Day1_Tasks

Task 1: Eclipse IDE Basic Operations

Topic: Navigating the Eclipse Interface

Description: Demonstrate the basic operations in Eclipse by running a pre-written Java program, observing the console output, and then stopping the execution.

Steps:

Open Eclipse IDE and select the workspace that contains the pre-written Java program.

Locate the Java program in the Package Explorer and open it in the editor. Click on the 'Run' button to start the program and observe the output in the Console window.

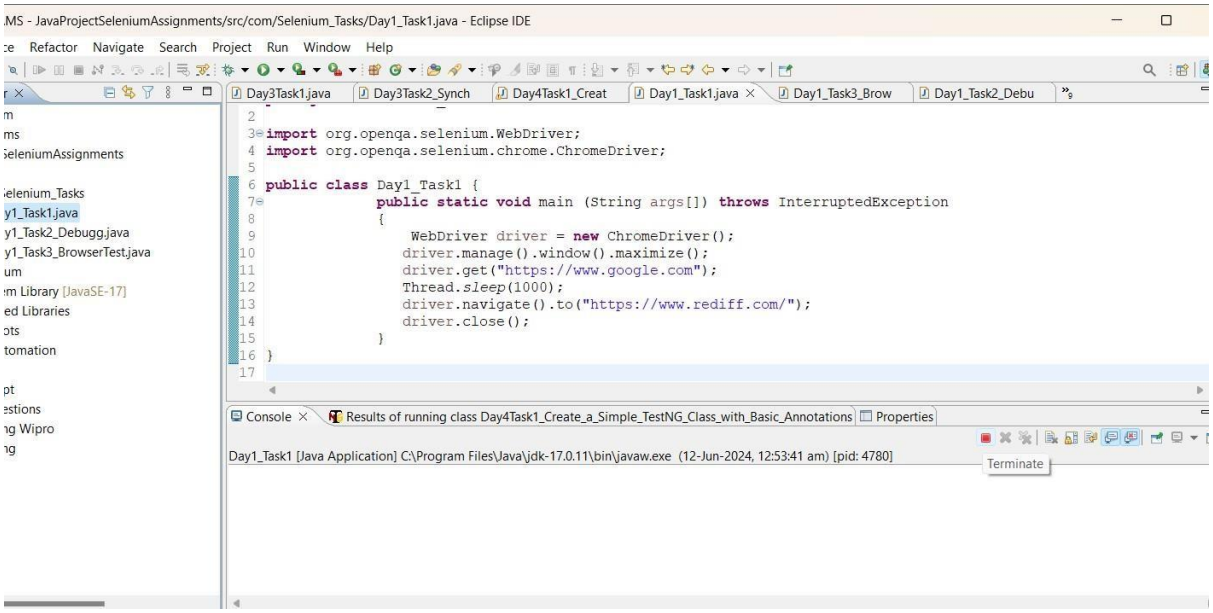
Use the 'Stop' button to terminate the program execution.

Describe what the 'Pause' button does in the context of running programs.

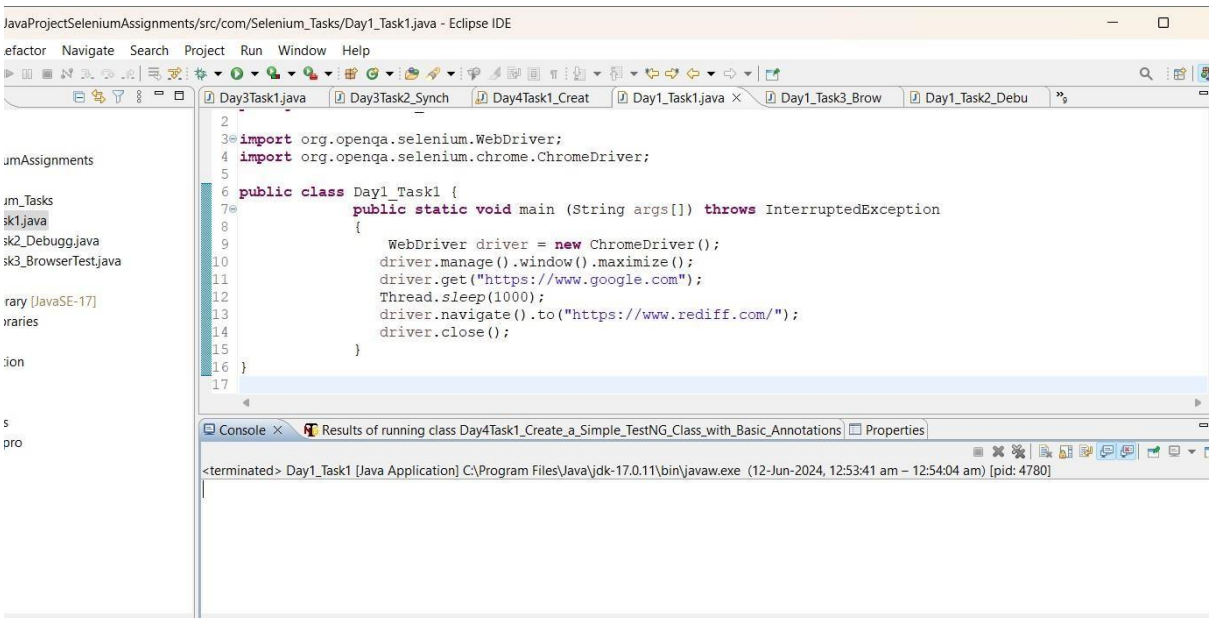
Sol:

```
package com.Selenium_Tasks;

import org.openqa.selenium.WebDriver;
import
org.openqa.selenium.chrome.ChromeDriver
;

public class Day1_Task1 {
              public static void main (String args[]) throws
InterruptedException
              {
               WebDriver driver = new ChromeDriver();
                   driver.manage().window().maximize();
                      driver.get("https://www.google.com");
                      Thread.sleep(1000);

     driver.navigate().to("https://www.rediff.com/");
                driver.close();
              }
}
```

Using Stop:



Terminated Program:



Task 2: Debugging with Eclipse

Topic: Error Handling and Debugging

Description: Utilize Eclipse's debugging features to identify and fix a simple logical error in a Java program.

Steps:

Open the provided Java program with a known logical error in Eclipse.

Start the debugger by clicking on the 'Debug' button.

Use breakpoints to pause the execution at critical points in the program.

Step through the code using 'Step into' and 'Step over' to observe variable values.

