Assignment 4 – MVC  
COS318 – Web Programming

For this fantastic assignment, you will be creating some MVC controllers and Razor views in a wonderfully wizarding world. All HTML rendering will be done on the server-side, so you won’t be writing any plain HTML files. This assignment references saving data a few times. Your data storage does not need to be persistent; i.e. it is okay if each time your web program is started the data storage is reset. Expelliarmus!

Your MVC project must contain the following endpoints and associated views (for GET requests):

1. **(40 Points) SpellsController (/spells)**
   1. **/ (index):** Display a list of magic spells currently saved.
      1. The list should be in an ordered html list, with odd numbered rows being different colors.
      2. Each row should contain a link that allows the user to navigate directly to the spell (which will be the view “viewSpell”). Use either a query parameter or a URL parameter for the specific spell index.
      3. Add a link on the page that navigates to /potions.
   2. **/add:** Allow a user to add a new spell to the list.
      1. The data can be sent as JSON data in the body or as form parameters.
      2. The field and button to add new spells should be displayed at the bottom of the /spells/index view from step 1a.
   3. **/viewSpell:** Display a single spell on the page.
      1. This page should use a URL or query parameter spell index that displays the spell at that index on the page.
      2. If it is a valid index, **display just that spell’s name on the page** with its index, but in bigger text than the list from step 1a. If the index isn’t valid, then redirect the user back to the index. Make sure the index displayed on the viewSpell page is the same displayed for the spell in the ordered list.
   4. **/viewSpell (cancel and delete):** Allow a user to delete a spell that they are currently viewing or return to the index
      1. Add a cancel button to the viewSpell page that returns the user to the /spells/index but doesn’t delete the spell.
      2. Add a delete button to the viewSpell page that deletes the currently displayed spell. For the delete endpoint, the controller should either accept a URL or query parameter for the spell index, or accept form data that contains the spell index to delete. After the delete, redirect the user back to the index
2. **(40 Points)** **PotionsController (/potions)**
   1. **/ (index):** Display a list of potion ingredients in html.
      1. Give some way for a user to select two distinct ingredients to mix. (Hint: form checkboxes are a good option here). **Add a ‘mix’ button** that will send the two selected ingredients to POST /potions/addIngredient.
      2. Add a link on the page that navigates to /spells.
   2. **/addIngredient (POST):** Allow a user to add a new ingredient to the potion, by mixing two other things together.
      1. Accept two ingredient strings as **form data**. If both are not specified, you should redirect the user back to the index.
      2. **Mix the two ingredient strings together randomly**, but preserving the order of the letters in each word. For example, if you had ‘apple’ and ‘banana’ as ingredients, one possible result returned would be ‘apbapnalnea.’ The mixed string should then be added to the list of possible ingredients. Then redirect to the user back to the index.
3. **(20 Points)** Code style, formatting, completeness, and quality.
   1. Either the Spell index or the Potion index should be the default page when your project starts. Your project should start at the root page, for example, <https://localhost:44312/>.
   2. The spells list and potion list should both start with at least 5 creative names.
   3. All HTML from this assignment must be rendered on the server side with Razor pages.
   4. All javascript and CSS must be hosted directly by the server.

Stretch Levels

If you already have a lot of experience with MVC and server-rendered views, or if you just like casting spells and mixing potions, try to complete these stretch levels for a reputation bonus. If you try for the stretch levels, make sure to type it in the comments on Moodle so I don’t miss it.

**Harry Potter Level**

Add some CSS to your page to make it look nicer. Background colors, font colors, or anything that looks good.

**Ron Weasley Level**

On the /spells/index page you generated a link using either a URL parameter or a query parameter. Add a second link on the /spells page for each spell. One link should use a query parameter, the other should use a URL parameter.

**Hermione Granger Level**

Add a second button on the /potions page. This button should send the data to the /potions endpoint as URL parameters instead of form data.

**Hagrid Level**

When you mix the strings together, use a weighted average. This means that if you have a very long string and a short string, you’d be able to find characters of the short string throughout the resulting string instead of just towards the beginning.

The Rules

1. No inline styles or inline javascript.
2. Error messages must be “in-page” i.e. no pop-ups or alerts.
3. Any resources not created by you (images, javascript libraries, etc.) must be referenced using a CDN or URL, not directly included in your assignment submission.
4. All requests that submit a body to your server must have their entities validated with appropriate annotations, such as MinLength, Range, or Required.
5. The root path of your server must display the main page of your application.
6. Service/data/model classes must not have any http, request, or response references.
7. Controller entity classes must not be used directly to store data on the server; translate them into a model (data storage) class before saving the data. Conversely, controllers must not send any model classes to the user; translate them into controller entity classes before sending the response.