## Mphasis ATE Capstone Project Medicare

**Name: Prajwal Diwakar**

## Create a Testing Framework for the Medicare Website

## Source code:

## Rest Assured for API endpoints.

**Get Products:**

```
package MphasisATE_CapstoneProjectRestAssured_TestNG_Medicare;

import io.restassured.RestAssured;
import org.testng.annotations.Test;

import static org.hamcrest.CoreMatchers.equalTo;

import org.apache.log4j.Logger;


public class GetProducts {
	private static final String Base_Url ="http://localhost:9010";
	static final Logger logger= Logger.getLogger(GetProducts.class);

	@Test(description="Get request method to get all the product details")


	public void testGetProducts() {

		logger.info("START::GET method for the Product details");
		logger.info("GET: URL " +Base_Url+ "/get-products");

		int code=101;
		String message="27 Products Fetched Successfully.";

		try {
			RestAssured.given().baseUri(Base_Url)
			.when().get("/get-products")
			.then().assertThat().statusCode(200)
			.body("code", equalTo(code)).and()
			.body("message", equalTo(message));
		}
		catch(Exception e) {
				logger.error("Exception Object :: " + e.toString());
				logger.error("End Exception :: "+e.getLocalizedMessage());
		}

		String response= RestAssured.given().baseUri(Base_Url)
				.when().get("/get-products").getBody().asPrettyString();
		logger.info("Response is " +response);
		logger.info("END:: GET method for the Product details");
	}
}
```

**Get Registered Users:**

```
package MphasisATE_CapstoneProjectRestAssured_TestNG_Medicare;
```

```java
import io.restassured.RestAssured;
import org.testng.annotations.Test;

import static org.hamcrest.CoreMatchers.equalTo;

import org.apache.log4j.Logger;


public class GetRegisteredUser {
	private static final String Base_Url ="http://localhost:9010";
	static final Logger logger= Logger.getLogger(GetProducts.class);

	@Test(description="Get request method to get all the registered users ")


	public void testGetProducts() {

		logger.info("START::GET method for the Registered user details");
		logger.info("GET: URL " +Base_Url+ "/get-users");

		int code=101;
		String message="4 Users Fetched Successfully.";

		try {
			RestAssured.given().baseUri(Base_Url)
			.when().get("/get-users")
			.then().assertThat().statusCode(200)
			.body("code", equalTo(code)).and()
			.body("message", equalTo(message));
		}
		catch(Exception e) {
				logger.error("Exception Object :: " + e.toString());
				logger.error("End Exception :: "+e.getLocalizedMessage());
		}

		String response= RestAssured.given().baseUri(Base_Url)
				.when().get("/get-users").getBody().asPrettyString();
		logger.info("Response is " +response);
		logger.info("END:: GET method for the Registered user details");
	}

}
```

**Add Product:**

```java
package MphasisATE_CapstoneProjectRestAssured_TestNG_Medicare;

import static org.hamcrest.CoreMatchers.equalTo;

import org.apache.log4j.Logger;
import org.testng.annotations.Test;
import io.restassured.RestAssured;
import io.restassured.http.ContentType;
```

```
public class PostAddProduct {

	private static final String Base_Url ="http://localhost:9010";
	static final Logger logger= Logger.getLogger(PostAddProduct.class);

	@Test(description="Post request method to add the product")


	public void testGetProducts() {

		logger.info("START::POST method to add the product details");
		logger.info("POST: URL " +Base_Url+ "/add-product");

		String requestBody="{\"id\": 999,\"image\": \"1.png\", \"name\":
\"Disprin\",\"brand\": \"BZ Medico\", " +
		  "\"status\": 1, \"price\": 100}";

		try {
			RestAssured.given().baseUri(Base_Url)
			.contentType(ContentType.JSON).body(requestBody)
			.when().post("/add-product")
			.then().assertThat().statusCode(200)
			.body("products.id", equalTo(999)).and()
			.body("products.name", equalTo("Disprin"));;
		}
		catch(Exception e) {
				logger.error("Exception Object :: " + e.toString());
				logger.error("End Exception :: "+e.getLocalizedMessage());
		}

		String response=
RestAssured.given().baseUri(Base_Url).contentType(ContentType.JSON).body(requestBody)
				.when().post("/add-product").getBody().asPrettyString();
		logger.info("Response is " +response);
		logger.info("END:: POST method to add the product details");
	}

}
```

**Update product:**

```
package MphasisATE_CapstoneProjectRestAssured_TestNG_Medicare;

import static org.hamcrest.CoreMatchers.equalTo;

import org.apache.log4j.Logger;
import org.testng.annotations.Test;

import io.restassured.RestAssured;
import io.restassured.http.ContentType;


public class PutUpdateProduct {

	private static final String Base_Url ="http://localhost:9010";
	static final Logger logger= Logger.getLogger(PutUpdateProduct.class);
```

```
	@Test(description="Put request method to update the product")


	public void testPutUpdate() {

		Logger.info("START::PUT method to update the product details");
		Logger.info("PUT: URL " +Base_Url+ "/update-product");

		String requestBody="{\"id\": 999,\"image\": \"2.png\", \"name\":
\"Disprin\",\"brand\": \"BZ Medico\", " +
		  "\"status\": 1, \"price\": 120}";

		try {
			RestAssured.given().baseUri(Base_Url)
			.contentType(ContentType.JSON).body(requestBody)
			.when().put("/update-product")
			.then().assertThat().statusCode(200)
			.body("product.id", equalTo(999)).and()
			.body("product.image", equalTo("2.png")).and()
			.body("product.price", equalTo(120)).and()
			.body("product.name", equalTo("Disprin"));
		}
		catch(Exception e) {
				Logger.error("Exception Object :: " + e.toString());
				Logger.error("End Exception :: "+e.getLocalizedMessage());
		}

		String response=
RestAssured.given().baseUri(Base_Url).contentType(ContentType.JSON).body(requestBody)
				.when().put("/update-product").getBody().asPrettyString();
		Logger.info("Response is " +response);
		Logger.info("END:: PUT method to update the product details");
	}


}
```

