## 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 Farm table.

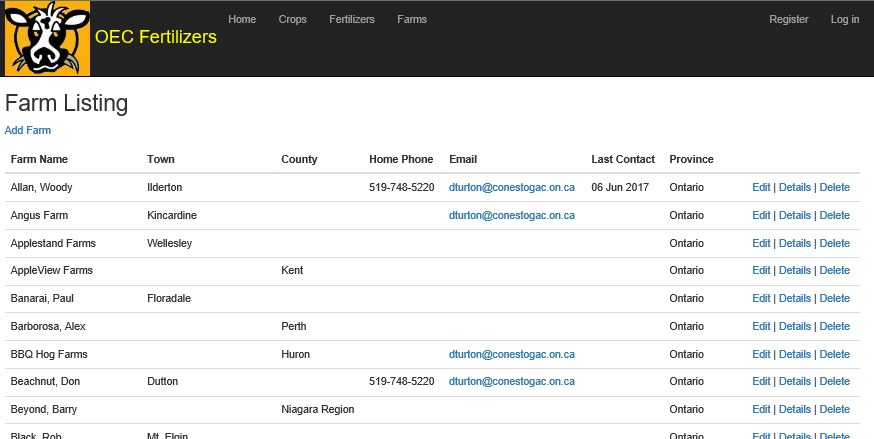
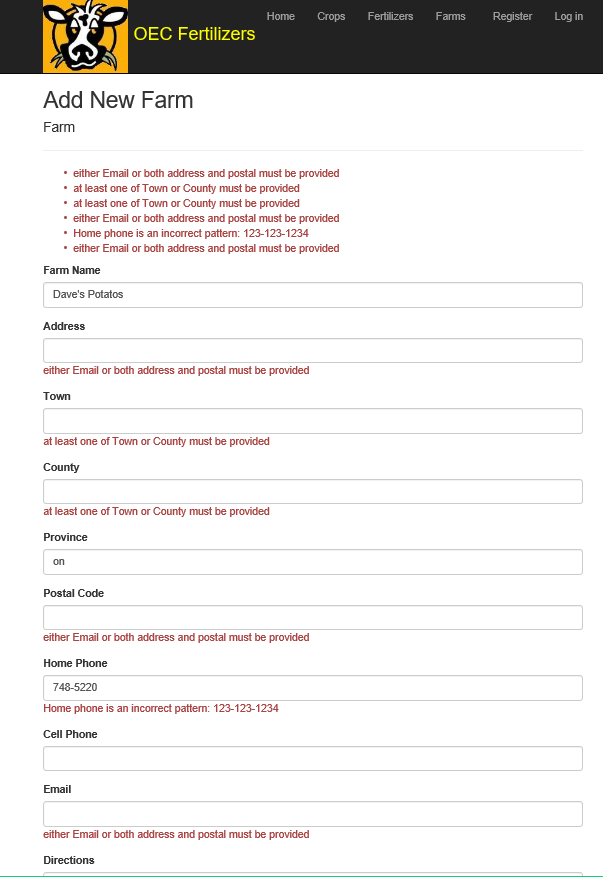
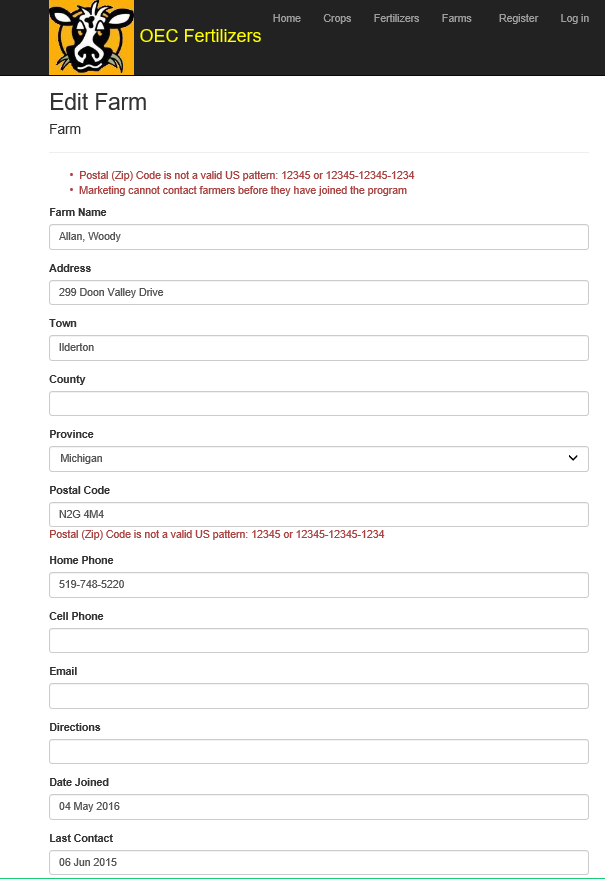
### Database

1. Continue with your ***XXOEC*** project … or you can download ***a2OEC*** and start from there.
   1. If starting from ***a2OEC***, put your name and section in the footer.

