Coding Challenge

You want to work at Mitchell? Great! We want you here, too. But first – we want to see your abilities in action. The following is a little programming challenge that will help us figure out where your strengths are and how we can best utilize your talents. If you finish and return this task, you will have a next round interview here at Mitchell.

We hope that you take this challenge seriously and work on it without the assistance of other developers. That said, developers don't work without resources, so feel free to leverage the internet or books if necessary.

Above all, this programming problem is less about IF you solve the problem and more about HOW. The elegance and extensibility of your solution is important. Let this be your chance to demonstrate your understanding of good software principles.

So enough talk – let's get to the fun stuff.

Challenge

Required

Implement a RESTful web service that performs CRUD operations (Create, Read, Update, and Delete) for a Vehicle entity.

A Vehicle is a simple object defined as follows:

```
public class Vehicle
{
    public int Id { get; set; }
    public int Year { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }
}
```

Your RESTful service must implement the following routes:

GET vehicles
GET vehicles/{id}
POST vehicles
PUT vehicles
DELETE vehicles/{id}

A working implementation of this service can be found here.

Additionally any solution must employ the following:

1) Usage of either C# or Java.
2) Some form of automated testing.
3) Some form of in-memory persistence of created vehicle objects.
4) Function properly with the provided test web client.

The test client can be accessed here (please use Chrome). You will need to change the service URL to the URL of your hosted service when you are ready to test. Keep in mind that your service will most likely be using the non-secure HTTP protocol (and the test client is served over HTTPS which is secure) so you might need to allow unsafe scripts to execute on the web page to test your service. You can do this for Chrome with the provided steps.

Optional

Got all the required stuff nailed down and want to enhance your service a little further? Here are some additional features you can add to make the service more interesting!

1) Add validation to your service.
   - Vehicles must have a non-null / non-empty make and model specified, and the year must be between 1950 and 2050.
2) Add filtering to your service.
   - The GET vehicles route should support filtering vehicles based on one or more vehicle properties. (EX: retrieving all vehicles where the 'Make' is 'Toyota')
3) Write an example client for your service!
   - Client should leverage AngularJS. (1.x or 2.0)
   - Any other libraries used are entirely up to you!

Evaluation

Your implementation will be evaluated on the follow criteria:

1) Whether or not the service implemented the required routes successfully.
2) The maintainability / readability of the service's code.
3) The flexibility of the service's code. How well does the code base adapt to changing requirements?
4) The testability of the service's code.
5) The reusability of the service's code. Can facets of the service's code be used for other services or applications?

Good luck!