**package** com.mastercard.testing.gdp.ui.framework;

**import** java.net.URL;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.apache.http.client.utils.URIBuilder;

**import** org.jbehave.core.annotations.AfterStories;

**import** org.jbehave.web.selenium.WebDriverProvider;

**import** org.openqa.selenium.MutableCapabilities;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.remote.CapabilityType;

**import** org.openqa.selenium.remote.RemoteWebDriver;

**import** org.openqa.selenium.remote.http.HttpClient;

**import** org.openqa.selenium.remote.internal.OkHttpClient;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** java.net.Proxy;

**import** java.net.InetSocketAddress;

**import** com.browserstack.local.Local;

**import** com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;

**public** **class** BrowserStackRunner **implements** WebDriverProvider {

**protected** ThreadLocal<WebDriver> delegate = **new** ThreadLocal<>();

**private** **static** Logger *logger* = LoggerFactory.*getLogger*(BrowserStackRunner.**class**);

**private** **static** **final** String ***BROWSERSTACK\_USER*** = System.*getProperty*("browserstack.user");

**private** **static** **final** String ***BROWSERSTACK\_KEY*** = System.*getProperty*("browserstack.key");

**private** **static** **final** String ***BROWSERSTACK\_LOCAL\_BIN*** = System.*getProperty*("browserstack.local.bin");

**private** **static** **final** String ***PROJECT\_VERSION*** = System.*getProperty*("browserstack.build.version");

**private** **static** **final** String ***PROJECT\_NAME*** = System.*getProperty*("browserstack.project");

**private** **static** **final** String ***SESSION\_NAME*** = System.*getProperty*("browserstack.name");

**private** **static** **final** String ***PROJECT\_LOCAL\_IDENTIFIER*** = System.*getProperty*("browserstack.local.identifier");

**private** **static** **final** String ***OS\_TYPE*** = System.*getProperty*("browserstack.web.os");

**private** **static** **final** String ***OS\_VERSION*** = System.*getProperty*("browserstack.web.os.version");

**private** **static** **final** String ***BROWSER\_TYPE*** = System.*getProperty*("default.web.execution.platform");

**private** **static** **final** String ***BROWSER\_VERSION*** = System.*getProperty*("browserstack.chrome.version");

**private** **static** **final** String ***OUTBOUND\_PROXY\_HOST*** = "outboundproxy.mclocal.int";

**private** **static** **final** **int** ***OUTBOUND\_PROXY\_PORT*** = 15768;

**private** Local browserStackLocalBinary;

@Override

**public** WebDriver get() {

URL browserStackHubURL;

WebDriver remoteDriver = delegate.get();

**if** (remoteDriver == **null**) {

MutableCapabilities capabilities = **new** MutableCapabilities();

Map<String, Object> bstackOptions = **new** HashMap<>();

*logger*.info("### BROWSERSTACK\_USER IS : {} ", ***BROWSERSTACK\_USER***);

*logger*.info("### BROWSERSTACK\_KEY IS : {} ", ***BROWSERSTACK\_KEY***);

*logger*.info("### BROWSERSTACK\_LOCAL\_BIN IS : {} ", ***BROWSERSTACK\_LOCAL\_BIN***);

*logger*.info("### PROJECT\_VERSION IS : {} ", ***PROJECT\_VERSION***);

*logger*.info("### PROJECT\_NAME IS : {} ", ***PROJECT\_NAME***);

*logger*.info("### SESSION\_NAME IS : {} ", ***SESSION\_NAME***);

*logger*.info("### PROJECT\_LOCAL\_IDENTIFIER IS : {} ", ***PROJECT\_LOCAL\_IDENTIFIER***);

*logger*.info("### OS\_TYPE IS : {} ", ***OS\_TYPE***);

*logger*.info("### OS\_VERSION IS : {} ", ***OS\_VERSION***);

*logger*.info("### BROWSER\_TYPE IS : {} ", ***BROWSER\_TYPE***);

*logger*.info("### BROWSER\_VERSION IS : {} ", ***BROWSER\_VERSION***);

bstackOptions.put("os", ***OS\_TYPE***);

bstackOptions.put("osVersion", ***OS\_VERSION***);

bstackOptions.put("debug", "true");

bstackOptions.put("networkLogs", "true");

bstackOptions.put("projectName", ***PROJECT\_NAME***);

bstackOptions.put("buildName", ***PROJECT\_VERSION***);

bstackOptions.put("sessionName", ***SESSION\_NAME***);

bstackOptions.put("local", **true**);

bstackOptions.put("localIdentifier", ***PROJECT\_LOCAL\_IDENTIFIER***);

capabilities.setCapability("bstack:options", bstackOptions);

capabilities.setCapability(CapabilityType.***ACCEPT\_SSL\_CERTS***, **true**);

capabilities.setCapability(CapabilityType.***ACCEPT\_INSECURE\_CERTS***, **true**);

capabilities.setCapability("browserName", ***BROWSER\_TYPE***.substring(13));

capabilities.setCapability("browserVersion", ***BROWSER\_VERSION***);

**try** {

browserStackHubURL = **new** URIBuilder("https://hub.browserstack.com").setPath("wd/hub")

.setUserInfo(***BROWSERSTACK\_USER***, ***BROWSERSTACK\_KEY***).build().toURL();

*logger*.info("### URL IS : {} ", browserStackHubURL);

} **catch** (Exception e) {

*logger*.error("Exception caught while preparing URL for browserstack driver");

**throw** **new** GDPUIException(e.getMessage(), e);

}

remoteDriver = **new** RemoteWebDriver(browserStackHubURL, capabilities);

delegate.set(remoteDriver);

}

**return** remoteDriver;

}

**public** **void** runBrowserStackBinary() {

browserStackLocalBinary = **new** Local();

Map<String, String> localArgs = **new** HashMap<>();

localArgs.put("v", "true");

localArgs.put("localIdentifier", ***PROJECT\_LOCAL\_IDENTIFIER***);

localArgs.put("binarypath", ***BROWSERSTACK\_LOCAL\_BIN***);

*logger*.info("Starting BrowserStack Local with Arguments : {}", localArgs);

localArgs.put("key", ***BROWSERSTACK\_KEY***);

*logger*.info("#### BS KEY is : {} ", ***BROWSERSTACK\_KEY***);

**try** {

browserStackLocalBinary.start(localArgs);

**if** (browserStackLocalBinary.isRunning()) {

*logger*.info("#### BrowserStack STARTED ####");

}

} **catch** (Exception e) {

*logger*.error("Exception caught while starting local binary driver for browserstack");

**throw** **new** GDPUIException(e.getMessage(), e);

}

*logger*.info("#### BrowserStack driver binary local started ####");

}

**protected** HttpClient.Factory clientFactory() {

**return** **new** OkHttpClient.Factory() {

@Override

**public** HttpClient.Builder builder() {

**return** **super**.builder().proxy(

**new** Proxy(Proxy.Type.***HTTP***, **new** InetSocketAddress(***OUTBOUND\_PROXY\_HOST***, ***OUTBOUND\_PROXY\_PORT***)));

}

};

}

@Override

@AfterStories

**public** **void** end() {

**try** {

**if** (browserStackLocalBinary.isRunning()) {

browserStackLocalBinary.stop();

*logger*.info("#### Closing BrowserStack driver binary local ####");

}

} **catch** (Exception e) {

*logger*.error("Exception caught while closing browserstack driver binary local");

**throw** **new** GDPUIException(e.getMessage(), e);

}

}

@Override

**public** **void** initialize() {

// **TODO** Auto-generated method stub

}

@Override

**public** **boolean** saveScreenshotTo(String path) {

// **TODO** Auto-generated method stub

**return** **false**;

}

}

**package** com.mastercard.testing.gdp.ui.framework;

**import** com.mastercard.quality.engineering.common.enumeration.WebExecutionPlatform;

**import** com.mastercard.quality.engineering.eyes.utils.factory.EyesFactory;

**import** com.mastercard.quality.engineering.eyes.utils.factory.EyesProperties;

**import** com.mastercard.quality.engineering.eyes.utils.testresults.VisualTestsResults;

**import** com.mastercard.quality.engineering.mtaf.df.factory.DriverFactory;

**import** com.mastercard.quality.engineering.mtaf.ui.providers.MasterCardWebDelegatingDriverProvider;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeOptions;

**import** org.openqa.selenium.firefox.FirefoxOptions;

**import** org.openqa.selenium.remote.CapabilityType;

**import** org.openqa.selenium.remote.DesiredCapabilities;

**import** org.openqa.selenium.remote.LocalFileDetector;

**import** org.openqa.selenium.remote.RemoteWebDriver;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** org.springframework.context.annotation.Primary;

**import** org.springframework.core.env.Environment;

**import** org.springframework.stereotype.Component;

**import** java.util.HashMap;

**import** **static** com.mastercard.testing.gdp.ui.tests.constants.UIConstants.DOWNLOAD\_FILEPATH;

@Component

@Primary

**public** **class** CustomCapabilitiesWebDriverProvider **extends** MasterCardWebDelegatingDriverProvider {

@Autowired

Environment env;

@Autowired

DriverFactory driverFactory;

@Autowired

EyesFactory eyesFactory;

@Autowired

@Qualifier("webEyesProperties")

EyesProperties eyesProperties;

@Autowired

VisualTestsResults visualTestsResults;

**public** CustomCapabilitiesWebDriverProvider(DriverFactory driverFactory, EyesFactory eyesFactory,

@Qualifier("webEyesProperties") EyesProperties eyesProperties, VisualTestsResults visualTestsResults) {

**super**(driverFactory, eyesFactory, eyesProperties, visualTestsResults);

}

**private** **static** **final** String ***ENABLE\_VNC*** = "enableVNC";

**private** **static** **final** String ***DOWNLOAD\_DEFAULT\_DIRECTORY*** = "download.default\_directory";

**private** **static** **final** String ***PREFS*** = "prefs";

**private** **static** **final** String ***DEFAULT\_CONTENT\_SETTINGS*** = "profile.default\_content\_settings.popups";

**private** **static** **final** String ***SHOW\_POP\_UP\_FOR\_EACH\_DOWNLOAD*** = "download.prompt\_for\_download";

**private** WebExecutionPlatform webPlatform;

@Override

**public** **void** initialize(WebExecutionPlatform platform) {

**this**.webPlatform = platform;

**super**.initialize(platform);

}

@Override

**public** **void** initialize() {

DesiredCapabilities customCapabilities = **new** DesiredCapabilities();

customCapabilities.setCapability(CapabilityType.***ACCEPT\_SSL\_CERTS***, **true**);

customCapabilities.setCapability(CapabilityType.***ACCEPT\_INSECURE\_CERTS***, **true**);

**if** (env.getProperty(***ENABLE\_VNC***, Boolean.**class**, **false**)) {

customCapabilities.setCapability(***ENABLE\_VNC***, **true**);

}

**if** (webPlatform == WebExecutionPlatform.GRID\_CHROME) {

customCapabilities.setCapability(ChromeOptions.***CAPABILITY***, **this**.getDownloadFileChromeOptions());

WebDriver driver = driverFactory.getWebDriver(webPlatform, customCapabilities);

RemoteWebDriver remoteDriver = (RemoteWebDriver) driver;

remoteDriver.setFileDetector(**new** LocalFileDetector());

**this**.delegate.set(remoteDriver);

} **else** **if** (webPlatform.toString().toLowerCase().contains("BROWSERSTACK\_".toLowerCase())) {

**this**.delegate.set((**new** BrowserStackRunner()).get());

} **else** **if** (webPlatform == WebExecutionPlatform.GRID\_FIREFOX) {

customCapabilities.setCapability(FirefoxOptions.***FIREFOX\_OPTIONS***, **this**.getDownloadFileFirefoxOptions());

WebDriver driver = driverFactory.getWebDriver(webPlatform, customCapabilities);

**this**.delegate.set(driver);

} **else** {

customCapabilities.setCapability(ChromeOptions.***CAPABILITY***, **this**.getDownloadFileChromeOptions());

WebDriver driver = driverFactory.getWebDriver(webPlatform, customCapabilities);

**this**.delegate.set(driver);

}

}

**private** ChromeOptions getDownloadFileChromeOptions() {

HashMap<String, Object> chromePrefs = **new** HashMap<>();

chromePrefs.put(***DEFAULT\_CONTENT\_SETTINGS***, 0);

chromePrefs.put(***SHOW\_POP\_UP\_FOR\_EACH\_DOWNLOAD***, **false**);

chromePrefs.put(***DOWNLOAD\_DEFAULT\_DIRECTORY***, System.*getProperty*("user.dir") + DOWNLOAD\_FILEPATH);

ChromeOptions options = **new** ChromeOptions();

options.setExperimentalOption(***PREFS***, chromePrefs);

**return** options;

}

**private** FirefoxOptions getDownloadFileFirefoxOptions() {

FirefoxOptions fireFoxoptions = **new** FirefoxOptions();

fireFoxoptions.addPreference(***DEFAULT\_CONTENT\_SETTINGS***, 0);

fireFoxoptions.addPreference(***SHOW\_POP\_UP\_FOR\_EACH\_DOWNLOAD***, **false**);

fireFoxoptions.addPreference(***DOWNLOAD\_DEFAULT\_DIRECTORY***, System.*getProperty*("user.dir") + DOWNLOAD\_FILEPATH);

**return** fireFoxoptions;

}

}

**package** com.mastercard.testing.gdp.ui.framework;

**import** org.jbehave.web.selenium.WebDriverProvider;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.ie.InternetExplorerDriver;

**import** org.openqa.selenium.remote.DesiredCapabilities;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.stereotype.Component;

@Component("customIEDriverProvider")

**public** **class** CustomIEDriverProvider **implements** WebDriverProvider {

**protected** ThreadLocal<WebDriver> delegate = **new** ThreadLocal<>();

**private** **static** Logger *logger* = LoggerFactory.*getLogger*(CustomIEDriverProvider.**class**);

/\*\*

\* **@deprecated** Need to update the IE driver provider logic

\*/

@Deprecated

@Override

**public** WebDriver ~~get~~() {

WebDriver driver = delegate.get();

**if** (driver == **null**) {

System.*setProperty*("webdriver.ie.driver",

"C:\\Users\\e074925\\Downloads\\IEDriverServer\_Win32\_3.8.0\\IEDriverServer.exe");

DesiredCapabilities caps = DesiredCapabilities.internetExplorer();

caps.setCapability(InternetExplorerDriver.***INTRODUCE\_FLAKINESS\_BY\_IGNORING\_SECURITY\_DOMAINS***, **true**);

caps.setCapability("introduceInstabilityByIgnoringProtectedModeSettings", **true**);

caps.setCapability("ignoreZoomSetting", **true**);

caps.setCapability("unexpectedAlertBehaviour", "accept");

caps.setCapability("ignoreProtectedModeSettings", **true**);

caps.setCapability("disable-popup-blocking", **true**);

caps.setCapability("enablePersistentHover", **false**);

caps.setCapability("requireWindowFocus", **true**);

driver = **new** ~~InternetExplorerDriver~~(caps);

delegate.set(driver);

}

**return** driver;

}

@Override

**public** **void** initialize() {

*logger*.info("emtpy implementation of initialize");

}

@Override

**public** **boolean** saveScreenshotTo(String path) {

**return** **false**;

}

@Override

**public** **void** end() {

*logger*.info("emtpy implementation of end");

}

}

**package** com.mastercard.testing.gdp.ui.framework;

**import** com.mastercard.quality.engineering.mtaf.ui.providers.MasterCardMobileDelegatingDriverProvider;

**import** com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;

**import** org.jbehave.web.selenium.WebDriverProvider;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.core.env.Environment;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** GDPDriverProviderFactory {

@Autowired

**private** CustomCapabilitiesWebDriverProvider webDriverProvider;

@Autowired

CustomIEDriverProvider customIEDriverProvider;

@Autowired

**private** MasterCardMobileDelegatingDriverProvider mobileDriverProvider;

@Autowired

**private** Environment env;

**public** WebDriverProvider getDriverProvider() {

**if** ((env.getProperty("default.web.execution.platform") != **null**)) {

**return** webDriverProvider;

} **else** **if** (env.getProperty("default.mobile.execution.platform") != **null**) {

**return** mobileDriverProvider;

} **else** **if** ("IE".equalsIgnoreCase(env.getProperty("browser"))) {

**return** customIEDriverProvider;

} **else** {

**throw** **new** GDPUIException(

"Neither default.web.execution.platform nor default.mobile.execution.platform is set to appropriate value.");

}

}

}

//

// Source code recreated from a .class file by IntelliJ IDEA

// (powered by FernFlower decompiler)