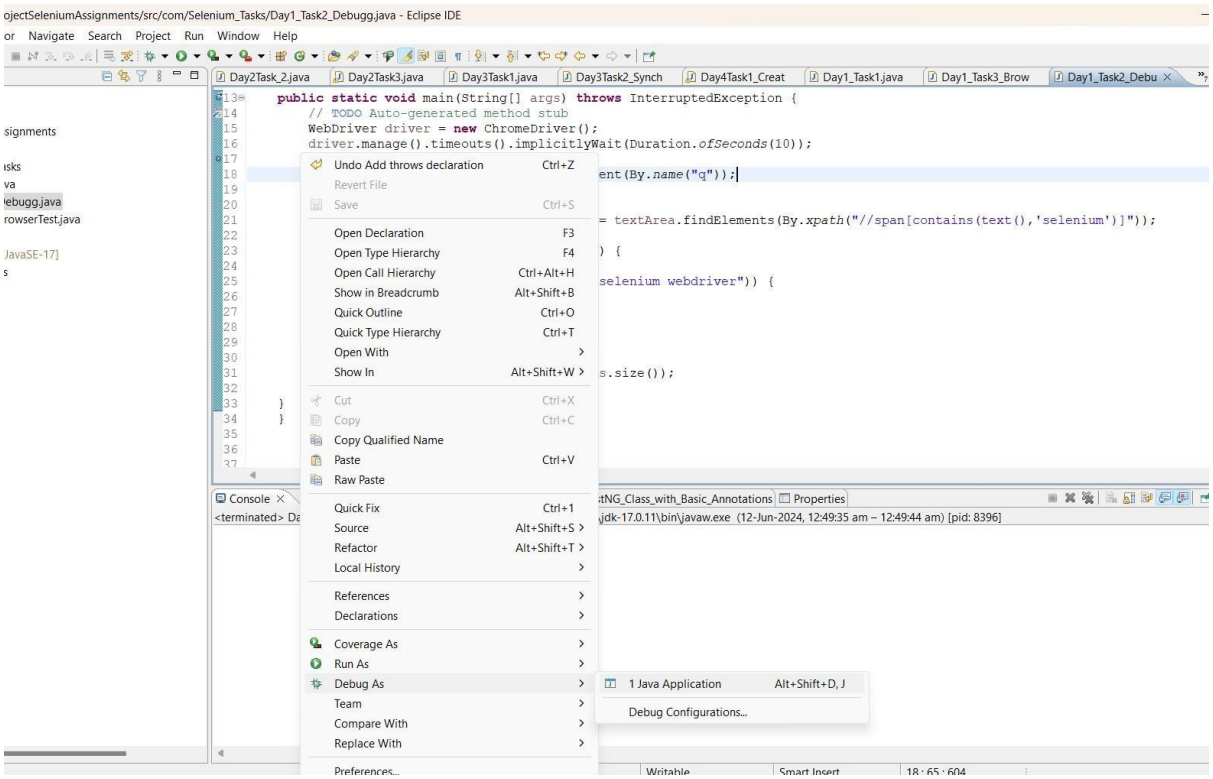
Identify the logical error based on variable values and code flow.

Correct the error and rerun the program to ensure it's fixed.

Sol:

Steps to Debug:

After writing the Java program we can debug by right-clicking on the Java class file from Package Explorer. Select Debug As → Java Application



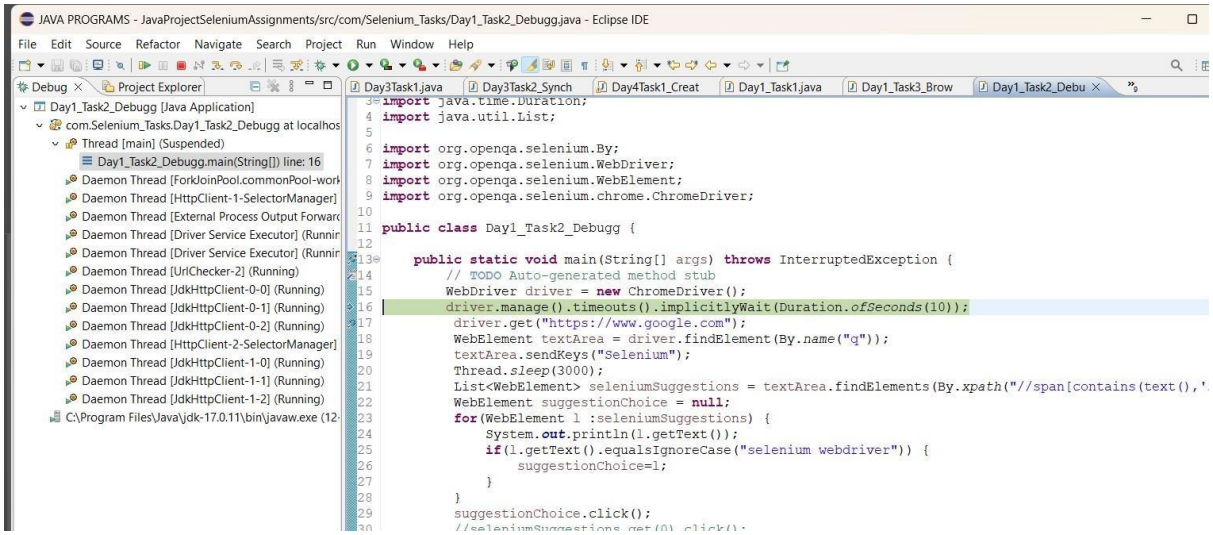
Breakup Points:

We can use breakup Points in the Application these Breakup Points are like a signal to the Debugger it will temporarily stop the execution at this point of code.

Step Over Debugging is a debugger function that glosses over a line of code, allowing it to execute without stepping into any underlying functions, thereby treating the entire function as a single unit.

Step Into Debugging' dives into a function, enabling you to dissect it line by line. Essentially, it leads you into nested functions to examine their operation up close.

We can see step by step how our code is executing and how the values are changing according to the code. By doing this we can easily find any mistake in our Code.




```
 package
com.Selenium_Tasks;
 import
java.time.Duration;
import
java.util.List;
 import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver
;

public class Day1_Task2_Debugg {

                  public static void main(String[] args) throws
InterruptedException {
        // TODO Auto-generated method stub
       WebDriver driver = new ChromeDriver();


     driver.manage().timeouts().implicitlyWait(Duration.ofSeco
nds(10));              driver.get("https://www.google.com");
       WebElement textArea = driver.findElement(By.name("q"));
         textArea.sendKeys("Selenium");
          Thread.sleep(3000);
        List<WebElement> seleniumSuggestions =
textArea.findElements(By.xpath("//span[contains(text(),'seleni
um')]"));              WebElement suggestionChoice = null;
      for(WebElement l :seleniumSuggestions) {
System.out.println(l.getText());
if(l.getText().equalsIgnoreCase("selenium webdriver")) {
                suggestionChoice=l;
                }
```

```
            }
            suggestionChoice.click();
           //seleniumSuggestions.get(0).click();
          System.out.println(seleniumSuggestions.size());

      }
 }
```

Task 3: Writing and Running a Simple Selenium Test

Topic: First Test Case and WebDriver Basics

Description: Write a simple Selenium WebDriver test in Eclipse to open a web browser and navigate to a specified URL.

