Github connect API’s Project

Table of Contents:

1. PURPOSE
2. REQUIREMENT ANALYSIS PHASE
3. PLANNING PHASE
4. SOFTWARES
5. DESIGN AND CODING PHASE
6. TESTING PHASE
7. IMPLEMENTATION AND INTEGRATION
8. DELIVERY
9. ASSUMPTIONS
10. Purpose  
    This project is to design an automated Framework which will connect to the end point source API bundles in the GitHub Gateway Repository. So from the Java program needs to connect to the endpoint GitHub API’s repository without using Rest Assured in Java. The client GitHub API Bundles supports (querying based on a series of keywords and qualifiers) , Ordering and Sorting. The program interface has to test the different scenarios of test conditions based on the API Bundles output.
    1. Phases of Project
       1. Requirement Analysis Phase
       2. Planning Phase
       3. Design and Coding Phase
       4. Testing Phase
       5. Implementation and Integration
       6. Delivery

A close up of a logo

Description automatically generated

1. Requirement Analysis Phase

Having formulated test conditions before provided an extra edge for planning the next scenarios of the project. Like The GitHub documentation link is provided in the below through which the requirements are researched and analysed thoroughly along with checking the API’s output.

<https://developer.github.com/v3/search/#search-repositories>

Below are the possible API’s bundle searches provided in GitHub.

Search

Search repositories

Search commits

Search code

Search issues and pull requests

Search users

Search topics

Search labels

Text match metadata

Through which test conditions were formulated and planned to how, what and what not needs to be done.

1. Planning Phase

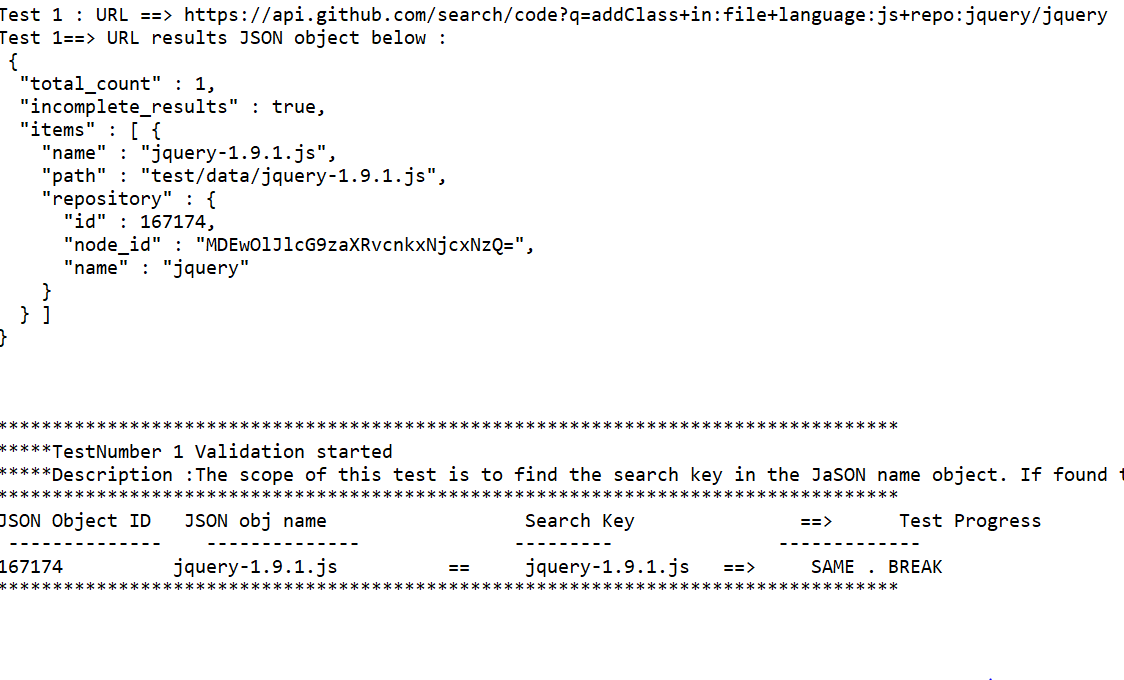
Having formulated test conditions before provided an extra edge for planning the next scenarios of the project. Like

