package com.globot.automation.interfaces.web;

import java.awt.RenderingHints.Key;

import java.io.File;

import java.io.IOException;

import java.lang.reflect.Constructor;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.net.URL;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

import java.util.concurrent.TimeUnit;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Unmarshaller;

import org.openqa.selenium.Alert;

import org.openqa.selenium.By;

import org.openqa.selenium.SearchContext;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebDriverBackedSelenium;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.android.AndroidDriver;

import org.openqa.selenium.chrome.ChromeDriverService;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.ie.InternetExplorerDriverService;

import org.openqa.selenium.remote.CapabilityType;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

import org.openqa.selenium.remote.service.DriverService;

import org.openqa.selenium.safari.SafariDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.Select;

import org.openqa.selenium.support.ui.WebDriverWait;

import com.globot.automation.data.schema.web.StepData;

import com.globot.automation.data.schema.web.WebSteps;

import com.globot.automation.i18n.Convert;

import com.globot.automation.interfaces.InterfaceC;

import com.globot.automation.logging.Logger;

import com.globot.automation.result.ConditionHandler;

import com.globot.automation.result.ExecutionResults;

import com.globot.automation.testcase.TestCaseHandler;

import com.globot.automation.tool.ToolVariables;

import com.opera.core.systems.OperaDriver;

import com.thoughtworks.selenium.Selenium;

public class WebSession extends InterfaceC {

private Selenium selenium = null;

private WebDriver driver = null;

private WebSteps unMarshelledActionFile=null;

private String pathOfActionFile=null;

private String actionName;

private TestCaseHandler tcHandler;

private DriverService service=null;

public String stepType=null;

public String getPathOfActionFile() {

return pathOfActionFile;

}

public void setPathOfActionFile(String pathOfActionFile) {

this.pathOfActionFile = pathOfActionFile;

}

public void unMarshel() {

try {

JAXBContext jaxbContext=JAXBContext.newInstance("com.globot.automation.data.schema.web");

Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();

unMarshelledActionFile=(WebSteps)jaxbUnmarshaller.unmarshal(new File(getPathOfActionFile()));

} catch(Exception e) {

e.printStackTrace();

unMarshelledActionFile=null;

}

}

@Override

public int setUp() {

try {

String browserName=tcHandler.isTcForParallel?tcHandler.browserName:ToolVariables.browserName;

if (browserName.equalsIgnoreCase("firefox")) {

if(tcHandler.isTcForParallel) {

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName("firefox");

driver = new RemoteWebDriver(new URL(tcHandler.parallelURL), capabilities);

} else {

driver = new FirefoxDriver();

driver.manage().timeouts().implicitlyWait(6000,TimeUnit.MILLISECONDS);

}

} else if (browserName.equalsIgnoreCase("google\_chrome")) {

if(tcHandler.isTcForParallel) {

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName("chrome");

driver = new RemoteWebDriver(new URL(tcHandler.parallelURL), capabilities);

} else {

service = new ChromeDriverService.Builder().usingDriverExecutable(new File(ToolVariables.chrome\_driverPath)).usingAnyFreePort().build();

try {

service.start();

} catch (IOException e) {

//what to do????

}

System.setProperty("webdriver.chrome.driver", ToolVariables.chrome\_driverPath);

driver = new RemoteWebDriver(service.getUrl(), DesiredCapabilities.chrome());

}

} else if (browserName.equalsIgnoreCase("internet\_explorer")) {

if(tcHandler.isTcForParallel) {

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName("");

driver = new RemoteWebDriver(new URL(tcHandler.parallelURL), capabilities);

} else {

service = new InternetExplorerDriverService.Builder().usingDriverExecutable(new File(ToolVariables.iEExecutable)).usingAnyFreePort().build();

try {

service.start();

} catch (IOException e) {

// TODO ????

