Assignment 10 – If-Match  
COS318 – Web Programming

One thousand years ago, superstition and the sword ruled. It was a time of darkness; it was a world of fear. It was the age of Gargoyles. (https://www.youtube.com/watch?v=ygrEVnrg3Ic) In this assignment, you’ll be ensuring that updates to your gargoyles don’t overwrite each other, since perhaps more than one person would be writing to your server at the same time. Maybe David Xanatos is using your controller…he never was too fond of the gargoyles.

This assignment will have one WebApi project with a GargoyleController and an html page with javascript that will make requests to the controller. The javascript must keep track of a current ETag with GET requests so it can use PATCH with If-Match.

1. **(10 Points) GargoyleModel**
   1. Create a class that represents a gargoyle, GargoyleModel.
   2. All gargoyles have four string properties; Name, Color, Size, and Gender. These properties should have a length of at least three if they are specified, but only Name is required. Gargoyles also have an “Updated” property that is of type DateTime which has a value of when the gargoyle was last updated.
2. **(20 Points) GargoyleDatabase**
   1. Create a GargoyleDatabase that maintains a list of gargoyles currently in the system. The Name of the gargoyle is the “key” of the database, so there must never be two gargoyles in the database with the same name. (Hint: Dictionary instead of List)
   2. It will need methods that allow the creation and lookup of GargoyleModels.
   3. It should be initialized with at least two values when the server starts.
3. **(30 Points) GargoyleController**
   1. GargoyleController must support five endpoints. GET (all gargoyles), GET (a specific gargoyle by Name), POST, PUT, and PATCH
   2. GET / return all gargoyles currently on the server. This endpoint will not send any ETags.
   3. GET /index (a specific gargoyle) must set an ETag header representing the gargoyle in some way. GET should use the gargoyle’s Name as the URL parameter, not an integer index.
   4. POST / must not allow two gargoyles with the same name to be in the system at the same time. If a second gargoyle is attempted to be created and it already exists in the database, return the appropriate error status code (Hint: There is a better status code to use than just the generic BadRequest).
   5. PUT /index must replace a gargoyle already at that index or create it if it doesn’t exist. PUT should validate that the gargoyle name that is being added/replaced matches the URL index parameter. It also must verify that the If-Match header of the request either is a wild card “\*”, or matches the ETag value of the gargoyle to be replaced.
   6. PATCH requests must verify that the If-Match header of the request either is a wild card “\*”, or matches the ETag value of the gargoyle to be edited. Return appropriate status codes if the If-Match header value is not a value that allows an update.
4. **(20 Points) Html and Javascript**
   1. Create an html page that can make requests to the server. It should have buttons for GET all, GET, POST, PUT, and PATCH. It should also have input fields for the four text properties of a gargoyle.
   2. The html page should display the current ETag (after making any GET request for a specific gargoyle).
   3. The html page must show error messages and status codes when requests fail (such as not having the correct If-Match header values)
   4. PATCH must only send the text fields that are non-empty. This means your JSON structure that you are sending to the server will change depending on which text fields have values on your html page.
5. **(20 Points)** Code style, formatting, completeness, and quality.

Stretch Levels

If you already have a lot of experience with optimistic concurrency control, or if you just turn to stone during the day, 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.

**Hudson Level**

Add some CSS to your page to make it look nicer. Background colors, font colors, or anything that looks good.

**Lexington Level**

Add another button to your html page that sends a PATCH request with “\*” as your If-Match header value, forcing the update to be accepted.

**Goliath Level**

Implement If-None-Match in GET requests to your server. This means that if the ETag matches on a GET request, the server will return an empty body with a 304 -Not Modified status code. When this is the response from your server, ensure that your html page also displays that the data has not been modified since the last GET request.

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. All requests that submit a body to your server must have their entities validated with appropriate annotations, such as MinLength, Range, or Required.
5. The root path of your server must display the main page of your application.
6. Service/data/model classes must not have any http, request, or response references.
7. 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.
8. All service class instances must be obtained using dependency injection.
9. You may not use any synchronous methods in your C# code wherever there is an async option.