**Test cases proposed for automation:**

All the below end points are tested for each of their operations:

* Products:

Test cases:

* + Verify add products
  + Verify edit products
  + Verify get all products
  + Verify getting a specific product
  + Verify limiting the search product results
    - Above 25 limit
    - Below 25 limit
  + Verify querying the products endpoint for its type
  + Verify delete product
* Stores:

Test cases:

* + Verify add stores
  + Verify edit stores
  + Verify get all stores
  + Verify getting a specific store
  + Verify limiting the search stores
  + Verify delete store
* Services:

Test Cases:

* + Verify add services
  + Verify edit services
  + Verify get all services
  + Verify getting a specific service
  + Verify limiting the search services
  + Verify delete services
* Categories:

Test Cases:

* + Verify add categories
  + Verify edit categories
  + Verify get all categories
  + Verify getting a specific category
  + Verify limiting the search categories
  + Verify delete categories
* Utilities (Version and Healthcheck):

Test Cases:

* + Verify the version of API
  + Verify the health of API

**Details of the solution:**

* **IDE:** Pycharm
* **Programming Language:** Python (Version 3.9.5)
* **Framework:** Pytest
* **Reporting:** Pytest HTML

**Summary:**

**Below is the high level architecture:**

* API\_Framework: This directory contains the framework level methods which can be used with any API testing and they are totally generic. This directory contains methods like call\_request(…), read\_config(…) and etc.
* Business\_logic: This directory contains the business logic implementation of the API under test like call\_api\_playground\_endpoints(…), add\_store(…), edit\_store(…) etc
* PayLoad: This directory contains the payloads for API service endpoints.
* Test\_Cases: This directory contains the test cases which user wants to execute.
* Test\_Reports: This directory contains the report of test cases in HTML format.

The reason for above described architecture is to de-couple different layers and make them as much reusable as possible e.g. user can automate any amount of test cases by using this architecture by simply adding test cases in API\_Playground\_test\_cases.py. If user wants to add new endpoints they can achieve that by minimal coding in business logic directory.

**Setup Instructions:**

1. First, user will need to Install Pytest
   1. Go to File > Settings > Project Interpreter > + > write pytest > install packages
2. Install Requests
   1. Go to File > Settings > Project Interpreter > + > write Requests > install packages
3. For reporting install pytest-html
   1. Go to File > Settings > Project Interpreter > + > write pytest-html > install packages
4. Install PyYAML
   1. Go to File > Settings > Project Interpreter > + > write PyYAML > install packages
5. Set Default test runner: pytest from File > Settings > Python Integrated Tools
6. Edit Configurations and set the Additional Arguments in Python Tests as
   1. --html=Test\_Reports/APIPlayGround\_test\_results.html for reporting.
7. To view the execution results: Test\_Cases/Test\_Reports/APIPlayGround\_test\_results.html