Assignment 3 – WebApi  
COS318 – Web Programming

In this assignment, we will be returning to Star Wars. “I think my eyes are getting better. Instead of a big dark blur, I’m seeing a big bright blur.” In the web requests assignment, you wrote javascript that communicated with a server that was hosted on Azure. In this assignment, you’ll be recreating that server. You’ll be able to use your previous assignment code to test your new server. Also, the server from that assignment is still deployed to Azure at “https://webrequestsserverbethel.azurewebsites.net/api/favoritecharacters” so you can verify if your server is acting in the same way. Your server will be tracking a list of people and their favorite Star Wars characters. Your server does not need to persist any data, so it is okay if data is lost when your server is stopped.

Your FavoriteCharacters controller must support the following endpoints:

1. **(20 Points) POST**
   1. Accept JSON data for FirstName, LastName, and FavoriteCharacter and save it on the server. The CreatedTime should be saved in the model, but the value should not be changeable by the client.
   2. Validate that FirstName and Character are sent in the JSON and have length of at least 1 using entity annotations.
2. **(10 Points) GET**
   1. Return a JSON object of all the entries currently stored on the server.
      1. The returned object should have a single key “favoriteCharacters” which is an array of all the favorite character entities on the server.
      2. The “views” key should be set for each entry in your response JSON equal to the number of views that entry has so far (as an int).
3. **(10 Points)** **GET/{index}**
   1. Return a JSON object containing the name, favorite character, and created time data for the specified index.
      1. The “views” key should be set for this entry in your response JSON equal to the number of views this entry has so far (as an int).
4. **(10 Points) DELETE /{index}**
   1. Delete the entry at the index from the server.
5. **(10 Points) GET /{index}/views**
   1. Return a JSON object of views for a particular entry stored on the server.
      1. The returned object should have a single key “viewDates” which is an array of all the view date entities on the server.
6. **(20 Points) POST /{index}/views**
   1. Accept JSON data for a view date. It only has a single key/value pair, “ViewDate.” The value for this “ViewDate” key will be a string.
   2. Validate that ViewDate is sent in the JSON and has a length of at least 1 using entity annotations.
7. **(20 Points)** Code style, formatting, completeness, and quality.
   1. Be careful with the names of your controller and the names of the entity properties. They must match the names written here so that the code from your previous assignment works with this server. Make sure to have “api” in the Route for your controller and that it is named correctly.
   2. All indices should be validated to make sure the entry exists. If it does not, return an appropriate status code.
   3. The list of entries should begin with one entry, and the size of the list should be capped at 30. During a POST request to create a new entry, clear the list first if it has too many entries.
   4. Remember that POST requests should return the created data.

Stretch Levels

If you already have some experience with WebApi, or if you aren’t frozen in carbonite, 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.

**Purple Lightsaber Level**

Instead of using StatusCode or StatusCodeResult, use ContentResult to also return a helpful message about what was wrong with the request.

**Green Lightsaber Level**

Add a new endpoint for retrieving a specific view from an entry. Be careful that you are still returning well-formatted JSON! If you did the Purple Lightsaber level, there should be different error messages here depending on which index is out of range.

**Blue Lightsaber Level**

Add a PATCH method to your controller that changes the FirstName, LastName, or FavoriteCharacter only if they are not null and not empty strings in the request body. PATCH was not a part of assignment two, so you will need to write some new javascript or use a tool like Swagger to test your new method. Hint: You will need a new entity class with different annotations. Second Hint (we’ve had one yes, what about SECOND hint?): PATCH requests edit a particular entry in the list, so it would be easiest to use the index chosen by Force Read as the index that you should PATCH.

The Rules

1. No inline styles or inline 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. **New Rule:** All requests that submit a body to your server must have their entities validated with appropriate annotations, such as MinLength, Range, or Required.
5. **New Rule:** The root path of your server must display the main page of your application.
6. **New Rule:** Service/data/model classes must not have any http, request, or response references.
7. **New Rule:** 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.