ISC 559 – IS Application Design and Implementation
Assignment 6: Passing Data from Controller to View – 10 points

In this project, we will add a view and controller methods that will allow the user to search the PerformingActs in our database. This assignment will also illustrate the use of ViewModels that are bound to views. ViewModels allow us to neatly package and pass information back and forth between a controller and a view in a format that is specific to the needs of a particular view.

Complete the following steps:

In the Solution Explorer, create a new folder at the root of your project named ViewModels. In this folder, create a new class named PerformingActSearchViewModel. At the top of your class file, make sure to write a using statement that will allow you to access your Models namescape (`using NameOfYourProject.Models; // replace NameOfYourProject with the actual name of your project`). Your class should be defined with the following properties and constructor method:

```
public class PerformingActSearchViewModel
{
    public PerformingAct PerformingAct { get; set; }
    public string SearchError { get; set; }
    public List<PerformingAct> ResultList { get; set; }

    public PerformingActSearchViewModel()
    {
        ResultList = new List<PerformingAct>();
    }
}
```

In your _ViewImports.cshtml file (under the Views folder), add the following line of code, which will automatically give all of your views access to the ViewModels namespace (replace NameOfYourProject with the actual name of your project; also remember that you don't need a semicolon at the end of your lines of code in the ViewImports file):

```
@using NameOfYourProject.ViewModels
```

In the Views folder, under the PerformingActs subfolder, create a Razor View named Search. Replace the contents of the Search view with the following code:

```
@model PerformingActSearchViewModel
```

```
<form asp-action="Search">
    <div class="form-group">
        <input asp-for="PerformingAct.Name" class="form-control"
/>
        <input type="submit" value="Search for Act" class="btn
btn-default" />
    </div>
</form>
<div class="text-danger">@Model.SearchError</div>
@if (Model.ResultList.Count > 0)
{
    <table class="table table-condensed table-bordered">
        <thead>
            <tr>
                <th>
                    Acts Found
                </th>
            </tr>
        </thead>
        <tbody>
            @foreach (var pa in Model.ResultList)
            {
                <tr>
                    <td>
                        <a asp-action="Details" asp-
controller="PerformingActs" asp-route-id="@pa.PerformingActID">
                            @pa.Name
                        </a>
                    </td>
                </tr>
            }
        </tbody>
    </table>
}
```

A few things to highlight in the Search view:

- Notice that the asp-action attribute of the form tag is set to Search. This means that when the user clicks the submit button on this form, it will get submitted to the HttpPost method named Search in the PerformingActsController.
- The asp-for attribute of the first form input element is set to PerformingAct.Name. This means that when the form gets submitted to the controller, we will be able to access what the user provided in this textbox by getting the Name property from within our view model's PerformingAct property (we'll do this in the later steps of this assignment).
- The view model's SearchError property is embedded within a div tag with the class "text-danger". This means that if the SearchError property has a value, it will get embedded in that div tag, which will display the text in the color red.

- If the view model's ResultList property contains PerformingAct objects (Count > 0), we'll show an HTML table of results and loop through the list to create a new HTML table row for each PerformingAct in the list. The PerformingAct's name will be shown as a link that will redirect the user to the Details view for that PerformingAct. This is accomplished by setting the asp-action attribute equal to the controller method we want to call (Details), the asp-controller attribute equal to the controller that houses the method (PerformingActs), and the asp-route-id attribute equal to the PerformingAct's unique identifier (PerformingActID).

In the PerformingActsController, write an HttpGet method named Search. Because this method is and HttpGet, you don't need to decorate it with [HttpGet] above the header -- your project assumes that methods without decoration are HttpGet methods. This method does not receive any parameters. In the body of the method, include the following line of code, which will return an empty Search view to the user (make sure you have the using clause included for ViewModels()) :

```
var model = new PerformingActSearchViewModel();

return View(model);
```

After the HttpGet version of the Search method, create an HttpPost version of the Search method. Because this method is HttpPost, you will need to decorate it with [HttpPost] above the method header. This method should receive a PerformingActSearchViewModel object named model as its parameter.

In the body of the HttpPost Search method, do the following:

- Get the list of performing acts based on the search criteria using the following line of code:
  ```
  var results = _context.PerformingActs.Where(pa =>
  pa.Name.ToLower().Contains(model.PerformingAct.Name.ToLower
  ())).ToList();
  ```

- Next, write an if statement that checks whether the count of results is less than or equal to 1
- Inside the if statement, write another if statement that checks whether the count is equal to exactly 1. In the body of this if statement, include the following line of code, which will automatically forward the user to the PerformingAct's Details page if they are the only search result:
  ```
  return RedirectToAction("Details", "PerformingActs", new {
  id = results[0].PerformingActID });
  ```

- Outside of the inner if statement, but still inside the if statement checking whether the results count is less than or equal to 1, set a the view model's property named SearchError equal to "No performing acts found with the search criteria provided".
  ```
  model.SearchError = "No performing acts found with the
  search criteria provided";
  ```

- Outside of all the if statements (just before the end of the Search method), set the view model's ResultList property equal to results.
  ```
  model.ResultList = results;
  ```
- Return the Search view and pass the view model as its model.
  ```
  return View(model);
  ```

Run the project. Navigate to the Search view by adding "/PerformingActs/Search" to the end of the address in your browser's address bar. When the Search view loads, you should see a textbox and a button. You should be able to type a search string into the textbox. When you click the Search for Act button, you will either be taken directly to an act's page, shown a list of acts that match the criteria with links that go to their page, or an error message that says there were no matches. You can use the following test cases to make sure it's working properly:

bey - should take you right to Beyonce

dav - should show a list that includes Dave Chappelle and David Blaine. You can click their names to be taken to their pages

asf - should show the error message with no matches