Assignment 5 – MVC  
COS318 – FA2018

Due Date: October 4th, 2018  
Turn in all files using Moodle

For the fantastic fifth 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):** Render a list of magic spells currently saved. They should be rendered in an ordered html list, with odd numbered rows being different colors. Each row should contain a link that allows the user to navigate directly to the spell. There also should be a link on the page that navigates to /potions.
   2. **/Add:** Accept data that will allow a user to add a new spell to the list. The data either be accepted as JSON data in the body or as form parameters. The field and button to accept new spells should be rendered at the bottom of the /spells view from step a.
   3. **/Delete:** There are two options here. Either accept a URL or query parameter that will allow a user to delete a spell, or accept form data that contains the spell index to delete. After the delete, redirect to /spells.
   4. **/ViewSpell:** Accept a URL or query parameter that will allow a spell index to be sent to the controller. If it is a valid index, **render just that spell’s name on the page** with its index, but in bigger text than the list from step a. This page should also **include a delete button** that will use the /Delete endpoint from step c. Lastly this page should **include a ‘cancel’ button** that returns the user to the /spells page from step a, but doesn’t delete the spell.
2. **(40 Points)** **PotionsController (/potions)**
   1. **/ (index):** Render a list of at least five potion ingredients in html. Make them creative. Give some way for a user to select two distinct ingredients to mix. **Add a ‘mix’ button** that will send the two selected ingredients to POST /potions/AddSpell. There also should be a link on the page that navigates to /spells.
   2. **/AddIngredient (POST):** Accept two ingredients as strings as **form data**. If both are not specified, you should redirect to /potions. Otherwise, **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 /potions.
3. **(20 Points)** Code style, formatting, completeness, and quality.
   1. All HTML from this assignment must be rendered on the server side with Razor pages.
   2. 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, 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 endpoint during step 1b and the /potions endpoint from step 2b, you allowed the endpoint to accept values in either URL parameters or query parameters. Update your controllers to accept whichever type you didn’t accept before so now you support both. Add a second link on the /spells page. One link should use the query parameter, the other should use the 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 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. Service/data/model classes must not have any http, request, or response references.
5. 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.