**Update status of product.**

```
package MphasisATE_CapstoneProjectRestAssured_TestNG_Medicare;

import static org.hamcrest.CoreMatchers.equalTo;

import org.apache.log4j.Logger;
import org.testng.annotations.Test;

import io.restassured.RestAssured;
import io.restassured.http.ContentType;


public class PutUpdateStatusProduct {
	private static final String Base_Url ="http://localhost:9010";
	static final Logger logger= Logger.getLogger(PutUpdateStatusProduct.class);

	@Test(description="PUT request method to update the status of product")


	public void testPostUpdateProductStatus() {

		Logger.info("START::PUT method to update the status of product");
		Logger.info("PUT: URL " +Base_Url+ "/update-product-status");
```

```
		String requestBody="{\"id\": 999,\"image\": \"1.png\", \"name\":
\"Disprin\",\"brand\": \"BZ Medico\", " +
		  "\"status\": 0, \"price\": 100}";

		try {
			RestAssured.given().baseUri(Base_Url)
			.contentType(ContentType.JSON).body(requestBody)
			.when().put("/update-product-status")
			.then().assertThat().statusCode(200)
			.body("product.id", equalTo(999)).and()
			.body("product.image", equalTo("2.png")).and()
			.body("product.price", equalTo(120)).and()
			.body("product.name", equalTo("Disprin")).and()
			.body("product.status", equalTo(0));
		}
		catch(Exception e) {
				logger.error("Exception Object :: " + e.toString());
				logger.error("End Exception :: "+e.getLocalizedMessage());
		}

		String response=
RestAssured.given().baseUri(Base_Url).contentType(ContentType.JSON).body(requestBody)
				.when().put("/update-product-status").getBody().asPrettyString();
		logger.info("Response is " +response);
		logger.info("END:: PUT method to update the status of product");
	}


}
```

**Delete product:**

```
package MphasisATE_CapstoneProjectRestAssured_TestNG_Medicare;

import static org.hamcrest.CoreMatchers.equalTo;

import org.apache.log4j.Logger;
import org.testng.annotations.Test;

import io.restassured.RestAssured;


public class DeleteProduct {

	private static final String Base_Url ="http://localhost:9010";
	static final Logger logger= Logger.getLogger(DeleteProduct.class);

	@Test(description="Delete request method to delete product")


	public void testGetProducts() {

		logger.info("START::DELETE method to delete Product");
		logger.info("DELETE: URL " +Base_Url+ "/get-products");

		int code=101;
		String message="Product with ID 101 Deleted Successfully.";

		try {
			RestAssured.given().baseUri(Base_Url)
			.when().delete("/delete-product?id=101")
```

```
                        .then().assertThat().statusCode(200)
                        .body("code", equalTo(code)).and()
                        .body("message", equalTo(message));;
                }
                catch(Exception e) {
                                logger.error("Exception Object :: " + e.toString());
                                logger.error("End Exception :: "+e.getLocalizedMessage());
                }

                String response= RestAssured.given().baseUri(Base_Url)
                                .when().delete("/delete-
product?id=101").getBody().asPrettyString();
                logger.info("Response is " +response);
                logger.info("END:: DELETE method to delete Product");
        }

}
```

## Selenium with TestNG:

**Login Page:**

```
package MphasisATE_CapstoneProject_Selenium_TestNG_Medicare;

import static org.testng.Assert.assertEquals;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;


public class MedicareLoginPage {

        // step1: formulate a test domain url & driver path
                                String siteUrl = "http://localhost:9010";
                                String driverPath = "drivers/windows/chromedriver.exe";
                                WebDriver driver;

                                @BeforeClass
                                public void setUp() throws InterruptedException {

                                        // step2: set system properties for selenium dirver
                                        System.setProperty("webdriver.chromedriver.driver",
driverPath);


                                        // step3: instantiate selenium webdriver
                                        driver = new ChromeDriver();

                                        // step4: launch browser
                                        driver.get(siteUrl);
```

```
                                Thread.sleep(2000);
                        }

                        @AfterClass
                        public void cleanUp() {
                                driver.quit();  // the quit() method closes all browser
windows and ends the WebDriver session.
                                // driver.close(); // the close() closes only the current
window on which Selenium is running automated tests.The WebDriver session, however, remains
active.
                        }

        @Test(description = "Test Medicare Login Page ")

        public void testLoginPage() throws InterruptedException {

                WebElement searchbox1 = driver.findElement(By.id("email"));
                searchbox1.sendKeys("john@example.com");

                WebElement searchbox2 = driver.findElement(By.id("password"));
                searchbox2.sendKeys("john123");

                WebElement login =
driver.findElement(By.xpath("/html/body/div[2]/form/button"));
                login.click();

                // add delay
                Thread.sleep(5000);

                String url= driver.getCurrentUrl();
                assertEquals("http://localhost:9010/login", url);
        }
}
```

**Register Page:**

```
package MphasisATE_CapstoneProject_Selenium_TestNG_Medicare;

import static org.testng.Assert.assertEquals;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;


public class MedicareRegisterPage {
        // step1: formulate a test domain url & driver path
                        String siteUrl = "http://localhost:9010";
                        String driverPath = "drivers/windows/chromedriver.exe";
                        WebDriver driver;

                        @BeforeClass
                        public void setUp() throws InterruptedException {

                                // step2: set system properties for selenium dirver
                                System.setProperty("webdriver.chromedriver.driver",
driverPath);
```

```
                                        // step3: instantiate selenium webdriver
                                        driver = new ChromeDriver();

                                        // step4: launch browser
                                        driver.get(siteUrl);

                                        Thread.sleep(2000);
                                }

                                @AfterClass
                                public void cleanUp() {
                                        driver.quit();  // the quit() method closes all browser
windows and ends the WebDriver session.
                                        // driver.close(); // the close() closes only the current
window on which Selenium is running automated tests.The WebDriver session, however, remains
active.
                                }
                                @Test(description = "Test Medicare Register Page ")


                                public void testRegisterPage() throws InterruptedException {

                                        WebElement link =
driver.findElement(By.xpath("/html/body/div[2]/form/a"));

                                        link.click();
                                        WebElement searchbox1 = driver.findElement(By.id("name"));
                                        searchbox1.sendKeys("Prajwal.Diwakar");

                                        WebElement searchbox2 =
driver.findElement(By.id("email"));
                                        searchbox2.sendKeys("prajwal.diwakar@mphasis.com");

                                        WebElement searchbox3 =
driver.findElement(By.id("password"));
                                        searchbox3.sendKeys("prajwal@123");

                                        WebElement register =
driver.findElement(By.xpath("/html/body/div[2]/form/button"));
                                        register.submit();

                                        // add delay
                                        Thread.sleep(2000);

                                        String url= driver.getCurrentUrl();
                                        assertEquals("http://localhost:9010/register-user", url);
                                }

}
```