//

package com.mastercard.quality.engineering.mtaf.ui.providers;

import com.applitools.eyes.RectangleSize;

import com.applitools.eyes.selenium.Eyes;

import com.mastercard.quality.engineering.common.enumeration.WebExecutionPlatform;

import com.mastercard.quality.engineering.eyes.utils.factory.EyesFactory;

import com.mastercard.quality.engineering.eyes.utils.factory.EyesProperties;

import com.mastercard.quality.engineering.eyes.utils.testresults.VisualTestsResults;

import com.mastercard.quality.engineering.mtaf.df.factory.DriverFactory;

import com.mastercard.quality.engineering.mtaf.ui.exceptions.DelegateWebDriverNotFoundException;

import com.mastercard.quality.engineering.mtaf.ui.exceptions.EyesNotDefinedException;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import org.apache.commons.io.IOUtils;

import org.jbehave.web.selenium.WebDriverProvider;

import org.openqa.selenium.Capabilities;

import org.openqa.selenium.MutableCapabilities;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

public class MasterCardWebDelegatingDriverProvider implements WebDriverProvider {

private WebExecutionPlatform platform;

private Eyes eyes;

private String eyesAppName;

private String eyesTestName;

private DriverFactory driverFactory;

private EyesFactory eyesFactory;

private EyesProperties eyesProperties;

private VisualTestsResults visualTestsResults;

protected ThreadLocal<WebDriver> delegate = new ThreadLocal();

public MasterCardWebDelegatingDriverProvider(DriverFactory driverFactory, EyesFactory eyesFactory, EyesProperties eyesProperties, VisualTestsResults visualTestsResults) {

this.driverFactory = driverFactory;

this.eyesFactory = eyesFactory;

this.eyesProperties = eyesProperties;

this.visualTestsResults = visualTestsResults;

}

public WebDriver get() {

WebDriver driver = (WebDriver)this.delegate.get();

if (driver == null) {

throw new DelegateWebDriverNotFoundException("Delegate WebDriver not found. Please verify that either\n1. You have set a Maven parameter 'default.web.execution.platform' with a valid value.\n2. You have included a meta tag in each JBehave story '@webPlatform' with a valid value\n3. You have included a meta tag for each JBehave scenario '@webPlatform' with a valid value.");

} else {

return driver;

}

}

public Eyes getEyes() {

if (this.eyes == null) {

throw new EyesNotDefinedException("AppliTools Eyes has not been initialized. Please make sure your story is meta tagged with the proper value for @eyesStrategy. Available values are 'all', 'mobile' or 'web.");

} else {

return this.eyes;

}

}

public void initializeWithEyes(WebExecutionPlatform platform, String appName, String testName) {

this.platform = platform;

this.eyes = this.eyesFactory.getEyes(this.eyesProperties);

this.eyesAppName = appName;

this.eyesTestName = testName;

}

public void initialize(WebExecutionPlatform platform) {

this.platform = platform;

this.initialize();

}

public void initialize() {

this.initializeDriver(new MutableCapabilities());

this.initializeEyes();

}

public void initialize(WebExecutionPlatform platform, Capabilities capabilities) {

this.platform = platform;

this.initialize(capabilities);

}

public void initialize(Capabilities capabilities) {

this.initializeDriver(capabilities);

this.initializeEyes();

}

private void initializeDriver(Capabilities capabilities) {

WebDriver driver = this.driverFactory.getWebDriver(this.platform, capabilities);

this.delegate.set(driver);

}

private void initializeEyes() {

if (this.eyes != null) {

if (this.eyesProperties.getViewportWidth() != 0 && this.eyesProperties.getViewportHeight() != 0) {

this.eyes.open((WebDriver)this.delegate.get(), this.eyesAppName, this.eyesTestName, new RectangleSize(this.eyesProperties.getViewportWidth(), this.eyesProperties.getViewportHeight()));

} else {

this.eyes.open((WebDriver)this.delegate.get(), this.eyesAppName, this.eyesTestName);

}

}

}

public void end() {

if (this.delegate.get() != null) {

((WebDriver)this.delegate.get()).quit();

}

if (this.eyes != null) {

this.visualTestsResults.saveResultsAndClose(this.eyes, this.eyesAppName, this.eyesTestName);

}

}

public boolean saveScreenshotTo(String path) {

WebDriver driver = (WebDriver)this.delegate.get();

if (driver instanceof TakesScreenshot && driver != null) {

File file = new File(path);

byte[] bytes = (byte[])((TakesScreenshot)driver).getScreenshotAs(OutputType.BYTES);

file.getParentFile().mkdirs();

try {

file.createNewFile();

IOUtils.write(bytes, new FileOutputStream(file));

return true;

} catch (IOException var6) {

throw new RuntimeException("Can't save file", var6);

}

} else {

return false;

}

}

public static class DelegateWebDriverNotFound extends RuntimeException {

public DelegateWebDriverNotFound() {

super("WebDriver has not been found for this thread.\nPlease verify you are using the correct WebDriverProvider, with the appropriate credentials if using remote access, e.g. to Browserstack: -DBROWSERSTACK\_USERNAME=xxxxxx -DBROWSERSTACK\_ACCESS\_KEY=xxx-xxxx-xxxx-xxxx-xxx ");

}

}

}

//

// Source code recreated from a .class file by IntelliJ IDEA

// (powered by FernFlower decompiler)

//

package com.mastercard.quality.engineering.common.enumeration;

public enum WebExecutionPlatform {

LOCAL\_CHROME,

LOCAL\_FIREFOX,

LOCAL\_IE,

LOCAL\_EDGE,

LOCAL\_SAFARI,

BROWSERSTACK\_CHROME,

BROWSERSTACK\_FIREFOX,

BROWSERSTACK\_IE,

BROWSERSTACK\_SAFARI,

BROWSERSTACK\_OPERA,

BROWSERSTACK\_YANDEX,

BROWSERSTACK\_EDGE,

GRID\_FIREFOX,

GRID\_CHROME,

GRID\_SAFARI,

GRID\_EDGE,

GRID\_IE,

LOCAL\_INCOGNITO\_CHROME,

LOCAL\_HEADLESS\_CHROME,

LOCAL\_PRIVATE\_FIREFOX,

LOCAL\_PRIVATE\_IE,

PHANTOM\_JS,

NONE;

private WebExecutionPlatform() {

}

}

//

// Source code recreated from a .class file by IntelliJ IDEA

// (powered by FernFlower decompiler)

//

package com.mastercard.quality.engineering.mtaf.df.factory;

import com.mastercard.quality.engineering.common.enumeration.MobileExecutionPlatform;

import com.mastercard.quality.engineering.common.enumeration.WebExecutionPlatform;

import com.mastercard.quality.engineering.common.exception.InvalidMobileExecutionPlatformException;

import com.mastercard.quality.engineering.common.exception.InvalidWebExecutionPlatformException;

import com.mastercard.quality.engineering.mtaf.df.drivers.ExtendedAndroidDriver;

import com.mastercard.quality.engineering.mtaf.df.drivers.ExtendedIOSDriver;

import com.mastercard.quality.engineering.mtaf.df.drivers.ScreenshootingBrowserStackDriver;

import com.mastercard.quality.engineering.mtaf.df.drivers.ScreenshootingChromeDriver;

import com.mastercard.quality.engineering.mtaf.df.drivers.ScreenshootingEdgeDriver;

import com.mastercard.quality.engineering.mtaf.df.drivers.ScreenshootingFirefoxDriver;

import com.mastercard.quality.engineering.mtaf.df.drivers.ScreenshootingIEDriver;

import com.mastercard.quality.engineering.mtaf.df.drivers.ScreenshootingPhantomJsDriver;

import io.appium.java\_client.AppiumDriver;

import io.appium.java\_client.android.AndroidDriver;

import io.appium.java\_client.ios.IOSDriver;

import java.net.URL;

import org.apache.log4j.Logger;

import org.openqa.selenium.Capabilities;

import org.openqa.selenium.MutableCapabilities;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.remote.RemoteWebDriver;

import org.openqa.selenium.remote.http.HttpClient.Factory;

import org.openqa.selenium.safari.SafariDriver;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.ApplicationContext;

public class DriverFactory {

private static Logger LOG = Logger.getLogger(DriverFactory.class);

@Autowired

private ApplicationContext applicationContext;

public DriverFactory() {

}

public WebDriver getWebDriver(WebExecutionPlatform platform) {

return this.getWebDriver(platform, new MutableCapabilities());

}

public WebDriver getWebDriver(WebExecutionPlatform platform, Capabilities capabilities) {

switch(platform) {

case LOCAL\_FIREFOX:

return new ScreenshootingFirefoxDriver(capabilities, false);

case LOCAL\_CHROME:

return new ScreenshootingChromeDriver(capabilities, false, false);

case LOCAL\_SAFARI:

return new SafariDriver(capabilities);

case LOCAL\_IE:

return new ScreenshootingIEDriver(capabilities, false);

case LOCAL\_EDGE:

return new ScreenshootingEdgeDriver(capabilities, false);

case GRID\_FIREFOX:

return new RemoteWebDriver(this.getURL("gridHubURL"), this.getBaseCapabilities("gridFirefoxBaseCapabilities").merge(capabilities));

case GRID\_CHROME:

return new RemoteWebDriver(this.getURL("gridHubURL"), this.getBaseCapabilities("gridChromeBaseCapabilities").merge(capabilities));

case GRID\_SAFARI:

return new RemoteWebDriver(this.getURL("gridHubURL"), this.getBaseCapabilities("gridSafariBaseCapabilities").merge(capabilities));

case GRID\_IE:

return new RemoteWebDriver(this.getURL("gridHubURL"), this.getBaseCapabilities("gridIEBaseCapabilities").merge(capabilities));

case GRID\_EDGE:

return new RemoteWebDriver(this.getURL("gridHubURL"), this.getBaseCapabilities("gridEdgeBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_FIREFOX:

return new ScreenshootingBrowserStackDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackFirefoxBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_CHROME:

return new ScreenshootingBrowserStackDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackChromeBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_SAFARI:

return new ScreenshootingBrowserStackDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackSafariBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_IE:

return new ScreenshootingBrowserStackDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackIEBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_EDGE:

return new ScreenshootingBrowserStackDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackEdgeBaseCapabilities").merge(capabilities));

case LOCAL\_INCOGNITO\_CHROME:

return new ScreenshootingChromeDriver(capabilities, true, false);

case LOCAL\_HEADLESS\_CHROME:

return new ScreenshootingChromeDriver(capabilities, false, true);

case LOCAL\_PRIVATE\_IE:

return new ScreenshootingIEDriver(capabilities, true);

case LOCAL\_PRIVATE\_FIREFOX:

return new ScreenshootingFirefoxDriver(capabilities, true);

case PHANTOM\_JS:

return new ScreenshootingPhantomJsDriver(capabilities);

default:

throw new InvalidWebExecutionPlatformException(platform);

}

}

public AppiumDriver<WebElement> getMobileDriver(MobileExecutionPlatform platform) {

return this.getMobileDriver(platform, new MutableCapabilities());

}

public AppiumDriver<WebElement> getMobileDriver(MobileExecutionPlatform platform, Capabilities capabilities) {

switch(platform) {

case LOCAL\_IOS:

return new IOSDriver(this.getURL("appiumHubURL"), this.getBaseCapabilities("localIOSBaseCapabilities").merge(capabilities));

case LOCAL\_ANDROID:

return new AndroidDriver(this.getURL("appiumHubURL"), this.getBaseCapabilities("localAndroidBaseCapabilities").merge(capabilities));

case PERFECTO\_IOS:

return new ExtendedIOSDriver(this.getClientFactory(), this.getURL("perfectoHubURL"), this.getBaseCapabilities("perfectoIOSBaseCapabilities").merge(capabilities));

case PERFECTO\_ANDROID:

return new ExtendedAndroidDriver(this.getClientFactory(), this.getURL("perfectoHubURL"), this.getBaseCapabilities("perfectoAndroidBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_IPHONE:

return new ExtendedIOSDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackIPhoneBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_IPAD:

return new ExtendedIOSDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackIPadBaseCapabilities").merge(capabilities));

case BROWSERSTACK\_ANDROID:

return new ExtendedAndroidDriver(this.getClientFactory(), this.getURL("browserStackHubURL"), this.getBaseCapabilities("browserStackAndroidBaseCapabilities").merge(capabilities));

default:

throw new InvalidMobileExecutionPlatformException(platform);

}

}

private Factory getClientFactory() {

return (Factory)this.applicationContext.getBean(Factory.class);

}

private URL getURL(String url) {

return (URL)this.applicationContext.getBean(url);

}

private Capabilities getBaseCapabilities(String capabilities) {

return (Capabilities)this.applicationContext.getBean(capabilities);

}

}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.firefox.FirefoxDriver;  
import org.openqa.selenium.firefox.FirefoxOptions;  
import org.openqa.selenium.firefox.FirefoxProfile;  
  
public class ScreenshootingFirefoxDriver extends FirefoxDriver {  
 private static final Logger *LOG* = Logger.*getLogger*(ScreenshootingFirefoxDriver.class);  
  
 public ScreenshootingFirefoxDriver(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 super(*getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 *LOG*.debug("CAPABILITIES: " + *getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 }  
  
 private static FirefoxOptions getAllCapabilities(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 FirefoxOptions options = *getBaseCapabilities*(isPrivateMode);  
 options.merge(additionalCapabilities);  
 return options;  
 }  
  
 private static FirefoxOptions getBaseCapabilities(boolean isPrivateMode) {  
 FirefoxOptions options = new FirefoxOptions();  
 FirefoxProfile firefoxProfile = new FirefoxProfile();  
 if (isPrivateMode) {  
 firefoxProfile.setPreference("browser.privatebrowsing.autostart", true);  
 options.setCapability("firefox\_profile", firefoxProfile);  
 }  
  
 return options;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.chrome.ChromeDriver;  
import org.openqa.selenium.chrome.ChromeOptions;  
  
public class ScreenshootingChromeDriver extends ChromeDriver {  
 private static Logger *LOG* = Logger.*getLogger*(ScreenshootingChromeDriver.class);  
  
 public ScreenshootingChromeDriver(Capabilities additionalCapabilities, boolean isPrivateMode, boolean isHeadless) {  
 super(*getAllCapabilities*(additionalCapabilities, isPrivateMode, isHeadless));  
 *LOG*.debug(*getAllCapabilities*(additionalCapabilities, isPrivateMode, isHeadless));  
 }  
  
 private static ChromeOptions getAllCapabilities(Capabilities additionalCapabilities, boolean isPrivateMode, boolean isHeadless) {  
 ChromeOptions options = *getBaseCapabilities*(isPrivateMode, isHeadless);  
 options.merge(additionalCapabilities);  
 return options;  
 }  
  
 private static ChromeOptions getBaseCapabilities(boolean isPrivateMode, Boolean isHeadless) {  
 ChromeOptions options = new ChromeOptions();  
 options.addArguments(new String[]{"--test-type"});  
 options.addArguments(new String[]{"--disable-device-discovery-notifications"});  
 options.addArguments(new String[]{"--disable-desktop-notifications"});  
 options.addArguments(new String[]{"--start-maximized"});  
 *LOG*.info("Setting ScreenshootingChromeDriver.isHeadless : " + isHeadless + "...");  
 if (isHeadless) {  
 options.addArguments(new String[]{"--headless"});  
 options.addArguments(new String[]{"--window-size=1200x600"});  
 }  
  
