Terkwaz CodeWorkFlow Documentation

Code WorkFlow Documentation

PreRequisites :

1- Setting the environment :

- Set up an editor like " intellij"
- Put all dependencies and Run Configuration we will need in coding
- Refresh Maven
- We will use **SHAFT-ENGINE** ,so we have to take all instructions from the github repository " https://github.com/MohabMohie/SHAFT_ENGINE "

2- Create the pages for page object Model :

Main:

- Create HomePage
- Create searchPage
- Create FileUploadPage
- Create DynamicLoadingPage
- ApiObjectPage

Test :

- Create searchTests Page
- Create BaseTests Page (for herokuapp Tests)
- Create fileUploadTests Page
- Create DynamicLoadingTests Page
- Create APITests Page

Code WorkFlow For Herokuapp Task :

Following is a detailed description about the Execution of the source code workflow

Note: this code is written by POM Structure using TestNG and SHAFT ENGINE.

1- Setup code :

Here is the first step, where the setup() function, *in "BaseTest" class,* is called to set the required Test configurations, including:

Launching the Browser and Passing The url of "Required Website" using :

```
driver= DriverFactory.getDriver();
BrowserActions.navigateToURL(driver,"https://the-internet.
herokuapp.com/");
```

Making an instance from all Pages needed:

```
homepage = new HomePage(driver);
fileUploadPage=new FileUploadPage(driver);
dynamicLoadingPage=new DynamicLoadingPage(driver);
    }
```

2- Provide inputs and Getting Outputs:

- Here is the second step including :

clicklink() Method is used to Press on desired links to open certain pages followed by (`ClickFileUploadPage()& ClickDynamicLoadingPage()`) to open the desired pages.

```
private void clicklink(String linkText)
{
   ElementActions.click(driver,By.linkText(linkText));

}

public void ClickFileUploadPage()
{
   clicklink("File Upload");
}
public void ClickDynamicLoadingPage()
{
   clicklink("Dynamic Loading");
}
```

A new Pages will be displayed containing:

1. Dynamic LoadingPage:

   The page will contain Two Examples and we need to click on the second ExamplePage using ..

```
public void ClickDynamicLoadingExample2 (){
    ElementActions.click(driver,link_Example2);
```

   Then Press on Start Button using `clickStart()` Method

```
public void clickStart() {
    ElementActions.click(driver,startButton);
  }
```

   Then waiting for the loaded Text and get its text name using `getLoadedText()` Method

```
public String getLoadedText(){

WebDriverElementActions.waitForElementToBePresent(driver,l
oadedText,5,true);
    return    ElementActions.getText(driver,loadedText);
}
```

2. File UploadPage :

We will use some Methods to :

➔ Upload file using `FileUploadInput(String path)` Method, passing the relative path to it

```
public void FileUploadInput(String path) {
   ElementActions.typeFileLocationForUpload(driver,
choosefile, getabsoulutepath(path));

}
```

To get the Absolute path and using it for the upload method, use

```
public String getabsoulutepath(String path) {
   return FileActions.getAbsolutePath( path);
}
```

Then use `clickUpload()` to click Upload Button, and `getUploadedfilename()` to get the file name after uploading it.

```
public void clickUpload() {

   ElementActions.click(driver, Upload);
}

public String getUploadedfilename() {

WebDriverElementActions.waitForElementToBePresent(driver,
Text, 5, true);
   return ElementActions.getText(driver, Text);
}
```

3- Execution :

The previous workflow is executed as follow :

Using `getTextAfterLoading()` Method to :

1. **Click on "Dynamic LoadingPage" in `ClickDynamicLoadingPage()` Method .**
2. **Click on "Example 2" in `ClickDynamicLoadingExample2()` Method .**
3. **Click "Start" and Wait for loading to finish in `clickStart()` Method .**
4. **Check that the text displayed is "Hello World!"**

```
public void getTextAfterLoading()
{
    homepage.ClickDynamicLoadingPage();
    dynamicLoadingPage.ClickDynamicLoadingExample2();
    dynamicLoadingPage.clickStart();
  String Text= dynamicLoadingPage.getLoadedText();
Validations.assertThat().object(Text).isEqualTo("Hello
World!").perform();

}
```