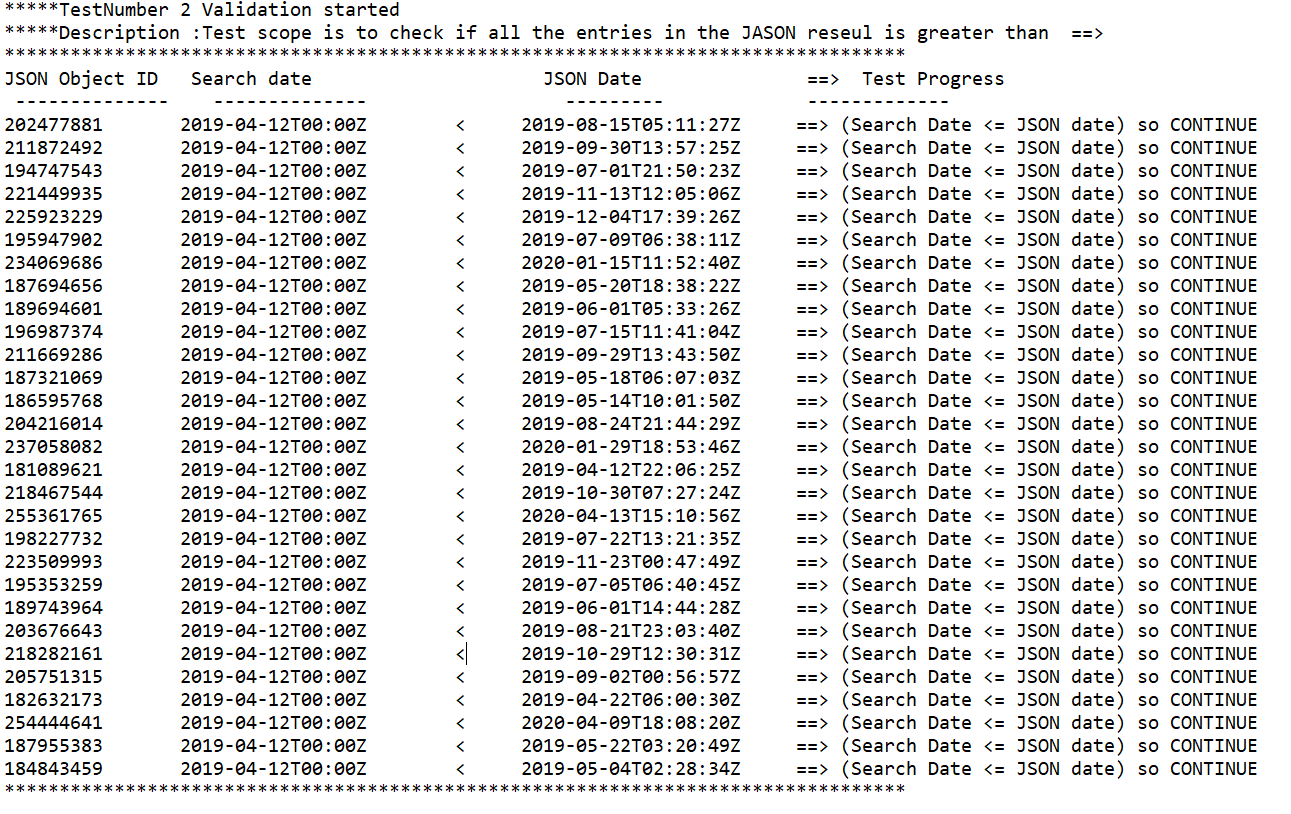
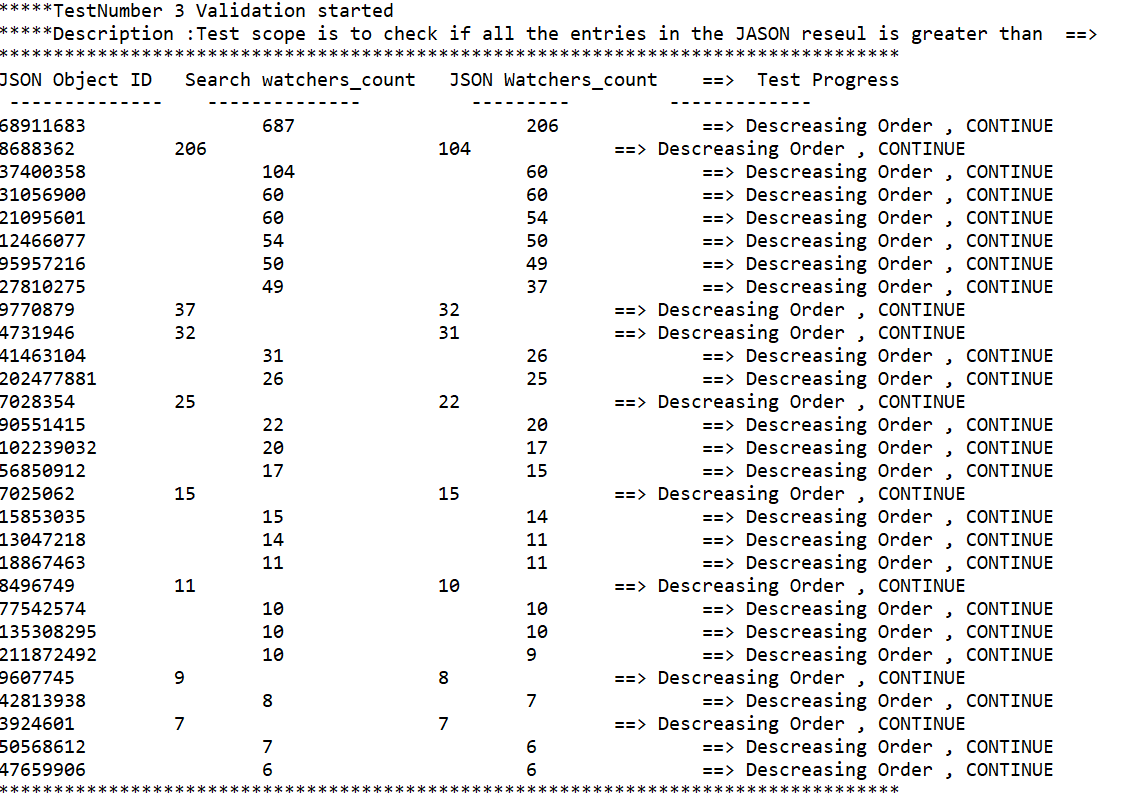
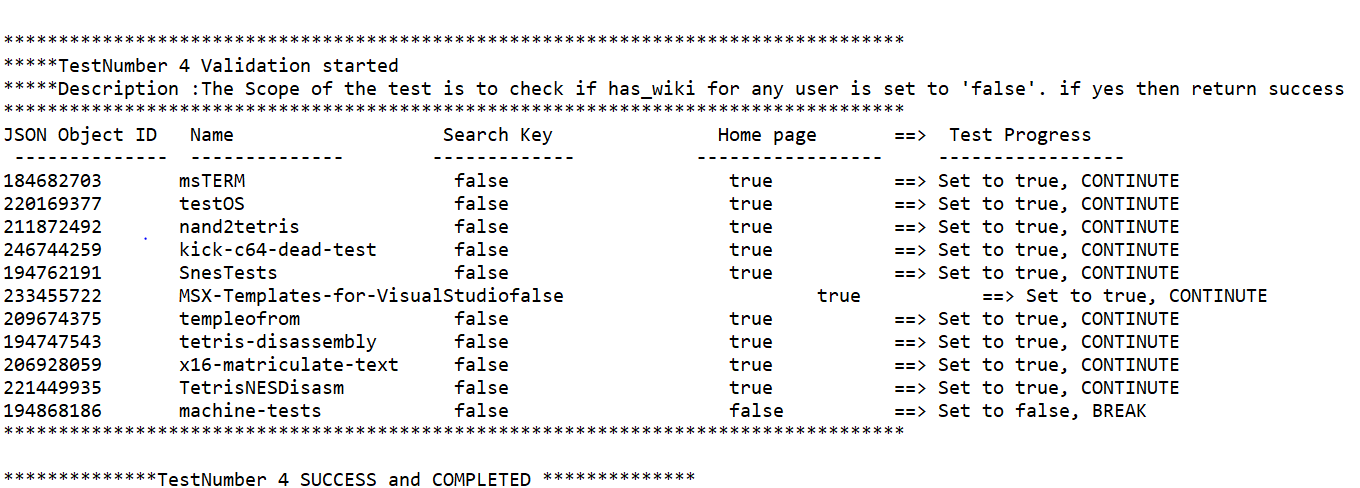