Steps:

Open Eclipse and create a new Java Project dedicated to Selenium tests.

Within the project, set up the Selenium WebDriver by adding the WebDriver JAR files to the build path.

Create a new Java class file for the test case.

Write a Java method using WebDriver to initiate a Firefox, Chrome, or Safari browser session.

Use WebDriver to navigate to 'http://example.com' or a similar simple web page. Run the test to ensure the browser opens and navigates to the URL successfully.

Sol:

```
package com.Selenium_Tasks;

import org.openqa.selenium.WebDriver;
import
org.openqa.selenium.chrome.ChromeDriver;
import
org.openqa.selenium.edge.EdgeDriver;
import
org.openqa.selenium.firefox.FirefoxDrive
r;
 public class
Day1_Task3_BrowserTest {
     static WebDriver
driver=null;

    //open browser
      public static void invokeBrowser(String browser)
          {
```

```
            if(browser.equalsIgnoreCase("chrome"))
                {
                   driver = new ChromeDriver();
                  driver.manage().window().maximize();
                }
               else if(browser.equalsIgnoreCase("firefox"))
                {
                   driver = new FirefoxDriver();
                  driver.manage().window().maximize();
                }
               else if(browser.equalsIgnoreCase("Edge"))
                {
                   driver = new EdgeDriver();
                  driver.manage().window().maximize();
                }
           }
   public static void main (String args[]) throws
InterruptedException {       invokeBrowser("chrome");
driver.get("https://www.google.com");
driver.navigate().to("https://www.flipkart.com/");
        driver.navigate().back();
driver.navigate().forward();
driver.navigate().refresh();
        driver.close();
Thread.sleep(1000);
     System.out.println(driver.getTitle());
       driver.close();
    System.out.println("Executed successfully");
   }


}
```

Day2_Tasks

Task 1: Page Title Verification

Topic: Page Interaction Commands

Description: Write a simple Selenium script to open a specified URL and verify the title of the page.

Steps:

Launch Eclipse IDE and create a new Java class in the Selenium project.

In the main method, instantiate the WebDriver and navigate to 'http://example.com'.

Use driver.getTitle() to retrieve the page title and store it in a variable.

Assert that the page title matches the expected title using a simple if statement that prints out the verification result.

Close the browser using driver.quit().

Sol:

```
package selenium_Tasks_Assignments;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Day2Task1 {

	public static void main(String[] args) throws
InterruptedException  {

		WebDriver driver = new ChromeDriver();

		try {
			driver.get("https://www.google.com/");
			Thread.sleep(3000);
			String pageTitle = driver.getTitle();
			System.out.println(pageTitle);

			if (pageTitle.equals("Google")) {
			System.out.println("Title verification passed!");
		  } else {
			System.out.println("Title verification failed!
Expected: '" + "Google" + "', but got: '" + pageTitle + "'");
```

```
            }
        }finally {
                driver.quit();
        }

    }

}
```

Task 2: Element Interaction and State Verification

Topic: WebElements Commands and Identification

Description: Write a Selenium script to interact with an input field on a form, enter text, and verify the text was entered correctly.

Steps:

Open the same Java class used in Task 1.

Navigate to a web page with an input field (e.g., a search box).

Use driver.findElement() to locate the input field by its name or ID.

Enter text using the sendKeys() method and clear it with clear().

Retrieve the input field value and verify it matches the text entered.

Verify the input field is displayed and enabled by using isDisplayed() and isEnabled().

Sol:

```
package selenium_Tasks_Assignments;

import java.time.Duration;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
public class Day2_Task2 {
public static void main(String[] args) throws InterruptedException {
WebDriver driver = new ChromeDriver();
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
driver.navigate().to("https://www.google.com/");
WebElement searchArea = driver.findElement(By.name("q"));
searchArea.sendKeys("Selenium", Keys.ENTER);
driver.navigate().to("https://login.salesforce.com/?locale=in");
WebElement userName = driver.findElement(By.name("username"));
userName.sendKeys("vsuneel244@gmail.com");
driver.findElement(By.name("pw")).sendKeys("vsuneel@123");
```

```
WebElement logIn = driver.findElement(By.xpath("//input[@id =
'Login']"));
logIn.click();
WebElement errorMsg = driver.findElement(By.id("error"));
String error = errorMsg.getText(); System.out.println(error);
if(error.contains("If you still can't log in, contact your
Salesforce administrator"
))
{
System.out.println("Excepted messege is Equal to Actual Messege");
}
else
{
System.out.println("Excepted messege is not Equal to Actual
Messege");
}
}
}
```

Task 3: Locator Strategy Implementation

Topic: Locator Techniques and Effective Element Selection

Description: Craft CSS Selector and XPath locators for an element on a web page and demonstrate the selection in a Selenium script.

Steps:

Use the Eclipse IDE to open the existing Selenium project.

Open a web page that has a uniquely identifiable element (like a button with an ID).

Write two separate locators in the Selenium script: one using CSS Selector and one using XPath.

Use driver.findElement() to find the element using both locators separately, printing out a confirmation that the element has been found.

Discuss the difference between absolute and relative XPath expressions and when to use each.

Sol;

```
package selenium_Tasks_Assignments;

import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Day2_Task3{
```

```
	public static void main(String[] args) {
		// TODO Auto-generated method stub
	    WebDriver driver = new ChromeDriver();

	driver.get("https://rahulshettyacademy.com/AutomationPractice/"
);
		WebElement radio1 =
driver.findElement(By.name("radioButton"));
		System.out.println("Displayed "+radio1.isDisplayed());
		System.out.println("Selected "+radio1.isSelected());
		System.out.println("Enabled "+radio1.isEnabled());
		radio1.click();
		System.out.println("Selected "+radio1.isSelected());
		List<WebElement> radioBtns =
driver.findElements(By.xpath("//input[@name='radioButton']"));
		System.out.println("Total no of radio buttons(XPATH):
"+radioBtns.size());
		List<WebElement> radioBtn =
driver.findElements(By.cssSelector("input[name='radioButton']"));
		System.out.println("Total no of radio buttons(CSS):
"+radioBtn.size());
	}

}
```

Absolute XPath:

A XPath expression specifies the exact location of an element from the root node of the document.

They are usually longer and harder to maintain. Due to changes in code.

When we can use:

Use when the structure of the document is unlikely to change.

Not using for dynamic web pages because the structure keeps changing its difficult to find xpath.

