RELAYR Test Document

Task 1

The scenario outlined is something which I faced when I joined my current assignment. The approach will be explained in detail in the below points, which correspond to the responses in the assignment questions as well.

Where would you start? What would be your first steps?

My first steps would be to start off with a test plan and test document based on the SRS document, if available, or in consultation with the product team and by exploring the product in detail. Further I would also find a test/defect management tool which can be used for the establishing the test process.

Also I will try to narrow down on the Automation Framework to be used so that the scenarios can be built based on that in mind. I will also use some systematic techniques like State Transition for test case preparation, if applicable. Automation Framework would be selected keeping in mind the CI/CD requirements in the future. Then I would go around with the test preparation as outlined in the below steps.

1. Test plan/document preparation
2. Test Case Preparation
3. Test Case Execution
4. Regression Test Case Execution
5. Post-Production Testing
6. Regression Test Case update process

Further I would try to establish an efficient regression suite, development of which would be prioritized in the below order.

1. Sanity Test Suite
2. Smoke Test Suite
3. Regression Test Suite

Which process would you establish around testing new functionality? How would you want the features to be tested?

Whenever a new feature is being taken up for testing, the test planning and test case development should ideally be started whenever software development is starting for the new feature, so that the test cases will be ready when the feature is ready for testing. Also this will provide ample time for updating the Automation Regression Suite as per the latest changes.

An Entry and Exit criterion should be set before we start testing any new feature and should be followed strictly. One of the Entry criterion that should included is that the build taken up

for testing should pass the Automation Sanity/Smoke Scenarios. One of the exit criterion should be that the high priority regression test cases should be passed before the changes are promoted to production.

A reliable set of post-production test cases, preferably automation, should be run after deployment to probe for any issues that may creep up during deployment.

If you would do test automation which techniques or best practices would you use?

As discussed in the previous section, Automation Framework would be selected keeping in mind the CI/CD requirements in the future. I would prefer to implement a BDD (Behavior-driven development) Framework for Test Automation as this would help in test case understandability across the team, including the product and marketing team, if required.

The Test Automation Suite will be having below folders.

- Features: Feature files will be created based on each feature. The different scenarios in the particular feature will be listed in the same feature file.
- Source: Source folder will be used to list the files used to implement the source files for corresponding feature files. Emphasize will be kept to resue the codes wherever possible.
- Pages: This folder will be used to save the locators for each pages in case of Front End Automation and api's in case of Backend Automation. Each file will be used to save the locators belonging to each page in the UI.
- Environment : This will be the folder used to save the URLs, and other environment related information.
- Credentials : Will be used to save credentials, token etc.
- Synchronization : Folder used to create and save custom wait or any wait libraries.
- Data: May be used to save data coming from external sources such as .xlsx file.
- Reports : Can be used to store the reports.
- Screenshots : May be created and used to save screenshots of failed test cases in case of Front End Automation.

Framework should be executable using command line and should support development of a dockerized version.

Task 2 - Alternative

Alternative task was selected as I am more proficient in Front End Automation. However I am indeed having some Api Automation experience using pytest along with request, base64 and json libraries.

The website selected for Testing is github gists website with gist search functionality (*"https://gist.github.com/"*). The Framework was developed using Selenium webdriver and pytest_bdd.

Pytest_bdd - Test Framework

The framework is uploaded in the private git repository mentioned below.

https://github.com/getarun4t/git_search.git

Selection Rationale

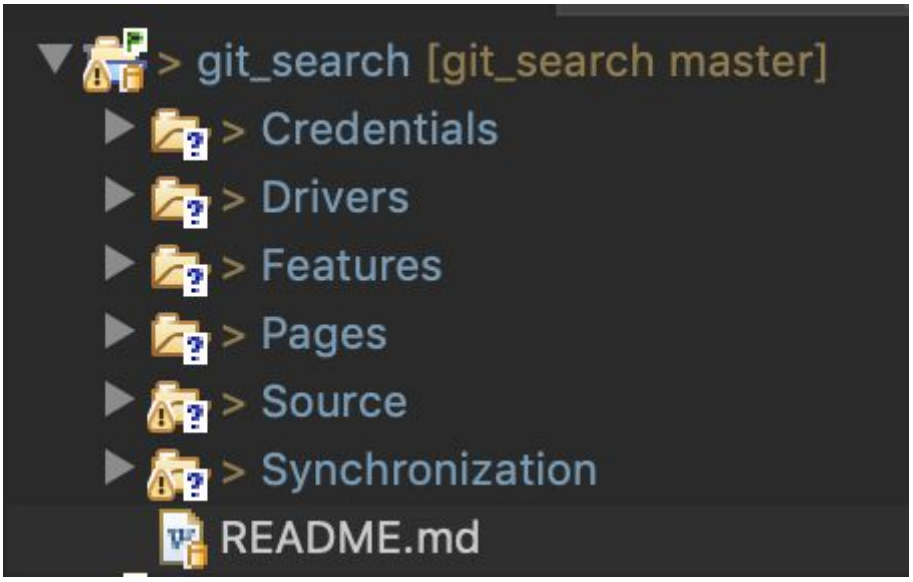
The rationale behind the selection of the framework was due to the below inherent advantages in pytest and Python.

1. Can be extended to be used for both api and functional automation.
2. Parallel execution capability
3. Detects test files automatically
4. Uses BDD which makes it easier to read and understand the test scripts

Framework Details

The Pytest_bdd is an efficient BDD framework which uses Gherkin feature files for writing the tests in plain english language which uses all the capabilities of the pytest framework.

The image below depicts the folder structure used in the framework. The use of each folder in the framework is discussed in details in the below section.



1. Credentials: Is a single location for storing all the credentials and links used in testing.
2. Drivers: Is used for storing all the drivers which is useful for cross-browser testing.

3. Features: Is used for listing the different features which are being tested. Each feature under test will have different scenarios and will be listed below the feature in the same file.
4. Pages: will be used for storing all the locators in UI Functional testing and api's in case of api testing.
5. Source: will be having the scenario and step implementation of different test cases that are framed in the features files folder.
6. Synchronization: Used to develop and store custom wait methods.

Additional folders can be added for storing screenshots, reports etc. whenever required.

Installation Guide

Listed below will be the step by step guide to install and use the framework to test in the local.

1. Download and Install Python3 (if not done already). Python2 is pre-installed in Mac Devices.
2. Install pip/pip3 for Python.
3. Download and install selenium webdriver and chromedriver. Update chromedriver location, if required.
4. Install pytest and pytest-bdd using the below command line requests.

```
pip install pytest
pip install pytest_bdd
```

5. Download the project from the repository.
6. Run the below command line request to run the project

```
python3 -m pytest
```

7. To create a report of the execution, please use the below command line request

```
python3 -m pytest --html=./report.html
```

8. To run test cases belonging to a particular file, use the below command line argument

python3 -m pytest "filename.py"

Enhancements

The enhancements to the project include adding screenshot capture capability whenever applicable, for UI Automation scenarios. Further the entire suite can be implemented inside a pipeline to ensure that the test cases are automatically triggered as per needs.

Dockerized version of the test framework developed can be implemented using *pytest docker compose* module and configuring the *.yml file.

Test Cases

Few of the test cases which can be used to test the search functionality is listed in the attached excel file. Two test cases which are implemented in Automation are highlighted in yellow colour in the attachment.
Precondition : Data Creation (included in the Automation Framework): First test case is used to create a public and private gist which is used in the subsequent test case.
Relayr Assessment HLTC