Using `uploadFile()` Method to :

- **Click on "File Upload" in `ClickFileUploadPage()` Method.**
- **Upload any small image file in `FileUploadInput(relative path)` Method.**
- **Check that the file has been uploaded successfully by getting its name after uploading in `getUploadedfilename()` Method.**

```
public void uploadFile()
{
   homepage.ClickFileUploadPage();

fileUploadPage.FileUploadInput("src\\main\\resources\\10440964_1613
974605524418_5426798511651274387_n.jpg");

   fileUploadPage.clickUpload();

   fileUploadPage.getUploadedfilename();
}
```

Code WorkFlow For SearchTask :

Following is a detailed description about the Execution of the source code workflow

Note: this code is written by POM Structure using TestNG and SHAFT ENGINE.

1- Setup code :

Here is the first step, where the setup() function, *in "searchTests" class,* is called to set the required Test configurations, including:

Launching the Browser and Passing The url of "Required Website" using :

```
driver= DriverFactory.getDriver();
```

```
BrowserActions.navigateToURL(driver,"https://www.google.com/ncr);
```

Making an instance from all Pages needed:

```
SearchPage = new searchPage(driver);
```

2- Provide inputs and Getting Outputs:

- Here is the second step including :

Search for items we need ,using `searchforitems(String Text)` Method :

```
public void searchforitems(String Text)
    {
        new ElementActions(driver)
                .type( searchinput, Text)
                .keyPress(searchinput, Keys.ENTER);
    }
```

It will be redirected to the Result Page ,Then get the locate of third element to assert its presence using `GetThirdElement()` Method:

```
public By GetThirdElement()
{
    return result;
}
```

3- Execution :

The previous workflow is executed as follow :

Using `searchGoogle()` Method to :

1. Type in the query for search for `"selenium webdriver"`.
2. Verify that the result page contains the third link of text containing `'What is Selenium WebDriver?'`.

```
public void searchGoogle() {

    SearchPage.searchforitems("selenium webdriver");

    Validations.verifyThat()
            .element(driver, SearchPage.GetThirdElement())

            .exists().withCustomReportMessage("third result
text contains 'What is Selenium WebDriver?'")

            .perform();
}
```

Code WorkFlow For APITest Task:

Following is a detailed description about the Execution of the source code workflow

Note: this code is written by POM Structure using TestNG and SHAFT ENGINE.

1- Setup code :

-Here is the first step, where the setup() Method is called to set the required Test

configurations, including:

-Passing the URL(Which is provided in the APIObjectPage and =

"https://cat-fact.herokuapp.com/";)

where we will go to the desired baseLink .

-Making an instance from all Pages needed

```
public void Setup()
{
    baseurl= DriverFactory.getAPIDriver(APIObjectsPage.Base_Url);
    apiObjectsPage=new APIObjectsPage(baseurl);

}
```

2- Provide inputs and Getting Outputs:

- Here is the second step including :

Perform the response using `getcatFacts()` GET Method to get the Response Body

```
public Response getcatFacts()
{
   return apiObject.buildNewRequest("facts",
RestActions.RequestType.GET)
            .performRequest();
}
```

3- Execution :

The previous workflow is executed as follow :

Using `TestRandomCatFacts()` Method to

1. get the response body
2. get random JsonPath to get the “text” property
3. Check that “text” is not empty.

```
public void TestRandomCatFacts()
{
   Response res= apiObjectsPage.getcatFacts();

   String RandomNo = apiObjectsPage.getRandomNumber();

   String Text =
RestActions.getResponseJSONValue(res,"text["+RandomNo+"]");

 Validations.assertThat().object(Text).isNotNull().perform();

   System.out.println("Random No Choosen is:" + RandomNo +" And the
Text is: \n"+Text);


}
```