 LOG.info("Setting ScreenshootingChromeDriver.isPrivateMode : " + isPrivateMode + "...");  
 if (isPrivateMode) {  
 options.addArguments(new String[]{"incognito"});  
 }  
  
 options.setCapability("takesScreenshot", true);  
 return options;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.ie.InternetExplorerDriver;  
import org.openqa.selenium.ie.InternetExplorerOptions;  
  
public class ScreenshootingIEDriver extends InternetExplorerDriver {  
 private static final Logger *LOG* = Logger.*getLogger*(ScreenshootingIEDriver.class);  
  
 public ScreenshootingIEDriver(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 super(*getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 *LOG*.debug("CAPABILITIES: " + *getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 }  
  
 private static InternetExplorerOptions getAllCapabilities(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 InternetExplorerOptions options = *getBaseCapabilities*(isPrivateMode);  
 options.merge(additionalCapabilities);  
 return options;  
 }  
  
 private static InternetExplorerOptions getBaseCapabilities(boolean isPrivateMode) {  
 InternetExplorerOptions options = new InternetExplorerOptions();  
 options.setCapability("ignoreProtectedModeSettings", true);  
 options.setCapability("ie.forceCreateProcessApi", true);  
 options.setCapability("ignoreZoomSetting", true);  
 options.setCapability("ie.ensureCleanSession", true);  
 if (isPrivateMode) {  
 options.setCapability("ie.browserCommandLineSwitches", "-private");  
 options.setCapability("ie.forceCreateProcessApi", true);  
 }  
  
 return options;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.UnsupportedCommandException;  
import org.openqa.selenium.edge.EdgeDriver;  
import org.openqa.selenium.edge.EdgeOptions;  
  
public class ScreenshootingEdgeDriver extends EdgeDriver {  
 private static Logger *LOG* = Logger.*getLogger*(ScreenshootingEdgeDriver.class);  
  
 public ScreenshootingEdgeDriver(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 super(*getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 *LOG*.debug("CAPABILITIES: " + *getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 }  
  
 private static EdgeOptions getAllCapabilities(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 EdgeOptions options = *getBaseCapabilities*(isPrivateMode);  
 options.merge(additionalCapabilities);  
 return options;  
 }  
  
 private static EdgeOptions getBaseCapabilities(boolean isPrivateMode) {  
 EdgeOptions options = new EdgeOptions();  
 options.setPageLoadStrategy("normal");  
 if (isPrivateMode) {  
 *LOG*.error("Private mode is not compatible for edge");  
 throw new UnsupportedCommandException("Private Mode is not yet supported for Edge");  
 } else {  
 return options;  
 }  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import java.net.URL;  
import java.util.HashMap;  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.remote.CommandExecutor;  
import org.openqa.selenium.remote.HttpCommandExecutor;  
import org.openqa.selenium.remote.LocalFileDetector;  
import org.openqa.selenium.remote.RemoteWebDriver;  
import org.openqa.selenium.remote.http.HttpClient.Factory;  
  
public class ScreenshootingBrowserStackDriver extends RemoteWebDriver {  
 private static final Logger *LOG* = Logger.*getLogger*(ScreenshootingBrowserStackDriver.class);  
  
 public ScreenshootingBrowserStackDriver(Factory clientFactory, URL browserStackHubURL, Capabilities capabilities) {  
 super(new HttpCommandExecutor(new HashMap(), browserStackHubURL, clientFactory), capabilities);  
 *LOG*.debug("CAPABILITIES: " + capabilities);  
 this.fileDetector();  
 }  
  
 public ScreenshootingBrowserStackDriver(URL browserStackHubURL, Capabilities capabilities) {  
 super(browserStackHubURL, capabilities);  
 *LOG*.debug("CAPABILITIES: " + capabilities);  
 this.fileDetector();  
 }  
  
 public ScreenshootingBrowserStackDriver(CommandExecutor executor, Capabilities capabilities) {  
 super(executor, capabilities);  
 *LOG*.debug("CAPABILITIES: " + capabilities);  
 this.fileDetector();  
 }  
  
 private void fileDetector() {  
 Boolean isLocalFileUpdload = Boolean.*parseBoolean*(System.*getProperty*("browserstack.web.localfileupload", "false"));  
 if (isLocalFileUpdload) {  
 this.setFileDetector(new LocalFileDetector());  
 }  
  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.chrome.ChromeDriver;  
import org.openqa.selenium.chrome.ChromeOptions;  
  
public class ScreenshootingChromeDriver extends ChromeDriver {  
 private static Logger *LOG* = Logger.*getLogger*(ScreenshootingChromeDriver.class);  
  
 public ScreenshootingChromeDriver(Capabilities additionalCapabilities, boolean isPrivateMode, boolean isHeadless) {  
 super(*getAllCapabilities*(additionalCapabilities, isPrivateMode, isHeadless));  
 *LOG*.debug(*getAllCapabilities*(additionalCapabilities, isPrivateMode, isHeadless));  
 }  
  
 private static ChromeOptions getAllCapabilities(Capabilities additionalCapabilities, boolean isPrivateMode, boolean isHeadless) {  
 ChromeOptions options = *getBaseCapabilities*(isPrivateMode, isHeadless);  
 options.merge(additionalCapabilities);  
 return options;  
 }  
  
 private static ChromeOptions getBaseCapabilities(boolean isPrivateMode, Boolean isHeadless) {  
 ChromeOptions options = new ChromeOptions();  
 options.addArguments(new String[]{"--test-type"});  
 options.addArguments(new String[]{"--disable-device-discovery-notifications"});  
 options.addArguments(new String[]{"--disable-desktop-notifications"});  
 options.addArguments(new String[]{"--start-maximized"});  
 *LOG*.info("Setting ScreenshootingChromeDriver.isHeadless : " + isHeadless + "...");  
 if (isHeadless) {  
 options.addArguments(new String[]{"--headless"});  
 options.addArguments(new String[]{"--window-size=1200x600"});  
 }  
  
 *LOG*.info("Setting ScreenshootingChromeDriver.isPrivateMode : " + isPrivateMode + "...");  
 if (isPrivateMode) {  
 options.addArguments(new String[]{"incognito"});  
 }  
  
 options.setCapability("takesScreenshot", true);  
 return options;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.ie.InternetExplorerDriver;  
import org.openqa.selenium.ie.InternetExplorerOptions;  
  
public class ScreenshootingIEDriver extends InternetExplorerDriver {  
 private static final Logger *LOG* = Logger.*getLogger*(ScreenshootingIEDriver.class);  
  
 public ScreenshootingIEDriver(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 super(*getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 *LOG*.debug("CAPABILITIES: " + *getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 }  
  
 private static InternetExplorerOptions getAllCapabilities(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 InternetExplorerOptions options = *getBaseCapabilities*(isPrivateMode);  
 options.merge(additionalCapabilities);  
 return options;  
 }  
  
 private static InternetExplorerOptions getBaseCapabilities(boolean isPrivateMode) {  
 InternetExplorerOptions options = new InternetExplorerOptions();  
 options.setCapability("ignoreProtectedModeSettings", true);  
 options.setCapability("ie.forceCreateProcessApi", true);  
 options.setCapability("ignoreZoomSetting", true);  
 options.setCapability("ie.ensureCleanSession", true);  
 if (isPrivateMode) {  
 options.setCapability("ie.browserCommandLineSwitches", "-private");  
 options.setCapability("ie.forceCreateProcessApi", true);  
 }  
  
 return options;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.firefox.FirefoxDriver;  
import org.openqa.selenium.firefox.FirefoxOptions;  
import org.openqa.selenium.firefox.FirefoxProfile;  
  
public class ScreenshootingFirefoxDriver extends FirefoxDriver {  
 private static final Logger *LOG* = Logger.*getLogger*(ScreenshootingFirefoxDriver.class);  
  
 public ScreenshootingFirefoxDriver(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 super(*getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 *LOG*.debug("CAPABILITIES: " + *getAllCapabilities*(additionalCapabilities, isPrivateMode));  
 }  
  
 private static FirefoxOptions getAllCapabilities(Capabilities additionalCapabilities, boolean isPrivateMode) {  
 FirefoxOptions options = *getBaseCapabilities*(isPrivateMode);  
 options.merge(additionalCapabilities);  
 return options;  
 }  
  
 private static FirefoxOptions getBaseCapabilities(boolean isPrivateMode) {  
 FirefoxOptions options = new FirefoxOptions();  
 FirefoxProfile firefoxProfile = new FirefoxProfile();  
 if (isPrivateMode) {  
 firefoxProfile.setPreference("browser.privatebrowsing.autostart", true);  
 options.setCapability("firefox\_profile", firefoxProfile);  
 }  
  
 return options;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.MutableCapabilities;  
import org.openqa.selenium.phantomjs.PhantomJSDriver;  
  
public class ScreenshootingPhantomJsDriver extends PhantomJSDriver {  
 public ScreenshootingPhantomJsDriver(Capabilities additionalCapabilities) {  
 super(*getAllCapabilities*(additionalCapabilities));  
 }  
  
 private static Capabilities getAllCapabilities(Capabilities additionalCapabilities) {  
 return *getBaseCapabilities*().merge(additionalCapabilities);  
 }  
  
 private static Capabilities getBaseCapabilities() {  
 MutableCapabilities baseCapabilities = new MutableCapabilities();  
 baseCapabilities.setCapability("browserName", "phantomjs");  
 return baseCapabilities;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.common.exception;  
  
import com.mastercard.quality.engineering.common.enumeration.WebExecutionPlatform;  
  
public class InvalidWebExecutionPlatformException extends MTAFRuntimeException {  
 private static final long *serialVersionUID* = 1L;  
 private static final String *baseMessage* = "Available web platforms are local\_firefox, local\_chrome, local\_safari, local\_ie, local\_edge, grid\_firefox, grid\_chrome, grid\_safari, grid\_ie, browserstack\_firefox, browserstack\_chrome, browserstack\_safari, browserstack\_ie";  
  
 public InvalidWebExecutionPlatformException(WebExecutionPlatform platform) {  
 super(String.*format*("Available web platforms are local\_firefox, local\_chrome, local\_safari, local\_ie, local\_edge, grid\_firefox, grid\_chrome, grid\_safari, grid\_ie, browserstack\_firefox, browserstack\_chrome, browserstack\_safari, browserstack\_ie", platform.toString()));  
 }  
  
 public InvalidWebExecutionPlatformException(WebExecutionPlatform platform, Throwable cause) {  
 super(String.*format*("Available web platforms are local\_firefox, local\_chrome, local\_safari, local\_ie, local\_edge, grid\_firefox, grid\_chrome, grid\_safari, grid\_ie, browserstack\_firefox, browserstack\_chrome, browserstack\_safari, browserstack\_ie", platform.toString()), cause);  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import io.appium.java\_client.MobileCommand;  
import io.appium.java\_client.android.AndroidDriver;  
import java.net.URL;  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.remote.HttpCommandExecutor;  
import org.openqa.selenium.remote.http.HttpClient.Factory;  
  
public class ExtendedIOSDriver extends AndroidDriver<WebElement> {  
 private static final Logger *LOG* = Logger.*getLogger*(ExtendedIOSDriver.class);  
  
 public ExtendedIOSDriver(Factory clientFactory, URL url, Capabilities capabilities) {  
 super(new HttpCommandExecutor(MobileCommand.*commandRepository*, url, clientFactory), capabilities);  
 *LOG*.debug("CAPABILITIES: " + capabilities);  
 }  
  
 public ExtendedIOSDriver(URL url, Capabilities capabilities) {  
 super(url, capabilities);  
 *LOG*.debug("CAPABILITIES: " + capabilities);  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.df.drivers;  
  
import io.appium.java\_client.MobileCommand;  
import io.appium.java\_client.android.AndroidDriver;  
import java.net.URL;  
import org.apache.log4j.Logger;  
import org.openqa.selenium.Capabilities;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.remote.HttpCommandExecutor;  
import org.openqa.selenium.remote.http.HttpClient.Factory;  
  
public class ExtendedAndroidDriver extends AndroidDriver<WebElement> {  
 private static final Logger *LOG* = Logger.*getLogger*(ExtendedAndroidDriver.class);  
  
 public ExtendedAndroidDriver(Factory clientFactory, URL url, Capabilities capabilities) {  
 super(new HttpCommandExecutor(MobileCommand.*commandRepository*, url, clientFactory), capabilities);  
 *LOG*.debug("CAPABILITIES: " + capabilities);  
 }  
  
 public ExtendedAndroidDriver(URL url, Capabilities capabilities) {  
 super(url, capabilities);  
 *LOG*.debug("CAPABILITIES: " + capabilities);  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.common.enumeration;  
  
public enum MobileExecutionPlatform {  
 *LOCAL\_IOS*,  
 *LOCAL\_ANDROID*,  
 *BROWSERSTACK\_IPHONE*,  
 *BROWSERSTACK\_IPAD*,  
 *BROWSERSTACK\_ANDROID*,  
 *PERFECTO\_IOS*,  
 *PERFECTO\_ANDROID*,  
 *NONE*;  
  
 private MobileExecutionPlatform() {  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.eyes.utils.factory;  
  
import com.applitools.eyes.ProxySettings;  
import com.applitools.eyes.selenium.Eyes;  
import com.mastercard.quality.engineering.eyes.utils.batch.BatchProvider;  
import java.net.URI;  
import java.util.Optional;  
  
public class EyesFactory {  
 private URI eyesServerUrl;  
 private String eyesAPIKey;  
 private BatchProvider batchProvider;  
 private Optional<ProxySettings> proxySettings;  
  
 public EyesFactory(URI eyesServerUrl, String eyesAPIKey, BatchProvider batchProvider) {  
 this.eyesServerUrl = eyesServerUrl;  
 this.eyesAPIKey = eyesAPIKey;  
 this.batchProvider = batchProvider;  
 this.proxySettings = Optional.*empty*();  
 }  
  
 public EyesFactory(URI eyesServerUrl, String eyesAPIKey, BatchProvider batchProvider, ProxySettings proxySettings) {  
 this(eyesServerUrl, eyesAPIKey, batchProvider);  
 this.proxySettings = Optional.*of*(proxySettings);  
 }  
  
 public Eyes getEyes() {  
 Eyes eyes = new Eyes();  
 eyes.setServerUrl(this.eyesServerUrl);  
 eyes.setApiKey(this.eyesAPIKey);  
 this.proxySettings.ifPresent(eyes::setProxy);  
 return eyes;  
 }  
  
 public Eyes getEyes(EyesProperties properties) {  
 Eyes eyes = this.getEyes();  
 if (!properties.getBranchName().isEmpty()) {  
 eyes.setBranchName(properties.getBranchName());  
 }  
  
 if (!properties.getBatchName().isEmpty()) {  
 eyes.setBatch(this.batchProvider.getBatch(properties.getBatchName()));  
 }  
  
 eyes.setMatchLevel(properties.getMatchLevel());  
 eyes.setForceFullPageScreenshot(properties.getForceFullScreenshot());  
 eyes.setHideScrollbars(properties.getHideScrollbars());  
 eyes.setScrollToRegion(properties.getScrollToRegion());  
 eyes.setMatchTimeout(properties.getMatchTimeout());  
 return eyes;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.eyes.utils.batch;  
  
import com.applitools.eyes.BatchInfo;  
import java.util.HashMap;  
import java.util.Map;  
  
public class BatchProvider {  
 private Map<String, BatchInfo> batches = new HashMap();  
  
 public BatchProvider() {  
 }  
  
 public BatchInfo getBatch(String batchName) {  
 if (this.batches.containsKey(batchName)) {  
 return (BatchInfo)this.batches.get(batchName);  
 } else {  
 BatchInfo batch = new BatchInfo(batchName);  
 this.batches.put(batchName, batch);  
 return batch;  
 }  
 }  
}

package com.applitools.eyes;  
  
public class ProxySettings extends AbstractProxySettings {  
  
 public ProxySettings(String uri, int port, String username, String password) {  
 super(uri+":"+port, port, username, password);  
 }  
  
 public ProxySettings(String uri, int port) {  
 super(uri+":"+port, null, null);  
 }  
  
 public ProxySettings(String uri, String username, String password) {  
 super(uri, username, password);  
 }  
  
 public ProxySettings(String uri) {  
 super(uri);  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.eyes.utils.factory;  
  
import com.applitools.eyes.MatchLevel;  
  
public class EyesProperties {  
 private String batchName;  
 private Boolean forceFullScreenshot;  
 private MatchLevel matchLevel;  
 private Integer viewportHeight;  
 private Integer viewportWidth;  
 private Boolean hideScrollbars;  
 private Boolean scrollToRegion;  
 private Integer matchTimeout;  
 private String branchName;  
  
 public EyesProperties() {  
 }  
  
 public String getBatchName() {  
 return this.batchName != null ? this.batchName : "";  
 }  
  
 public String getBranchName() {  
 return this.branchName != null ? this.branchName : "";  
 }  
  
 public Boolean getForceFullScreenshot() {  
 return this.forceFullScreenshot != null ? this.forceFullScreenshot : true;  
 }  
  
 public MatchLevel getMatchLevel() {  
 return this.matchLevel != null ? this.matchLevel : MatchLevel.*STRICT*;  
 }  
  
 public Integer getViewportHeight() {  
 return this.viewportHeight != null && this.viewportHeight >= 0 ? this.viewportHeight : 0;  
 }  
  
 public Integer getViewportWidth() {  
 return this.viewportWidth != null && this.viewportWidth >= 0 ? this.viewportWidth : 0;  
 }  
  
 public Boolean getHideScrollbars() {  
 return this.hideScrollbars != null ? this.hideScrollbars : true;  
 }  
  
 public Boolean getScrollToRegion() {  
 return this.scrollToRegion != null ? this.scrollToRegion : false;  
 }  
  
 public Integer getMatchTimeout() {  
 return this.matchTimeout != null && this.matchTimeout >= 0 ? this.matchTimeout : 15000;  
 }  
  
 public void setBatchName(String batchName) {  
 this.batchName = batchName;  
 }  
  
 public void setForceFullScreenshot(Boolean forceFullScreenshot) {  
 this.forceFullScreenshot = forceFullScreenshot;  
 }  
  
 public void setMatchLevel(MatchLevel matchLevel) {  
 this.matchLevel = matchLevel;  
 }  
  
 public void setViewportHeight(Integer viewportHeight) {  
 this.viewportHeight = viewportHeight;  
 }  
  
 public void setViewportWidth(Integer viewportWidth) {  
 this.viewportWidth = viewportWidth;  
 }  
  
 public void setHideScrollbars(Boolean hideScrollbars) {  
 this.hideScrollbars = hideScrollbars;  
 }  
  
 public void setScrollToRegion(Boolean scrollToRegion) {  
 this.scrollToRegion = scrollToRegion;  
 }  
  
 public void setMatchTimeout(Integer matchTimeout) {  
 this.matchTimeout = matchTimeout;  
 }  
  
 public void setBranchName(String branchName) {  
 this.branchName = branchName;  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.eyes.utils.testresults;  
  
import com.applitools.eyes.TestResults;  
import com.applitools.eyes.selenium.Eyes;  
import java.util.HashMap;  
import java.util.Map;  
  
public class VisualTestsResults {  
 private static Map<String, TestResults> *resultsMap* = new HashMap();  
 private static final String *SEPARATOR* = ";";  
  
 public VisualTestsResults() {  
 }  
  
 public TestResults getResults(String appName, String testName) {  
 return (TestResults)*resultsMap*.get(appName.concat(";").concat(testName));  
 }  
  
 public void addResults(String appName, String testName, TestResults results) {  
 *resultsMap*.put(appName.concat(";").concat(testName), results);  
 }  
  
 public void saveResultsAndClose(Eyes eyes, String appName, String eyesTestName) {  
 if (eyes.getIsOpen()) {  
 TestResults results = eyes.close(false);  
 *resultsMap*.put(appName.concat(";").concat(eyesTestName), results);  
 }  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.ui.exceptions;  
  
public class DelegateWebDriverNotFoundException extends RuntimeException {  
 private static final long *serialVersionUID* = -5113687032084239119L;  
  
 public DelegateWebDriverNotFoundException() {  
 }  
  
 public DelegateWebDriverNotFoundException(String message) {  
 super(message);  
 }  
  
 public DelegateWebDriverNotFoundException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.ui.exceptions;  
  
public class EyesNotDefinedException extends RuntimeException {  
 private static final long *serialVersionUID* = -2151021476842994071L;  
  
 public EyesNotDefinedException() {  
 }  
  
 public EyesNotDefinedException(String message) {  
 super(message);  
 }  
  
 public EyesNotDefinedException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
@FunctionalInterface  
public interface StepArgumentProvider {  
 Object[] getLastStepArguments();  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
import java.util.List;  
  
public interface TestDataExpressionResolver {  
  
 String resolveExpression(String expression);  
  
