## **Introduction** An API test plan is a document that outlines the testing strategy, objectives, and procedures for testing an API. It is useful because it provides a structured approach to testing and ensures that all aspects of the API are tested thoroughly. By creating a test plan, you can reduce the amount of manual work required for testing and ensure that testing is consistent across different environments.

## Including API testing in your CI/CD pipeline can help you catch issues early in the development process and ensure that your API is working as expected.

## **Test Objectives**

* Ensure that the REST API is functioning correctly.
* Verify that the API can create, read, update, and delete records as expected.
* Confirm that the API is secure and can handle errors gracefully.
* Out-Of-Scope: Data Model and nullable fields are out of scope, so that POST scenario should verify only critical path (without attempts to verify each possible combination of input data)

## **Test Plan: Environment**

* The API will be tested in a Test environment *https://reqres.in/api-docs/*
* The test environment will be set up with test data that is representative of the production data.
* Automated test should be compliant to test Swagger

## **Test Plan: Resources**

* Evgenii Subbotin should implement 100% of work
* Work should be done by 01/01/2023

**Test Data**

* The test data will be pre-generated in the Test Environment
* Variables for tests are hardcoded to include both valid and invalid input data.
* Out-Of-Scope: ID for data depended test may be obtained via Groovy scripts

**Test Approach**

* All possible scenarios should be automated with SoapUI
* Test Run Report should be obtained from SoapUI Test Run Data
* All test will be done with “Whitebox”/”Greybox” approach

## **Test Scenarios by types**

Following list of scenarios define all tests that will be used for the plan. Each entry point should be covered with at least one positive and one negative scenario. Create a new record using valid input data.

1. POST:Create a new record using valid input data. (negative scenario)
   1. Verify HTTP code 200
   2. Verify content
2. POST:Create a new record using invalid input data. (negative scenario)
   1. Verify HTTP code 400
   2. Verify error message
3. GET:Read an existing record.
   1. Verify HTTP code 200
   2. Verify content
4. GET:Attempt to read a deleted record. (negative scenario)
   1. Verify HTTP code 400
   2. Verify error message
5. PATCH/PUT:Update an existing record.
   1. Verify HTTP code 200
   2. Verify content
6. PATCH/PUT:Attempt to update a deleted record. (negative scenario)
   1. Verify HTTP code 400
   2. Verify error message
7. DELETE:Delete an existing record.
   1. Verify HTTP code 200
   2. Verify deletion message
8. DELETE:Attempt to delete a deleted record. (negative scenario)
   1. Verify HTTP code 204/404
   2. Verify error message

## Test Run Report

Legend:  
Columns with + used for positive tests, Columns with - for negative tests

|  | POST+ | POST- | GET+ | GET- | PUT+ | PUT- | DEL+ | DEL- |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Users | N/A | N/A | Passed | N/A | N/A | N/A | N/A | N/A |
| Users By ID | N/A | N/A | Passed | To Fix | Passed | To Fix | To Fix | Passed |
| Resource | N/A | N/A | Passed | N/A | N/A | N/A | N/A | N/A |
| Resource By ID | N/A | N/A | Passed | To Fix | Passed | To Fix | To Fix | Passed |
| Login | To Fix | Passed | N/A | N/A | N/A | N/A | N/A | N/A |
| Register | To Fix | Passed | N/A | N/A | N/A | N/A | N/A | N/A |
| Logout | Passed | N/A | N/A | N/A | N/A | N/A | N/A | N/A |

## Conclusion

Login and register functionality doesn’t work in the proper way.

Change(CUD) functionality doesn’t work in the proper way.  
Without the possibility to change content this API environment can be counted as read-only API and it works well only in read only mode.