**Add to cart Page:**

```
package MphasisATE_CapstoneProject_Selenium_TestNG_Medicare;

import static org.testng.Assert.assertEquals;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;


public class MedicareAddProductToCart {

                        // step1: formulate a test domain url & driver path
                        String siteUrl = "http://localhost:9010";
                        String driverPath = "drivers/windows/chromedriver.exe";
                        WebDriver driver;

                        @BeforeClass
                        public void setUp() throws InterruptedException {

                                // step2: set system properties for selenium dirver
                                System.setProperty("webdriver.chromedriver.driver", driverPath);

                                // step3: instantiate selenium webdriver
                                driver = new ChromeDriver();

                                // step4: launch browser
                                driver.get(siteUrl);

                                Thread.sleep(2000);
                        }

                        @AfterClass
                        public void cleanUp() {
                                driver.quit();  // the quit() method closes all browser windows
and ends the WebDriver session.
                                // driver.close(); // the close() closes only the current window
on which Selenium is running automated tests.The WebDriver session, however, remains active.
                        }

                        @Test(priority = 1, description = "Add product to cart page Test ")


                        public void testRegisterPage() throws InterruptedException {

                                WebElement link =
driver.findElement(By.xpath("/html/body/div[2]/form/a"));
                                link.click();

                                WebElement searchbox1 = driver.findElement(By.id("name"));
                                searchbox1.sendKeys("Prajwal.Diwakar");

                                WebElement searchbox2 = driver.findElement(By.id("email"));
                                searchbox2.sendKeys("prajwal.diwakar@mphasis.com");

                                WebElement searchbox3 = driver.findElement(By.id("password"));
                                searchbox3.sendKeys("prajwal@123");

                                WebElement register =
driver.findElement(By.xpath("/html/body/div[2]/form/button"));
                                register.submit();

                                // add delay
                                Thread.sleep(2000);
```

```
                String expectedTitle = "";
                String actualTitle = driver.getTitle();
                assertEquals(actualTitle, expectedTitle);
        }


        @Test(priority = 2, description= "Add product to cart page Test")

        public void testCartPage() throws InterruptedException {

                // add delay
                Thread.sleep(2000);


         WebElement addcart1 =
driver.findElement(By.xpath("//*[@id=\"cart101\"]"));
         ((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", addcart1);

       // add delay
                Thread.sleep(2000);

         addcart1.click();

       // add delay
                Thread.sleep(2000);


driver.findElement(By.xpath("//*[@id=\"mynavbar\"]/ul/li[1]/a")).click();
       // add delay
                Thread.sleep(2000);

         WebElement addcart2 =
driver.findElement(By.xpath("//*[@id=\"cart102\"]"));
         ((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", addcart2);

            // add delay
                    Thread.sleep(2000);

         addcart2.click();

       // add delay
                Thread.sleep(2000);

         }
}
```

**Place order Page:**

```
package MphasisATE_CapstoneProject_Selenium_TestNG_Medicare;

import static org.testng.Assert.assertEquals;

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

```
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;


public class MedicarePlaceOrderPage {

		// step1: formulate a test domain url & driver path
				String siteUrl = "http://localhost:9010";
				String driverPath = "drivers/windows/chromedriver.exe";
				WebDriver driver;

				@BeforeClass
				public void setUp() throws InterruptedException {

					// step2: set system properties for selenium
dirver

	System.setProperty("webdriver.chromedriver.driver", driverPath);

					// step3: instantiate selenium webdriver
					driver = new ChromeDriver();

					// step4: launch browser
					driver.get(siteUrl);

					Thread.sleep(2000);
				}

				@AfterClass
				public void cleanUp() {
					driver.quit();  // the quit() method closes all
browser windows and ends the WebDriver session.
					// driver.close(); // the close() closes only the
current window on which Selenium is running automated tests.The WebDriver session, however,
remains active.
				}

				@Test(priority = 1, description = "Test Medicare Register
Page ")


				public void testRegisterPage() throws InterruptedException
{

					WebElement link =
driver.findElement(By.xpath("/html/body/div[2]/form/a"));

					link.click();
					WebElement searchbox1 =
driver.findElement(By.id("name"));
					searchbox1.sendKeys("Prajwal.Diwakar");

					WebElement searchbox2 =
driver.findElement(By.id("email"));

	searchbox2.sendKeys("prajwal.diwakar@mphasis.com");

					WebElement searchbox3 =
driver.findElement(By.id("password"));
					searchbox3.sendKeys("prajwal@123");
```

```
                                        WebElement register =
driver.findElement(By.xpath("/html/body/div[2]/form/button"));
                                        register.submit();

                                        // add delay
                                        Thread.sleep(2000);

                                        String url= driver.getCurrentUrl();
                                        assertEquals("http://localhost:9010/register-
user", url);
                                }

                                @Test(priority = 2, description= "Add product to cart page
Test")

                                public void testCartPage() throws InterruptedException {

                                        // add delay
                                        Thread.sleep(2000);


                                 WebElement addcart1 =
driver.findElement(By.xpath("//*[@id=\"cart101\"]"));
                                ((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", addcart1);

                               // add delay
                                        Thread.sleep(2000);

                                 addcart1.click();

                               // add delay
                                        Thread.sleep(2000);


driver.findElement(By.xpath("//*[@id=\"mynavbar\"]/ul/li[1]/a")).click();
                               // add delay
                                        Thread.sleep(2000);

                                 WebElement addcart2 =
driver.findElement(By.xpath("//*[@id=\"cart102\"]"));
                                ((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", addcart2);

                                    // add delay
                                            Thread.sleep(2000);

                                 addcart2.click();

                               // add delay
                                        Thread.sleep(2000);

                                }

                                @Test(priority = 3, description= "Place order for the
product in the cart")

                                public void PlaceOrder() throws InterruptedException {
```

```
        driver.findElement(By.xpath("//*[@id=\"mynavbar\"]/ul/li[1]/a")).click();  //home
                                        // add delay
                                        Thread.sleep(2000);


        driver.findElement(By.xpath("//*[@id=\"mynavbar\"]/ul/li[1]/a")).click();  //cart

                                        // add delay
                                        Thread.sleep(2000);

                                WebElement placeorder= driver.findElement(By.id("place-
order"));  //place order
                                ((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", placeorder);

                                    // add delay
                                        Thread.sleep(2000);
                                    placeorder.click();

                                    String url= driver.getCurrentUrl();
                                        assertEquals("http://localhost:9010/place-order",
url);
                                    }
}
```

**Search Product:**

```
package MphasisATE_CapstoneProject_Selenium_TestNG_Medicare;

import static org.testng.Assert.assertEquals;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;


public class MedicareSearchProduct {

        // step1: formulate a test domain url & driver path
                            String siteUrl = "http://localhost:9010";
                            String driverPath = "drivers/windows/chromedriver.exe";
                            WebDriver driver;

                            @BeforeClass
                            public void setUp() throws InterruptedException {

                                    // step2: set system properties for selenium dirver
                                    System.setProperty("webdriver.chromedriver.driver",
driverPath);


                                    // step3: instantiate selenium webdriver
                                    driver = new ChromeDriver();

                                    // step4: launch browser
                                    driver.get(siteUrl);

                                    Thread.sleep(2000);
```

```
	}

	@AfterClass
	public void cleanUp() {
		driver.quit();  // the quit() method closes all browser windows and ends the WebDriver session.
		// driver.close(); // the close() closes only the current window on which Selenium is running automated tests.The WebDriver session, however, remains active.
	}

	@Test(priority = 1, description = "Test Medicare Register Page ")


	public void testRegisterPage() throws InterruptedException {

		WebElement link = driver.findElement(By.xpath("/html/body/div[2]/form/a"));

		link.click();
		WebElement searchbox1 = driver.findElement(By.id("name"));
		searchbox1.sendKeys("Prajwal.Diwakar");

