README

The purpose of this solution is to prove that the Photos API is working successfully on the deployed environment. The following sections will describe the the different components of the solution.

Configuration

You will find a configuration file within the Configs folder. For now, it contains the base address of the Photos API, but it can be extended to hold other settings required by the solution.

Constants

This folder holds scenario context constants and test constants. You will notice that some of the test constants are only used once, but they were created to improve readability and to avoid the use of “magic numbers” (i.e. numbers with no obvious purpose).

Features

There is only one feature file for the Photos API. It contains scenarios that will prove that the photos can be created, retrieved, updated, deleted, and filtered.

Models

There are models for API requests, API responses, and the configuration file settings.

Scenario Steps

There is a single steps file for the Photos API feature file. There are a few steps that do not execute any code, but have been added for readability purposes. The scenario context object and an API client object have been injected into the steps file to simplify the handling of API requests and context variables.

Support Folder

There is a helper class to create any test data that is required by the test scenarios.
There is an API client class to handle the submission of requests to the Photos API.
The *TestDependencies* class uses *Autofac* to register the configuration file object and the API client object. Additionally, it will register any class that has the Specflow *Binding* attribute, which gives you the ability to inject a step file object into another step file class. This may be useful for future work. However, I am aware that *YAGNI* may apply here.

Potential Improvements

The following paragraphs describe a list of improvements to the solution that I ran out of time to achieve.

The number of constants for the scenario context keys could be reduced, but I wanted to ensure that they were unique keys for each test scenario.

The following scenarios need to be included in the feature file:

- A user cannot update a non-existent photo
- A user cannot update the unique ID of an existing photo
- A user can update a single field within an existing photo (via the PATCH method)

The API client class could be refactored to set up the HTTP client object as a property for the class instead of using it as a local variable within each method. The setting up of the base address for the API could be done in the class constructor. There is a better way to set up the parameters for *SearchPhotoAsync* method, at the moment it can only filter on one field-value pair.