## Ms Web Tech Assignment – MVC Validation & Class Libraries

In this assignment you’ll be using partial & metadata classes, validation & format annotations, custom annotations, the Validate method in a self-validating model and a class library. You’re going to validate records for the Member table.

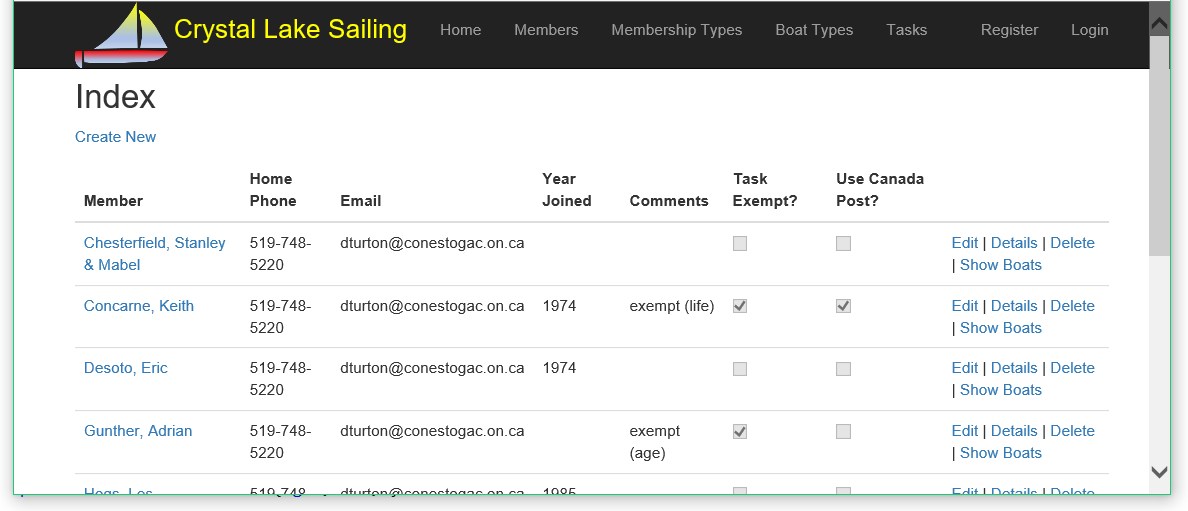
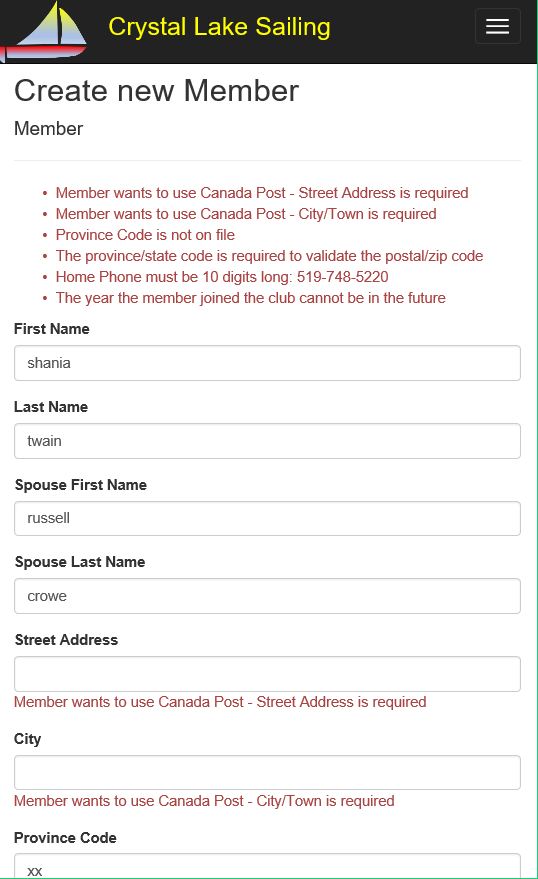
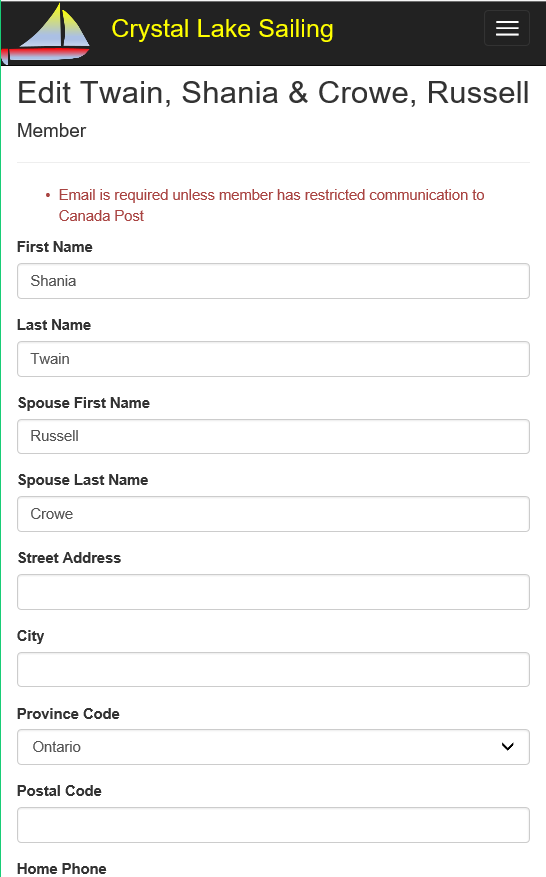
### Setup