		WebElement searchbox2 = driver.findElement(By.id("email"));
		searchbox2.sendKeys("prajwal.diwakar@mphasis.com");

		WebElement searchbox3 = driver.findElement(By.id("password"));
		searchbox3.sendKeys("prajwal@123");

		WebElement register = driver.findElement(By.xpath("/html/body/div[2]/form/button"));
		register.submit();

		// add delay
		Thread.sleep(2000);

		String expectedTitle = "";
		String actualTitle = driver.getTitle();
		assertEquals(actualTitle, expectedTitle);
	}
	@Test(priority = 2, description= "Search the product in the search bar")


	public void Searchtheproduct() throws InterruptedException {
		// add delay
		Thread.sleep(2000);

		driver.findElement(By.id("search-product")).sendKeys("Limcee Chewable Tablet Orange");
		// add delay
		Thread.sleep(2000);

		driver.findElement(By.id("search-product-button")).click();
		// add delay
		Thread.sleep(2000);

	}
```

```
}
```

**Filter Products:**

```
package MphasisATE_CapstoneProject_Selenium_TestNG_Medicare;

import static org.testng.Assert.assertEquals;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;


public class MedicareFilterProduct {

	// step1: formulate a test domain url & driver path
	String siteUrl = "http://localhost:9010";
	String driverPath = "drivers/windows/chromedriver.exe";
	WebDriver driver;

	@BeforeClass
	public void setUp() throws InterruptedException {

		// step2: set system properties for selenium dirver
		System.setProperty("webdriver.chromedriver.driver", driverPath);

		// step3: instantiate selenium webdriver
		driver = new ChromeDriver();

		// step4: launch browser
		driver.get(siteUrl);

		Thread.sleep(2000);
	}

	@AfterClass
	public void cleanUp() {
		driver.quit();  // the quit() method closes all browser windows and ends the
WebDriver session.
		// driver.close(); // the close() closes only the current window on which
Selenium is running automated tests.The WebDriver session, however, remains active.
	}

                                    @Test(priority = 1, description = "Filter the products")


                                    public void testRegisterPage() throws InterruptedException
{
```

```
                                        WebElement link =
driver.findElement(By.xpath("/html/body/div[2]/form/a"));

                                        link.click();
                                        WebElement searchbox1 =
driver.findElement(By.id("name"));
                                        searchbox1.sendKeys("Prajwal.Diwakar");

                                        WebElement searchbox2 =
driver.findElement(By.id("email"));

        searchbox2.sendKeys("prajwal.diwakar@mphasis.com");

                                        WebElement searchbox3 =
driver.findElement(By.id("password"));
                                        searchbox3.sendKeys("prajwal@123");

                                        WebElement register =
driver.findElement(By.xpath("/html/body/div[2]/form/button"));
                                        register.submit();

                                        // add delay
                                        Thread.sleep(2000);

                                        String expectedTitle = "";
                                        String actualTitle = driver.getTitle();
                                        assertEquals(actualTitle, expectedTitle);
                                }

                                @Test(priority = 2, description= "Filter the products")

                                public void Filtertheproduct() throws InterruptedException
{
                                // add delay
                                        Thread.sleep(2000);

                                WebElement filter1 = driver.findElement(By.id("filter-
button"));
                                ((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", filter1);

                                // add delay
                                Thread.sleep(2000);

                                filter1.click();

                                // add delay
                                Thread.sleep(2000);

                                driver.findElement(By.id("1th")).click();

                                // add delay
                                Thread.sleep(2000);

                                String url= driver.getCurrentUrl();
                                assertEquals("http://localhost:9010/search-
product?name=0", url);


                                // add delay
                                Thread.sleep(2000);
```

```
        driver.findElement(By.xpath("//*[@id=\"mynavbar\"]/ul/li[1]/a")).click();  //home

                                    // add delay
                                        Thread.sleep(2000);

                                        WebElement filter2 = driver.findElement(By.id("filter-
button"));
                                        ((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", filter2);

                                    // add delay
                                        Thread.sleep(2000);

                                        filter2.click();

                                        // add delay
                                        Thread.sleep(2000);

                                        driver.findElement(By.id("htl")).click();

                                        String url2= driver.getCurrentUrl();
                                        assertEquals("http://localhost:9010/search-
product?name=1", url2);


                                        }

}
```

## Cucumber Test for the API endpoints

**Get products:**

```
#Author: prajwal.diwakar@your.domain.com
Feature: Testing the medicare web application API.
        User is testing the API of the web appication to get the list of all the products.

  @Restfull @smoketest
  Scenario: Retrieve the list of all products in the store
    Given user wants to get the list of all products in the store
```

```
    When User executes the endpoint url of the get product page
    Then user gets the all the products in the store.
    Then the request response has a 200 response code
```

```
package StepDefinations;

import static org.hamcrest.CoreMatchers.equalTo;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.restassured.RestAssured;
import io.restassured.response.Response;
import io.restassured.response.ValidatableResponse;
import io.restassured.specification.RequestSpecification;

public class GetProductStep {

	private RequestSpecification request;
	private Response response;
	private ValidatableResponse json;


	private String BASE_URL = "http://localhost:9010";

	@Given("user wants to get the list of all products in the store")
	public void user_wants_to_get_the_list_of_all_products_in_the_store() {
		request = RestAssured.given().baseUri(BASE_URL);

	}

	@When("User executes the endpoint url of the get product page")
	public void user_executes_the_endpoint_url_of_the_get_product_page() {
		response=request.when().get("/get-products");
	}

	@Then("user gets the all the products in the store.")
	public void user_gets_the_all_the_products_in_the_store() {

		int code=101;
		String message="12 Products Fetched Successfully.";

		json= response.then()
				.and().assertThat()
				.body("code", equalTo(code)).and()
				.body("message", equalTo(message));

	}

	@Then("the request response has a {int} response code")
	public void the_request_response_has_a_response_code(Integer statusCode) {
		response.then().statusCode(statusCode);
	}



}
```

**Get registered users:**

```
#Author: prajwal@gmail.com
Feature: Testing the medicare web application API.
        User is testing the API of the web appication to get the list of all the registered
users.


