CS526 Enterprise and Cloud Computing
Stevens Institute of Technology—Fall 2020
Assignment Four—Cloud Storage

Description:

- This assignment requires the use of Visual Studio 2019. You are provided with an ASP.NET Core MVC project (Version 3.1, C#). The structure of the presentation logic (default layout and styles) is the same as in the previous assignments. Your project includes the models from the previous assignment, and the basic UI. In this project, you will move the application you have developed in the last three assignments into the cloud. The external API as far as users of your system are concerned will not change over the previous assignments, (aside from on additional piece of functionality), but your application will now run as a cloud application.

- Instead of storing information in SQL Server, you should now use **SQL Database** to store this information in the cloud. Use Entity Framework as before to access the underlying database. Your main task here will be defining a connection string to use SQL Database. You will continue to store the Users, Images and Tags tables in this database, but see below. You should be careful to set up SQL Database in the *serverless tier*, otherwise you may incur substantial charges for the use of Azure[1].

- Use **Azure Blob Storage** to save images, instead of storing them on the server file system. Store all of the images in a single blob container. *You have two options here; pick one and document your choice.* One is to simply store images in blob storage as though the container is a file folder, and store information about those images in SQL Database, as you have been doing. This is the easy option, but does not get you far into blob storage. Another option is to use a blob naming convention where a blob name consists of the user name of the user who uploaded the image, followed by a delimiter ("/"), followed by an image identifier that is unique for that user. You can generate this key by storing a sequence identifier in the User table in SQL Database, and incrementing that identifier each time the user uploads an image. However, if you delete a user account with this scheme, you will still need to delete each image uploaded by that user individually, rather than issuing a single delete request for all images, with that user name's name as their "virtual directory" name. For this reason, Azure blob storage now supports a hierarchical file system, where you can have a directory for each user, containing the images uploaded by that user. With this option, you would place the information that was in an image record into the

[1] Beware that using SQL Database is not the same experience as with SQL Server. In particular, you may need to deal with transient failures, particularly due to cold start of a database in the serverless tier. The SQL client tries to mask these failures, if you configure it with `options.UseSqlServer(…, opts => opts().EnableRetryOnFailure())` in the startup configuration, but you may have to deal with some of these issues yourself.

metadata for the blob for that image instead. If you pursue this second option, you will still need the Images table, now just containing a database primary key and the URI for the blob in Azure storage, because you still need to represent the one-to-many relationship from tags to images. Although theoretically this relationship could be represented using Azure Table Storage, in fact we would like this relationship to be many-to-many (you have not been asked to support this many-to-many relationship, yet), and doing this efficiently in Azure Table Storage would require secondary indexes. For now, leave the one-to-many relationship from tags to images represented unchanged in SQL Database, no matter which of the two options you choose above. If you choose the second option, you no longer need to represent the one-to-many relationship from users to images in SQL Database, but you still need the Users table for other purposes (e.g. for authentication and authorization). *You will need to add the functionality for removing an image from blob storage, adding this operation to the* `ImageStorage` *service class and the* `IImageStorage` *interface.*

- Add a table in **Azure Table Storage** which records every time that an image and its details are viewed, and by whom. Call this table `ImageViews`. Assume that the most common query is "What images were viewed today?" so you can use the same key organization as in the example in the lectures. Use a partition key based on date, and use a primary key based on image id. Provide an additional action in the `Account` controller, called `ImageViews`, that displays a report of viewing for a particular day, organized by image id (display each image caption in the report as well). Create a separate role, called `Supervisor`, to authorize access to this action. *The code you are provided with just lists image views for today. Edit this to provide a drop-down list of the last two weeks, allowing the user to pick one of those days to view the logs. You will need to modify the* `LogContext` *service class and* `ILogContext` *interface to provide the ability to list image views for a particular day.*

- For development purposes, you can continue to use the local SQL Server database for storing metadata, as well as storing images on the local file system. The development `appsettings.json` file configures this, with an empty connection string for blob storage triggering local storage of images (see the `ImageStorage` class). You may also want to use the Cosmos DB emulator to test usage of the Table storage locally. Edit the `launchSettings.json` file to have the app run in Production mode, when only cloud storage is used. This is what you need to use for demonstration purposes, when everything must run in the cloud. You will need to edit the production `appsettings.json` file with your connection strings, after setting up storage in Azure.

- Deploy your application using Windows Azure App Service. You can use your Stevens email to register for a student account, which provides some benefits including credit for the use of Azure that should be sufficient, provided you are careful how you use the service.

- Do not allow public access to images uploaded to blob storage. Restrict access to your current IP address.

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 `ImageSharingCloudStorage`, with project `ImageSharingWithCloudStorage`.

In addition, record mpeg or Quicktime videos demonstrating your deployment working. See the rubric for further details of what is expected. 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.