}

System.setProperty("WebDriver.ie.driver",ToolVariables.iEExecutable);

driver = new RemoteWebDriver(service.getUrl(), DesiredCapabilities.internetExplorer());

}

}else if (browserName.equalsIgnoreCase("Safari")) {

if(tcHandler.isTcForParallel) {

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName("safari");

driver = new RemoteWebDriver(new URL(tcHandler.parallelURL), capabilities);

} else {

driver = new SafariDriver();

driver.manage().timeouts().implicitlyWait(6000,TimeUnit.MILLISECONDS);

}

}else if (browserName.equalsIgnoreCase("opera")) {

if(tcHandler.isTcForParallel) {

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName("opera");

driver = new RemoteWebDriver(new URL(tcHandler.parallelURL), capabilities);

} else {

driver = new OperaDriver();

driver.manage().timeouts().implicitlyWait(6000,TimeUnit.MILLISECONDS);

}

}else if (browserName.equalsIgnoreCase("android\_browser")) {

try {

System.out.println("cmd /c start "+System.getProperty("user.dir")+"\\resources\\AndroidBrowserCommands.bat");

Runtime.getRuntime().exec("cmd /c start "+System.getProperty("user.dir")+"\\resources\\AndroidBrowserCommands.bat");

} catch (IOException e) {

e.printStackTrace();

}

Thread.sleep(7000);

driver = new AndroidDriver();

}else if (browserName.equalsIgnoreCase("iOS\_browser")){

System.out.println("In IOS browser session");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(CapabilityType.BROWSER\_NAME, "iOS");

capabilities.setCapability(CapabilityType.VERSION, "6.1");

capabilities.setCapability("device", "iPhone Simulator");

capabilities.setCapability(CapabilityType.PLATFORM, "Mac");

capabilities.setCapability("app", "safari");

driver = new RemoteWebDriver(new URL("http://0.0.0.0:4723/wd/hub"), capabilities);

}

return ExecutionResults.PASS;

} catch (Exception e) {

Logger.logExecutionError("Error while launching the browser instance");

e.printStackTrace();

return ExecutionResults.FAIL;

}

}

@Override

public void preExecutionSetup(int executionType, String actionFilePath, String actionName, TestCaseHandler tcHandler) {

this.actionName=actionName;

this.tcHandler=tcHandler;

setPathOfActionFile(actionFilePath);

}

@Override

public int execute() {

//new

unMarshel();

List<StepData> stepData=unMarshelledActionFile.getStepData();

Iterator<StepData> i=stepData.iterator();

int l\_result=ExecutionResults.PASS;

while(i.hasNext() && l\_result==ExecutionResults.PASS) {

StepData s=i.next();

try {

Object o1=Class.forName("com.globot.automation.interfaces.web.WebSession").newInstance();

Class<?> c2=Class.forName("com.globot.automation.interfaces.web.WebSession$"+Character.toUpperCase(s.getType().charAt(0)) + s.getType().substring(1));

Constructor<?> cons2=c2.getConstructor(new Class[]{o1.getClass()});

Object o2=cons2.newInstance(new Object[]{o1});

Method m2=c2.getDeclaredMethod("execute", new Class[]{StepData.class, WebSession.class});

l\_result=(Integer)m2.invoke(o2,new Object[]{s, this});

//Class<Step> c=(Class<Step>) Class.forName("com.globot.automation.interface.web.WebSession");//$"+Character.toUpperCase(s.getType().charAt(0)) + s.getType().substring(1));

Object webSession = Class.forName("com.globot.automation.interfaces.web.WebSession").newInstance();

} catch (ClassNotFoundException e) {

e.printStackTrace();

return ExecutionResults.FAIL;

} catch (InstantiationException e) {

e.printStackTrace();

return ExecutionResults.FAIL;

} catch (IllegalAccessException e) {

e.printStackTrace();

return ExecutionResults.FAIL;

} catch (SecurityException e) {

e.printStackTrace();

return ExecutionResults.FAIL;

} catch (NoSuchMethodException e) {

e.printStackTrace();

return ExecutionResults.FAIL;

} catch (IllegalArgumentException e) {

e.printStackTrace();

return ExecutionResults.FAIL;

} catch (InvocationTargetException e) {

e.printStackTrace();

return ExecutionResults.FAIL;

}

}

if(l\_result==ExecutionResults.PASS) {

return ExecutionResults.PASS;

} else {

return ExecutionResults.FAIL;