 List<String> resolveListExpression(String expression);  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
import java.util.List;  
import java.util.Map;  
  
public interface TestDataProvider {  
  
 String getValue(String expression);  
  
 List<String> getList(String expression);  
  
 Map<String, Object> getMap(String expression);  
  
 boolean support(String prefix);  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
import org.assertj.core.api.Assertions;  
  
import org.jbehave.core.annotations.Given;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import com.mastercard.testing.gdp.ui.tests.steps.common.BaseSteps;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Component  
public class ContextVariablesSteps extends BaseSteps {  
  
 @Autowired  
 private ContextVariableTestData variableTestData;  
  
 @Autowired  
 private JsonTestData jsonTestData;  
  
 */\*\*  
 \* Save string value in context  
 \*  
 \** ***@param*** *name variable name  
 \** ***@param*** *value variable value  
 \*/* @Given("variable '$name' equals '$value'")  
 public void saveContextVariable(String name, String value) {  
 variableTestData.setContextVariable(name, value);  
 }  
  
 */\*\*  
 \* Remove variable from context  
 \*  
 \** ***@param*** *name variable name  
 \*/* @Given("variable '$name' is removed")  
 public void removeContextVariable(String name) {  
 variableTestData.setContextVariable(name, null);  
 }  
  
 */\*\*  
 \* Push value into array  
 \*  
 \** ***@param*** *name list name  
 \** ***@param*** *value value  
 \*/* @Given("value '$value' is pushed into list '$name'")  
 public void pushValueIntoList(String value, String name) {  
 Object list = variableTestData.getContextVariable(name);  
 if (list == null) {  
 list = new ArrayList<String>();  
 variableTestData.setContextVariable(name, list);  
 }  
 Assertions.assertThat(list instanceof List).overridingErrorMessage(  
 String.format("Variable %s (type = %s) is " + "not a List", name, list.getClass())).isTrue();  
 ((List<String>) list).add(value);  
 }  
  
 */\*\*  
 \* Save list of string values in context  
 \*  
 \** ***@param*** *name variable name  
 \** ***@param*** *value variable values  
 \*/* @Given("list '$name' equals '$value'")  
 public void saveContextVariable(String name, List<String> value) {  
 variableTestData.setContextVariable(name, value);  
 }  
  
 */\*\*  
 \* Save exists context variable with new name  
 \*  
 \** ***@param*** *name exists variable name  
 \** ***@param*** *alias alias name  
 \*/* @Given("alias of '$name' is '$alias'")  
 public void saveContextVariableAs(String name, String alias) {  
 variableTestData.setContextVariable(alias, variableTestData.getContextVariable(name));  
 }  
  
 */\*\*  
 \* Save json object in context variables. Json object is got by jsonPath  
 \*  
 \** ***@param*** *jsonPath path in json to object for save, array indexing is not  
 \* allowed  
 \** ***@param*** *variableName new context variable name  
 \*/* @Given("json data '$jsonPath' saved as '$variableName'")  
 public void saveJsonDataAsVariable(String jsonPath, String variableName) {  
 variableTestData.setContextVariable(variableName, jsonTestData.getMap(jsonPath));  
 }  
  
 public String getContextVariableValue(String key) {  
 return variableTestData.getValue(key);  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
import org.apache.commons.lang3.StringUtils;  
import org.springframework.stereotype.Component;  
  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
@Component  
public class ContextVariableTestData implements TestDataProvider {  
  
 public static final String PROVIDER\_PREFIX = "v";  
  
 private String varibleText = "Variable ";  
  
 private ThreadLocal<Map<String, Object>> contextVariables = new ThreadLocal<>();  
  
 public Object getContextVariable(String key) {  
 String[] pathElements = key.split("\\.");  
 return getVariableStorageByPath(pathElements).get(pathElements[pathElements.length - 1]);  
 }  
  
 public void setContextVariable(String key, Object value) {  
 String[] pathElements = key.split("\\.");  
 getVariableStorageByPath(pathElements).put(pathElements[pathElements.length - 1], value);  
 }  
  
 private Map<String, Object> getVariableStorageByPath(String[] pathElements) {  
 Map<String, Object> variableStorage = getVariableStorage();  
 for (int i = 0; i < pathElements.length - 1; i++) {  
 Object storageCandidate = variableStorage.get(pathElements[i]);  
 if (storageCandidate == null) {  
 storageCandidate = new HashMap<String, Object>();  
 variableStorage.put(pathElements[i], storageCandidate);  
 }  
 if (storageCandidate instanceof Map) {  
 variableStorage = (Map) storageCandidate;  
 } else {  
 throw new GDPUIException(varibleText + StringUtils.join(Arrays.copyOfRange(pathElements, 0, i), ".")  
 + " class=" + storageCandidate.getClass().getName() + " already exists and is is not a Map");  
 }  
 }  
 return variableStorage;  
 }  
  
 @Override  
 public String getValue(String expression) {  
 Object value = getContextVariable(expression);  
 return value != null ? value.toString() : null;  
 }  
  
 @Override  
 public List<String> getList(String expression) {  
 Object variable = getContextVariable(expression);  
 if (variable instanceof String) {  
 return Arrays.asList((String) variable);  
 }  
 if (variable instanceof List) {  
 return (List) variable;  
 } else {  
 throw new IllegalArgumentException(varibleText + expression + " is not list");  
 }  
 }  
  
 @Override  
 public Map<String, Object> getMap(String expression) {  
 Object variable = getContextVariable(expression);  
 if (variable instanceof Map) {  
 return (Map) variable;  
 } else {  
 throw new IllegalArgumentException(varibleText + expression + " is not map");  
 }  
 }  
  
 @Override  
 public boolean support(String prefix) {  
 return PROVIDER\_PREFIX.equals(prefix);  
 }  
  
 private Map<String, Object> getVariableStorage() {  
 if (contextVariables.get() == null) {  
 contextVariables.set(new HashMap<String, Object>());  
 }  
 return contextVariables.get();  
 }  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
@Component  
public class DefaultTestDataProvider implements TestDataProvider {  
  
 public static final String *PROVIDER\_PREFIX* = "default";  
 public static final String *VALUE\_KEY* = "value";  
 public static final Pattern *EXPRESSION\_PARAMS* = Pattern.*compile*("[a-z]\\{[\\w.\\s]\*\\}");  
 @Autowired  
 private TestDataExpressionResolver expressionResolver;  
  
 @Override  
 public String getValue(String expression) {  
 String matcherExpression = expression;  
 Matcher matcher = EXPRESSION\_PARAMS.matcher(matcherExpression);  
 while (matcher.find()) {  
 String substitute = expressionResolver.resolveExpression(matcher.group());  
 matcherExpression = matcherExpression.replace("(" + matcher.group() + ")", substitute);  
 }  
 return matcherExpression;  
 }  
  
 @Override  
 public List<String> getList(String expression) {  
 return Arrays.asList(expression);  
 }  
  
 @Override  
 public Map<String, Object> getMap(String expression) {  
 Map<String, Object> result = new HashMap<>();  
 result.put(VALUE\_KEY, expression);  
 return result;  
 }  
  
 @Override  
 public boolean support(String prefix) {  
 return PROVIDER\_PREFIX.equals(prefix);  
 }  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.JsonNode;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.databind.node.TextNode;  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.\*;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
@Component  
public class JsonTestData implements TestDataProvider {  
  
 public static final Pattern EXPRESSION\_PARAMS = Pattern.compile("[a-z]\\{[\\w.\\s]\*\\}");  
 public static final String PROVIDER\_PREFIX = "j";  
 @Autowired  
 private TestDataExpressionResolver expressionResolver;  
 private ObjectMapper objectMapper = new ObjectMapper();  
  
 private ThreadLocal<JsonNode> jsonNodeThreadLocal = new ThreadLocal<>();  
  
 public void init(String path) {  
 try {  
 InputStream inputStream = this.getClass().getResourceAsStream(path);  
 jsonNodeThreadLocal.set(objectMapper.readTree(inputStream));  
 } catch (IOException ex) {  
 throw new GDPUIException(ex.getMessage(), ex);  
 }  
 }  
  
 @Override  
 public String getValue(String jsonPath) {  
 JsonNode jsonNode = getNodeByPath(buildPath(jsonPath));  
 if (jsonNode instanceof TextNode) {  
 return jsonNode.asText();  
 }  
 return jsonNode != null ? jsonNode.toString() : null;  
 }  
  
 public List<String> getList(String jsonPath) {  
 JsonNode jsonNode = getNodeByPath(buildPath(jsonPath));  
  
 if (jsonNode != null) {  
 List<String> result = new ArrayList<>();  
 Iterator<JsonNode> childIterator = jsonNode.elements();  
 while (childIterator.hasNext()) {  
 result.add(childIterator.next().toString());  
 }  
 return result;  
 } else {  
 return new ArrayList<>();  
 }  
 }  
  
 @Override  
 public Map<String, Object> getMap(String jsonPath) {  
 JsonNode jsonNode = getNodeByPath(buildPath(jsonPath));  
  
 if (jsonNode != null) {  
 try {  
 return objectMapper.treeToValue(jsonNode, Map.class);  
 } catch (JsonProcessingException ex) {  
 throw new GDPUIException(ex.getMessage(), ex);  
 }  
 } else {  
 return new HashMap<>();  
 }  
 }  
  
 @Override  
 public boolean support(String prefix) {  
 return PROVIDER\_PREFIX.equals(prefix);  
 }  
  
 public JsonNode getNodeByPath(String jsonPath) {  
 JsonNode jsonNode = jsonNodeThreadLocal.get();  
 for (String path : jsonPath.split("\\.")) {  
 if (jsonNode == null) {  
 break;  
 }  
 jsonNode = jsonNode.get(path);  
 }  
 return jsonNode;  
 }  
  
 public String buildPath(String untouchedPath) {  
 String replacedPath = untouchedPath;  
 Matcher matcher = EXPRESSION\_PARAMS.matcher(untouchedPath);  
 while (matcher.find()) {  
 String substitute = expressionResolver.resolveExpression(matcher.group());  
 replacedPath = replacedPath.replace("(" + matcher.group() + ")", substitute);  
 }  
 return replacedPath;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.aspect;  
  
import org.apache.commons.logging.Log;  
import org.apache.commons.logging.LogFactory;  
import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.Around;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Pointcut;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.List;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
@Aspect  
@Component  
public class TestDataProviderAspect implements TestDataExpressionResolver, StepArgumentProvider {  
  
 public static final Pattern TEST\_DATA\_EXPRESSION = Pattern.compile("([A-z])\\{(.\*)\\}$");  
  
 private static final Log LOG = LogFactory.getLog(TestDataProviderAspect.class);  
  
 @Autowired  
 private List<TestDataProvider> testDataProviders;  
  
 private ThreadLocal<Object[]> lastArguments = new ThreadLocal<>();  
  
 @Pointcut(value = "@annotation(org.jbehave.core.annotations.Given)")  
 public void givenMethod() {  
 LOG.info("TestDataProviderAspect::givenMethod is called");  
 }  
  
 @Pointcut(value = "@annotation(org.jbehave.core.annotations.When)")  
 public void whenMethod() {  
 LOG.info("TestDataProviderAspect::whenMethod is called");  
 }  
  
 @Pointcut(value = "@annotation(org.jbehave.core.annotations.Then)")  
 public void thenMethod() {  
 LOG.info("TestDataProviderAspect::thenMethod is called");  
 }  
  
 @Around("givenMethod() || whenMethod() || thenMethod()")  
 public Object methodInterceptor(ProceedingJoinPoint jp) throws Throwable {  
 String methodName = jp.toShortString();  
 LOG.info("Start handle test data expressions: Method = " + methodName);  
 Object[] arguments = jp.getArgs();  
 for (int i = 0; i < arguments.length; i++) {  
 Object argument = arguments[i];  
 if (argument != null) {  
 if (argument instanceof String) {  
 arguments[i] = resolveExpression((String) argument);  
 } else if (argument instanceof Collection && ((Collection) argument).size() == 1) {  
 Object collectionElement = ((Collection) argument).iterator().next();  
 if (collectionElement instanceof String) {  
 arguments[i] = resolveListExpression((String) collectionElement);  
 }  
 }  
 }  
 }  
 lastArguments.set(arguments);  
 try {  
 return jp.proceed(arguments);  
 } catch (Throwable e) {  
 LOG.info(e.getMessage());  
 throw e;  
 }  
 }  
  
 @Override  
 public String resolveExpression(String expression) {  
 StringBuilder result = new StringBuilder();  
 String param = "";  
 for (String subExpression : expression.split("\\+")) {  
 Matcher matcher = TEST\_DATA\_EXPRESSION.matcher(subExpression);  
  
 String prefix = "default";  
  
 if (matcher.find()) {  
 prefix = matcher.group(1);  
 subExpression = matcher.group(2);  
 }  
 param = prefix + "{" + subExpression + "}";  
 result.append(getDataProvider(prefix).getValue(subExpression));  
 }  
  
 return expression.replace(param, result.toString());  
 }  
  
 @Override  
 public List<String> resolveListExpression(String expression) {  
 List<String> result = new ArrayList<>();  
 for (String subExpression : expression.split("\\+")) {  
 Matcher matcher = TEST\_DATA\_EXPRESSION.matcher(subExpression);  
  
 String prefix = "default";  
  
 if (matcher.find()) {  
 prefix = matcher.group(1);  
 subExpression = matcher.group(2);  
 }  
 result.addAll(getDataProvider(prefix).getList(subExpression));  
 }  
 return result;  
 }  
  
 @Override  
 public Object[] getLastStepArguments() {  
 Object[] arguments = lastArguments.get();  
 lastArguments.remove();  
 return arguments;  
 }  
  
 private TestDataProvider getDataProvider(String prefix) {  
 for (TestDataProvider testDataProvider : testDataProviders) {  
 if (testDataProvider.support(prefix)) {  
 return testDataProvider;  
 }  
 }  
 throw new IllegalArgumentException("Unsupported expression type '" + prefix + "{...}");  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.configuration;  
  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.springframework.beans.factory.FactoryBean;  
  
public interface MCClientFactory extends FactoryBean<CloseableHttpClient> {  
 CloseableHttpClient getObject();  
}

package com.mastercard.testing.gdp.ui.tests.configuration;  
  
import com.fasterxml.jackson.databind.ObjectMapper;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
@ComponentScan  
public class BDDBaseRestToolsConfig {  
 @Bean  
 public ObjectMapper objectMapper() {  
 return new ObjectMapper();  
 }  
}

package com.mastercard.testing.gdp.ui.tests.configuration;  
  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.springframework.beans.factory.FactoryBean;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Import;  
  