  Scenario: Retrieve the list of all registered users
    Given user wants to get the list of all regsitered users
    When User executes the endpoint url of the get user page
    Then user gets the all the registered users.
    Then validate the outcomes status code is 200
```

```
package StepDefinations;

import static org.hamcrest.CoreMatchers.equalTo;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.restassured.RestAssured;
import io.restassured.response.Response;
import io.restassured.response.ValidatableResponse;
import io.restassured.specification.RequestSpecification;

public class GetRegisteredUser {

	private RequestSpecification request;
	private Response response;
	private ValidatableResponse json;


	private String BASE_URL = "http://localhost:9010";

	@Given("user wants to get the list of all regsitered users")
	public void user_wants_to_get_the_list_of_all_regsitered_users() {
		request = RestAssured.given().baseUri(BASE_URL);
	}

	@When("User executes the endpoint url of the get user page")
	public void user_executes_the_endpoint_url_of_the_get_user_page() {
		response=request.when().get("/get-users");
	}

	@Then("user gets the all the registered users.")
	public void user_gets_the_all_the_registered_users() {
		int code=101;
		String message="4 Users Fetched Successfully.";

		json= response.then()
				.and().assertThat()
				.body("code", equalTo(code)).and()
				.body("message", equalTo(message));
	}

	@Then("validate the outcomes status code is {int}")
	public void validate_the_outcomes_status_code_is(Integer statusCode) {
		response.then().statusCode(statusCode);
	}

}
```

**Add Product:**

```
#Author: prajwal@gmail.com
Feature: Testing the medicare web application API.
        User is testing the API of the web appication to add the products.

  Scenario: Add the product
    Given user wants to add the products
    When User executes the endpoint url of the adding product page
    Then user adds the products.
    Then validate the outcomes of status code is 200
```

```
package StepDefinations;

import static org.hamcrest.CoreMatchers.equalTo;

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.response.Response;

import io.restassured.response.ValidatableResponse;

import io.restassured.specification.RequestSpecification;

public class PostAddProduct {

        private RequestSpecification request;

        private Response response;

        private ValidatableResponse json;

        private String BASE_URL = "http://localhost:9010";



        @Given("user wants to add the products")

        public void user_wants_to_add_the_products() {

                request = RestAssured.given().baseUri(BASE_URL);

        }
```

```
	@When("User executes the endpoint url of the adding product page")
	public void user_executes_the_endpoint_url_of_the_adding_product_page() {

		String requestBody="{\"id\": 999,\"image\": \"1.png\", \"name\": \"Disprin\",\"brand\": \"BZ Medico\", " +
				"\"status\": 1, \"price\": 100}";

		response =  request.contentType(ContentType.JSON).body(requestBody)
				.when().post("/add-product");
	}

	@Then("user adds the products.")
	public void user_adds_the_products() {
		json= response.then()
				.and().assertThat()
				.body("products.id", equalTo(999)).and()
				.body("products.name", equalTo("Disprin"));
	}

	@Then("validate the outcomes of status code is {int}")
	public void validate_the_outcomes_of_status_code_is(Integer statusCode) {
		response.then().statusCode(statusCode);;
	}

}
```

**Update product page**

```
#Author: prajwal@gmail.com
Feature: Testing the medicare web application API.
        User is testing the API of the web appication to update the products.

  Scenario: update the product
    Given user wants to update the products
    When User executes the endpoint url of the update product page
    Then user updates the products.
    Then user validates the outcomes of status code is 200
```

```
package StepDefinations;

import static org.hamcrest.CoreMatchers.equalTo;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.restassured.RestAssured;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import io.restassured.response.ValidatableResponse;
import io.restassured.specification.RequestSpecification;

public class PutUpdateProductStep {

        private RequestSpecification request;
        private Response response;
        private ValidatableResponse json;


        private String BASE_URL = "http://localhost:9010";

        @Given("user wants to update the products")
        public void user_wants_to_update_the_products() {
                request = RestAssured.given().baseUri(BASE_URL);
        }

        @When("User executes the endpoint url of the update product page")
        public void user_executes_the_endpoint_url_of_the_update_product_page() {
                String requestBody="{\"id\": 999,\"image\": \"2.png\", \"name\":
\"Disprin\",\"brand\": \"BZ Medico\", " +
                  "\"status\": 1, \"price\": 120}";

                response =  request.contentType(ContentType.JSON).body(requestBody)
                                .when().put("update-product");
        }

        @Then("user updates the products.")
        public void user_updates_the_products() {
                json= response.then()
                                .and().assertThat()
                                .body("product.id", equalTo(999)).and()
                                .body("product.image", equalTo("2.png")).and()
                                .body("product.price", equalTo(120)).and()
                                .body("product.name", equalTo("Disprin"));
        }

        @Then("user validates the outcomes of status code is {int}")
        public void user_validates_the_outcomes_of_status_code_is(Integer statusCode) {
                response.then().statusCode(statusCode);;
        }
```

```
}
```

**Update status of product.**

```
#Author: prajwal@gmail.com
Feature: Testing the medicare web application API.
        User is testing the API of the web appication to update status of the products.

  Scenario: Update status of the product
    Given user wants to Update status of the product
    When User executes the endpoint url of the update product status
    Then user updates the product status.
    Then user validating the outcomes of status code is 200
```

```
package StepDefinations;

import static org.hamcrest.CoreMatchers.equalTo;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.restassured.RestAssured;
import io.restassured.http.ContentType;
import io.restassured.response.Response;
import io.restassured.response.ValidatableResponse;
import io.restassured.specification.RequestSpecification;

public class PutUpdateStatusProductStep {

        private RequestSpecification request;
        private Response response;
        private ValidatableResponse json;


        private String BASE_URL = "http://localhost:9010";


                @Given("user wants to Update status of the product")
                public void user_wants_to_update_status_of_the_product() {
                request = RestAssured.given().baseUri(BASE_URL);
                }

                @When("User executes the endpoint url of the update product status")
                public void user_executes_the_endpoint_url_of_the_update_product_status() {
                        String requestBody="{\"id\": 999,\"image\": \"2.png\", \"name\":
\"Disprin\",\"brand\": \"BZ Medico\", " +
                                "\"status\": 0, \"price\": 120}";

                        response =  request.contentType(ContentType.JSON).body(requestBody)
                                        .when().put("update-product-status");
                }

                @Then("user updates the product status.")
                public void user_updates_the_product_status() {
                        json= response.then()
                                        .and().assertThat()
                                        .body("product.id", equalTo(999)).and()
                                        .body("product.image", equalTo("2.png")).and()
                                        .body("product.price", equalTo(120)).and()
                                        .body("product.name", equalTo("Disprin")).and()
                                        .body("product.status", equalTo(0));
```

```
	}

	@Then("user validating the outcomes of status code is {int}")
	public void user_validating_the_outcomes_of_status_code_is(Integer statusCode) {
		response.then().statusCode(statusCode);
	}

}
```

**Delete page:**

```
#Author: prajwal@gmail.com
Feature: Testing the medicare web application API.
        User is testing the API of the web appication to delete the products.

