CS526 Enterprise and Cloud Computing
Stevens Institute of Technology—Fall 2020
Assignment One—Server Side Processing with MVC

Description:
This assignment requires the use of Visual Studio 2019 Community Edition and ASP.NET Core MVC 3.1. You are provided with a partially completed Web app, which you should complete. The app includes a banner image and two style sheets (one default, one for ADA clients). You may modify the look and appearance of the Web site (e.g. by changing the banner image), but you should leave the general structure: a header area, a navigation bar with navigation links for the Web site, a sidebar on the left, a content area on the right, and a footer. You will use this template to build a simple social network that is intended to allow people to share photographic images. The layout for all Web pages is specified in the `Layout` view (rather than the default `_Layout` view); you will need to update the `_ViewStart` configuration view to specify that this be used as the template for all views.

The app has three controllers, `Home`, `Account` and `Images`.

The `Home` controller has a single action, `Index`, that displays a welcome message, personalized for a registered user.

The `Account` controller has an action, `Register`, for the `Account` controller, that allows a user to register a cookie with their userid and ADA preferences. We will see more realistic ways of recording registered users in later assignments.

You will need to complete the `Register` action to save the registration information in a cookie.

The `Images` controller has an action, `Upload`, that allows a registered user to upload an image to the server. This page should save not just the image file, but also a caption and description specified by the user, as well as the date that the photograph was taken. To simplify things, the user should also be required to provide an image identifier, with this identifier used to name the image file, and the image information file, on the server. The controller should define an action (annotated with `HttpGet`) for displaying an empty form, and another action of the same name (annotated with `HttpPost`) for processing the input. When an upload is complete, display the uploaded image with its metadata. If there are validation errors, then the original data should be retained in the form, to allow the user to make amendments.

You will need to complete the Register action to save the image and its metadata in a JSON file on the server. You will also need to complete the view for uploading the image metadata, using form tag helpers to enable input validation. Finally you will

need to add error diagnostics to the validation logic already specified in the `Image` model.

The `Images` controller has another action, `Query`, that allows a user to search for an image, by image identifier, in the Web app. This action returns a form for searching based on the image identifier. Another action, `Details`, processes the contents of this form, when it is submitted by the user. With an image identifier as a parameter[1], this action returns a page that displays the specified image. This should display the image and the information about that image: its caption and description, the date the photograph was taken, *and the user who uploaded the image*. If there is no image with that identifier on the Web site, then display the original form with a message that that identifier does not exist, allowing the user to enter another image identifier. Both the `Query` and `Details` actions should be annotated to be executed on a GET request.

Attempting to perform the upload or query action without registering should result in redirection to the registration page.

The code performs some simple input validation. For the image identifier in the upload and query pages, it uses a `RegularExpression` validation attribute to ensure that the identifier is composed only of alphanumeric and underscore characters. For the image specified in the upload, it places a limit on the content size of allowable images, and only accepts the image if it can be validated as a JPEG image. It requires both a caption and a description, and a well-formed date, for the image. This validation is specified by attributes on the `Image` model class, so it is necessary to use tag form helpers in the input elements to perform this validation (and use the `asp-validation-for` attribute to report on validation errors). Handle any processing errors such as file input-output failures on the server, by providing an informative message (rather than a stack dump) to the user.

Submission:
Submit your assignment as a zip archive file. This archive file should contain a single folder with your name, with an underscore between first and last name. For example, if your name is Humphrey Bogart, the folder should be named Humphrey_Bogart. This folder should contain a single folder for your solution named `ImageSharingWithUpload`.

Record short MP4 or Quicktime videos (*note the allowable formats*) demonstrating your deployment working on your local machine. Your videos should demonstrate:
1. Launching the Web app from Visual Studio.
2. Verifying that the Registration action works (including ADA flag).
3. Verifying that the Upload action works (including validation checks).
4. Verifying that the Query (GET and POST) actions work.

---

[1] The `Details` action takes an Image model as its argument, in order to enable input validation on the identifier.

5. Verifying that the Error page works.

See the rubric for detailed explanation of what is required for testing. Make sure that your name appears at the beginning of the video. For example, display the contents of a file that provides your name. *Do not provide private information such as your email or cwid in the video*. Be careful of any "free" apps that you download to do the recording, there are known cases of such apps containing Trojan horses, including key loggers.