}

}

@Override

public int tearDown() {

try {

driver.quit();

driver=null;

if (System.getProperty("os.name").indexOf("Windows") != -1){

Runtime.getRuntime().exec("taskkill /F /IM cmd.exe");

}

this.selenium.stop();

this.selenium=null;

if(service!=null) {

service.stop();

service=null;

}

return ExecutionResults.PASS;

} catch(Exception e) {

Logger.logExecutionError("Error while closing the browser instance");

return ExecutionResults.FAIL;

}

}

@Override

public void run() {

result=ExecutionResults.EXECUTING;

if(actionType==-1) {

return;

} else if(actionType==0) {

result=setUp();

} else if(actionType==2) {

result=tearDown();

} else if(actionType==1) {

result=execute();

}

if(isThread) {

//setting up execution for others from this thread

ConditionHandler c=tcHandler.actionConditionMap.get(actionName);

String nextActionName=null;

if(result==ExecutionResults.PASS) {

nextActionName=c.getPassActionName();

} else if(result==ExecutionResults.FAIL) {

nextActionName=c.getFailActionName();

}

if(nextActionName!=null && !nextActionName.equals("")) {

tcHandler.actionStatusMap.put(nextActionName, ExecutionResults.EXECUTING);

tcHandler.actionStatusMap.put(actionName, result);

tcHandler.execute(tcHandler.actionConditionMap.get(nextActionName), nextActionName);

}

}

}

interface Step {

public int execute(StepData s, WebSession w);

}

class Goto implements Step {

public Goto() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String url=s.getData().getUrl();

w.selenium = new WebDriverBackedSelenium(w.driver, url);

w.driver.get(url);

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class ClickButton implements Step {

public ClickButton() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

we.click();

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class KeyPressAndWait implements Step {

public KeyPressAndWait() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=Convert.getInstance().convertToi18n(s.getData().getElementValue());

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed. Element specified not found on page");

return ExecutionResults.FAIL;

}

we.sendKeys(org.openqa.selenium.Keys.ENTER);

Thread.currentThread().sleep(ToolVariables.executionSpeed);

}catch (Exception e) {

System.out.println(e);

Logger.logExecutionError(s.getDescription()+" failed.");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class ClickRadioButton implements Step {

public ClickRadioButton() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

w.selenium.check(s.getData().getElementType()+"="+s.getData().getElementValue()+" "+"value="+Convert.getInstance().convertToi18n(s.getData().getValue()));

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class ClickLink implements Step {

public ClickLink() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

//System.out.println(s.getData().getElementValue());

String stepValue=Convert.getInstance().convertToi18n(s.getData().getElementValue());

//System.out.println(stepValue);

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed. Element specified not found on page");

return ExecutionResults.FAIL;

}

we.click();

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class SelectOption implements Step {

public SelectOption() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=Convert.getInstance().convertToi18n(s.getData().getElementValue());

String value=s.getData().getValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed. Element specified not found on page");

return ExecutionResults.FAIL;

}

Select select = new Select(we);

System.out.println(select.getFirstSelectedOption().getText());

select.selectByVisibleText(value);

System.out.println(select.getFirstSelectedOption().getText());

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class SetCheckedBox implements Step {

public SetCheckedBox() { }

@SuppressWarnings("null")

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

String value=s.getData().getValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed. Element specified not found on page");

return ExecutionResults.FAIL;

}

//logic -> Value is !=null then only the below code will handle the selection of the particular checkbox else the check box will be get selected based on the xpath given in elementValue

//Search is happening from 1st and last elements of rows (the best complexity is when the element at border and the worst case is when element is in middle.