@Configuration  
@Import({ BDDBaseRestToolsConfig.class })  
public class BDDRestToolsConfig {  
 @Bean  
 public FactoryBean<CloseableHttpClient> getFactory() {  
 return new HttpClientFactory();  
 }  
}

package com.mastercard.testing.gdp.ui.tests.configuration;  
  
import org.springframework.beans.factory.config.BeanFactoryPostProcessor;  
import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;  
import org.springframework.context.support.SimpleThreadScope;  
  
public class CustomScopeRegistering implements BeanFactoryPostProcessor {  
  
 @Override  
 public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) {  
 beanFactory.registerScope("thread", new SimpleThreadScope());  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.configuration;  
  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.apache.http.impl.client.HttpClients;  
  
public class HttpClientFactory implements MCClientFactory {  
 public Class<?> getObjectType() {  
 return CloseableHttpClient.class;  
 }  
  
 @Override  
 public boolean isSingleton() {  
 return true;  
 }  
  
 public CloseableHttpClient getObject() {  
 return HttpClients.createDefault();  
 }  
}

package com.mastercard.testing.gdp.ui.tests.configuration;  
  
import com.mastercard.quality.engineering.mtaf.df.configuration.MTAFDriverFactoryConfiguration;  
import com.mastercard.quality.engineering.mtaf.ui.configuration.spring.MTAFWebToolsConfiguration;  
import org.springframework.beans.factory.config.BeanFactoryPostProcessor;  
import org.springframework.context.annotation.\*;  
  
@Configuration  
@ComponentScan(basePackages = { "com.mastercard.testing.gdp.ui.tests" })  
@ComponentScan(basePackages = { "com.mastercard.testing.gdp.ui.framework" })  
@PropertySource({ "config/${env}/application.properties" })  
@Import({ MTAFWebToolsConfiguration.class, MTAFDriverFactoryConfiguration.class, BDDRestToolsConfig.class })  
  
@EnableAspectJAutoProxy  
public class TestConfiguration {  
  
 @Bean  
 public static BeanFactoryPostProcessor beanFactoryPostProcessor() {  
 return new CustomScopeRegistering();  
 }  
}

package com.mastercard.testing.gdp.ui.tests.configuration;  
  
public class RestServiceException extends RuntimeException {  
 private static final long *serialVersionUID* = -942235246006018987L;  
  
 public RestServiceException() {  
 }  
  
 public RestServiceException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

package com.mastercard.testing.gdp.ui.tests.dbconnection;  
  
import org.apache.commons.dbutils.ResultSetHandler;  
  
import javax.sql.DataSource;  
import java.sql.Connection;  
import java.sql.SQLException;  
  
public interface DataSourceConfigBase {  
  
 public DataSource getDataSource();  
  
 public Connection getConnection() throws SQLException;  
  
 public ResultSetHandler<Object> getResultSetHandler();  
  
 public ResultSetHandler<Object> createResultSetHandler();  
  
}

package com.mastercard.testing.gdp.ui.tests.dbconnection;  
  
import org.apache.commons.dbcp2.BasicDataSource;  
import org.apache.commons.dbutils.ResultSetHandler;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import javax.sql.DataSource;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.SQLException;  
  
@Component  
public class DataSourceConfig implements DataSourceConfigBase {  
  
 @Value("${db.url:local}")  
 private String dbUrl;  
  
 @Value("${db.user:local}")  
 private String dbUser;  
  
 @Value("${db.password:local}")  
 private String dbPassword;  
  
 @Value("${db.url.fcrm:local}")  
 private String dbUrlFCRM;  
  
 @Value("${db.user.fcrm:local}")  
 private String dbUserFCRM;  
  
 @Value("${db.password.fcrm:local}")  
 private String dbPasswordFCRM;  
  
 private DataSource postgres = null;  
 private ResultSetHandler resultSetHandler = null;  
 private DataSource postgresFCRM = null;  
  
 public DataSource getDataSource() {  
 if (postgres == null) {  
 postgres = createDataSource(dbUrl, dbUser, dbPassword);  
 }  
 return postgres;  
 }  
  
 public Connection getConnection() throws SQLException {  
 return this.getDataSource().getConnection();  
 }  
  
 public DataSource getFcrmDataSource() {  
 if (postgresFCRM == null) {  
 postgresFCRM = createDataSource(dbUrlFCRM, dbUserFCRM, dbPasswordFCRM);  
 }  
 return postgresFCRM;  
 }  
  
 public ResultSetHandler getResultSetHandler() {  
 if (resultSetHandler == null) {  
 resultSetHandler = createResultSetHandler();  
 }  
 return resultSetHandler;  
 }  
  
 public ResultSetHandler createResultSetHandler() {  
 return new ResultSetHandler<Object[]>() {  
 @Override  
 public Object[] handle(ResultSet rs) throws SQLException {  
 if (!rs.next()) {  
 return new Object[0];  
 }  
  
 ResultSetMetaData meta = rs.getMetaData();  
 int cols = meta.getColumnCount();  
 Object[] result = new Object[cols];  
  
 for (int i = 0; i < cols; i++) {  
 result[i] = rs.getObject(i + 1);  
 }  
  
 return result;  
 }  
 };  
 }  
  
 public DataSource createDataSource(String dbUrl, String dbUser, String dbPassword) {  
 BasicDataSource dataSource = new BasicDataSource();  
 dataSource.setDriverClassName("org.postgresql.Driver");  
 dataSource.setUrl(dbUrl);  
 dataSource.setUsername(dbUser);  
 dataSource.setPassword(dbPassword);  
 return dataSource;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.dbconnection;  
  
import org.apache.commons.dbcp2.BasicDataSource;  
import org.apache.commons.dbutils.ResultSetHandler;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import javax.sql.DataSource;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.SQLException;  
  
@Component  
public class DataSourceConfigOracle implements DataSourceConfigBase {  
  
 @Value("${oracle.db.url:local}")  
 private String dbUrl;  
  
 @Value("${oracle.db.user:local}")  
 private String dbUser;  
  
 @Value("${oracle.db.password:local}")  
 private String dbPassword;  
  
 @Value("${oracle.db.schema:local}")  
 private String dbSchema;  
  
 private DataSource oracle = null;  
 private ResultSetHandler resultSetHandler = null;  
  
 public DataSource getDataSource() {  
 if (oracle == null) {  
 oracle = createDataSource(dbUrl, dbUser, dbPassword, dbSchema);  
 }  
 return oracle;  
 }  
  
 public Connection getConnection() throws SQLException {  
 return this.getDataSource().getConnection();  
 }  
  
 public ResultSetHandler getResultSetHandler() {  
 if (resultSetHandler == null) {  
 resultSetHandler = createResultSetHandler();  
 }  
 return resultSetHandler;  
 }  
  
 public ResultSetHandler createResultSetHandler() {  
 return new ResultSetHandler<Object[]>() {  
 @Override  
 public Object[] handle(ResultSet recordSet) throws SQLException {  
 if (!recordSet.next()) {  
 return new Object[0];  
 }  
  
 ResultSetMetaData meta = recordSet.getMetaData();  
 int cols = meta.getColumnCount();  
 Object[] result = new Object[cols];  
  
 for (int i = 0; i < cols; i++) {  
 result[i] = recordSet.getObject(i + 1);  
 }  
  
 return result;  
 }  
 };  
 }  
  
 public DataSource createDataSource(String dbUrl, String dbUser, String dbPassword, String dbSchema) {  
 BasicDataSource dataSource = new BasicDataSource();  
 dataSource.setDriverClassName("oracle.jdbc.OracleDriver");  
 dataSource.setUrl(dbUrl);  
 dataSource.setUsername(dbUser);  
 dataSource.setPassword(dbPassword);  
 dataSource.setDefaultSchema(dbSchema);  
 return dataSource;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.dbconnection;  
  
import com.ibatis.common.jdbc.ScriptRunner;  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
import com.mastercard.testing.gdp.ui.tests.steps.common.BaseSteps;  
import org.apache.commons.dbutils.QueryRunner;  
import org.apache.commons.dbutils.ResultSetHandler;  
import org.apache.commons.dbutils.handlers.ColumnListHandler;  
import org.jbehave.core.annotations.AfterStories;  
import org.jbehave.core.annotations.BeforeStories;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.Reader;  
import java.nio.charset.StandardCharsets;  
import java.sql.\*;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class DBOperationBaseUtils extends BaseSteps {  
  
 protected QueryRunner queryRunner;  
 protected Connection conn = null;  
 protected PreparedStatement pstmt = null;  
 protected ResultSet rs=null;  
  
   
 protected DataSourceConfigBase dataSourceConfigBase;  
  
 @AfterStories  
 public void closeDatabaseConnection() {  
 try {  
 logger.info("Closed the connection "+ conn);  
 conn.close();  
 } catch (SQLException sqlError) {  
 logger.info("SQL exception occurred {}", sqlError);  
 }  
 }  
  
 @BeforeStories  
 public void establishDatabaseConnection() {  
 try {  
 conn = dataSourceConfigBase.getDataSource().getConnection();  
 queryRunner = new QueryRunner(dataSourceConfigBase.getDataSource());  
 logger.info("Connection object is "+ conn);  
 } catch (SQLException sqlError) {  
 logger.info("SQL exception occurred {}", sqlError);  
 }  
 }  
  
 public void executeInsertStatement(String query) {  
 logger.info("INSERT\_QUERY :"+ query);  
 try {  
 queryRunner.insert(query, dataSourceConfigBase.getResultSetHandler());  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public void performUpdate(String query) {  
 logger.info("UPDATE\_QUERY : "+ query);  
 try {  
 queryRunner.update(query);  
 } catch (SQLException e) {  
 logger.error("Exception occurred while updating the records: ", e);  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public int executeDeleteStatement(String query) {  
 logger.info("DELETE\_QUERY : "+ query);  
 return executeUpdate(query);  
 }  
  
 public int executeUpdateStatement(String query) {  
 logger.info("UPDATE\_QUERY : "+ query);  
 return executeUpdate(query);  
 }  
  
 private int executeUpdate(String query) {  
 int numberOfRecordsEffected = -1;  
 try {  
 queryRunner.insert(query, dataSourceConfigBase.getResultSetHandler());  
 numberOfRecordsEffected = queryRunner.update(query);  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return numberOfRecordsEffected;  
 }  
  
 public int getCountOfRecords(String query) {  
 int numberOfRecords = -1;  
 try {  
 Object[] result = (Object[]) queryRunner.query(query, dataSourceConfigBase.getResultSetHandler());  
 long recordsCount = (long) result[0];  
 numberOfRecords = (int) recordsCount;  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return numberOfRecords;  
 }  
  
 public String getSingleDbValue(String query) {  
 logger.info("Query : "+ query);  
 String value = null;  
 try {  
 Object[] result = (Object[]) queryRunner.query(query, dataSourceConfigBase.getResultSetHandler());  
 if (result != null && result[0] != null) {  
 value = result[0].toString();  
 }  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 logger.info("Value fetched from database is "+ value);  
 return value;  
 }  
  
 public <T> List<T> getAllDbValuesForAColumn(String query, String columnName) {  
 List<T> returnList = null;  
 ColumnListHandler<T> resultSetHandler = new ColumnListHandler<>(columnName);  
 try {  
 returnList = queryRunner.query(query, resultSetHandler);  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return returnList;  
 }  
  
 public <T> List<T> getCompleteRowData(String query, Class clazz) {  
 Object obj;  
 try {  
 obj = clazz.newInstance();  
 return queryRunner.query(query, (ResultSetHandler<List<T>>) obj);  
 } catch (InstantiationException | IllegalAccessException | SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public void executeSqlScript(String sqlFilePath) throws SQLException {  
 try {  
 logger.info("Connection object is "+ conn);  
 ScriptRunner scriptRunner = new ScriptRunner(conn, false, true);  
 InputStream inputStream = this.getClass().getResourceAsStream(sqlFilePath);  
 Reader reader = new InputStreamReader(inputStream, StandardCharsets.UTF\_8.name());  
 scriptRunner.runScript(reader);  
 conn.setAutoCommit(false);  
 conn.commit();  
 } catch (Exception e) {  
 logger.error(String.format("Failed to Execute %s %s %s", sqlFilePath, " \nThe error is ", e));  
 try {  
 conn.rollback();  
 } catch (SQLException sqlException) {  
 logger.error("SQL Rollback exception :", sqlException);  
 }  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public Map<String, Object> executeSelectStmt(String query, List<String> keys) {  
 try {  
 try (PreparedStatement stmt = conn.prepareStatement(query)) {  
 Array array = conn.createArrayOf("text", keys.toArray(new String[0]));  
 stmt.setArray(1, array);  
 DriverManager.registerDriver(new org.postgresql.Driver());  
 ResultSet resultSet = stmt.executeQuery();  
 return prepareKeyValuePairs(resultSet);  
 }  
 } catch (SQLException e) {  
 logger.error("SQL exception occurred ", e);  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public Map<String, Object> prepareKeyValuePairs(ResultSet rs) throws SQLException {  
 Map<String, Object> row = new HashMap<>();  
 while (rs.next()) {  
 row.put(rs.getString(1).trim(), rs.getString(2));  
 }  
 return row;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.dbconnection;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
  
@Component  
public class DBOperationOracleUtils extends DBOperationBaseUtils {  
  
 @Autowired  
 private DataSourceConfigOracle dataSourceConfigOracle;  
  
 @PostConstruct  
 public void init() {  
 super.dataSourceConfigBase = this.dataSourceConfigOracle;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.dbconnection;  
  
import com.ibatis.common.jdbc.ScriptRunner;  
import com.mastercard.testing.gdp.ui.tests.aspect.ContextVariableTestData;  
import com.mastercard.testing.gdp.ui.tests.aspect.JsonTestData;  
import com.mastercard.testing.gdp.ui.tests.constants.LinkConstants;  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
import com.mastercard.testing.gdp.ui.tests.steps.api.CommonFunctionsBFF;  
import com.mastercard.testing.gdp.ui.tests.utils.QueryConstants;  
import org.apache.commons.collections4.MapUtils;  
import org.apache.commons.dbutils.QueryRunner;  
import org.apache.commons.dbutils.ResultSetHandler;  
import org.apache.commons.dbutils.handlers.ColumnListHandler;  
import org.apache.commons.lang3.BooleanUtils;  
import org.apache.commons.lang3.math.NumberUtils;  
import org.jbehave.core.annotations.\*;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
import org.springframework.core.io.ClassPathResource;  
  
import javax.annotation.PostConstruct;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.Reader;  
import java.nio.charset.StandardCharsets;  
import java.sql.\*;  
import java.util.\*;  
  
@Component  
public class DBOperationUtils extends DBOperationBaseUtils {  
  
 private static final String SQL\_FILE = "sqlFile";  
  
 @Value("${truncatePrivacyRequest:false}")  
 private String truncateTableSwitch;  
  
 @Value("${env:local}")  
 private String environment;  
  
 @Autowired  
 private DataSourceConfig dataSourceConfig;  
  
 @PostConstruct  
 public void init() {  
 super.dataSourceConfigBase = this.dataSourceConfig;  
 }  
  
 @Autowired  
 private ContextVariableTestData variableTestData;  
  
 @Autowired  
 private JsonTestData jsonTestData;  
  
 @Autowired  
 @Qualifier("commonFunctionsBFF")  
 private CommonFunctionsBFF commonFunctionsBFF;  
  
 static final Logger log = LoggerFactory.getLogger(DBOperationUtils.class);  
 String resourcePath = null;  
 private static final String DEV\_USERNAME = "dev-username";  
  
 private static final String BASE\_SELECT\_PRIVACY\_REQS\_QUERY = "SELECT \r\n" + " prvcy\_rqst\_id, \r\n" + "\r\n"  
 + " CASE\r\n"  
 + " WHEN (rqst\_type\_cd='V') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_type\_lkp where type\_cd='V')\r\n"  
 + " WHEN (rqst\_type\_cd='D') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_type\_lkp where type\_cd='D')\r\n"  
 + " WHEN (rqst\_type\_cd='M') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_type\_lkp where type\_cd='M')\r\n"  
 + " WHEN (rqst\_type\_cd='C') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_type\_lkp where type\_cd='C')\r\n"  
 + " END AS rqst\_type\_cd, \r\n" + " \r\n" + " CASE\r\n"  
 + " WHEN (rqst\_cntxt\_cd='B2C') THEN (select type\_nam from gdp\_owner.busn\_cntxt\_lkp where type\_cd='B2C')\r\n"  
 + " WHEN (rqst\_cntxt\_cd='B2E') THEN (select type\_nam from gdp\_owner.busn\_cntxt\_lkp where type\_cd='B2E')\r\n"  
 + " WHEN (rqst\_cntxt\_cd='B2B') THEN (select type\_nam from gdp\_owner.busn\_cntxt\_lkp where type\_cd='B2B')\r\n"  
 + " END AS rqst\_cntxt\_cd, \r\n" + " rqst\_email\_addr, rqst\_dt, \r\n" + " CASE\r\n"  
 + " WHEN (rqst\_stat\_cd='C') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='C')\r\n"  
 + " WHEN (rqst\_stat\_cd='I') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='I')\r\n"  
 + " WHEN (rqst\_stat\_cd='R') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='R')\r\n"  
 + " WHEN (rqst\_stat\_cd='S') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='S')\r\n"  
 + " WHEN (rqst\_stat\_cd='A') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='A')\r\n"  
 + " WHEN (rqst\_stat\_cd='NR') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='NR')\r\n"  
 + " WHEN (rqst\_stat\_cd='ND') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='ND')\r\n"  
 + " WHEN (rqst\_stat\_cd='CL') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='CL')\r\n"  
 + " WHEN (rqst\_stat\_cd='PC') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='PC')\r\n"  
 + " WHEN (rqst\_stat\_cd='LH') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='LH')\r\n"  
 + " WHEN (rqst\_stat\_cd='D') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='D')\r\n"  
 + " WHEN (rqst\_stat\_cd='RV') THEN (select type\_nam from gdp\_owner.prvcy\_rqst\_stat\_lkp where type\_cd='RV')\r\n"  
 + " END AS rqst\_stat\_cd, \r\n"  
 + " crte\_user\_id, crte\_ts, lst\_updt\_user\_id, lst\_updt\_ts, first\_nam, lst\_nam\r\n"  
 + " FROM gdp\_owner.prvcy\_rqst \r\n";  
  
 private String whereConditions = " WHERE ";  
 private String orderByCondition = "";  
 private String offsetValue = "";  
 private String limitValue = "";  
 private String sortWay = "";  
  