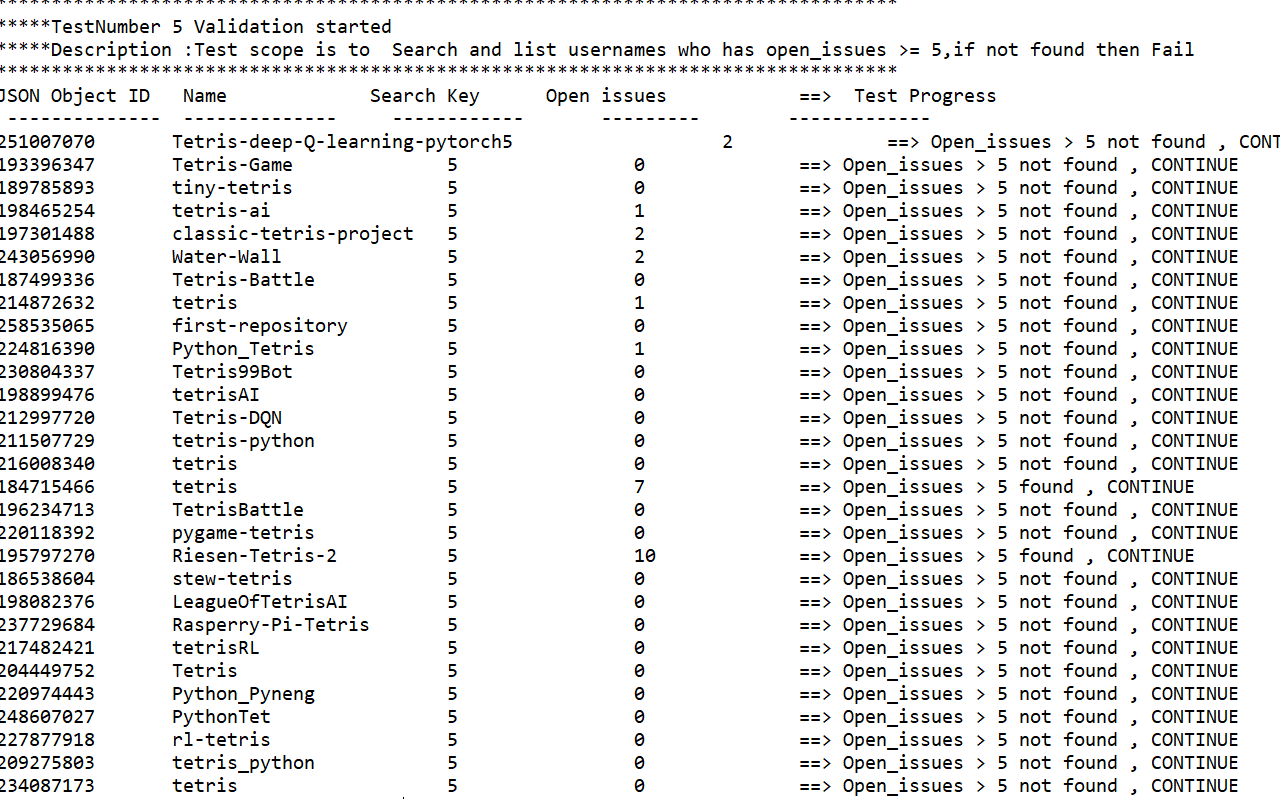
* 1. Which Framework to select to automate the model.
  2. How to design the model, test and implement.
  3. Drive the project with TDD approach.