Relative XPath:

A relative XPath expression specifies the location of an element relative to another element in the document.

They are easier to read, write, and maintain, especially for complex documents.

When we can use:

It is easier to use because to will give exact path

Day3_Tasks

Task 1: Form Element Interaction and Data Extraction

Topic: Advanced Element Interaction - Handling Tables and Form Elements

Description: Write a Selenium script to extract data from a table and interact with form elements like checkboxes and radio buttons on a sample web page.

Steps:

Launch the web browser and navigate to a predefined web page with a table and form elements.

Use Selenium's WebDriver to locate the table element and retrieve the text from a specific cell.

Locate a checkbox and a radio button on the form and use Selenium commands to select and unselect the checkbox, and select the radio button.

Print out the extracted data and the status of checkbox and radio button selections to the console.

Sol:

```
package selenium_Tasks_Assignments;

import java.time.Duration;
import java.util.List;
 import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver; import
org.openqa.selenium.support.ui.ExpectedConditions; import
org.openqa.selenium.support.ui.WebDriverWait;

public class Day3Task1 {

	public static void main(String[] args) {

		WebDriver driver = new ChromeDriver();

		try {
	driver.get("https://tutorialsninja.com/demo/");
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(50));
List<WebElement> featuredProducts =
```

```
wait.until(ExpectedConditions.visibilityOfAllElementsLocatedBy(By.cssSelect
or(".product-thumb")));

for (WebElement product : featuredProducts) {
String productName = product.findElement(By.cssSelector("h4 a")).getText();
String productPrice =
product.findElement(By.cssSelector(".price")).getText();
System.out.println("Product Name: " + productName + " | Price: " +
productPrice);
                    }

driver.findElement(By.linkText("My Account")).click();
wait.until(ExpectedConditions.visibilityOfElementLocated(By.linkText("Regis
ter"))).click();

WebElement privacyPolicyCheckbox = wait
        .until(ExpectedConditions.visibilityOfElementLocated(By.name("agree")
));
      privacyPolicyCheckbox.click();
System.out.println("Privacy Policy Checkbox selected: " +
privacyPolicyCheckbox.isSelected());

WebElement subscribeYesRadio = wait.until(ExpectedCondition
         .visibilityOfElementLocated(By.xpath("//input[@name='newsletter' and
@value='1']")));
WebElement subscribeNoRadio =
driver.findElement(By.xpath("//input[@name='newsletter' and @value='0']"));
subscribeYesRadio.click();
System.out.println("Subscribe Yes Radio selected: " +
subscribeYesRadio.isSelected());
System.out.println("Subscribe No Radio selected: " +
subscribeNoRadio.isSelected());

            } finally {
                  // Close the browser
                  driver.quit();
            }
      }

}
```

Task 2: Synchronization and Alert Handling

Topic: Selenium Synchronization Strategies and Handling Alerts

Description: Implement an explicit wait to synchronize a web element's presence and handle a simple JavaScript alert on the page.

Steps:

Open a web browser and navigate to a predefined web page that triggers a JavaScript alert after a button click.

Implement an explicit wait using WebDriverWait to wait for the button to become clickable. Click the button to trigger the alert and then implement another explicit wait to ensure the alert is present.
Use Selenium's Alert interface to accept the alert and then print a confirmation to the console.

Sol:

```
package selenium_Tasks_Assignments;
import java.time.Duration; import
java.util.List;
 import org.openqa.selenium.Alert;
import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement;
import
org.openqa.selenium.chrome.ChromeDriv
er; import
org.openqa.selenium.support.ui.Expect
edConditions; import
org.openqa.selenium.support.ui.WebDri
verWait;

public class Day3Task2_Synchronization_and_Alert_Handling {
	public static void main(String[] args) {

		WebDriver driver = new ChromeDriver();

		try {

			driver.get("https://tutorialsninja.com/demo/");

			System.out.println("Page title: " + driver.getTitle());

			List<WebElement> buttons =
driver.findElements(By.xpath("//button[contains(text(),'Add to Cart')]"));
			if (buttons.isEmpty()) {
				System.out.println("No 'Add to Cart' button
found.");
				return;
			} else {
				 System.out.println("'Add to Cart' button found.");
			}

			WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(20));
			WebElement button = wait.until(
```

```
      ExpectedConditions.elementToBeClickable(By.xpath("//button[contains(t
ext(),'Add to Cart')]")));

                  button.click();

                  wait.until(ExpectedConditions.alertIsPresent());

                  Alert alert = driver.switchTo().alert();
      alert.accept();

                  System.out.println("Alert was present and accepted.");

            } catch (Exception e) {
                  System.out.println("An error occurred: " +
e.getMessage());
            } finally {
                  // Close the browser
                  driver.quit();
            }
      }

}
```

Day4_Tasks

Task 1: Create a Simple TestNG Class with Basic Annotations

Topic: Annotation and Execution with TestNG

Description: Create a basic TestNG class that demonstrates the use of fundamental annotations. This class w include two test methods: one that passes and one that intentionally fails to illustrate the assertion mechanism.

Steps:

Open your IDE and create a new Java class within your test project.

Import the TestNG library and set up the class with the @Test annotation.

Write the first test method testPass() that asserts a true condition, like Assert.assertTrue(1 == 1);. Write the second test method testFail() that asserts a false condition, like Assert.assertTrue(1 == 2);.

Annotate both methods with @Test and give them meaningful names, such as shouldPass and shouldFail.

Execute the TestNG class using the IDE's TestNG plugin or via command line, and observe the output.

Briefly describe what the @BeforeMethod and @AfterMethod annotations do, and how they can be utilized in setting up and tearing down test conditions.

Sol:

```
package selenium_Tasks_Assignments;

import org.testng.Assert;
import org.testng.annotations.AfterMethod; import
org.testng.annotations.BeforeMethod; import
org.testng.annotations.Test;
 public class
Day4Task1_Create_a_Simple_TestNG_Class_with_Basic_Annotations
{

	@BeforeMethod
	public void setUp() {

		System.out.println("Setting up test conditions.");
	}

	@Test
	public void shouldPass() {

		Assert.assertTrue(1 == 1, "This test should always pass.");
	}

	@Test
	public void shouldFail() {

		Assert.assertTrue(1 == 2, "This test should always fail.");
	}

	@AfterMethod
	public void tearDown() {

		System.out.println("Tearing down test conditions.");
	}

}
```

Day5_Task

Task 1: Execute Maven Commands to Compile and Run Tests

Topic: Maven Life Cycle

Description: Use Maven to compile a simple Java project and run its tests. This exercise will test the fresher's ability to interact with Maven, which is commonly used for building and managing any Java-based project.

Steps:

Open the command line or terminal.