 @BeforeStory  
 public void truncatePrivacyRequestTables() {  
 try {  
 if (BooleanUtils.toBoolean(truncateTableSwitch)) {  
 performUpdate(QueryConstants.TRUNCATE\_SERVICE\_FUNCTION\_PRIVACY\_REQUEST);  
 performUpdate(QueryConstants.TRUNCATE\_SERVICE\_FUNCTION\_INTEGRATION);  
 performUpdate(QueryConstants.TRUNCATE\_PRIVACY\_REQUEST);  
 }  
 } catch (Exception exception) {  
 throw new GDPUIException(exception.toString());  
 }  
 }  
  
 @AfterStory  
 public void deleteTestData() {  
 try {  
 String sqlFile = null;  
  
 if (Objects.nonNull(variableTestData.getValue(SQL\_FILE))) {  
 sqlFile = variableTestData.getValue(SQL\_FILE).replace("INSERT", "DELETE");  
 if ((sqlFile != null && !sqlFile.isEmpty() && (new ClassPathResource(sqlFile).exists()))) {  
 dBOperationUtils.executeSqlScript(sqlFile);  
 turnOffMaskAndEncyptionForAllDataElements();  
 variableTestData.setContextVariable(SQL\_FILE, null);  
 }  
 }  
  
 } catch (Exception exception) {  
 log.info("Exception occured in AfterStory annotation while executing delete file: {}" , exception);  
 }  
 resetDashboardColumns();  
  
 }  
  
 private void resetDashboardColumns() {  
 commonSteps.setJsonPathToContext("/jsontestdata/aob/S555555\_TestData.json");  
 callCustomColumnBffApi(jsonTestData.getValue("ResetDashboardColumns"), "gdpsuperuser");  
 }  
  
 private void turnOffMaskAndEncyptionForAllDataElements() {  
 executeUpdateStatement(QueryConstants.TURNOFF\_MASKING\_FOR\_ALL\_DATA\_ELEMENTS);  
 executeUpdateStatement(QueryConstants.TURNOFF\_ENCRYPT\_FOR\_ALL\_DATA\_ELEMENTS);  
 }  
  
 public void turnOnOrOffMaskingForCardNumberDataElement(String enableOrDisable) {  
 executeUpdateStatement(enableOrDisable.equalsIgnoreCase("enable")  
 ? String.format(QueryConstants.TURNON\_OR\_OFF\_MASKING\_FOR\_CARD\_NUMBER\_DATA\_ELEMENT, true)  
 : String.format(QueryConstants.TURNON\_OR\_OFF\_MASKING\_FOR\_CARD\_NUMBER\_DATA\_ELEMENT, false));  
 }  
  
 public void callCustomColumnBffApi(String requestBody, String user) {  
 StringBuilder builder = new StringBuilder();  
 commonFunctions.setBaseURL("gdp.aob.app.base.url");  
 resourcePath = builder.append(LinkConstants.GDP\_AOB\_APP\_API).append(LinkConstants.PREFFERED\_DASHBOARD\_COLUMN)  
 .toString();  
 Map<String, String> headers = new HashMap<>();  
 headers.put(DEV\_USERNAME, user);  
  
 commonFunctions.executePostRequestWithCustomHeaders(resourcePath, requestBody, headers);  
 }  
  
 public List<Map<String, String>> executeSelectStmt(String query) {  
 try {  
 try (Statement stmt = conn.createStatement()) {  
 DriverManager.registerDriver(new org.postgresql.Driver());  
 return resultSetToArrayList(stmt.executeQuery(query));  
 }  
 } catch (SQLException e) {  
 log.error("SQL exception occurred ", e);  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public boolean recordExists(String query) {  
 QueryRunner run = new QueryRunner(dataSourceConfig.getDataSource());  
 try {  
 Object[] result = (Object[]) run.query(query, dataSourceConfig.getResultSetHandler());  
 if (NumberUtils.INTEGER\_ZERO < result.length) {  
 return true;  
 }  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return false;  
 }  
  
 public String executeSelectStatement(String query) {  
 QueryRunner run = new QueryRunner(dataSourceConfig.getDataSource());  
 Object[] rs;  
 StringBuilder resultValue = new StringBuilder();  
 resultValue.append("\t");  
 try {  
 rs = (Object[]) run.query(query, dataSourceConfig.getResultSetHandler());  
 for (Object value : rs) {  
 resultValue.append(value.toString());  
 }  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return resultValue.toString().trim();  
 }  
  
 public List<Map<String, String>> resultSetToArrayList(ResultSet rs) throws SQLException {  
 ResultSetMetaData md = rs.getMetaData();  
 int columns = md.getColumnCount();  
 List<Map<String, String>> list = new ArrayList<>();  
 while (rs.next()) {  
 HashMap<String, String> row = new HashMap<>();  
 for (int i = 1; i <= columns; ++i) {  
 row.put(md.getColumnName(i), rs.getString(i));  
 }  
 list.add(row);  
 }  
  
 return list;  
 }  
  
 public String getSelectQuerybyConditions(Map<String, String> localWhereConditionHash,  
 Map<String, String> localSingleConditionHash) {  
 // Construct Query in the right order of conditions. Where followed by Orderby  
 // followed by Limit, Offset  
 if (MapUtils.isEmpty(localWhereConditionHash)) {  
 localWhereConditionHash.put("1", "1");  
 }  
 Integer numOfConditions = localWhereConditionHash.size();  
 Integer currentKey = 1;  
 Set<String> whereKeys = localWhereConditionHash.keySet();  
 StringBuilder conditions = new StringBuilder();  
 // Fetch Where conditions.  
 for (String field : whereKeys) {  
 if (currentKey.equals(numOfConditions)) {  
 conditions.append(whereConditions).append(field).append("='").append(localWhereConditionHash.get(field))  
 .append("'");  
 whereConditions = conditions.toString();  
 } else {  
 conditions.append(whereConditions).append(field).append("='").append(localWhereConditionHash.get(field))  
 .append("' AND ");  
 whereConditions = conditions.toString();  
 }  
 currentKey += 1;  
 conditions.setLength(0);  
 }  
 if (!localSingleConditionHash.isEmpty()) {  
 Set<String> singleConditionKeys = localSingleConditionHash.keySet();  
 StringBuilder orderByConditions = new StringBuilder();  
 StringBuilder offsetValues = new StringBuilder();  
 StringBuilder limitValues = new StringBuilder();  
 setOrderByOffSetAndLimit(localSingleConditionHash, singleConditionKeys, orderByConditions, offsetValues,  
 limitValues);  
 }  
 String queryToReturn = BASE\_SELECT\_PRIVACY\_REQS\_QUERY + whereConditions + orderByCondition + sortWay;  
 if (!orderByCondition.isEmpty()) {  
 queryToReturn = queryToReturn + " ,lst\_updt\_ts desc " + offsetValue + limitValue;  
 } else {  
 queryToReturn = queryToReturn + " ORDER BY lst\_updt\_ts desc " + offsetValue + limitValue;  
 }  
  
 localWhereConditionHash.clear();  
 localSingleConditionHash.clear();  
 resetStringValues();  
 return queryToReturn;  
 }  
  
 private void setOrderByOffSetAndLimit(Map<String, String> localSingleConditionHash, Set<String> singleConditionKeys,  
 StringBuilder orderByConditions, StringBuilder offsetValues, StringBuilder limitValues) {  
 for (String condition : singleConditionKeys) {  
 switch (condition.toUpperCase()) {  
 case "ORDERBY":  
 orderByCondition = orderByConditions.append(orderByCondition).append(" ORDER BY ")  
 .append(localSingleConditionHash.get(condition)).append(" ").toString();  
 break;  
 case "OFFSET":  
 offsetValue = offsetValues.append(offsetValue).append(" OFFSET ")  
 .append(Integer.parseInt(localSingleConditionHash.get(condition))  
 \* Integer.parseInt(localSingleConditionHash.get("LIMIT")))  
 .append(" ROWS ").toString();  
 break;  
 case "LIMIT":  
 limitValue = limitValues.append(limitValue).append(" FETCH NEXT ")  
 .append(localSingleConditionHash.get(condition)).append(" ROWS ONLY").toString();  
 break;  
 case "SORTWAY":  
 sortWay = " " + localSingleConditionHash.get(condition);  
 break;  
 default: // default clause should be the last one  
 break;  
 }  
 orderByConditions.setLength(0);  
 offsetValues.setLength(0);  
 limitValues.setLength(0);  
 }  
 }  
  
 private void resetStringValues() {  
 whereConditions = " WHERE ";  
 orderByCondition = "";  
 offsetValue = "";  
 limitValue = "";  
 sortWay = "";  
  
 }  
  
 public void update(String query) {  
 QueryRunner run = new QueryRunner(dataSourceConfig.getDataSource());  
 try {  
 run.update(query);  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public int getCountOfRecords(String query) {  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 int numberOfRecords = -1;  
 try {  
 Object[] result = (Object[]) queryRunner.query(query, dataSourceConfig.getResultSetHandler());  
 long recordsCount = (long) result[0];  
 numberOfRecords = (int) recordsCount;  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return numberOfRecords;  
 }  
  
 public void executeInsertStatement(String query) {  
 log.info("INSERT\_QUERY : {}", query);  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 try {  
 queryRunner.insert(query, dataSourceConfig.getResultSetHandler());  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public int executeDeleteStatement(String query) {  
 log.info("DELETE\_QUERY : {}", query);  
 return executeUpdate(query);  
 }  
  
 public int executeUpdateStatement(String query) {  
 log.info("UPDATE\_QUERY : {}", query);  
 return executeUpdate(query);  
 }  
  
 private int executeUpdate(String query) {  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 int numberOfRecordsEffected = -1;  
 try {  
 queryRunner.insert(query, dataSourceConfig.getResultSetHandler());  
 numberOfRecordsEffected = queryRunner.update(query);  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return numberOfRecordsEffected;  
 }  
  
 public String getSingleDbValue(String query) {  
 log.info("Query to execute {}", query);  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 String value = null;  
 try {  
 Object[] result = (Object[]) queryRunner.query(query, dataSourceConfig.getResultSetHandler());  
 if (result != null && result[0] != null) {  
 value = result[0].toString();  
 }  
 } catch (SQLException e) {  
 log.error("Exception occurred while fetching single value from db", e);  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return value;  
 }  
  
 public void performUpdate(String query) {  
 log.info("UPDATE\_QUERY : {}", query);  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 try {  
 queryRunner.update(query);  
 } catch (SQLException e) {  
 log.error("Exception occurred while updating the records: ", e);  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public <T> List<T> getCompleteRowData(String query, Class clazz) {  
 Object obj;  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 try {  
 obj = clazz.newInstance();  
 return queryRunner.query(query, (ResultSetHandler<List<T>>) obj);  
 } catch (InstantiationException | IllegalAccessException | SQLException e) {  
 log.error(String.format("Exception occurred : %s", e.getMessage()));  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public <T> List<T> getAllDbValuesForColumn(String query, String columnName) {  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 List<T> returnList = null;  
 ColumnListHandler<T> resultSetHandler = new ColumnListHandler<>(columnName);  
 try {  
 returnList = queryRunner.query(query, resultSetHandler);  
 } catch (SQLException e) {  
 log.error(String.format("Exception occurred : %s", e.getMessage()));  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return returnList;  
 }  
  
 public void executeSqlScript(String sqlFilePath) throws SQLException {  
 try {  
 log.info("Connection object is {}", conn);  
 ScriptRunner scriptRunner = new ScriptRunner(conn, false, true);  
 InputStream inputStream = this.getClass().getResourceAsStream(sqlFilePath);  
 Reader reader = new InputStreamReader(inputStream, StandardCharsets.UTF\_8);  
 scriptRunner.runScript(reader);  
 conn.setAutoCommit(false);  
 conn.commit();  
 } catch (Exception e) {  
 log.error(String.format("Failed to Execute %s The error is %s", sqlFilePath, e));  
 try {  
 conn.rollback();  
 } catch (SQLException sqlException) {  
 log.error("SQL Rollback exception :", sqlException);  
 }  
 throw new GDPUIException(e.getMessage());  
 }  
 }  
  
 @BeforeScenario(uponType = ScenarioType.ANY)  
 public void setTheStage() {  
 log.info("@@@@@@@@@@@ BEFORE SCENARIO @@@@@@@@@@@");  
 }  
  
 public <T> List<T> getAllDbValuesForAColumn(String query, String columnName) {  
 QueryRunner queryRunner = new QueryRunner(dataSourceConfig.getDataSource());  
 List<T> returnList = null;  
 ColumnListHandler<T> resultSetHandler = new ColumnListHandler<>(columnName);  
 try {  
 returnList = queryRunner.query(query, resultSetHandler);  
 } catch (SQLException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return returnList;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.exception;  
  
public class GDPAPIException extends RuntimeException {  
  
 public GDPAPIException(String message) {  
 super(message);  
 }  
  
 public GDPAPIException(String message, Exception exp) {  
 super(message, exp);  
 }  
}

package com.mastercard.testing.gdp.ui.tests.exception;  
  
public class GDPUIException extends RuntimeException {  
  
 private static final long serialVersionUID = 1L;  
  
 public GDPUIException(String message) {  
 super(message);  
 }  
  
 public GDPUIException(String message, Exception exp) {  
 super(message, exp);  
 }  
}

package com.mastercard.testing.gdp.ui.tests.helper.datagrid;  
  
import lombok.Data;  
import lombok.EqualsAndHashCode;  
  
@Data  
@EqualsAndHashCode  
public class ActionHistoryGrid {  
  
 private String date;  
 private String action;  
 private String internalComment;  
 private String externalComment;  
 private String commentBy;  
  
 public ActionHistoryGrid(String date, String action, String internalComment, String externalComment,  
 String commentBy) {  
 super();  
 this.date = date;  
 this.action = action;  
 this.internalComment = internalComment;  
 this.externalComment = externalComment;  
 this.commentBy = commentBy;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.helper.datagrid;  
  
import lombok.Data;  
  
import java.util.Comparator;  
import java.util.Date;  
  
@Data  
public class DeleteProductResultsGridRecord {  
 private String product;  
 private int records;  
 private String status;  
 private Date dateVerified;  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_STATUS\_ASCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> o1.getStatus().compareTo(o2.getStatus());  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_STATUS\_DESCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> (o1.getStatus().compareTo(o2.getStatus())) \* (-1);  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_RECORDS\_ASCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> o1.getRecords() - (o2.getRecords());  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_RECORDS\_DESCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> (o1.getRecords() - (o2.getRecords())) \* (-1);  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_PRODUCT\_ASCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> o1.getProduct().compareTo(o2.getProduct());  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_PRODUCT\_DESCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> (o1.getProduct().compareTo(o2.getProduct())) \* (-1);  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_DATE\_VERIFIED\_ASCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> o1.getDateVerified().compareTo(o2.getDateVerified());  
  
 public static final Comparator<DeleteProductResultsGridRecord> *SORT\_DATE\_VERIFIED\_DESCENDING* = (  
 DeleteProductResultsGridRecord o1,  
 DeleteProductResultsGridRecord o2) -> (o1.getDateVerified().compareTo(o2.getDateVerified())) \* (-1);  
  
 public DeleteProductResultsGridRecord(String product, int records, String status, Date dateVerified) {  
 this.product = product;  
 this.records = records;  
 this.status = status;  
 this.dateVerified = dateVerified;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.helper.datagrid;  
  
import lombok.Data;  
  
import java.util.Date;  
  