1. Continue with your ***XXSail*** project … or you can download ***A2Sail*** and start from there.
   1. If starting from ***A2Sail***, put your name and section in the footer.
2. Ensure you have the correct code in **SailContext.cs** method **OnConfiguring** … see the slide titled “Generating the Models” in lecture “01a\_Core\_CreatingAWebSite.ppx”.

### XXProvinceController & XXCountryController

1. Generate controllers for the *province* and *country* tables, with full CRUD support and Views … no marks, but you might like to refer to them.

### XXMemberController

1. Generate the *XXMemberController* with full CRUD support and Views for the *Member* model.
2. Order the listing by Member *FullName* and reduce its fields to the ones shown in the example below. Model annotations will modify the field headings & formats for you. Show province *name*, not *provinceCode*.
3. In the Create view:
   1. Remove *fullName* from the page … it’s a generated field, not an input field.
   2. Replace the *provinceCode* drop-down with a textbox and a field-validation <span> to display any error messages for it. Just look at another textbox on the page.
4. On all other views, do not show *fullname* as an input/display field. Instead, show it in the browser title and page heading as “Edit Chesterfield, Stanley & Mabel”, Details of Chesterfield, Stanley & Mabel”, etc.
5. In the Edit view, display the province name in the drop-down, ordered by name.
6. Catch any exception that is thrown on Create or Edit, place its ***innermost*** message into ModelState, and allow processing to continue to the sad path, which should redisplay the user’s data with the error.
7. For Delete, put the innermost exception’s message into TempData and return to the Delete view.
8. If the insert, update or delete works, display a success message on the Member Index page via TempData.

Notice I have ValidationSummary showing **All** error messages, so even errors on hidden fields show.

### XXClassLibrary

1. Add a .NET Core class library called ***XXClassLibrary*** … as a separate project in the XXSail solution or as a project in a separate solution***.*** Remember that when you create classes here, make them and their methods **public** or they won’t be available to other projects like your web site.
2. Add a **public static** class called ***XXValidations***, with the following **public static** methods:
   1. Add a method called ***XXCapitalize*** that accepts a string and returns a string:
      1. If the input string is null, return it as an empty string.
      2. Change the input string to lower case and remove leading & trailing spaces (not imbedded ones).
      3. Shift the first letter of every word in the string to upper case.
      4. Return the newly-capitalised string.
   2. Add a method called ***XXExtractDigits*** that accepts a string and returns a string:
      1. Null is possible, so don’t blow up on it.
      2. Return a string containing all digits found in the input string.
   3. Add a method called ***XXPostalCodeValidation*** that accepts a string and returns a Boolean:
      1. Use this method in your partial class’ Validate method to verify a Canadian postal code, once you’ve decided the address is in Canada.
      2. This can be in upper or lower case, with or without a space, and the letters have limited values … the first letter can only be one of 18 letters and the others one of 20.
      3. If the input string is null or an empty string, have the method return true, permitting the postal code to be optional. Just spaces should fail.
      4. If the string fails the edit, return false, else return true.
   4. Add a method called ***XXPostalCodeFormat*** that accepts a string and returns a string:
      1. This is supposed to be run if the postal validation returns true, so you may get null … deal with it.
      2. Add a space (if missing) to the input string and return it, shifted to upper case.
   5. Add a method called ***XXZipCodeValidation*** that accepts a string **by reference** and returns a Boolean:
      1. Use this method to verify & format a US zip code, once you’ve decided the address is in the states.
      2. If the input string was null or empty, make it an empty string and return true.
      3. Extract all digits from the input string.
      4. If it contained 5 digits, put them into the input string and return true.
      5. If it contained 9 digits, format them as 12345-1234 into the input string and return true.
      6. Else, return false and leave the input field unchanged.
3. Add a reference to your class library to your ***XXSail*** project.

