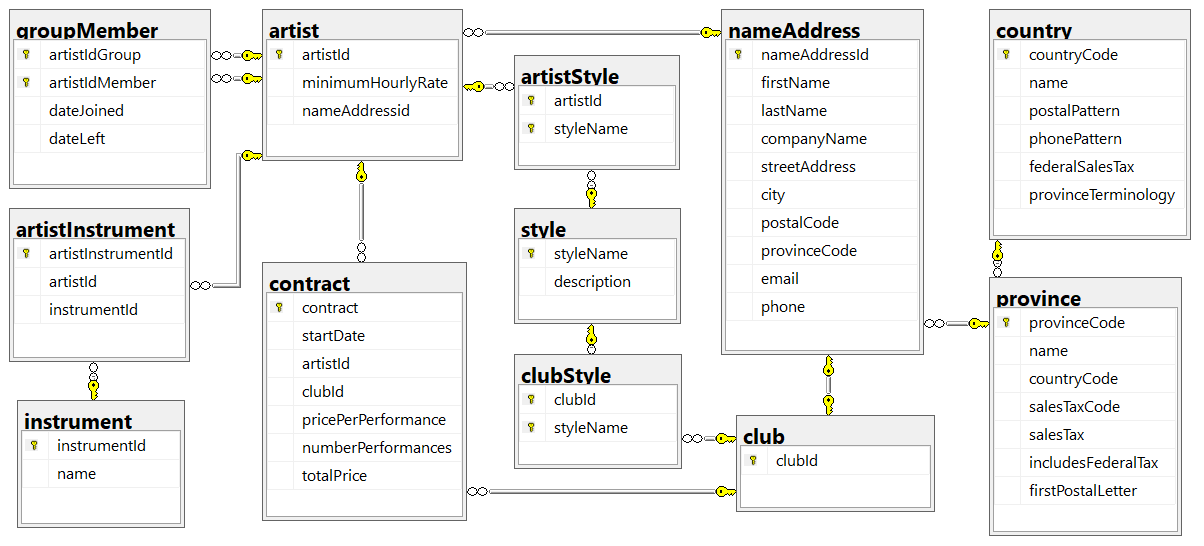
## Ms Web Tech Assignment – Model-Based Validation & Class Libraries