@Data  
public class DiscloseHistoryGrid {  
  
 private Date date;  
 private String action;  
 private String internal;  
 private String messageToRequester;  
 private String commentBy;  
  
 public DiscloseHistoryGrid(Date date, String action, String internal, String messageToRequester, String commentBy) {  
 super();  
 this.date = date;  
 this.action = action;  
 this.internal = internal;  
 this.messageToRequester = messageToRequester;  
 this.commentBy = commentBy;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.helper.datagrid;  
  
import lombok.Data;  
  
@Data  
public class DoNotDiscloseHistoryGrid {  
  
 private String date;  
 private String action;  
 private String internal;  
 private String commentBy;  
  
 public DoNotDiscloseHistoryGrid(String date, String action, String internal, String commentBy) {  
 super();  
 this.date = date;  
 this.action = action;  
 this.internal = internal;  
 this.commentBy = commentBy;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.helper.datagrid;  
  
import lombok.Data;  
  
import java.util.Comparator;  
  
@Data  
public class ProductResultsGridRecord {  
  
 private String record;  
 private String status;  
 private String doNotDisclose;  
 private String dateReceived;  
 private String dateReviewed;  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_STATUS\_ASCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> o1.getStatus().compareTo(o2.getStatus());  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_STATUS\_DESCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> (o1.getStatus().compareTo(o2.getStatus())) \* (-1);  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_RECORD\_ASCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> o1.getRecord().compareTo(o2.getRecord());  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_RECORD\_DESCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> (o1.getRecord().compareTo(o2.getRecord())) \* (-1);  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_DISCLOSE\_ASCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> o1.getDoNotDisclose().compareTo(o2.getDoNotDisclose());  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_DISCLOSE\_DESCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> (o1.getDoNotDisclose().compareTo(o2.getDoNotDisclose())) \* (-1);  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_RECEIVED\_ASCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> o1.getDateReceived().compareTo(o2.getDateReceived());  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_RECEIVED\_DESCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> (o1.getDateReceived().compareTo(o2.getDateReceived())) \* (-1);  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_REVIEWED\_ASCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> o1.getDateReviewed().compareTo(o2.getDateReviewed());  
  
 public static final Comparator<ProductResultsGridRecord> SORT\_REVIEWED\_DESCENDING = (ProductResultsGridRecord o1,  
 ProductResultsGridRecord o2) -> (o1.getDateReviewed().compareTo(o2.getDateReviewed())) \* (-1);  
  
 public ProductResultsGridRecord(String record, String status, String doNotDisclose, String dateReceived,  
 String dateReviewed) {  
 super();  
 this.record = record;  
 this.status = status;  
 this.doNotDisclose = doNotDisclose;  
 this.dateReceived = dateReceived;  
 this.dateReviewed = dateReviewed;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.helper;  
  
@FunctionalInterface  
public interface AbstractPageGDP {  
 boolean isLoaded();  
}

package com.mastercard.testing.gdp.ui.tests.helper;  
  
import com.mastercard.testing.gdp.ui.framework.GDPDriverProviderFactory;  
import com.mastercard.testing.gdp.ui.tests.pages.pagecomponents.PageFooter;  
import com.mastercard.testing.gdp.ui.tests.pages.pagecomponents.PageHeader;  
import com.mastercard.testing.gdp.ui.tests.steps.CommonSteps;  
import com.mastercard.testing.gdp.ui.tests.utils.CommonFunctions;  
import com.mastercard.testing.gdp.ui.tests.utils.WebDriverUtils;  
import org.apache.commons.lang3.StringUtils;  
import org.apache.log4j.Logger;  
import org.springframework.beans.factory.annotation.Autowired;  
  
public abstract class GDPBasePage {  
  
 @Autowired  
 public PageHeader pageHeader;  
  
 @Autowired  
 public PageFooter pageFooter;  
  
 @Autowired  
 public WebDriverUtils webDriverUtils;  
  
 @Autowired  
 protected GDPDriverProviderFactory gdpDriverProviderFactory;  
  
 @Autowired  
 protected CommonFunctions commonFunctions;  
  
 @Autowired  
 protected CustomWaiters customWaiters;  
  
 @Autowired  
 protected CommonSteps commonSteps;  
  
 protected Logger logger = Logger.getLogger(this.getClass());  
 private String suffixForFirst = "0";  
 private String suffixForSecond = "1";  
 private String suffixForThird = "2";  
  
 public void open(String pageURL) {  
 webDriverUtils.openLinkAndMaximizeWindow(pageURL);  
 }  
  
 public void openWithLocale(String pageURL, String localeToUse) {  
 String baseUrlWithLocale = pageURL;  
 if (!localeToUse.isEmpty()) {  
 baseUrlWithLocale = pageURL.replace("en-us.html", localeToUse + ".html");  
 }  
 this.open(baseUrlWithLocale);  
 }  
  
 public String getPositionSuffix(String position) {  
 switch (position.toUpperCase()) {  
 case "FIRST":  
 return this.suffixForFirst;  
 case "SECOND":  
 return this.suffixForSecond;  
 case "THIRD":  
 return this.suffixForThird;  
 default:  
 return StringUtils.EMPTY;  
 }  
 }  
   
}

package com.mastercard.testing.gdp.ui.tests.helper;  
  
import com.google.common.base.Function;  
import com.mastercard.testing.gdp.ui.framework.GDPDriverProviderFactory;  
import com.mastercard.testing.gdp.ui.tests.constants.UIConstants;  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
import com.mastercard.testing.gdp.ui.tests.utils.WebDriverUtils;  
  
import io.appium.java\_client.functions.ExpectedCondition;  
import org.apache.log4j.Logger;  
import org.assertj.core.api.Assertions;  
import org.checkerframework.checker.nullness.qual.Nullable;  
import org.openqa.selenium.By;  
import org.openqa.selenium.JavascriptExecutor;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.ui.ExpectedConditions;  
import org.openqa.selenium.support.ui.WebDriverWait;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import java.util.List;  
  
@Component  
public class CustomWaiters {  
  
 @Autowired  
 private GDPDriverProviderFactory gdpDriverProviderfactory;  
  
 @Autowired  
 private WebDriverUtils webDriverUtils;  
  
 private final Logger logger = Logger.*getLogger*(CustomWaiters.class);  
  
 @Value("${element.finder.wait.timeout\_in\_secs:90}")  
 private int elementWaitTime;  
  
 public int getElementWaitTime() {  
 return elementWaitTime;  
 }  
  
 public WebDriver getDriver() {  
 return gdpDriverProviderfactory.getDriverProvider().get();  
 }  
  
 public void waitForFunction(final Function function) {  
 waitForFunction(function, elementWaitTime);  
 }  
  
 public void waitForFunction(final Function function, int sec) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), sec);  
 try {  
 wait.ignoring(Throwable.class).until(function);  
 } catch (Exception exp) {  
 logger.debug(exp.getMessage());  
 throw new GDPUIException(exp.getCause().getMessage(), exp);  
 }  
 }  
  
 public WebElement getVisibleElement(By locator) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), elementWaitTime);  
 return wait.until(ExpectedConditions.*visibilityOfElementLocated*(locator));  
 }  
  
 public WebElement getVisibleElement(By by, int timeOutInSeconds) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), timeOutInSeconds);  
 return wait.until(ExpectedConditions.*visibilityOfElementLocated*(by));  
 }  
  
 public boolean waitInvisibility(By by) {  
 return waitInvisibilityTime(by, elementWaitTime);  
 }  
  
 private boolean waitInvisibilityTime(By by, int timeOutInSeconds) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), timeOutInSeconds);  
 return wait.until(ExpectedConditions.*invisibilityOfElementLocated*(by));  
 }  
  
 public WebElement waitUntilElementClickable(By element) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), elementWaitTime);  
 return wait.until(ExpectedConditions.*elementToBeClickable*(element));  
 }  
  
 public WebElement waitUntilElementClickable(WebElement element) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), elementWaitTime);  
 return wait.until(ExpectedConditions.*elementToBeClickable*(element));  
 }  
  
 public WebElement getClickableElement(By by) {  
 webDriverUtils.waitForAngularToLoad();  
 WebDriverWait wait = new WebDriverWait(getDriver(), elementWaitTime);  
 wait.until(ExpectedConditions.*visibilityOfElementLocated*(by));  
 return wait.until(ExpectedConditions.*elementToBeClickable*(by));  
 }  
  
 public Boolean waitUntilElementContainsValue(By by, String value) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), elementWaitTime);  
 return wait.until(ExpectedConditions.*attributeContains*(by, UIConstants.*VALUE\_ATTRIBUTE*, value));  
 }  
  
 public List<WebElement> getVisibleElements(By locator) {  
 return getVisibleElements(locator, elementWaitTime);  
 }  
  
 public List<WebElement> getVisibleElements(By by, int timeOutInSeconds) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), timeOutInSeconds);  
 return wait.until(ExpectedConditions.*visibilityOfAllElementsLocatedBy*(by));  
 }  
  
 public List<WebElement> waitForDropdownValueToAppear(By locator) {  
 return waitForDropdownValueToAppear(locator, elementWaitTime);  
 }  
  
 public List<WebElement> waitForDropdownValueToAppear(By locator, int timeOutInSeconds) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), timeOutInSeconds);  
 return wait.until(ExpectedConditions.*presenceOfNestedElementsLocatedBy*(locator, By.*tagName*("option")));  
 }  
  
 public List<WebElement> waitForNestedElements(By locator, int timeOutInSeconds, int numberOfExpectedElements) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), timeOutInSeconds);  
 return wait.until(ExpectedConditions.*numberOfElementsToBeMoreThan*(locator, numberOfExpectedElements));  
 }  
  
 public boolean waitElementSelectionStateToBe(By locator, boolean selected) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), elementWaitTime);  
 return wait.until(ExpectedConditions.*elementSelectionStateToBe*(locator, selected));  
 }  
  
 public List<WebElement> getElements(By by) {  
 return getDriver().findElements(by);  
 }  
  
 public WebElement getElement(By by) {  
 WebDriver webDriver = getDriver();  
 waitForFunction(function -> {  
 ((JavascriptExecutor) getDriver()).executeScript("arguments[0].scrollIntoView(true);",  
 webDriver.findElement(by));  
 Assertions.*assertThat*(webDriver.findElements(by)).isNotEmpty();  
 return true;  
 }, elementWaitTime);  
 return webDriver.findElement(by);  
 }  
  
 public boolean verifyElementNotPresent(By by) {  
 return getElements(by).isEmpty();  
 }  
  
 public boolean verifyElementNotVisible(By by) {  
 return getVisibleElements(by).isEmpty();  
 }  
  
 public void waitForAngularToLoad() {  
 waitForAngularToLoad(elementWaitTime);  
 }  
  
 public void waitForAngularToLoad(int timeOutInSeconds) {  
 webDriverUtils.waitForAngularToLoad(timeOutInSeconds);  
 }  
  
 public void waitForAttributeValue(By by, String attribute, String value) {  
 new WebDriverWait(getDriver(), elementWaitTime)  
 .until(ExpectedConditions.attributeContains(by, attribute, value));  
 }  
  
 public WebElement getWebElement(By locator) {  
 return getDriver().findElement(locator);  
 }  
  
 public WebElement getPresenceOfElement(By locator) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), elementWaitTime);  
 return wait.until(ExpectedConditions.presenceOfElementLocated(locator));  
 }  
  
 public void waitForElementTextTobePopulated(By locator){  
 new WebDriverWait(getDriver(), getElementWaitTime()).until(new ExpectedCondition<Boolean>() {  
 @Nullable  
 @Override  
 public Boolean apply(@Nullable WebDriver webDriver) {  
 return getDriver().findElement(locator).getText().length()>0;  
 }  
 });  
 }  
}

package com.mastercard.testing.gdp.ui.tests.model;  
  
import java.util.LinkedList;  
import java.util.List;  
  
public class Scenario {  
  
 private String title;  
 private List<Step> steps;  
 private Result result;  
 private Throwable exception;  
  
 public Scenario(String title, Result result) {  
 this.title = title;  
 this.result = result;  
 this.steps = new LinkedList<>();  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public List<Step> getSteps() {  
 return steps;  
 }  
  
 public Result getResult() {  
 return result;  
 }  
  
 public void setResult(Result result) {  
 this.result = result;  
 }  
  
 public Throwable getException() {  
 return exception;  
 }  
  
 public void setException(Throwable exception) {  
 this.exception = exception;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.model;  
  
public class Step {  
  
 private String description;  
 private Result result;  
  
 public Step(String description, Result result) {  
 this.description = description;  
 this.result = result;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public Result getResult() {  
 return result;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.model;  
  
import java.util.ArrayDeque;  
import java.util.Deque;  
  
public class Story {  
  
 private org.jbehave.core.model.Story jbehaveStory;  
 private Deque<Scenario> scenarios;  
 private Result result;  
  
 public Story(org.jbehave.core.model.Story story) {  
 this.jbehaveStory = story;  
 this.result = Result.NOT\_EXECUTED;  
 this.scenarios = new ArrayDeque<>();  
 }  
  
 public Result getResult() {  
 return result;  
 }  
  
 public void setResult(Result result) {  
 this.result = result;  
 }  
  
 public org.jbehave.core.model.Story getStory() {  
 return jbehaveStory;  
 }  
  
 public String getName() {  
 return jbehaveStory.getName().split("[.]")[0];  
 }  
  
 public void addScenario(Scenario scenario) {  
 this.scenarios.addFirst(scenario);  
 }  
  
 public Scenario getLastScenario() {  
 return scenarios.peekFirst();  
 }  
  
 public Scenario getLastFailedScenario() {  
 for (Scenario scenario : scenarios) {  
 if(scenario.getResult().equals(Result.FAILED)){  
 return scenario;  
 }  
 }  
  
 return null;  
 }  
  
 public Scenario removeLastScenario() {  
 Scenario scenario = null;  
 if(!scenarios.isEmpty()) {  
 scenario = scenarios.removeFirst();  
 }  
 return scenario;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.model;  
  
public enum Result {  
 *PASSED*,  
 *FAILED*,  
 *NOT\_EXECUTED*}

package com.mastercard.testing.gdp.ui.tests.reporters;  
  
import org.jbehave.core.model.\*;  
import org.jbehave.core.reporters.StoryReporter;  
import org.jbehave.core.steps.StepCollector;  
  
import java.util.List;  
import java.util.Map;  
  
abstract class BaseReporter implements StoryReporter {  
  
 @Override  
 public void beforeStory(Story story, boolean givenStory) {}  
  
 @Override  
 public void storyNotAllowed(Story story, String filter) {}  
  
 @Override  
 public void storyCancelled(Story story, StoryDuration storyDuration) {}  
  
 @Override  
 public void afterStory(boolean givenOrRestartingStory) {}  
  
 @Override  
 public void narrative(Narrative narrative) {}  
  
 @Override  
 public void lifecyle(Lifecycle lifecycle) {}  
  
 @Override  
 public void scenarioNotAllowed(Scenario scenario, String filter) {}  
  
 @Override  
 public void beforeScenario(Scenario var1) {}  
  
 @Override  
 public void beforeScenario(String scenarioTitle) {}  
  
 @Override  
 public void scenarioMeta(Meta meta) {}  
  
 @Override  
 public void afterScenario() {}  
  
 @Override  
 public void beforeGivenStories() {}  
  
 @Override  
 public void givenStories(GivenStories givenStories) {}  
  
 @Override  
 public void givenStories(List<String> storyPaths) {}  
  
 @Override  
 public void afterGivenStories() {}  
  
 @Override  
 public void beforeExamples(List<String> steps, ExamplesTable table) {}  
  
 @Override  
 public void example(Map<String, String> tableRow) {}  
  
 @Override  
 public void afterExamples() {}  
  
 @Override  
 public void beforeStep(String step) {}  
  
 @Override  
 public void successful(String step) {}  
  
 @Override  
 public void ignorable(String step) {}  
  
 @Override  
 public void comment(String step) {}  
  
 @Override  
 public void pending(String step) {}  
  
 @Override  
 public void notPerformed(String step) {}  
  
 @Override  
 public void failed(String step, Throwable cause) {}  
  
 @Override  
 public void failedOutcomes(String step, OutcomesTable table) {}  
  
 @Override  
 public void restarted(String step, Throwable cause) {}  
  
 @Override  
 public void restartedStory(Story story, Throwable cause) {}  
  
 @Override  
 public void dryRun() {}  
  
 @Override  
 public void pendingMethods(List<String> methods) {}  
 @Override  
 public void beforeStorySteps(StepCollector.Stage stage) {}  
  
 @Override  
 public void afterStorySteps(StepCollector.Stage stage) {}  
  
 @Override  
 public void beforeScenarioSteps(StepCollector.Stage stage) {}  
  
 @Override  
 public void afterScenarioSteps(StepCollector.Stage stage) {}  
  
 @Override  
 public void example(Map<String, String> map, int i) {}  
  
}

package com.mastercard.testing.gdp.ui.tests.reporters;  
  
import com.mastercard.testing.gdp.ui.tests.model.\*;  
import com.mastercard.testing.gdp.ui.tests.model.Result;  
import org.apache.log4j.Logger;  
import org.springframework.stereotype.Component;  
import java.io.File;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.nio.file.StandardOpenOption;  
  
