CS526 Enterprise and Cloud Computing
Stevens Institute of Technology—Fall 2017
Assignment Two—Data Models and Databases

Description:

- This assignment requires the use of Visual Studio 2017.
- You should create an ASP.NET MVC project (MVC5, .NET Framework 4.5, C# and Entity Framework 6). Choose the wizard for creating a Basic Project (i.e., a minimal project). Name this project `ImageSharingWithModel`. You should also create the unit tests project, `ImageSharingWithModel.Tests`, and place both in the same solution. The structure of your presentation logic (default layout and styles) should be the same as in the previous assignment, but *you must make this a new project*, because of the way that Windows binds projects in the registry. You should create a new project and copy the relevant files from the earlier project to this new project. Alternatively, you can try copying your old project, editing the project descriptors for the solution, and changing the application guid, but you are responsible if the project fails to run for the grader because of e.g. registry problems.
- For this assignment, you will work with three models that are stored in a database:
    - The `Image` model saves information about uploaded images: caption, description, date picture taken, location of image file (on the server), identity of uploader, and also a tag for the image. Every image has exactly one tag.
    - The `User` model saves user information, which in this case is just userid, an indication if they are visually impaired, and also a list of all the images this user has uploaded.
    - The `Tag` model saves information about image tags, which in this case is just the tag name, and also a list of all the images that have this tag.
- Use the LocalDB development database management. You may need to install SQL Server 2016 Express LocalDB. Unlike SQL Server 2012 LocalDB referenced in the demos, version information no longer appears in the server name in SQL Server 2016 LocalDB; use "`(localdb)\mssqllocaldb`" as the name of your server. Set up an initializer for the database, so that it is re-initialized with sample data every time the application is executed. Entity Framework now allows you to work without a default connection string, but to have some control over the location of your database file, you can define the following connection string (do not use "DefaultConnection" as the name of your connection string):

```
<add name="ImageSharingWithModel"
     providerName="System.Data.SqlClient"
     connectionString="Data Source=(LocalDb)\MSqlLocalDb;Initial
Catalog=ImageSharingWithModel;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|\ImageSharingWithModel
.mdf" />
```

  Index all of the database tables using integer primary keys.
- Create three controllers, `Home`, `Account`, and `Images`.

- There is a default action, `Index`, for the `Home` controller, that displays a welcome message, personalized for a logged-in user.
- There is an action, `Register`, for the `Account` controller, that allows a user to register themselves. This action will create a record in the database table to store user information for this user, and automatically logs them in. This action should fail (gracefully) if the specified userid is already in the database.
- There is another action, `Login`, for the `Account` controller, that allows a user to save a cookie identifying themselves in their browser. The userid they specify must already exist in the database. If the user indicated when they registered that they are visually impaired, then whenever they log in to the Web site, it should display text in a large font. Do not bother with password-based authentication for now, we will consider alternatives for authentication in a future assignment.
- The `Images` controller should provide an action called `Upload`, that allows a logged-in user to upload an image. In addition to the caption, description, date taken and image file, the user should provide a tag for the image, chosen from a dropdown list. This latter list in turn should be populated from the tags defined in the database. The user should not explicitly specify an image identifier, instead this is generated automatically when the image is added to the database. If there are validation errors, then the original data should be retained in the form, to allow the user to make amendments.
- The `Images` controller should provide an action called `Details`, that shows the details of an image identified by its image identifier (as part of the URI). The detailed information should include image identifier, caption, description, date taken, image tag and uploader (as well as the image itself).
- The `Images` controller should provide an action called `Edit`, that allows the details of an image, identified by its image identifier, to be modified by the user. The details that may be modified include caption, description and date taken. The logged-in user must be the same as the user who originally uploaded the image.
- The `Images` controller should provide an action called `Delete`, that allows the details of an image, identified by its image identifier, to be deleted by the user. The logged-in user must be the same as the user who originally uploaded the image.
- The `Images` controller should provide an action called `ListAll`, that displays a table with the caption, tag (name) and uploader (userid) of every image in the database. Each row in the table should have links for the `Details`, `Edit` and `Delete` actions. Only show the Edit and Delete buttons for those rows where the image was uploaded by the logged-in user.
- The `Images` controller should provide an action called `ListByUser`. This displays a form where a userid is chosen from a dropdown list, and displays a table of all images uploaded by that user, similarly to the `ListAll` action.
- The `Images` controller should provide an action called `ListByTag`. This displays a form where a tag is chosen from a dropdown list, and displays a table of all images with that tag, similarly to the `ListAll` action.

- Attempting to perform an image action without logging in should result in redirection to the login page.
- Perform some simple model validation. For the registration page, use a `RegularExpression` validation attribute to ensure that the userid is composed only of alphanumeric and underscore characters. For the image specified in the upload, place a limit on the content size of allowable images, and only accept the image if it can be validated as a JPEG image. For uploading an image and editing image data, make sure that the caption and description are no longer than the space allocated for them in the database, and ensure that the date specified is a valid date.
- Handle any processing errors such as input-output failures on the `server`, by redirecting the user to a friendly error page (action `Error` of the `Home` controller).

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 `ImageSharingWithModel`, with projects `ImageSharingWithModel` and `ImageSharingWithModel.Tests` (and DLLs in the folder `packages`).

In addition, record mpeg, avi or Quicktime videos demonstrating your deployment working. Make sure that your name appears at the beginning of the video, for example as the name of the administration user who manages the Web app. *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.