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

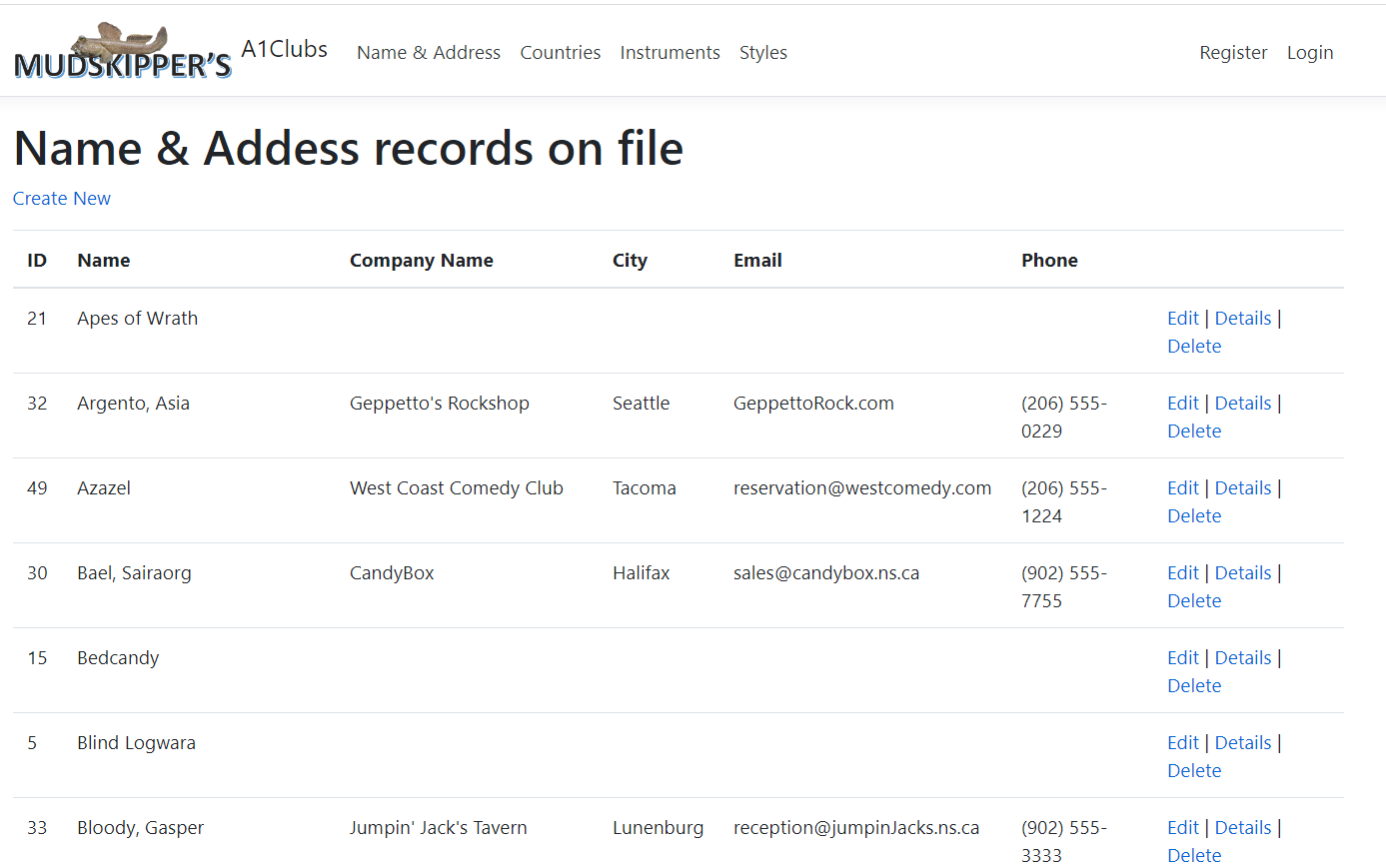


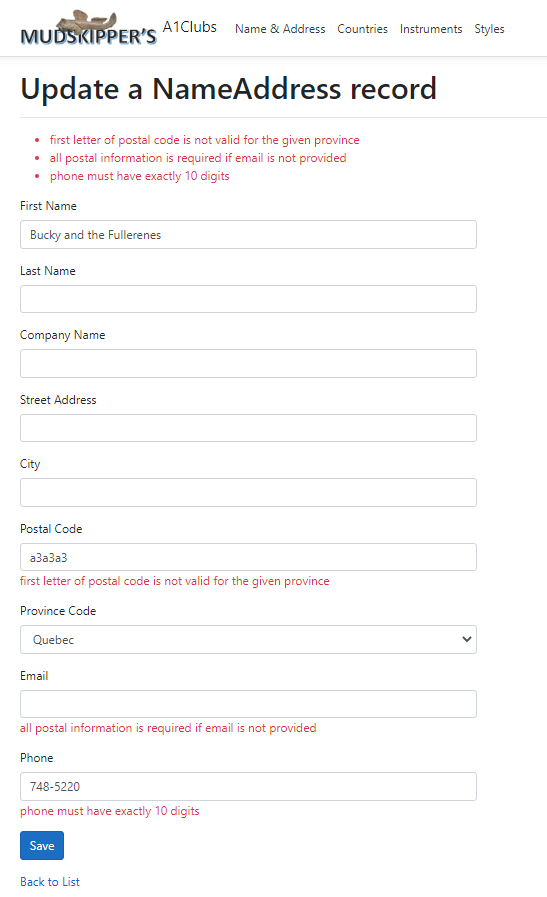
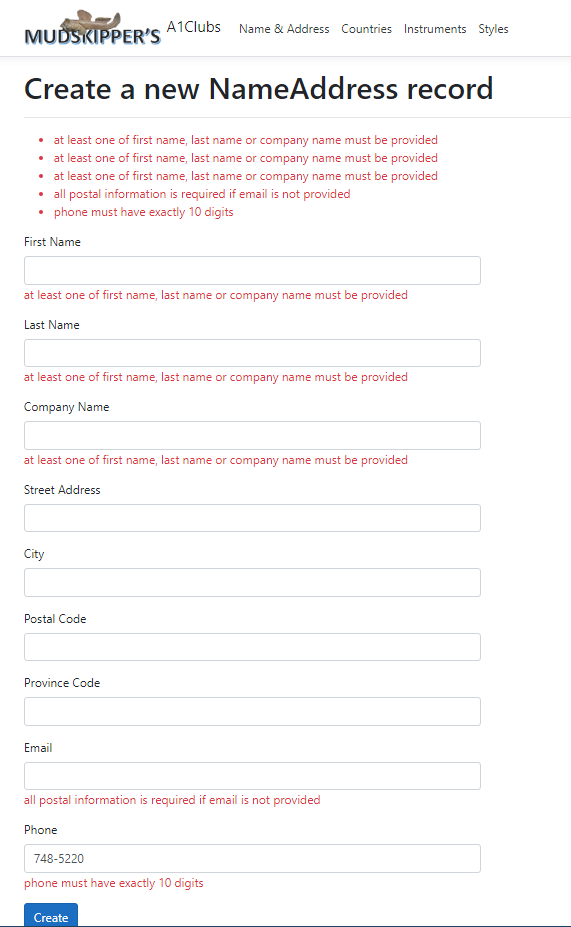
### Setup

