# LumiraDX Technical Test – High Level Test Plan

## Executive summary:

Our team has been tasked with building a simple back-end for a blog. This is built using RESTful API to implement CRUD operations.

## In scope:

The following are in scope for testing:

* Swagger API documentation.
* GET /blog/categories/ - Returns list of blog categories
* POST /blog/categories/ - Creates a new blog category
* DELETE /blog/categories/{id} - Deletes blog category
* GET /blog/categories/{id} - Returns a category with a list of posts
* PUT /blog/categories/{id} - Updates a blog category

## Defect definitions:

Defects found in the project will be raised in the issues part of GitHub.

Defects reports will have the following sections:

* **Title** – Descriptive title of bug found.
* **Description** – Summary of bug found.
* **Priority** – This will be rated: High, Medium, Low, and Cosmetic.
* **Steps to reproduce** – Steps to help reproduce the bug.
* **Screenshots** (If applicable)

## Risks and assumptions:

* Due to working commitments on other projects, the QAs time on this project is limited and testing may not be complete, however, at a minimum the endpoints detailed in the in scope sections will be tested.
* There is a risk due to the time limitations; the QA will not be able to complete the automated tests.
* This project assumes the additional endpoints listed in automated approach has been tested previously and should not require further verification.
* This project assumes that all the required data in the database has been pre-populated for testing.
* This project assumes that test results on MacOS will be identical to Windows and Linux systems.

## Manual tests approach:

QA will make use of Postman to test the end points manually. Postman will allow the QA to organise test scenarios and group the endpoints together in a more readable and understandable manner.

The checks during the manual testing will focus on:

* Verifying the correct status codes.
* Verifying the response matches ones found in the documentation.
* Verifying the response headers.

**GET endpoint:**

|  |  |
| --- | --- |
| Test ID | Test Scenario |
| 1 | GET: All categories |
| 2 | GET: Category ID with associated blog posts |
| 3 | GET: Category ID with no associated blog posts |
| 4 | GET: Category ID with non-existing ID |
| 5 | GET: Category ID with the maximum int value |
| 6 | GET: Category ID over the maximum int value |
| 6 | GET: Category ID with string value as ID |
| 7 | GET: Category ID with an ID value of 0 |
| 8 | GET: Category ID with an ID value of -1 |
| 9 | GET: Category ID |

**POST endpoint:**

|  |  |
| --- | --- |
| TEST ID | Test Scenario |
| 10 | POST: With no values |
| 11 | POST: With no ID and with a category name in body |
| 12 | POST: With new ID and with category name in body |
| 13 | POST: With existing ID and new category name in body |
| 14 | POST: With new ID and with an existing category name in body |
| 15 | POST: With string value in ID field and int value in category name |
| 16 | POST: With ID field populated only |
| 17 | POST: With existing category name only |
| 18 | POST: With blank value in the name |

**PUT endpoint:**

|  |  |
| --- | --- |
| TEST ID | Test Scenario |
| 19 | PUT: Update existing category name |
| 20 | PUT: Update non existing category |
| 21 | PUT: Update with blank string on name field |
| 22 | PUT: Update category where blog posts exists |

**DELETE endpoint:**

|  |  |
| --- | --- |
| TEST ID | Test Scenario |
| 23 | DELETE: Existing category |
| 24 | DELETE: Attempt to delete category that doesn’t exist |
| 25 | DELETE: string |
| 26 | DELETE: use wildcard to attempt to delete all |
| 27 | DELETE: Blog post ID |

## Automated test approach:

As stated in the risk and assumption section, there is assumption that the below end points listed in this section has been through a previous test cycle, so, no dedicated test scripts will be written for them. However, in the event that defects were uncovered whilst using these end points, defects will be raied.

QA will make use of the Pytest framework using Pycharm to create some automated integration test scenarios.

By taking into account the below end points, we can prove the integration of the categories endpoints.

* GET /blog/posts/ - Returns list of blog posts
* POST /blog/posts/ - Creates a new blog post
* GET /blog/posts/archive/{year}/ - Returns list of blog posts from a specified year
* GET /blog/posts/archive/{year}/{month}/ - Returns list of blog posts from a specified month
* GET /blog/posts/archive/{year}/{month}/{day}/ - Returns list of blog posts from a specified day
* DELETE /blog/posts/{id} - Deletes blog post
* GET /blog/posts/{id} - Returns a blog post
* PUT /blog/posts/{id} - Updates a blog post

|  |  |
| --- | --- |
| TEST ID | Test Scenario |
| 28 | * Create new blog post and link with existing category. * GET category can retrieve newly created blog post. |
| 29 | * DELETE newly created blog post. * GET category and check that newly created blog post was deleted. |
| 30 | * Create new blog post and link with existing category * GET category to retrieve blog post * PUT to update the blog post category * GET category to check that blog post no longer exist in original category and now moved to new category |