1. Software
   1. Java – JDK v14
   2. Eclipse IDE
   3. Junit5
   4. Maven
   5. Sure-fire plugins for Reporting
   6. Windows 10 Home Edition
2. Design and Code Phase
   1. Framework Architecture  
      The below Framework Architecture picture demonstrates as -  
      Diagram:  
        
      A screenshot of a cell phone

      Description automatically generated  
        
      Note: Jenkins can be used to integrate these Maven project into it for daily regression testing.  
      * User will run the Test cases in Junit5 framework/ Jenkins.
      * Within Eclipse, internally Junit tests are connected to Maven Project through which Java programs are fired. The Java program connects to the GitHub API via HTTP.
      * Test results are captured via Surefire report plugin and are stored in test run repository.
      * The API response is captured and test cases are executed to validate the test scenario.
   2. Framework Test Design Flow  
      The below Framework Architecture picture demonstrates as -  
      Diagram  
        
      A screenshot of a cell phone

      Description automatically generated  
        
        
      * Junit5 - Through this framework Test1.java program all test cases are executed, which is a plugin of JDK 14 integrated in Maven project in Eclipse IDE.
      * Eclipse
        1. Main Class – JSONGenericTestSetup.java: All test data needed for the Test cases are built and individual Test cases are called.(handy enough for a tester to play with test data)
        2. Main Methods class – JSONGenericTestSetupMethods.java: This class contains logic to call GitHub API’s and validate the test scenarios.
        3. Structure or POZO classes – JSONObjectRootClass.java consists of 2 sublevel objects JSONObjectRootLelve1Class and JSONObjectRootLelve2Class.
        4. These classes are used to represent the structure of the GitHub API result.

1. Testing Phase
   1. Test Case 1  
      TC–1:  
      Test scope is to validate the search for a specific text in the JSON object tag name. If found then success and display Tag object id and name, else false.   
        
      API url:  
      <https://api.github.com/search/code?q=addClass+in:file+language:js+repo:jquery/jquery>  
        
      Output:  
      Insert picture here  
        
      

* 1. Test Case 2  
     TC–2:   
     Test scope is to check if all the dates in the JSON result set is greater than the date specified in the test parameter. If found then 1(TRUE) else 0(FALSE). Filter by a given creation date - if the creation date is >= 2019/05/01, list the users count or users  
        
     API url:  
     <https://api.github.com/search/repositories?q=tetris+created:%3E=2019-04-01+language:assembly&sort=stars&order=desc>  
       
     Output:  
     Insert picture here  
       
     
  2. Test Case 3  
     TC–3:   
     Test scope is to check if all the entries in the JSON results are in descending order or not. If descending then 1(TRUE) else 0(FALSE)  
        
     API url:  
     <https://api.github.com/search/repositories?q=tetris+language:assembly&sort=stars&order=desc>  
       
     Output:  
     Insert picture here  
       
     
  3. Test Case 4  
     TC–4:   
     Test scope is to check if for any user has the wikipages in his repository false 'has\_wiki' then list them with their Tag id and display. 'has\_wiki' set to false then 1(TRUE) else 0(FALSE) test scope is to Search and list users who has open\_issues >= SearchValue ,if found then return 1(TRUE) else 0(FALSE)  
       
     API url:  
     [<https://api.github.com/search/repositories?q=tetris+language:assembly&sort=stars&order=desc>](https://api.github.com/search/repositories?q=tetris+created:%3E=2019-05-01+language:Python&sort=open_issues)  
       
     Output:  
     Insert picture here  
     
  4. Test Case 5  
     TC–5:  
     Test scope is to Search and list users who has open\_issues >= SearchValue ,if found then return 1(TRUE) else 0(FALSE)  
       
     API url:  
     <https://api.github.com/search/repositories?q=tetris+created:%3E=2019-05-01+language:Python&sort=open_issues>  
       
     Output:  
     Insert picture here  
       
     
  5. Test Case 6  
     <<For future test cases>>

1. Implementation and Integration Phase
   1. Plugins  
      The plugins like Junit5 and Surefire needed for Framework are integrated into Maven project of Eclipse IDE through POM xml file.
2. Delivery Phase
   1. Jar files  
      The jar files are placed in the path🡪 Delivery\Jar\TestProjectV1\0.0.1-SNAPSHOT\
   2. Source Code  
      The source files are placed in path 🡪 Delivery\Java Programs\src\main\java\demoFnl  
       and   
      JUnit test file is placed in path 🡪 Delivery\Java Programs\src\test\java
   3. Reports  
      The surefire report is placed in path 🡪 Delivery\Surefire Report
   4. Document  
      The project document is stored in path 🡪 Delivery\ GitHub Connect API Project.doc
3. Assumptions
   1. Third Party  
      The GitHub API gateway is up and running when the test is running
   2. Execution  
      The executables should be loaded and run in the same test setup as described above.