Navigate to the directory of a pre-existing simple Java project (this could be provided as a ZIP file for the purpose of the interview task).

Execute mvn compile to compile the Java source files of the project.

Next, run mvn test to execute the tests in the project.

Describe what the mvn package command does and when you would use it.

Sol;

Steps to Compile and Run Tests

1. Open the command line or terminal.
2. Navigate to the project directory.

Use the cd command to change to the directory where your Maven project is located. For example:

Code: cd path/to/your/project

3. Compile the Java source files.

Execute the following Maven command to compile the project's source code

Code: mvn compile

4. Run the tests.

Execute the following Maven command to run the project's tests:

Code: mvn test

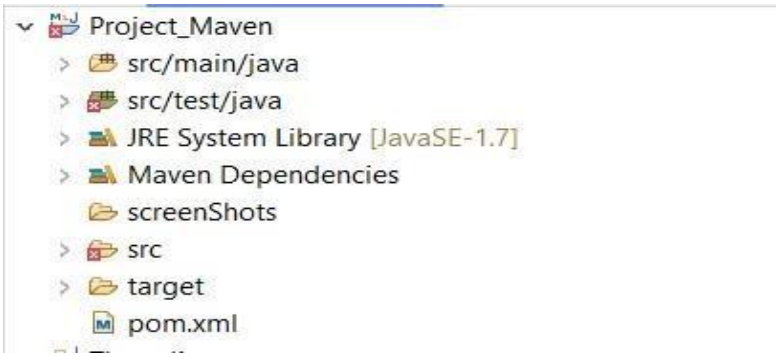
Typical pom.xml Configuration for mvn package:

This is pom.xml file. This acts as the heart of maven project, here we can add dependencipies plugins Related to your project.

Basic xml File Looks like:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <packaging>jar</packaging> <!-- This can be jar, war, ear, etc. -->



  <dependencies>

    <!-- here we can add dependencies-->
  </dependencies>



  <build>
    <plugins>

      <!-- Here we can add plug inns -->
    </plugins>
  </build>
</project>
```

By using this pom.xml file we can add up applications with their dependencies and we can work on it.



This is Basic Maven Project Looks like.

We have src/main/java → Where we can write Main methods and project related codes.

We have src/test/java → Where we can run write test cases fo the project.

Pom.xml → where we can add dependencies.

Task 2: Write a Simple Feature File with a Scenario

Topic: Crafting Feature Files and Defining Scenarios

Description: Create a basic feature file using Gherkin syntax for a login functionality of a web application, defining one scenario that includes a given, when, and then steps.

Steps:

Open your text editor or IDE with Cucumber support.

Create a new file named login.feature.

Define the Feature at the top of the file: Feature: User Login.

Write a Scenario with the title: Scenario: Successful login with valid credentials.

Use Gherkin syntax to define the steps:

Given: "Given the user is on the login page".

When: "When the user enters valid username and password".

Then: "Then the user should be redirected to the dashboard".

Sol:

Save the feature file.

Steps to Create the Feature File

1. Open your text editor or IDE with Cucumber support.
2. Create a new file named login. Feature.

Write Below Steps in Feature File

```
Feature: User Login
As a registered user
I want to log into the web application
So that I can access my dashboard
Scenario: Successful login with valid credentials
Given the user is on the login page
When the user enters a valid username and password
Then the user should be redirected to the dashboard
```

Create Step Definition Project:

```
package com.StepDefinition;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import org.junit.Assert;
public class LoginSteps {
private String currentPage;
private boolean loginSuccess;
@Given("the user is on the login page")
public void theUserIsOnTheLoginPage()
{ currentPage = "loginPage";
System.out.println("User is on the login page");
}
@When("the user enters valid username and password")
public void theUserEntersValidUsernameAndPassword() {
      if ("loginPage".equals(currentPage)) { loginSuccess = true; //
Simulate successful login
} else { loginSuccess = false;
}
System.out.println("User enters valid credentials");
}
@Then("the user should be redirected to the dashboard")
public void theUserShouldBeRedirectedToTheDashboard()
{ if (loginSuccess) { currentPage = "dashboard";
}
Assert.assertEquals("dashboard", currentPage);
System.out.println("User is redirected to the dashboard");
}
```

```
}
```

Create My Runner Class File

```
package com.MyRunner;
import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
@RunWith(Cucumber.class)
@CucumberOptions(
features = "src/test/resources/features",
glue = "com.example.steps", plugin = {"pretty",
"html:target/cucumber-reports.html"}
)
public class TestRunner { }
```

We can Run from Runner File to get Check Valid Login Credentials

Day6_Task

Task 1: Write a Simple Cucumber Feature for a Login Functionality

Topic: Cucumber with Selenium - Crafting Feature Files

Description: Create a basic Cucumber feature file for a login functionality, including scenarios for a successful

login and a failed login attempt due to incorrect credentials.

Steps:

Open your IDE and create a new feature file named login.feature in your Cucumber project.

Define a Feature: User Login at the top of the file.

Write a Scenario: Successful Login with Gherkin steps that include Given, When, Then, such as:

Given the user is on the login page

When the user enters valid credentials

Then the user is redirected to the homepage

Write a Scenario: Unsuccessful Login with steps for attempting to log in with invalid credentials and asserting the error message.

Sol:

```
Feature: User Login

Scenario: Successful login with valid credentials
Given the user is on the login page
When the user enters a valid username and password
And the user clicks the login button
Then the user should be redirected to the homepage
And a welcome message should be displayed
```

```
package stepDef;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class Write_a_Simple_Feature_File_with_a_Scenario {

	@Given("the user is on the login page")
	public void the_user_is_on_the_login_page() {
		System.out.println("Inside login_page method");

	}

	@When("the user enters a valid username and password")
	public void the_user_enters_a_valid_username_and_password() {
		System.out.println("Inside username and password method");

	}

	@When("the user clicks the login button")
	public void the_user_clicks_the_login_button() {
		System.out.println("Inside login_button method");

	}
	@Then("the user should be redirected to the homepage")
	public void the_user_should_be_redirected_to_the_homepage() {
		System.out.println("Inside homepage method");
	}

	@Then("a welcome message should be displayed")
	public void a_welcome_message_should_be_displayed() {
		System.out.println("Inside display method");
	}