  Scenario: Delete the product
    Given user wants to delete the products
    When User executes the endpoint url of the delete product page
    Then user deletes the products.
    Then user will validate the outcomes of status code is 200
```

```
package StepDefinations;

import static org.hamcrest.CoreMatchers.equalTo;




import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.response.Response;

import io.restassured.response.ValidatableResponse;

import io.restassured.specification.RequestSpecification;



public class DeleteProductStep {



	private RequestSpecification request;

	private Response response;
```

```
private ValidatableResponse json;


private String BASE_URL = "http://localhost:9010";


@Given("user wants to delete the products")
public void user_wants_to_delete_the_products() {
        request = RestAssured.given().baseUri(BASE_URL);
}


@When("User executes the endpoint url of the delete product page")
public void user_executes_the_endpoint_url_of_the_delete_product_page() {
        response =  request
                        .when().delete("delete-product?id=101");
}


@Then("user deletes the products.")
public void user_deletes_the_products() {


        int code=101;
        String message="Product with ID 101 Deleted Successfully.";


        json= response.then()
                        .and().assertThat()
                        .body("code", equalTo(code)).and()
                        .body("message", equalTo(message));
```

```
        }

        @Then("user will validate the outcomes of status code is {int}")

        public void user_will_validate_the_outcomes_of_status_code_is(Integer statusCode) {

                response.then().statusCode(statusCode);

        }

}
```

## Postman:

### Collection file:


```
{
    "info": {
        "_postman_id": "86c6e9bd-d03a-4000-9900-6b4491768b62",
        "name": "Testing_Framework_for_the_Medicare_Website",
        "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
        "_exporter_id": "32291674",
        "_collection_link": "https://restless-desert-
470943.postman.co/workspace/Mphasis-ATE-Capstone-Project-Me~6b6bae50-d97e-40a9-9280-
d5dbecce0cad/collection/32291674-86c6e9bd-d03a-4000-9900-
6b4491768b62?action=share&source=collection_link&creator=32291674"
    },
    "item": [
        {
            "name": "Retrieve the list of all products in the store",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "exec": [
                            "pm.test(\"Status code is 200\", function () {\r",
                            "    pm.response.to.have.status(200);\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Response time is less than 200ms\", function
() {\r",
                            "
 pm.expect(pm.response.responseTime).to.be.below(200);\r",
                            "});\r",
```

```
                            "pm.test(\"Response matches these values\", function ()
{\r",
                            "    var jsonData = pm.response.json();\r",
                            "    pm.expect(jsonData.code).to.eql(101);\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Body contains these string\", function ()
{\r",
                            "
 pm.expect(pm.response.text()).to.include(\"id\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"image\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"name\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"category\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"brand\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"status\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"price\");\r",
                            "});"
                        ],
                        "type": "text/javascript"
                    }
                }
            ],
            "request": {
                "method": "GET",
                "header": [],
                "url": {
                    "raw": "{{BASE_URL}}/get-products",
                    "host": [
                        "{{BASE_URL}}"
                    ],
                    "path": [
                        "get-products"
                    ]
                }
            },
            "response": []
        },
```

```
        {
            "name": "Retrieve the list of all registered users",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "exec": [
                            "pm.test(\"Status code is 200\", function () {\r",
                            "    pm.response.to.have.status(200);\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Response time is less than 200ms\", function
() {\r",
                            "
 pm.expect(pm.response.responseTime).to.be.below(200);\r",
                            "});\r",
                            "pm.test(\"Responsematches these values\", function ()
{\r",
                            "    var jsonData = pm.response.json();\r",
                            "    pm.expect(jsonData.code).to.eql(101);\r",
                            "    pm.expect(jsonData.message).to.eql(\"4 Users Fetched
Successfully.\");\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Body contains these string\", function ()
{\r",
                            "
 pm.expect(pm.response.text()).to.include(\"name\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"email\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"password\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"type\");\r",
                            "});"
                        ],
                        "type": "text/javascript"
                    }
                }
            ],
            "request": {
                "method": "GET",
                "header": [],
```

```
                "url": {
                    "raw": "{{BASE_URL}}/get-users",
                    "host": [
                        "{{BASE_URL}}"
                    ],
                    "path": [
                        "get-users"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "Add the product",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "exec": [
                            "pm.test(\"Status code is 200\", function () {\r",
                            "    pm.response.to.have.status(200);\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Response time is less than 200ms\", function
() {\r",
                            "
 pm.expect(pm.response.responseTime).to.be.below(200);\r",
                            "});\r",
                            "pm.test(\"Responsematches these values\", function ()
{\r",
                            "    var jsonData = pm.response.json();\r",
                            "    pm.expect(jsonData.code).to.eql(101);\r",
                            "    pm.expect(jsonData.message).to.eql(\"Disprin Added
Successfully.\");\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Body contains these string\", function ()
{\r",
                            "
 pm.expect(pm.response.text()).to.include(\"id\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"image\");\r",
```

```
                            "
pm.expect(pm.response.text()).to.include(\"name\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"category\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"brand\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"status\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"price\");\r",
                            "});"
                        ],
                        "type": "text/javascript"
                    }
                }
            ],
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\r\n        \"id\": 999,\r\n        \"image\":
\"1.png\",\r\n        \"name\": \"Disprin\",\r\n        \"category\":
\"medicine\",\r\n        \"brand\": \"BZ Medico\",\r\n        \"status\": 1,\r\n
   \"price\": 100\r\n}\r\n",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "{{BASE_URL}}/add-product",
                    "host": [
                        "{{BASE_URL}}"
                    ],
                    "path": [
                        "add-product"
                    ]
                }
            },
            "response": []
        },
```

```
        {
            "name": "Delete the product",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "exec": [
                            "pm.test(\"Status code is 200\", function () {\r",
                            "    pm.response.to.have.status(200);\r",
                            "});\r",
                            "pm.test(\"Response time is less than 200ms\", function
() {\r",
                            "
 pm.expect(pm.response.responseTime).to.be.below(200);\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Body matches string\", function () {\r",
                            "
 pm.expect(pm.response.text()).to.include(\"code\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"message\");\r",
                            ";\r",
                            "});\r",
                            "pm.test(\"Response body values\", function () {\r",
                            "    var jsonData = pm.response.json();\r",
                            "    pm.expect(jsonData.code).to.eql(101)\r",
                            "    pm.expect(jsonData.message).to.eql(\"Product with ID
101 Deleted Successfully.\")\r",
                            "});"
                        ],
                        "type": "text/javascript"
                    }
                }
            ],
            "request": {
                "method": "DELETE",
                "header": [],
                "url": {
                    "raw": "{{BASE_URL}}/delete-product?id=101",
                    "host": [
                        "{{BASE_URL}}"
                    ],
                    "path": [
```

```
                        "delete-product"
                    ],
                    "query": [
                        {
                            "key": "id",
                            "value": "101"
                        }
                    ]
                }
            },
            "response": []
        },
        {
            "name": "Update the product",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "exec": [
                            "pm.test(\"Status code is 200\", function () {\r",
                            "    pm.response.to.have.status(200);\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Response time is less than 200ms\", function
() {\r",
                            "
 pm.expect(pm.response.responseTime).to.be.below(200);\r",
                            "});\r",
                            "pm.test(\"Responsematches these values\", function ()
{\r",
                            "    var jsonData = pm.response.json();\r",
                            "    pm.expect(jsonData.code).to.eql(101);\r",
                            "    pm.expect(jsonData.message).to.eql(\"Disprin+
Updated Successfully.\");\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Body contains these string\", function ()
{\r",
                            "
 pm.expect(pm.response.text()).to.include(\"id\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"image\");\r",
```

