Gallery API

Business Case

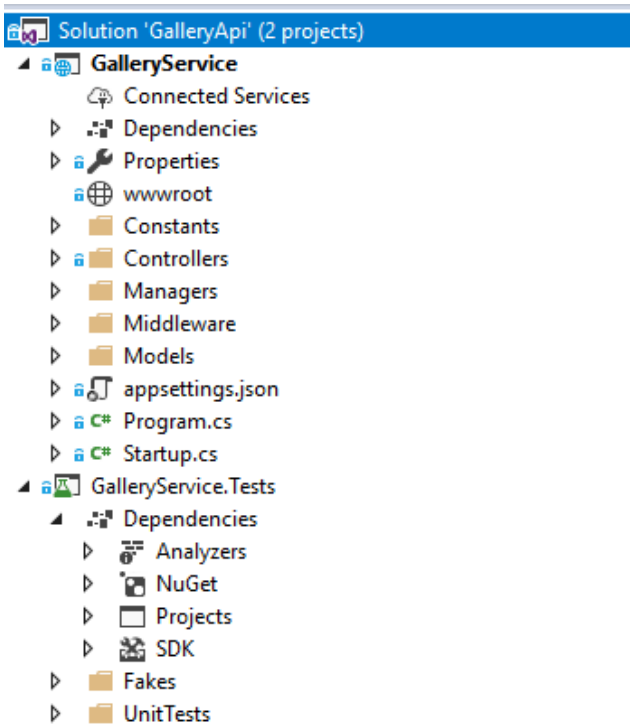
We are building a gallery service, an API based application that allows an administrator to manage the albums and photos from a gallery.

The *product requirements* for this initial phase are the following:

- **Retrieve entire gallery items** – A client should be able to view all albums and photos available into a gallery
- **Retrieve gallery items for specified user** – A client should be able to view all albums and photos available for a user

Solution Structure

Below is a snapshot of the solution open in Visual Studio (2017 Professional Edition) IDE.



Based on the simplicity of this initial phase there are only two projects defined.

Project	Responsibilities
GalleryService	○ Retrieve entire gallery details ○ Retrieve gallery details for user
GalleryService.Tests	○ Contains unit tests for gallery service functionalities

GalleryService project contains the main functionality of the application.

When the client call *the retrieve entire gallery details functionality*, the application will:

- Call, combine and return the results of:
- http://jsonplaceholder.typicode.com/photos
- http://jsonplaceholder.typicode.com/albums

When the client call the retrieve gallery details for user, the application will:

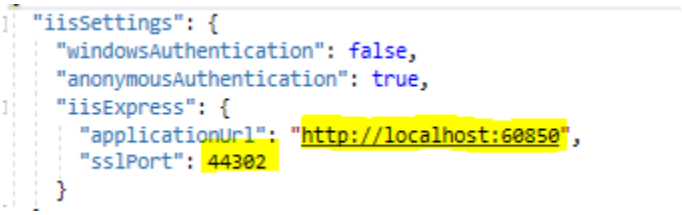
- Allow an integrator to filter on the user id – returns the albums and photos relevant to a single user – it uses the below gallery data source provider

Prerequisites

- Visual Studio 2017
- Net Core 2.1
- Postman

Steps:

1. Clone github repository to your local machine
2. Open solution using Visual Studio
3. Check launchSettings.json file



For further testing you will use one of these ports (https or http) – Postman testing scenario.