1. Continue with your ***XXClubs*** project … or you can download ***A1Clubs*** and start from there.
   1. If starting from ***A1Clubs***, put your name and section in the footer, TempData message before the guest page, and activate Session variables in Startup.cs
2. Add a hyperlink to the menu directed to the Index action of your ***XXNameAddressController.***

### XXNameAddressController

1. Generate the ***XXNameAddressController*** with full CRUD support and Views for the *nameAddress* table.
2. Reduce the Index listing to the fields shown in the example below.
   1. Show full name, not individual first and last names
      1. If both are provided, show LastName, FirstName
      2. If only first or last name is provided, show that
      3. If neither, leave the column blank
   2. Model annotations (later) will fix the field headings



1. In the Create view:
   1. Replace the *provinceCode* drop-down with a textbox and add a field-validation <span> to display any error messages for it. Use another textbox on the page for reference.
2. In the Edit view, display the province name in the drop-down, ordered by name.
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, along with the error.
4. For Delete, put the innermost exception’s message into TempData and return to the Delete view.
5. If the insert, update or delete works, display a success message on the Member Index page via TempData.
   1. Notice I have ValidationSummary showing **All** error messages, so even errors on hidden fields show.

### XXClubs.Utilities

1. Add a .NET Core class library called ***XXClubs.Utilities*** as a separate project in the XXClubs 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 ***XXStringManipulation***, with the following **public static** (class-level) methods:
   1. 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.
   2. Add a method called ***XXPostalCodeIsValid*** that accepts two strings and returns a Boolean:
      1. The first string parameter is the given postal code. The second string is the postal code Regex pattern from the country table.
      2. Return true if the string matches the country’s postal pattern … or is null/empty.
   3. 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.
      3. Shift the first letter of every word in the string to upper case.
      4. Return the newly-capitalised string.
3. Add a custom annotation class called ***XXEmailAnnotation*** to validate the email address:
   1. Let null or empty pass (return ValidationResult.Success), so email can be optional.
   2. Return an error message with the field name if invalid
   3. Suggestion: look into the System.Net.Mail.**MailAddress** class. When you instantiate an object of this class, and provide an email address, it’ll throw an exception if the email is invalid.
4. Add a reference to your class library to your ***XXClubs*** web project.

### NameAddress Class

All ***NameAddress*** 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 *NameAddress* model and do not modify or validate data in controller actions.

1. Create a *MetadataClasses* subfolder inside the *Models* folder.
   1. Create an ***XXNameAddressMetadata*** 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 *NameAddress* model to your ***XXNameAddressMetadata*** class.
   4. Add a separate public *partial* class called ***NameAddress*** (ie: same as the model) to this class file:
      1. Apply the XXNameAddressMetadata class to it, using an annotation.
      2. Turn the *NameAddress* partial class into a self-validating model by implementing the IValidatableObject interface. Implement the interface, creating 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 above.
   2. Trim all strings of leading & trailing spaces, and convert null to an empty string.
   3. Use your ***XXStringManipulation.Capitalize*** method to capitalise *FirstName*, *LastName*, CompanyName, *StreetAddress*, and *City*.
   4. Use your **XXStringManipulation***.****XXExtractDigits*** to reduce *phone* to just digits.
   5. At least one of *FirstName*, *LastName* or *CompanyName* must be specified. All can be specified, but is not mandatory.
   6. *ProvinceCode* is conditionally optional, but if provided:
      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 into an error for that field.
      3. Retain the province record … it and its country are required to validate *postalCode*
   7. *postalCode* is conditionally optional but, if provided:
      1. Produce an error if *provinceCode* is invalid or empty … it’s required to edit a postal code
      2. Otherwise, fetch the country record for the given province.
      3. Use your ***XXStringManipulation.PostalCodeIsValid*** method to validate the postal code to the country’s postal pattern.
      4. Shift the postal code to upper case and, if the address is in Canada:
         1. Confirm the first letter is valid for the specified province
         2. Add a space in the middle, if it’s not there already
   8. *email* is optional, but if provided, it must be a valid pattern:
      1. Use your ***XXEmailAnnotation*** to validate it & provide the error message if not valid
   9. If *email* is not provided, all the postal addressing information is required.
      1. Regardless of the email contents, all postal information that is provided must be validated & reformatted.
   10. *phone* is required:
       1. It must contain exactly 10 digits (use your class library method to remove punctuation and text).
       2. Reformat into dash notation: 519-748-5220

### Hand In

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