```
                            "
pm.expect(pm.response.text()).to.include(\"name\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"category\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"brand\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"status\");\r",
                            "
pm.expect(pm.response.text()).to.include(\"price\");\r",
                            "});"
                        ],
                        "type": "text/javascript"
                    }
                }
            ],
            "request": {
                "method": "PUT",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\r\n        \"id\": 999,\r\n        \"image\":
\"2.png\",\r\n        \"name\": \"Disprin+\",\r\n        \"category\":
\"medicine\",\r\n        \"brand\": \"BZ Medico\",\r\n        \"status\": 1,\r\n
   \"price\": 120\r\n}\r\n",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "{{BASE_URL}}/update-product",
                    "host": [
                        "{{BASE_URL}}"
                    ],
                    "path": [
                        "update-product"
                    ]
                }
            },
            "response": []
        },
```

```
        {
            "name": "Update the product status",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "exec": [
                            "pm.test(\"Status code is 200\", function () {\r",
                            "    pm.response.to.have.status(200);\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Response time is less than 200ms\", function
() {\r",
                            "
 pm.expect(pm.response.responseTime).to.be.below(200);\r",
                            "});\r",
                            "pm.test(\"Responsematches these values\", function ()
{\r",
                            "    var jsonData = pm.response.json();\r",
                            "    pm.expect(jsonData.code).to.eql(101);\r",
                            "    pm.expect(jsonData.message).to.eql(\"Disprin+ Status
Updated Successfully.\");\r",
                            "});\r",
                            "\r",
                            "pm.test(\"Body contains these string\", function ()
{\r",
                            "
 pm.expect(pm.response.text()).to.include(\"id\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"image\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"name\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"category\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"brand\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"status\");\r",
                            "
 pm.expect(pm.response.text()).to.include(\"price\");\r",
                            "});"
                        ],
                        "type": "text/javascript"
```

```
                    }
                }
            ],
            "request": {
                "method": "PUT",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\r\n        \"id\": 999,\r\n        \"image\":
\"2.png\",\r\n        \"name\": \"Disprin+\",\r\n        \"category\":
\"medicine\",\r\n        \"brand\": \"BZ Medico\",\r\n        \"status\": 0,\r\n
   \"price\": 120\r\n}\r\n",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "{{BASE_URL}}/update-product-status",
                    "host": [
                        "{{BASE_URL}}"
                    ],
                    "path": [
                        "update-product-status"
                    ]
                }
            },
            "response": []
        }
    ]
}
```

**Environment file:**

```
{
    "id": "f501fd68-400d-4fe5-aa38-4627f42bf634",
    "name": "QA_Environment",
    "values": [
        {
            "key": "BASE_URL",
            "value": "http://localhost:9010",
            "type": "default",
            "enabled": true
```