}
```

```java
package myRunner;

import org.junit.runner.RunWith;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)
@CucumberOptions(
    features = "C:\\Users\\sunee\\Desktop\\JAVA
PROGRAMS\\Maven\\src\\test\\java\\feature\\Login.feature",
    glue="stepDef",
    plugin = {"pretty","html:target/Login.html"},
      monochrome = true
)
public class TestRunner {
}
```

Task 2: Perform a Simple GET Request Using Postman
Topic: Postman for API Testing - Making API Calls
Description: Use Postman to send a GET request to a public API and validate the response.

Open Postman.

Create a new request by selecting the HTTP method GET.

Enter the URL of a public API endpoint that returns JSON data (e.g., https://jsonplaceholder.typicode.com/posts/1).

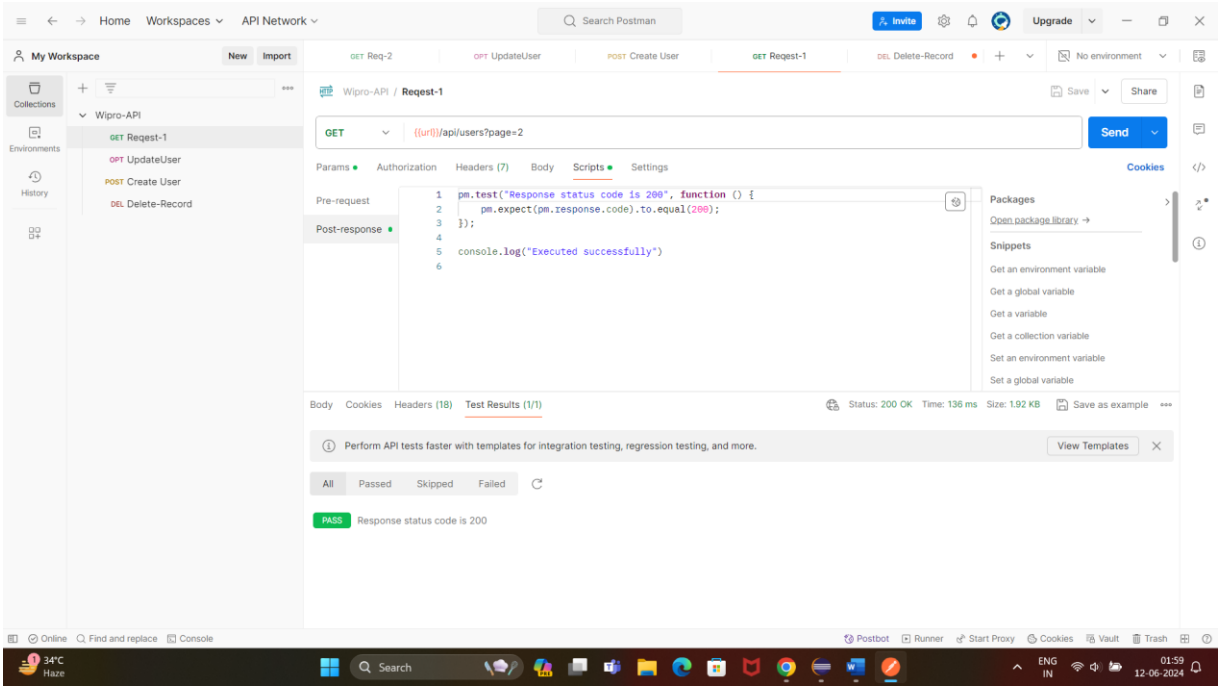
Send the request and observe the response.

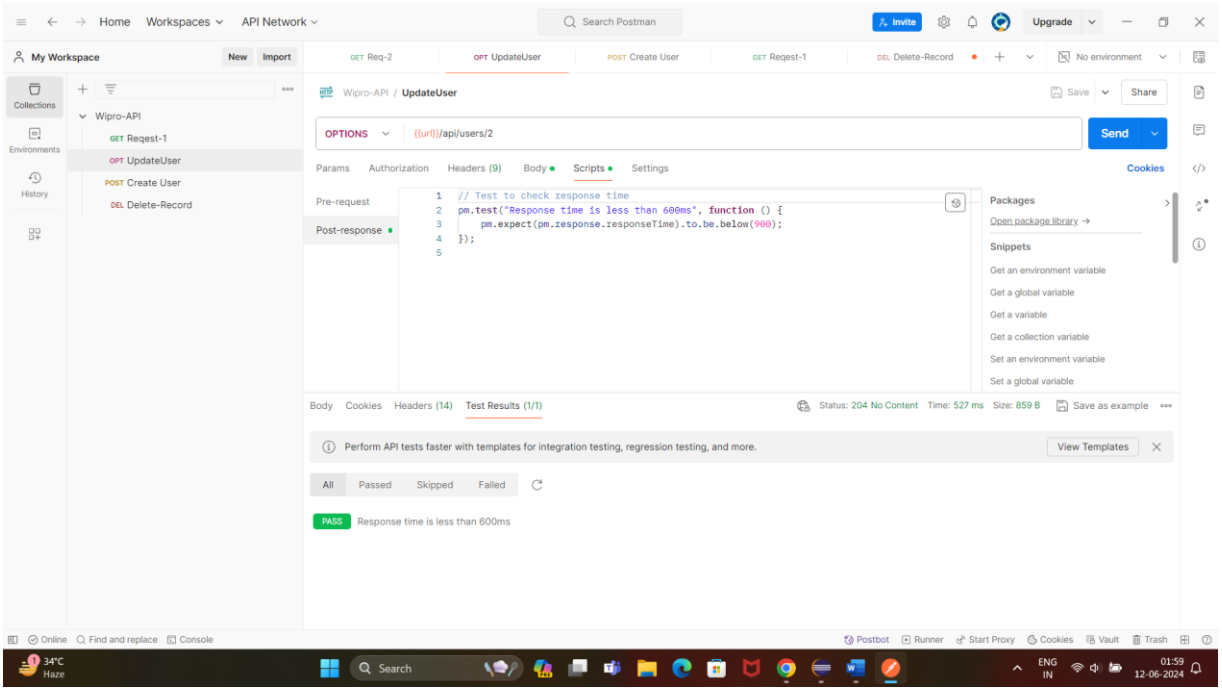
Check the status code is 200 OK and use the Tests tab in Postman to write a simple test confirming the status code, such as:

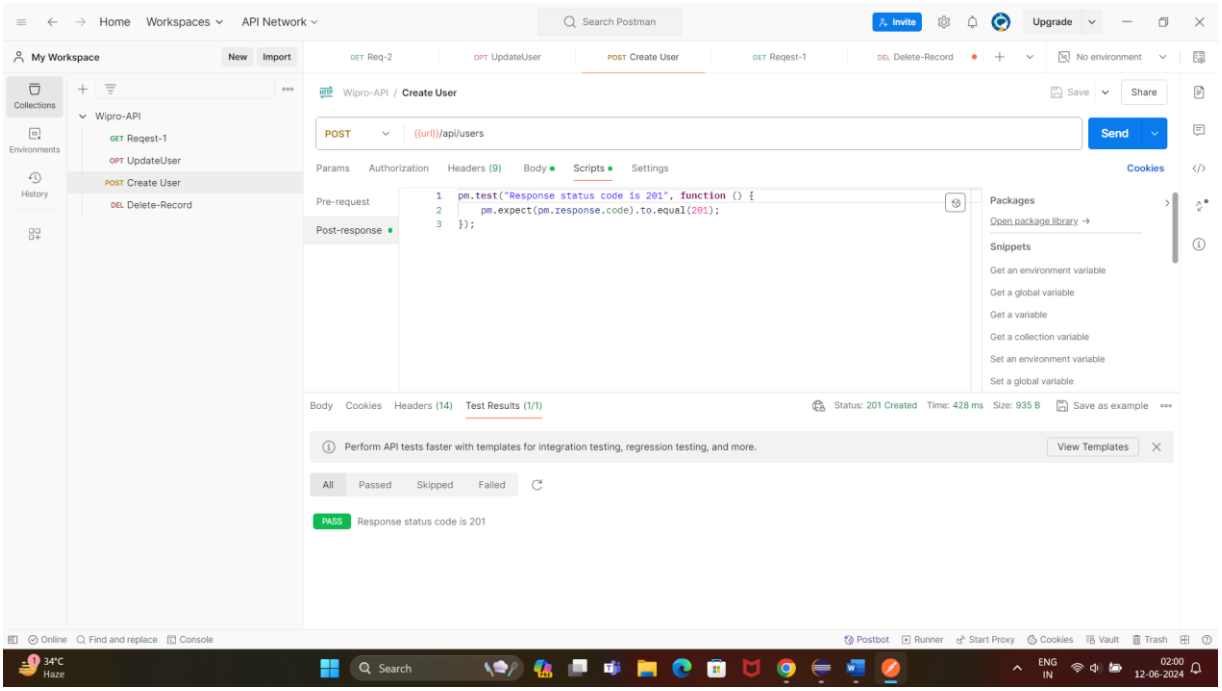
pm.test("Status code is 200", function () { pm.response.to.have.status(200); });

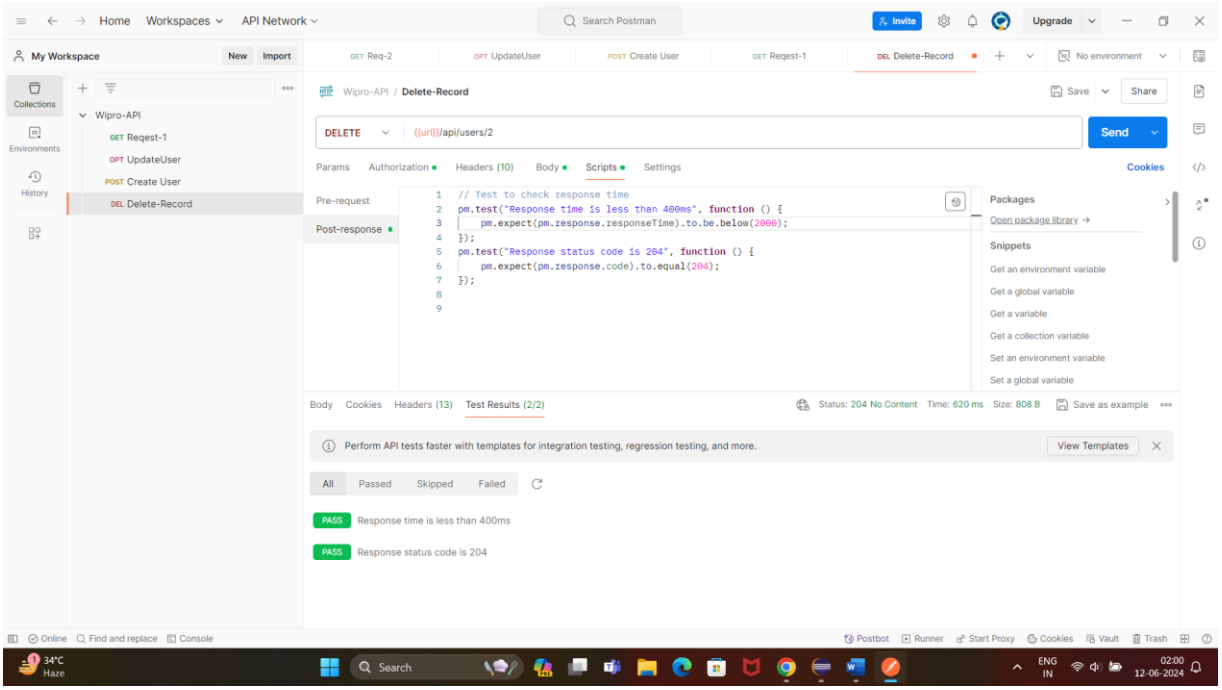
Sol:

New Request:


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
GET Req-2
OPT UpdateUser
POST Create User
GET Reqest-1
DEL Delete-Record
No environment
Collections
Environments
History
Wipro-API
GET Reqest-1
OPT UpdateUser
POST Create User
DEL Delete-Record
Wipro-API / Reqest-1
Save
Share
GET
{{url}}/api/users?page=2
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
1 pm.test("Response status code is 200", function () {
2 pm.expect(pm.response.code).to.equal(200);
3 });
4
5 console.log("Executed successfully")
6
Packages
Open package library
Snippets
Get an environment variable
Get a global variable
Get a variable
Get a collection variable
Set an environment variable
Set a global variable
Body
Cookies
Headers (18)
Test Results (1/1)
Status: 200 OK
Time: 136 ms
Size: 1.92 KB
Save as example
Perform API tests faster with templates for integration testing, regression testing, and more.
View Templates
All
Passed
Skipped
Failed
PASS
Response status code is 200
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
34°C
Haze
Search
ENG
IN
01:59
12-06-2024


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
GET Req-2
OPT UpdateUser
POST Create User
GET Reqest-1
DEL Delete-Record
No environment
Collections
Environments
History
Wipro-API
GET Reqest-1
OPT UpdateUser
POST Create User
DEL Delete-Record
Wipro-API / UpdateUser
Save
Share
OPTIONS
{{url}}/api/users/2
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
1 // Test to check response time
2 pm.test("Response time is less than 600ms", function () {
3 pm.expect(pm.response.responseTime).to.be.below(900);
4 });
5
Packages
Open package library
Snippets
Get an environment variable
Get a global variable
Get a variable
Get a collection variable
Set an environment variable
Set a global variable
Body
Cookies
Headers (14)
Test Results (1/1)
Status: 204 No Content
Time: 527 ms
Size: 859 B
Save as example
Perform API tests faster with templates for integration testing, regression testing, and more.
View Templates
All
Passed
Skipped
Failed
PASS
Response time is less than 600ms