### Member Class

All ***Member*** validation code must be centralised in a metadata class and a like-named partial class, except for Remote & custom annotation code. Do not modify the generated *Member* model and do not modify or validate data in controller actions.

1. Create a *MetadataClasses* subfolder inside the *Models* folder.
   1. Create an ***XXMemberMetadata*** class file inside this subfolder.
   2. Modify its namespace to be the same as the member model
   3. Copy all physical property declarations from the member model to your ***XXMemberMetadata*** class.
   4. Add a separate pubilc *partial* class called ***Member*** (ie: same as the model) to this class file:
      1. Apply the XXMemberMetadata class to it, using an annotation.
      2. Turn the *Member* partial class into a self-validating model by implementing the IValidatableObject interface, which creates a ***Validate*** method.
      3. Replace the “throw” statement with “yield return ValidationResult.Success” … if you leave the throw, it’ll always abend … if you just take it out, the project won’t build.
2. Display annotations & edits:
   1. Change the field display-names to those shown in the sample Index, Create & Edit pages.
   2. Trim all strings of leading & trailing spaces.
   3. The member’s first and last names are required. The spouse’s names are optional.
   4. Use your *XXValidations.Capitalize* method to capitalise *firstName*, *lastName*, *spouseFirstName*, *spouseLastName*, *street* and *city*.
   5. If the member’s first and last names have been provided and all names have been capitalized, generate *fullName:*
      1. If only the member’s name is provided: **Twain, Shania**
      2. If the spouse’s first & last name is provided: **Twain, Shania & Crowe, Russell**
      3. If the spouse’s first name is provided, but the last name isn’t (or is the same as the member’s): **Twain, Shania & Russell**
   6. If *provinceCode* is not empty or null, forceit to upper case and …
      1. Validate it by fetching its record from the database … error if not found
      2. If fetching the province code throws an exception, put its innermost message in a ModelState so it displays beside the input field.
      3. Retain the record you found … its country code is required to validate *postalCode*
   7. *postalCode* is conditionally optional but, if provided, *provinceCode* becomes required, to validate it.
      1. Validate and format the postal/zip code using the relevant method(s) in your ***XXValidations*** Class (error if provinceCode is invalid/missing).
   8. *homePhone* must be provided:
      1. It must contain exactly 10 digits (delete punctuation and text like “emergency only”).
      2. Reformat into dash notation: 519-748-5220.
   9. *Email* is optional but if provided, it must be a valid email pattern.
   10. *yearJoined* should not be in the future. **Allow null when editing** (old data).
   11. I suspect *taskExempt* and *useCanadaPost* will default to false, but SQL will choke on null … so …
   12. Some members live with a slow or no Internet connection, so they prefer newsletters by post
       1. If *useCanadaPost* is false, a valid *email* is required, but the postal fields (*street, city, provinceCode,* and *postalCode*) are all optional … though, if provided, must be valid.
       2. If *useCanadaPost* is true, all the postal fields are required and *email* is optional

### Hand In

1. Zip and upload your project folder to the [Drop-Box](https://eConestoga.ca/) (Course Tools 🡪 Assignments) for this assignment.