List<WebElement> rows=we.findElements(By.tagName("tr"));

/\* if(value!=null && value !=""&& rows.size() > 10)

{

Logger.logExecutionInfo("Executing step ......please wait ");

//System.out.println("Number of rows:"+rows.size());

String str=null;

String str1=null;

List<String> col\_values =new ArrayList<String>();

List<String> col\_values1 =new ArrayList<String>();

int lenRows = rows.size();

int n=lenRows/2;

System.out.println("lenRows"+lenRows+" "+n);

for(int rnum=1,frum=lenRows-1;rnum<n && frum>n;rnum++,frum--)

{

System.out.println("rnum"+rnum);

System.out.println("fnum"+frum);

List<WebElement> columns=rows.get(rnum).findElements(By.tagName("td"));

List<WebElement> columns1=rows.get(frum).findElements(By.tagName("td"));

System.out.println("Number of columns:"+columns.size());

System.out.println("Number of columns:"+columns1.size());

for(int cnum=1,cfrum=1;cnum<columns.size()||cfrum>columns1.size();cnum++,cfrum++)

{

str=columns.get(cnum).getText();

str1=columns1.get(cfrum).getText();

System.out.println(str);

System.out.println(str1);

if (str!=null || str != ""||str1!=null || str1 != ""){

col\_values.add(str);

col\_values1.add(str1);

}

}

if(col\_values.contains(value))

{

System.out.println("Got it "+value);

rnum++;

String actualxpath = stepValue.concat("/tr[")+rnum+("]/td[1]/input");

System.out.println(actualxpath+" "+rnum);

we=w.driver.findElement(By.xpath(actualxpath));

we.click();

col\_values.clear();

break;

}

if(col\_values1.contains(value))

{

System.out.println("Got it "+value);

frum++;

String actualxpath = stepValue.concat("/tr[")+frum+("]/td[1]/input");

System.out.println(actualxpath+" "+frum);

we=w.driver.findElement(By.xpath(actualxpath));

we.click();

col\_values1.clear();

break;

}

}

}\*/

// Liner search logic

if(value!=null && value !="")

{

// Logger.logExecutionInfo("Executing step and searching for check box please wait ");

System.out.println("Number of rows:"+rows.size());

String str=null;

List<String> col\_values =new ArrayList<String>();

for(int rnum=1;rnum<rows.size();rnum++)

{

List<WebElement> columns=rows.get(rnum).findElements(By.tagName("td"));

System.out.println("Number of columns:"+columns.size());

for(int cnum=1;cnum<columns.size();cnum++)

{

str=columns.get(cnum).getText();

System.out.println(str+"1");

if (str!=null || str != ""){

col\_values.add(str);

}

}

if(col\_values.contains(value))

{

System.out.println("Got it "+value);

rnum++;

String actualxpath = stepValue.concat("/tr[")+rnum+("]/td[1]/input");

System.out.println(actualxpath+" "+rnum);

we=w.driver.findElement(By.xpath(actualxpath));

we.click();

col\_values.clear();

break;

}

}

}

if (value==null || value == ""){

we.click();

}

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

e.printStackTrace();

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

//TO DO

class SetFileInput implements Step {

public SetFileInput() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

String value=s.getData().getValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

we.sendKeys(value);

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class SetPasswordInput implements Step {

public SetPasswordInput() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed. Element specified not found on page");

return ExecutionResults.FAIL;

}

we.clear();

we.sendKeys(Convert.getInstance().convertToi18n(s.getData().getValue()));

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class SetTextInput implements Step {

public SetTextInput() { }

@Override

public int execute(StepData s, WebSession w) {

//Logger.logExecutionInfo("Hi Input");

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError("Element not found on web page");

return ExecutionResults.FAIL;

}

we.clear();

we.sendKeys(Convert.getInstance().convertToi18n(s.getData().getValue()));

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class SetTextArea implements Step {

public SetTextArea() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed. Element specified not found on page");

return ExecutionResults.FAIL;

}

we.clear();

we.sendKeys(Convert.getInstance().convertToi18n(s.getData().getValue()));

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifyButtonNotPresent implements Step {

public VerifyButtonNotPresent() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.PASS;

} else {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.FAIL;

}

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

}

}

class VerifyButtonPresent implements Step {

public VerifyButtonPresent() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we!=null){

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.PASS;

} else {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.FAIL;

}

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

}

}

class VerifyLinkNotPresent implements Step {

public VerifyLinkNotPresent() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.PASS;

} else {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.FAIL;

}

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

}

}

class VerifyLinkPresent implements Step {

public VerifyLinkPresent() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else if(stepType.equals("linkText")) {

we=w.driver.findElement(By.linkText(stepValue));