### XXProvinceController & XXCountryController

1. Generate controllers for the province and country tables, with full CRUD support and Views.
   1. Show the primary key on all pages. It should be an input field on Create, but locked on Edit
   2. In XXProvince’s Create & Edit, display the country name in the drop-down, ordered by name.

### XXFarmController

1. Generate the *XXFarmController* with full CRUD support and Views for the *Farm* model.
2. Order the listing by Farm *name* and modify 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. For the Create & Edit actions & views:
   1. In the Create view, 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.
   2. In the Edit view, display the province name in the drop-down and order it by name. Annotations don’t fire on drop-downs, but add a field-validation <span> to display any error messages that you produce in the Validate method.
   3. 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.
      1. For Delete, put the innermost exception’s message into TempData and return to the Delete view.
   4. If the insert, update or delete works, place a message to that effect in your TempData message variable and return to the Farm listing.

### XXClassLibrary

1. Add a .NET Core class library called ***XXClassLibrary.*** 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 class called ***XXDateNotInFutureAttribute*** that extends the *ValidationAttribute* class
   1. If the date is null, let it pass (null dates usually mean “not yet occurred”, like dateOfDeath)
   2. If the date is greater than the current date/time, return a pertinent error message containing the variable’s name, otherwise let it pass.
3. Add a public class called ***XXValidations***:
   1. Add a public class-level method called ***XXCapitalize*** that accepts a string and returns a string:
      1. If the input string is null, return it unchanged.
      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 public class-level method called ***XXPostalCodeValidation*** that accepts a string by reference and returns a Boolean:
      1. Code this as a custom annotation that validates a Canadian postal code.
      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 … or is it 16 and 18?
      3. If the string being passed 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 and do not change it.
      5. If the string passes the edit, shift it to upper case and insert the space if it’s missing. Return true … the calling program should receive the updated string.
   3. Add another public class-level method called ***XXZipCodeValidation*** that accepts a string by reference and returns a Boolean:
      1. Code this as a custom annotation that validates a US zip code.
      2. This can be 5 digits or 9, regardless of any punctuation. If it this fails this, return false.
      3. If it contains 5 digits, return true, along with the 5 digits without any punctuation.
      4. If it contains 9 digits return true, along with the 9 digits in the format 12345-1234.
4. Add a reference to your class library to your ***XXOEC*** project.

### Farm Class

***Farm*** 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 farm model and do not modify data in controller actions.

1. Create a *MetadataClasses* subfolder inside *Models*.
   1. Create an ***XXFarmMetadata*** class file inside this subfolder.
   2. Modify its namespace to be the same as the farm model
   3. Copy all physical property declarations from the farm model to your ***XXFarmMetadata*** class.
   4. Add a separate *partial* class called ***Farm*** (ie: same as the model) to this class file:
      1. Apply the XXFarmMetadata class to it, using an annotation.
      2. Turn the *Farm* 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 *name* and *provinceCode* are required.
   4. Either the town or county must be provided. Both are OK, but not necessary.
   5. If *email* is not provided, *address* and *postalCode* must be provided. If *email* is provided, the other two are optional.
   6. *email*, if provided, must be a valid email pattern.
   7. Use your *XXValidations.Capitalize* method to capitalise *name*, *address*, *town* and *county*.
   8. Use a [Remote] annotation to perform 3 different checks (with different messages) on the province code:
      1. It must be exactly 2 **letters** long (not just any characters)
      2. It must be on file in the province table
      3. If the select used to check the province code throws an exception, return the innermost exception message with a lead-in like “error validating province code …”
   9. Force the province code to upper case before writing to the database. Modifications like capitalisation & computed fields must be done in the *Validate* method.
   10. *postalCode* is conditionally optional (see above) but, if it’s provided, validate & format it using either your *XXPostalCodeValidation* or *XXZipCodeValidation* method, depending on which country *provinceCode* is from.
   11. Either *homePhone* or *cellPhone* must be provided:
       1. These must contain exactly 10 digits, ignoring punctuation and text like “Bert’s cell”.
       2. Before writing to the database, reformat these into dash notation: 519-748-5220.
       3. … huh … this is the second time to extract numbers … SOLID class library idea?
   12. *dateJoined* and *lastContactDate* can both be null …
       1. Use your *XXDateNotInFutureAttribute* to ensure they’re not in the future, if provided.
       2. *lastContactDate* cannot be provided unless *dateJoined* is also provided, but *dateJoined* can be provided without *lastContactDate*.
       3. A farmer cannot be contacted by marketing before they have joined the program.

### Hand In

1. Zip and upload your project folder to the [Drop-Box](https://eConestoga.ca/) (Course Tools 🡪 Assignments) for this assignment.
2. Hand in [the marking sheet](https://eConestoga.ca/) with your name & your instructor’s name on it, in class or through the slot in your instructor’s office door.