```
        }
    ],
    "_postman_variable_scope": "environment",
    "_postman_exported_at": "2024-01-27T06:36:03.498Z",
    "_postman_exported_using": "Postman/10.22.6"
}
```

## JMeter:

**<jmeterTestPlan version="1.2" properties="5.0" jmeter="5.6.2">**

**<hashTree>**

**<TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test Plan" enabled="true">**

**<boolProp name="TestPlan.functional_mode">false</boolProp>**

**<boolProp name="TestPlan.tearDown_on_shutdown">false</boolProp>**

**<boolProp name="TestPlan.serialize_threadgroups">false</boolProp>**

**<elementProp name="TestPlan.user_defined_variables" elementType="Arguments" guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">**

**<collectionProp name="Arguments.arguments"/>**

**</elementProp>**

**</TestPlan>**

**<hashTree>**

**<CookieManager guiclass="CookiePanel" testclass="CookieManager" testname="HTTP Cookie Manager" enabled="true">**

**<collectionProp name="CookieManager.cookies"/>**

**<boolProp name="CookieManager.clearEachIteration">true</boolProp>**

**<boolProp name="CookieManager.controlledByThreadGroup">false</boolProp>**

**</CookieManager>**

**<hashTree/>**

<CacheManager guiclass="CacheManagerGui" testclass="CacheManager" testname="HTTP Cache Manager" enabled="true">

<boolProp name="clearEachIteration">true</boolProp>

<boolProp name="useExpires">true</boolProp>

<boolProp name="CacheManager.controlledByThread">false</boolProp>

</CacheManager>

<hashTree/>

<Arguments guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="BASE_URL" elementType="Argument">

<stringProp name="Argument.name">BASE_URL</stringProp>

<stringProp name="Argument.value">localhost</stringProp>

<stringProp name="Argument.metadata">=</stringProp>

</elementProp>

</collectionProp>

</Arguments>

<hashTree/>

<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Thread Group" enabled="true">

<stringProp name="ThreadGroup.on_sample_error">continue</stringProp>

<elementProp name="ThreadGroup.main_controller" elementType="LoopController" guiclass="LoopControlPanel" testclass="LoopController" testname="Loop Controller" enabled="true">

<stringProp name="LoopController.loops">1</stringProp>

<boolProp name="LoopController.continue_forever">false</boolProp>

</elementProp>

<stringProp name="ThreadGroup.num_threads">1</stringProp>

<stringProp name="ThreadGroup.ramp_time">1</stringProp>

<boolProp name="ThreadGroup.delayedStart">false</boolProp>

<boolProp name="ThreadGroup.scheduler">false</boolProp>

<stringProp name="ThreadGroup.duration"/>

<stringProp name="ThreadGroup.delay"/>

<boolProp name="ThreadGroup.same_user_on_next_iteration">true</boolProp>

</ThreadGroup>

<hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Login page HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments"/>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">false</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Register Page HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments"/>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.path">/register</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Register user HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="name" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">name</stringProp>

<stringProp name="Argument.value">Admin</stringProp>

</elementProp>

<elementProp name="email" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">email</stringProp>

<stringProp name="Argument.value">admin@example.com</stringProp>

</elementProp>

<elementProp name="password" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">password</stringProp>

<stringProp name="Argument.value">admin123</stringProp>

</elementProp>

</collectionProp>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.path">/register-user</stringProp>

<stringProp name="HTTPSampler.method">POST</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare View Product 1 HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="id" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">id</stringProp>

<stringProp name="Argument.value">101</stringProp>

</elementProp>

</collectionProp>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.path">/view-product</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare View Product 2 HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="id" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">id</stringProp>

<stringProp name="Argument.value">102</stringProp>

</elementProp>

</collectionProp>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.path">/view-product</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare AddToCart 1 HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="id" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">id</stringProp>

<stringProp name="Argument.value">101</stringProp>

</elementProp>

</collectionProp>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.path">/add-to-cart</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare AddToCart 2 HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="id" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">id</stringProp>

<stringProp name="Argument.value">102</stringProp>

</elementProp>

</collectionProp>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.path">/add-to-cart</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Place order HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments"/>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.path">/place-order</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Home Page HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments"/>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.path">/home</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Filter Low to High HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="name" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">name</stringProp>

<stringProp name="Argument.value">0</stringProp>

</elementProp>

</collectionProp>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.path">/search-product</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">false</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer"
enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Filter High to Low HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments">

<elementProp name="name" elementType="HTTPArgument">

<boolProp name="HTTPArgument.always_encode">true</boolProp>

<stringProp name="Argument.metadata">=</stringProp>

<boolProp name="HTTPArgument.use_equals">true</boolProp>

<stringProp name="Argument.name">name</stringProp>

<stringProp name="Argument.value">1</stringProp>

</elementProp>

</collectionProp>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.path">/search-product</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">false</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Medicare Logout HTTP Request" enabled="true">

<boolProp name="HTTPSampler.postBodyRaw">false</boolProp>

<elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

<collectionProp name="Arguments.arguments"/>

</elementProp>

<stringProp name="HTTPSampler.domain">${BASE_URL}</stringProp>

<stringProp name="HTTPSampler.port">9010</stringProp>

<stringProp name="HTTPSampler.path">/logout</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.BROWSER_COMPATIBLE_MULTIPART">false</boolProp>

<boolProp name="HTTPSampler.image_parser">false</boolProp>

<boolProp name="HTTPSampler.concurrentDwn">false</boolProp>

<stringProp name="HTTPSampler.concurrentPool">6</stringProp>

<boolProp name="HTTPSampler.md5">false</boolProp>

<intProp name="HTTPSampler.ipSourceType">0</intProp>

</HTTPSamplerProxy>

<hashTree>

<ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer" enabled="true">

<stringProp name="ConstantTimer.delay">3000</stringProp>

</ConstantTimer>

<hashTree/>

</hashTree>

<ResultCollector guiclass="ViewResultsFullVisualizer" testclass="ResultCollector" testname="View Results Tree" enabled="true">

<boolProp name="ResultCollector.error_logging">false</boolProp>

<objProp>

<name>saveConfig</name>

<value class="SampleSaveConfiguration">

<time>true</time>

<latency>true</latency>

<timestamp>true</timestamp>

<success>true</success>

<label>true</label>

<code>true</code>

<message>true</message>

<threadName>true</threadName>

<dataType>true</dataType>

<encoding>false</encoding>

<assertions>true</assertions>

<subresults>true</subresults>

<responseData>false</responseData>

<samplerData>false</samplerData>

<xml>false</xml>

<fieldNames>true</fieldNames>

<responseHeaders>false</responseHeaders>

<requestHeaders>false</requestHeaders>

<responseDataOnError>false</responseDataOnError>

<saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMessage>

<assertionsResultsToSave>0</assertionsResultsToSave>

<bytes>true</bytes>

<sentBytes>true</sentBytes>

<url>true</url>

<threadCounts>true</threadCounts>

<idleTime>true</idleTime>

<connectTime>true</connectTime>

</value>

</objProp>

<stringProp name="filename"/>

</ResultCollector>

<hashTree/>

<ResultCollector guiclass="SummaryReport" testclass="ResultCollector" testname="Summary Report" enabled="true">

<boolProp name="ResultCollector.error_logging">false</boolProp>

<objProp>

<name>saveConfig</name>

<value class="SampleSaveConfiguration">

<time>true</time>

<latency>true</latency>

<timestamp>true</timestamp>

<success>true</success>

<label>true</label>

<code>true</code>

<message>true</message>

<threadName>true</threadName>

<dataType>true</dataType>

<encoding>false</encoding>

<assertions>true</assertions>

<subresults>true</subresults>

<responseData>false</responseData>

<samplerData>false</samplerData>

<xml>false</xml>

<fieldNames>true</fieldNames>

<responseHeaders>false</responseHeaders>

<requestHeaders>false</requestHeaders>

<responseDataOnError>false</responseDataOnError>

<saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMessage>

<assertionsResultsToSave>0</assertionsResultsToSave>

<bytes>true</bytes>

<sentBytes>true</sentBytes>

<url>true</url>

<threadCounts>true</threadCounts>

<idleTime>true</idleTime>

<connectTime>true</connectTime>

</value>

</objProp>

<stringProp name="filename"/>

</ResultCollector>

<hashTree/>

<ResultCollector guiclass="StatVisualizer" testclass="ResultCollector" testname="Aggregate Report" enabled="true">

<boolProp name="ResultCollector.error_logging">false</boolProp>

<objProp>

<name>saveConfig</name>

<value class="SampleSaveConfiguration">

<time>true</time>

<latency>true</latency>

<timestamp>true</timestamp>

<success>true</success>

<label>true</label>

<code>true</code>

<message>true</message>

<threadName>true</threadName>

<dataType>true</dataType>

<encoding>false</encoding>

<assertions>true</assertions>

<subresults>true</subresults>

<responseData>false</responseData>

<samplerData>false</samplerData>

<xml>false</xml>

<fieldNames>true</fieldNames>

<responseHeaders>false</responseHeaders>

<requestHeaders>false</requestHeaders>

<responseDataOnError>false</responseDataOnError>

<saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMessage>

<assertionsResultsToSave>0</assertionsResultsToSave>

<bytes>true</bytes>

<sentBytes>true</sentBytes>

<url>true</url>

<threadCounts>true</threadCounts>

<idleTime>true</idleTime>

<connectTime>true</connectTime>

</value>

</objProp>

<stringProp name="filename"/>

</ResultCollector>

<hashTree/>

<ResultCollector guiclass="TableVisualizer" testclass="ResultCollector" testname="View Results in Table" enabled="true">

<boolProp name="ResultCollector.error_logging">false</boolProp>

<objProp>

<name>saveConfig</name>

<value class="SampleSaveConfiguration">

<time>true</time>

<latency>true</latency>

<timestamp>true</timestamp>

<success>true</success>

<label>true</label>

<code>true</code>

<message>true</message>

<threadName>true</threadName>

<dataType>true</dataType>

<encoding>false</encoding>

<assertions>true</assertions>

<subresults>true</subresults>

<responseData>false</responseData>

<samplerData>false</samplerData>

<xml>false</xml>

<fieldNames>true</fieldNames>

<responseHeaders>false</responseHeaders>

<requestHeaders>false</requestHeaders>

<responseDataOnError>false</responseDataOnError>

<saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMessage>

<assertionsResultsToSave>0</assertionsResultsToSave>

<bytes>true</bytes>

<sentBytes>true</sentBytes>

<url>true</url>

<threadCounts>true</threadCounts>

<idleTime>true</idleTime>

<connectTime>true</connectTime>

</value>

</objProp>

<stringProp name="filename"/>

</ResultCollector>

<hashTree/>

</hashTree>

</hashTree>

</hashTree>

</jmeterTestPlan>