} else if(stepType.equals("partialLinkText")) {

we=w.driver.findElement(By.partialLinkText(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we!=null){

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.PASS;

} else {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

return ExecutionResults.FAIL;

}

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

}

}

class VerifyTextNotPresent implements Step {

public VerifyTextNotPresent() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifyTextPresent implements Step {

public VerifyTextPresent() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

System.out.println(w.selenium.isTextPresent(Convert.getInstance().convertToi18n(s.getData().getValue())));

if(!w.selenium.isTextPresent(Convert.getInstance().convertToi18n(s.getData().getValue()))) {

Logger.logExecutionError(s.getDescription()+" failed : " +s.getData().getElementValue()+ "does not exists" );

return ExecutionResults.FAIL;

} else {

return ExecutionResults.PASS;

}

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed : " +s.getData().getElementValue()+ "does not exists" );

return ExecutionResults.FAIL;

}

}

}

class VerifyTitlePresent implements Step {

public VerifyTitlePresent() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class ExpectAlert implements Step {

public ExpectAlert() { }

@Override

public int execute(StepData s, WebSession w) {

//Logger.logExecutionInfo("Hi Alert");

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

if(stepType.equals("expectAlert")){

WebDriverWait wait = new WebDriverWait(w.driver, 15);

wait.until(ExpectedConditions.alertIsPresent());

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

if (alert!= null){

//Logger.logExecutionInfo("Print "+alert.getText());

alert.accept();

}

}

else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

}

catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class ExpectConfirm implements Step {

public ExpectConfirm() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifyDialog implements Step {

public VerifyDialog() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifyRadioButton implements Step {

public VerifyRadioButton() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifyCheckedBox implements Step {

public VerifyCheckedBox() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

String stepType=s.getData().getElementType();

String stepValue=s.getData().getElementValue();

String value=s.getData().getValue();

WebElement we=null;

if(stepType.equals("name")){

we=w.driver.findElement(By.name(stepValue));

} else if(stepType.equals("id")) {

we=w.driver.findElement(By.id(stepValue));

} else if(stepType.equals("className")) {

we=w.driver.findElement(By.className(stepValue));

} else if(stepType.equals("cssSelector")) {

we=w.driver.findElement(By.cssSelector(stepValue));

} else if(stepType.equals("xpath")) {

we=w.driver.findElement(By.xpath(stepValue));

} else if(stepType.equals("tagName")) {

we=w.driver.findElement(By.tagName(stepValue));

} else {

Logger.logExecutionError(s.getDescription()+" failed due to incompatible elementType");

return ExecutionResults.FAIL;

}

if(we==null){

Logger.logExecutionError(s.getDescription()+" failed. Element specified not found on page");

return ExecutionResults.FAIL;

}

Thread.currentThread().sleep(ToolVariables.executionSpeed);

//Code to check the checkBox Values

boolean str=we.isSelected();

System.out.println(str);

if(str==Boolean.parseBoolean(value))

{

return ExecutionResults.PASS;

}

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.FAIL;

}

}

class VerifyTextInput implements Step {

public VerifyTextInput() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifyTextArea implements Step {

public VerifyTextArea() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifySelectedOption implements Step {

public VerifySelectedOption() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class VerifyFileInput implements Step {

public VerifyFileInput() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(ToolVariables.executionSpeed);

} catch(Exception e) {

Logger.logExecutionError(s.getDescription()+" failed");

return ExecutionResults.FAIL;

}

return ExecutionResults.PASS;

}

}

class WaitForTime implements Step {

public WaitForTime() { }

@Override

public int execute(StepData s, WebSession w) {

Logger.logExecutionInfo("Executing step - -> "+s.getDescription());

try {

Thread.currentThread().sleep(Integer.parseInt(s.getData().getWaitTime()));

} catch (NumberFormatException e) {

Logger.logExecutionError(s.getDescription()+" failed. Time should have a numeric value");

return ExecutionResults.FAIL;

} catch (InterruptedException e) { }

return ExecutionResults.PASS;

}

}

}