ – digits and letters only

Custom Error Messages

- All attributes accept a parameter named “ErrorMessage”.
- The value is a custom error message for the user.
- For example:
  [Range(2, 6, ErrorMessage=“Selected gizmos must range from 2 to 6”)]
- Some attributes will allow string.Format style error messages (e.g. {0}, {1}, etc). Typically these attributes are filled in with the name of the parameter.

Data Annotations – View Model Scaffolding

- **[Key]**
  - *Use this if the primary key is named something other than “Id”.*

- **[Display(Name=”Improved property display name”)]**
  - Change the text that describes a property.
  - Default is the property name.

- **[DataType(DataType.Password)]**
  - Other options include EmailAddress, Url, Currency, Date, Time, and MultilineText.

Data Annotations – View Model Scaffolding

- **[HiddenInput]**
  - Rendered as <input type=hidden…/>. It is often used for an object's identifier that will not be shown to the user.

- **[ReadOnly(true)]**
  - Same as removing the { set; } accessor of the property.

- **[Editable(false)]**
  - Not rendered in the browser.

- **[ScaffoldColumn(false)]**
  - Prevents property from being scaffolded.

LINQ (Language Integrated Query)

- LINQ – Language Integrated Query – is a C# language feature that supports in-language operations on data collections.

- **IMPORTANT:** LINQ works on data collections which are located in-memory.

- Web Apps typically use LINQ to:
  - Locate (or select) one item (e.g. “Find”, “Single” or “SingleOrDefault”)
  - Filter a collection and return items that match a condition (e.g. “Where”)
  - Sorting (ordering) a collection (e.g. “OrderBy” or “OrderByDescending”)

- When querying data, we can use one of two syntax forms

“Standard” Query Expression Syntax

- For those familiar with relational database querying, the LINQ *query expression syntax* will appear familiar.
- A typical standard query expression consists of these parts:
  - from …
  - where …
  - orderby …
  - select …
- You will notice, unlike SQL, the “select” clause is specified at the end of the statement.

“Standard” Query Example

```
var query = from p in people
            where p.LastName.StartsWith("R")
            orderby p.LastName
            select p;
```

- In the “from” clause, “p” is known as a local range variable. It represents a single item in the data source during an iteration of the query.
- A local range variable follows the same naming convention as any other variable.

“Fluent” Query Expression Syntax

- Also known as “Method-Based” syntax.
- Uses methods and method chains to do the work.
- The same query in our last example would look like:

```
var query = people.Where(p => p.LastName.StartsWith("R")).OrderBy(p => p.LastName);
```

- To make it easier to read, break on each “.” that begins a method name:

```
var query = people.Where(p => p.LastName.StartsWith("R"))
                  .OrderBy(p => p.LastName);
```

Lambda Expressions

- Each method may extract, filter or project data.
- Many of the methods require the use of a *lambda expression* often referred to as *inline methods* or an *anonymous function*.
- The syntax below shows a lambda expression that returns an entity object that matches a specific condition:

```
p => p.SIN == 3845723
```

- The return type is inferred from the context in which it is used. For example, if the lambda expression is used as an argument to a LINQ statement that is supposed to return an entity object then the return type of the lambda expression above is the entity object type.

Lambda Expressions

- Anonymous functions can “see” the local variables in the surrounding methods.
- For example, assume that the method that surrounds the lambda expression includes a local variable named “sin”. The new syntax would be:

```
    p => p.SIN == sin
```

- If there is already a local variable called “p” in the surrounding methods, then you must use a different *local range variable*.

Reading the Lambda Expression

- If you’re curious how to read or pronounce the lambda expression you may see this StackOverflow article: here

```
p => p.PersonId == persId
```

- p *such that* p.PersonId *is equal to* persId
- p *where* p.PersonId *is equal to* persId
- p *becomes (the result of)* p.PersonId *equals* persId
- p *for which* p.PersonId *is equal to* persId
- p *maps to* p.PersonId *equals* persId
- p *lambda of* p.PersonId *is equal to* persId

More about LINQ

- You may learn more about LINQ at the following URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/