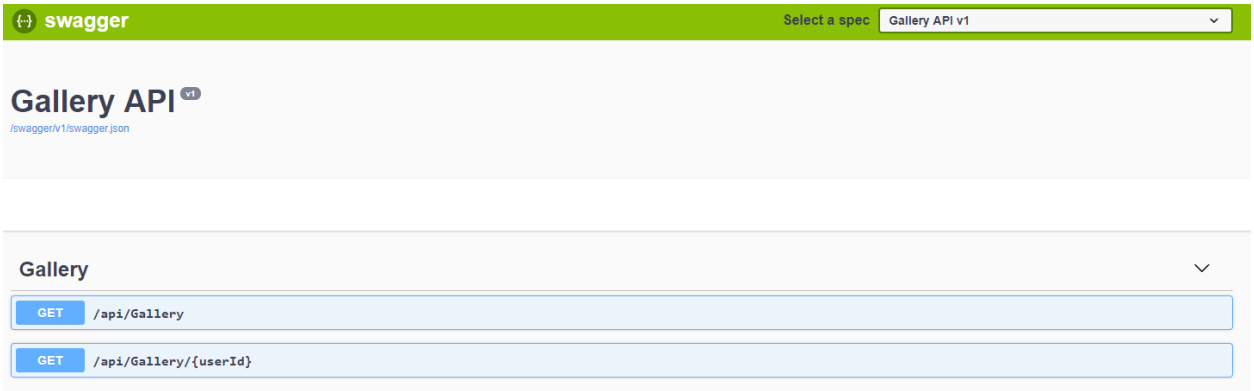
4. Build and Run solution – using IIS Express

Testing

The Gallery API can be test using Postman or Swagger.

Swagger

When solution starts, Swagger UI will be display:

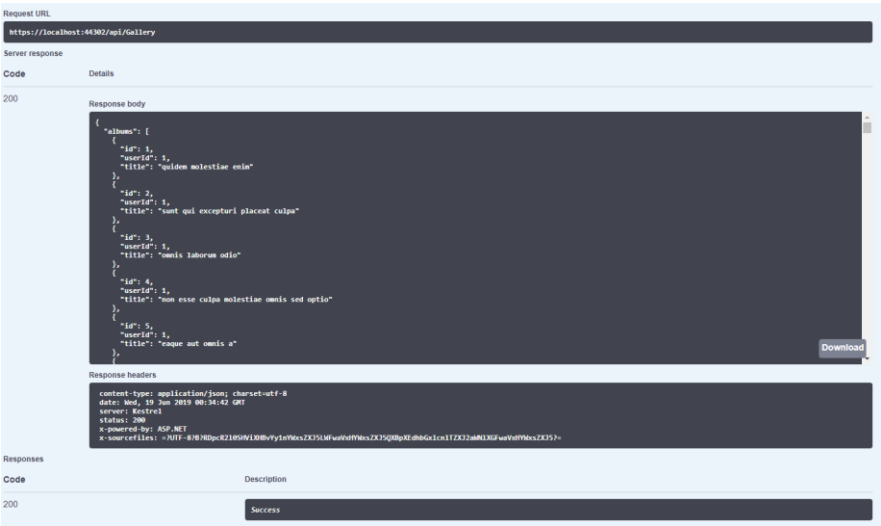


Requirement 1 [Retrieve entire gallery details] testing:

Steps:

1. Expand /api/Gallery tab item
2. Click right side button 'Try it out'
3. Click 'Execute' button

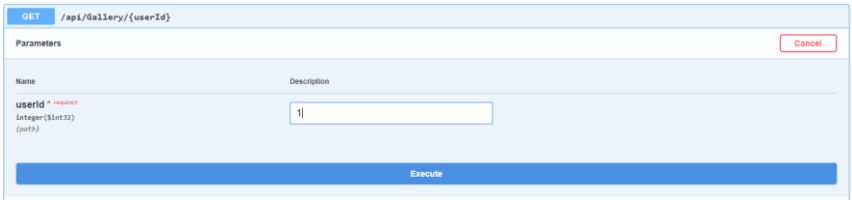
A response will be display inside 'Response body' section:



Requirement 2 [Retrieve gallery details for user] testing:

Steps:

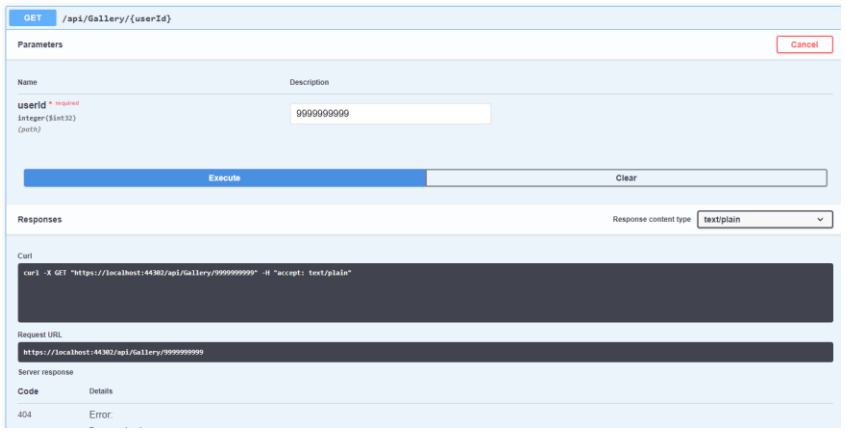
1. Expand /api/Gallery/{userId} tab item
2. Click right side button 'Try it out'
3. Fill field 'userId' with an integer value



4. Click 'Execute' button

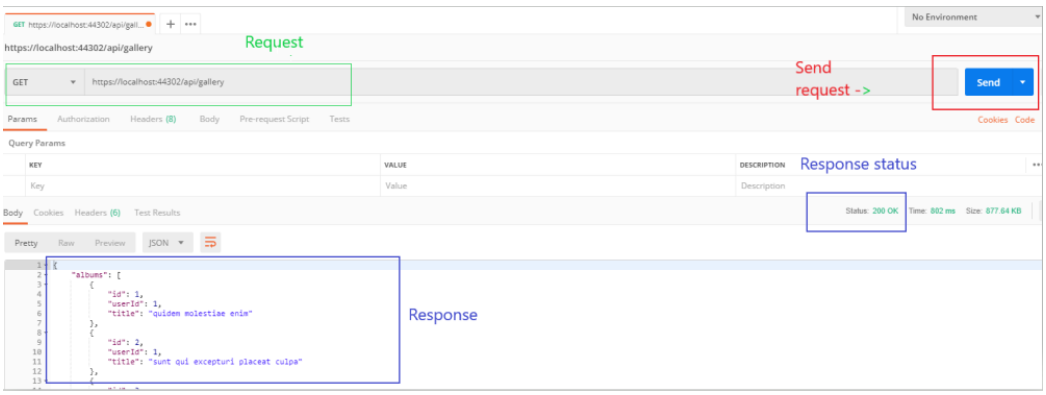
A response will be display inside 'Response body' section.

Ex: userId = 9999999999 does not exists. The API should return a 404 – Not Found message response.



Postman

The application can also be tested using Postman (or any other RestClient – based on personal preference)



Requirement 1 – request example:

Get https://localhost:44302/api/gallery

**Change port 44302 with your port from launchSettings.json file

Requirement 2 – request example:

Get https://localhost:44302/api/gallery /?userId=1

**Change port 44302 with your port from launchSettings.json file

Improvements

Current solution is a PoC, but it can be extended to a more complex application, that will offer a large list of functionalities (ex: adding/removing albums or photos, retrieving albums based on different criteria such as size or number of photos etc).

From a technical perspective:

- Authorization – currently there is a simulated token-based authorization middleware (not for development environment). It can be changed using a proper JWT Token Security implementation
- Logging – a logging mechanism is essential and desirable (ex: using Serilog libraries)
- Monitoring – performance also represents a key factor for an API which can be easily implemented using Azure Application Insights
- Code Coverage could be increase and also Azure Test Plans could be used for efficient performance tests
- CD/CI – use Docker for containerization