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
34°C
Haze
Search
ENG
IN
01:59
12-06-2024


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
GET Req-2
OPT UpdateUser
POST Create User
GET Reqest-1
DEL Delete-Record
No environment
Collections
Environments
History
Wipro-API
GET Reqest-1
OPT UpdateUser
POST Create User
DEL Delete-Record
Wipro-API / Create User
Save
Share
POST
{{url}}/api/users
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
1 pm.test("Response status code is 201", function () {
2 pm.expect(pm.response.code).to.equal(201);
3 });
Packages
Open package library
Snippets
Get an environment variable
Get a global variable
Get a variable
Get a collection variable
Set an environment variable
Set a global variable
Body
Cookies
Headers (14)
Test Results (1/1)
Status: 201 Created
Time: 428 ms
Size: 935 B
Save as example
Perform API tests faster with templates for integration testing, regression testing, and more.
View Templates
All
Passed
Skipped
Failed
PASS
Response status code is 201
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
34°C
Haze
Search
ENG
IN
02:00
12-06-2024


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
GET Req-2
OPT UpdateUser
POST Create User
GET Reqest-1
DEL Delete-Record
No environment
Collections
Environments
History
Wipro-API
GET Reqest-1
OPT UpdateUser
POST Create User
DEL Delete-Record
Wipro-API / Delete-Record
Save
Share
DELETE
{{url}}/api/users/2
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
1 // Test to check response time
2 pm.test("Response time is less than 400ms", function () {
3 pm.expect(pm.response.responseTime).to.be.below(2000);
4 });
5 pm.test("Response status code is 204", function () {
6 pm.expect(pm.response.code).to.equal(204);
7 });
8
9
Packages
Open package library
Snippets
Get an environment variable
Get a global variable
Get a variable
Get a collection variable
Set an environment variable
Set a global variable
Body
Cookies
Headers (13)
Test Results (2/2)
Status: 204 No Content
Time: 620 ms
Size: 808 B
Save as example
Perform API tests faster with templates for integration testing, regression testing, and more.
View Templates
All
Passed
Skipped
Failed
PASS
Response time is less than 400ms
PASS
Response status code is 204
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
34°C
Haze
Search
ENG
IN
02:00
12-06-2024

Day7_Tasks

Task 1: Verify HTTP Status Codes Using RestAssured

Topic: Understanding HTTP Status Response Codes

Description: Write a RestAssured test to GET a resource from a public API and verify the HTTP status code.

Steps:

Choose a public API endpoint for a GET request (e.g., https://api.publicapis.org/entries).

Write a RestAssured test using the GIVEN-WHEN-THEN methodology.

Use GIVEN to set up the RestAssured request.

Use WHEN to specify the GET method and the URL.

Use THEN to verify that the HTTP status code is 200.

Execute the test and confirm that the assertion passes.

Sol;

Steps to Write the RestAssured Test

1. Choose a public API endpoint for a GET request:
    - **For this example, we'll use the endpoint https://api.publicapis.org/entries, which provides information about public APIs.**
2. Write a RestAssured test using the GIVEN-WHEN-THEN methodology:

```
import org.junit.jupiter.api.Test; import
static io.restassured.RestAssured.given;
import static
org.hamcrest.Matchers.equalTo; public class
APITest {

   @Test public void
   testStatusCode() { given()
         .when()
            .get("https://api.publicapis.org/entries")
         .then()
            .statusCode(200);
   }
}
```

Explanation

- given()**: Sets up the RestAssured request.**
- when()**: Specifies the HTTP method and the URL.**
- get()**: Sends a GET request to the specified URL.**
- then()**: Verifies the response.**
- statusCode(200)**: Asserts that the HTTP status code of the response is 200 (OK).**

3. Execute the test:
    - **Run the test class using a test runner or an IDE that supports JUnit.**
4. Confirm that the assertion passes:
    - **Check the test execution output to ensure that the assertion for the HTTP status code passes.**

Task 2: **Create a POST Request with JSON Payload Using RestAssured**

Topic: POST Requests with JSON Strings and POJO Classes

Description: Write a RestAssured test to POST a JSON payload to a public API and verify the response body content.

Steps:

Choose a public API that accepts POST requests and requires a JSON payload (e.g., https://jsonplaceholder.typicode.com/posts).

Create a simple POJO (Plain Old Java Object) class to represent the JSON payload structure.

Write a RestAssured test that uses the POJO class to create the JSON payload.

Use RestAssured to send the POST request with the JSON payload.

Verify the response body to ensure the correct data is returned (e.g., using JSONPath). Confirm the status code is 201 (Created) or as appropriate for the API.

Sol;

Steps to Create the RestAssured Test

1. Choose a public API that accepts POST requests and requires a JSON payload:
    - **For this example, we'll use the endpoint `https://jsonplaceholder.typicode.com/posts`, which allows us to create new posts.**

2. Create a simple POJO (Plain Old Java Object) class to represent the JSON payload structure:

```
public class Post { private
   int
    userId; private
    int id; private
    String title;
    private String
    body;

   // Constructor, getters, and setters
}
```

3. Write a RestAssured test that uses the POJO class to create the JSON payload:

```
import org.junit.jupiter.api.Test; import
io.restassured.RestAssured; import
io.restassured.http.ContentType; import
static org.hamcrest.Matchers.equalTo;
public class PostTest {

   @Test public void
   testPostRequest() { Post post =
   new Post(); post.setUserId(1);
   post.setId(101);
   post.setTitle("Test Title");
   post.setBody("Test Body");

      RestAssured.given()
         .contentType(ContentType.JSON)
         .body(post)
      .when()
         .post("https://jsonplaceholder.typicode.com/posts")
      .then()
         .statusCode(201) // Ensure status code is 201 Created
         .body("title", equalTo("Test Title")) // Verify response body content
         .body("body", equalTo("Test Body"));
   }
```

}

Explanation

- RestAssured.given()**: Sets up the RestAssured request.**
- contentType(ContentType.JSON)**: Specifies that the request body is in JSON format.**
- body(post)**: Sets the JSON payload using the POJO object.**
- when().post()**: Sends a POST request to the specified URL.**
- then()**: Verifies the response.**
- statusCode(201)**: Asserts that the HTTP status code of the response is 201 (Created).**
- body("title", equalTo("Test Title"))**: Verifies that the "title" field in the response body is equal to "Test Title".**
- body("body", equalTo("Test Body"))**: Verifies that the "body" field in the response body is equal to "Test Body".**

4. Execute the test:
    - **Run the test class using a test runner or an IDE that supports JUnit.**
5. Confirm the assertion passes:
    - **Check the test execution output to ensure that all assertions pass, indicating that the POST request was successful and the response body content is as expected.**