@Component  
public class FailedStoriesReporter extends BaseReporter {  
 public static final String REPORT\_FILE\_PATH = "./target/FailedStories.txt";  
 public static final String PATHS\_BREAKER = ",";  
  
 private static final Logger LOGGER = Logger.getLogger(FailedStoriesReporter.class);  
  
 private ThreadLocal<Story> currentStory = new ThreadLocal<>();  
  
 public FailedStoriesReporter() {  
 File file = new File(REPORT\_FILE\_PATH);  
 try {  
 Files.deleteIfExists(file.toPath());  
 } catch (IOException ignored) {  
 LOGGER.debug(String.format("Could not delete file '%s'", REPORT\_FILE\_PATH));  
 }  
 }  
  
 @Override  
 public void beforeStory(org.jbehave.core.model.Story story, boolean givenStory) {  
 if (!givenStory) {  
 this.currentStory.set(new Story(story));  
 }  
 }  
  
 @Override  
 public void failed(String step, Throwable cause) {  
 currentStory.get().setResult(Result.FAILED);  
 }  
  
 @Override  
 public void afterStory(boolean givenOrRestartingStory) {  
 if (currentStory.get().getResult() == Result.FAILED && !givenOrRestartingStory) {  
 saveFailedStoryName(currentStory.get().getStory().getName());  
 currentStory.remove();  
 }  
 }  
  
 private void saveFailedStoryName(String name) {  
 if (Paths.get(REPORT\_FILE\_PATH).toFile().exists()) {  
 try {  
 Files.write(Paths.get(REPORT\_FILE\_PATH), ("," + name).getBytes(), StandardOpenOption.APPEND, StandardOpenOption.SYNC);  
 } catch (IOException e) {  
 LOGGER.error(e.getCause());  
 }  
 }  
 else {  
 try {  
 Files.write(Paths.get(REPORT\_FILE\_PATH), name.getBytes());  
 } catch (IOException e) {  
 LOGGER.error(e.getCause());  
 }  
 }  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.reporters;  
  
import org.apache.log4j.Logger;  
  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.ALMService;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportScenario.Verdict;  
import com.mastercard.quality.engineering.mtaf.ui.reporters.RallySeleniumStoryReporter;  
import com.mastercard.quality.engineering.mtaf.ui.screenshots.CustomWebDriverScreenshotOnFailure;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportScenario;  
  
public class GdpRallySeleniumStoryReporter extends RallySeleniumStoryReporter {  
 public GdpRallySeleniumStoryReporter(ALMService rallyService,  
 CustomWebDriverScreenshotOnFailure onFailureScreenshooter,  
 RallySeleniumStoryReporter.ScreenShotMode screenShotMode, boolean createDefectOnFailure) {  
 super(rallyService, onFailureScreenshooter, screenShotMode, createDefectOnFailure);  
 this.initialReporterSetup();  
 }  
   
 protected final Logger logger = Logger.getLogger(this.getClass());  
  
 @Override  
 public void notPerformed(String step) {  
 super.notPerformed(step);  
 (this.almScenarioContext.get()).setVerdict(Verdict.Inconclusive);  
 }  
   
 @Override  
 public synchronized void updateTestCase(RallyReportScenario scenario) {  
 try {  
 super.updateTestCase(scenario);  
 } catch (IllegalStateException ie) {  
 logger.info("Exception caught while pushing result to ALM as testcase not exist : {}", ie);  
 }  
 }  
}

package com.mastercard.testing.gdp.ui.tests.reporters;  
  
import com.epam.reportportal.jbehave.JBehaveUtils;  
import org.jbehave.core.model.StoryMaps;  
import org.jbehave.core.reporters.FreemarkerViewGenerator;  
  
import java.io.File;  
import java.util.List;  
import java.util.Properties;  
import java.util.concurrent.atomic.AtomicBoolean;  
  
public class RPFreemarkerViewGenerator extends FreemarkerViewGenerator {  
 private final AtomicBoolean finished;  
  
 public RPFreemarkerViewGenerator() {  
 finished = new AtomicBoolean(false);  
 }  
  
 @Override  
 public void generateMapsView(File outputDirectory, StoryMaps storyMaps, Properties viewResources) {  
 super.generateMapsView(outputDirectory, storyMaps, viewResources);  
 if (finished.compareAndSet(false, true)) {  
 JBehaveUtils.finishLaunch();  
 }  
 }  
  
 @Override  
 public void generateReportsView(File outputDirectory, List<String> formats, Properties viewResources) {  
 super.generateReportsView(outputDirectory, formats, viewResources);  
 if (finished.compareAndSet(false, true)) {  
 JBehaveUtils.finishLaunch();  
 }  
 }  
  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.ui.reporters;  
  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.RallyStoryReporter;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.ALMService;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportScenario;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportStory;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportScenario.Verdict;  
import com.mastercard.quality.engineering.mtaf.ui.screenshots.CustomWebDriverScreenshotOnFailure;  
import com.mastercard.quality.engineering.mtaf.ui.screenshots.RallyScreenshotAfterScenario;  
import java.util.LinkedList;  
import java.util.Map;  
import java.util.UUID;  
  
public class RallySeleniumStoryReporter extends RallyStoryReporter {  
 private RallySeleniumStoryReporter.ScreenShotMode screenShotMode;  
 private CustomWebDriverScreenshotOnFailure onFailureScreenshooter;  
 private RallyScreenshotAfterScenario afterScenarioScreenshooter = null;  
  
 public RallySeleniumStoryReporter(ALMService almService, CustomWebDriverScreenshotOnFailure onFailureScreenshooter) {  
 super(almService);  
 this.onFailureScreenshooter = onFailureScreenshooter;  
 }  
  
 public RallySeleniumStoryReporter(ALMService rallyService, CustomWebDriverScreenshotOnFailure onFailureScreenshooter, RallySeleniumStoryReporter.ScreenShotMode screenShotMode, boolean createDefectOnFailure) {  
 super(rallyService);  
 this.screenShotMode = screenShotMode;  
 this.onFailureScreenshooter = onFailureScreenshooter;  
 if (screenShotMode == RallySeleniumStoryReporter.ScreenShotMode.*ALWAYS*) {  
 this.afterScenarioScreenshooter = new RallyScreenshotAfterScenario(onFailureScreenshooter.getDriverProvider());  
 }  
  
 }  
  
 public void example(Map<String, String> tableRow) {  
 if (this.examplesContext.get() != null && !((LinkedList)this.examplesContext.get()).isEmpty()) {  
 this.afterScenarioScreenShooting();  
 super.example(tableRow);  
 } else {  
 super.example(tableRow);  
 }  
 }  
  
 public void afterExamples() {  
 this.afterScenarioScreenShooting();  
 super.afterExamples();  
 }  
  
 public void afterScenario() {  
 this.afterScenarioScreenShooting();  
 super.afterScenario();  
 }  
  
 private void afterScenarioScreenShooting() {  
 switch(this.screenShotMode) {  
 case *ON\_FAILURE*:  
 this.setScreenshotPathToScenario(false);  
 break;  
 case *ALWAYS*:  
 this.setScreenshotPathToScenario(true);  
 }  
  
 }  
  
 private boolean setScreenshotPathToScenario(boolean always) {  
 if (this.almScenarioContext.get() != null && ((RallyReportScenario)this.almScenarioContext.get()).getTestCaseIDs() != null && !((RallyReportScenario)this.almScenarioContext.get()).getTestCaseIDs().isEmpty()) {  
 String path;  
 if (((RallyReportScenario)this.almScenarioContext.get()).getVerdict().equals(Verdict.*Fail*)) {  
 path = this.onFailureScreenshooter.getScreenshotPath();  
 ((RallyReportScenario)this.almScenarioContext.get()).setScreenshotPath(path);  
 if (this.isDefectCreationEnabled() && this.storyBugContext.get() != null) {  
 ((RallyReportStory)this.storyBugContext.get()).getLastFailedScenario().setScreenshotPath(path);  
 }  
  
 return true;  
 } else if (always) {  
 path = this.afterScenarioScreenshooter.saveScreenshot(UUID.*randomUUID*().toString());  
 ((RallyReportScenario)this.almScenarioContext.get()).setScreenshotPath(path);  
 return true;  
 } else {  
 return false;  
 }  
 } else {  
 return false;  
 }  
 }  
  
 public static enum ScreenShotMode {  
 *NONE*,  
 *ON\_FAILURE*,  
 *ALWAYS*;  
  
 private ScreenShotMode() {  
 }  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.jbehave.reporters;  
  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.ALMService;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportScenario;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportStory;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyReportScenario.Verdict;  
import java.util.Iterator;  
import java.util.LinkedList;  
import java.util.Map;  
import org.jbehave.core.configuration.Keywords;  
import org.jbehave.core.model.Meta;  
import org.jbehave.core.model.Scenario;  
import org.jbehave.core.model.Story;  
  
public class RallyStoryReporter extends LoggingStoryReporter {  
 protected ALMService almService;  
 protected ThreadLocal<LinkedList<RallyReportScenario>> examplesContext = new ThreadLocal();  
 protected ThreadLocal<RallyReportStory> storyBugContext = new ThreadLocal();  
 protected ThreadLocal<RallyReportScenario> almScenarioContext = new ThreadLocal();  
 protected ThreadLocal<Scenario> jBehaveScenarioContext = new ThreadLocal();  
  
 public RallyStoryReporter(ALMService almService) {  
 this.almService = almService;  
 this.initialReporterSetup();  
 }  
  
 protected void initialReporterSetup() {  
 this.almService.createTestSet();  
 }  
  
 public void beforeScenario(String scenarioTitle) {  
 super.beforeScenario(scenarioTitle);  
 if (this.storyThreadLocal.get() != null) {  
 Scenario scenario = this.findScenario((Story)this.storyThreadLocal.get(), scenarioTitle);  
 this.jBehaveScenarioContext.set(scenario);  
 this.almScenarioContext.set(new RallyReportScenario(scenario));  
 } else {  
 this.jBehaveScenarioContext.set((Object)null);  
 this.almScenarioContext.set((Object)null);  
 }  
  
 }  
  
 public void example(Map<String, String> tableRow) {  
 Story story = (Story)this.storyThreadLocal.get();  
 Scenario scenario = (Scenario)this.jBehaveScenarioContext.get();  
 if (story != null) {  
 Meta meta = new Meta();  
 if (tableRow.containsKey("Meta:")) {  
 meta = meta.inheritFrom(Meta.*createMeta*((String)tableRow.get("Meta:"), new Keywords()));  
 }  
  
 meta = meta.inheritFrom(((Scenario)this.jBehaveScenarioContext.get()).getMeta());  
 meta = meta.inheritFrom(story.getMeta());  
 if (this.examplesContext.get() == null) {  
 this.examplesContext.set(new LinkedList());  
 }  
  
 Scenario exampleScenario = new Scenario(scenario.getTitle(), meta, scenario.getGivenStories(), scenario.getExamplesTable(), scenario.getSteps());  
 RallyReportScenario rallyReportScenario = new RallyReportScenario(exampleScenario);  
 rallyReportScenario.setExamples(tableRow);  
 this.almScenarioContext.set(rallyReportScenario);  
 if (meta.hasProperty("TestId")) {  
 ((LinkedList)this.examplesContext.get()).offer(rallyReportScenario);  
 }  
 }  
  
 }  
  
 public void afterExamples() {  
 while(!((LinkedList)this.examplesContext.get()).isEmpty()) {  
 this.updateTestCase((RallyReportScenario)((LinkedList)this.examplesContext.get()).poll());  
 }  
  
 this.examplesContext.set((Object)null);  
 this.almScenarioContext.set((Object)null);  
 }  
  
 public void scenarioNotAllowed(Scenario scenario, String filter) {  
 super.scenarioNotAllowed(scenario, filter);  
 this.almScenarioContext.set((Object)null);  
 }  
  
 public void failed(String step, Throwable cause) {  
 super.failed(step, cause);  
 if (this.almScenarioContext != null) {  
 ((RallyReportScenario)this.almScenarioContext.get()).setVerdict(Verdict.*Fail*);  
 ((RallyReportScenario)this.almScenarioContext.get()).setCause(cause.getMessage());  
 if (this.storyBugContext.get() == null) {  
 this.storyBugContext.set(new RallyReportStory((Story)this.storyThreadLocal.get()));  
 }  
  
 ((RallyReportStory)this.storyBugContext.get()).addFailedScenario((RallyReportScenario)this.almScenarioContext.get());  
 }  
  
 }  
  
 public void afterScenario() {  
 if (this.almScenarioContext.get() != null) {  
 if (((RallyReportScenario)this.almScenarioContext.get()).getVerdict() == null) {  
 ((RallyReportScenario)this.almScenarioContext.get()).setVerdict(Verdict.*Pass*);  
 }  
  
 if (!((RallyReportScenario)this.almScenarioContext.get()).getTestCaseIDs().isEmpty()) {  
 this.updateTestCase((RallyReportScenario)this.almScenarioContext.get());  
 }  
 }  
  
 }  
  
 protected void reportAfterStory(Story story) {  
 super.reportAfterStory(story);  
 if (this.isDefectCreationEnabled()) {  
 RallyReportStory bugContext = (RallyReportStory)this.storyBugContext.get();  
 if (bugContext != null && !this.almService.checkDefectExists(bugContext)) {  
 this.almService.createDefect(bugContext);  
 }  
 }  
  
 this.storyBugContext.set((Object)null);  
 }  
  
 public void successful(String step) {  
 super.successful(step);  
 if (((RallyReportScenario)this.almScenarioContext.get()).getVerdict() == null) {  
 ((RallyReportScenario)this.almScenarioContext.get()).setVerdict(Verdict.*Pass*);  
 }  
  
 }  
  
 public void pending(String step) {  
 super.pending(step);  
 ((RallyReportScenario)this.almScenarioContext.get()).setVerdict(Verdict.*Inconclusive*);  
 }  
  
 protected boolean isDefectCreationEnabled() {  
 return this.almService.getConfiguration().isDefectsCreationEnabled();  
 }  
  
 private Scenario findScenario(Story story, String scenarioTitle) {  
 Iterator var3 = story.getScenarios().iterator();  
  
 Scenario scenario;  
 do {  
 if (!var3.hasNext()) {  
 return null;  
 }  
  
 scenario = (Scenario)var3.next();  
 } while(!scenario.getTitle().equals(scenarioTitle));  
  
 return scenario;  
 }  
  
 protected void updateTestCase(RallyReportScenario scenario) {  
 this.almService.addTestCasesToTestSet(scenario);  
 this.almService.updateTestCasesTag(scenario);  
 this.almService.updateTestCase(scenario);  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.jbehave.reporters;  
  
import java.util.List;  
import java.util.Map;  
import org.apache.log4j.Logger;  
import org.jbehave.core.failures.UUIDExceptionWrapper;  
import org.jbehave.core.model.ExamplesTable;  
import org.jbehave.core.model.GivenStories;  
import org.jbehave.core.model.Lifecycle;  
import org.jbehave.core.model.Meta;  
import org.jbehave.core.model.Narrative;  
import org.jbehave.core.model.OutcomesTable;  
import org.jbehave.core.model.Scenario;  
import org.jbehave.core.model.Story;  
import org.jbehave.core.model.StoryDuration;  
import org.jbehave.core.reporters.NullStoryReporter;  
  
public class LoggingStoryReporter extends NullStoryReporter {  
 protected final Logger log = Logger.*getLogger*(this.getClass());  
 protected ThreadLocal<Boolean> runningStoryStatus = new ThreadLocal();  
 protected ThreadLocal<Story> storyThreadLocal = new ThreadLocal();  
  
 public LoggingStoryReporter() {  
 }  
  
 public void storyNotAllowed(Story story, String filter) {  
 this.log.info(String.*format*("%s (NOT ALLOWED [filter: %s])", story, filter));  
 }  
  
 public void storyCancelled(Story story, StoryDuration storyDuration) {  
 this.log.info(String.*format*("%s (CANCELLED [duration: %s])", story, storyDuration));  
 }  
  
 public void beforeStory(Story story, boolean givenStory) {  
 if (!givenStory) {  
 this.storyThreadLocal.set(story);  
 if (!story.getName().equals("BeforeStories") && !story.getName().equals("AfterStories")) {  
 this.runningStoryStatus.set(true);  
 this.reportBeforeStory(story);  
 }  
  
 }  
 }  
  
 protected void reportBeforeStory(Story story) {  
 this.log.info("=====================================================");  
 this.log.info("Begin Story: " + story.getName());  
 this.log.info("=====================================================");  
 }  
  
 public void afterStory(boolean givenStory) {  
 if (!givenStory) {  
 Story story = (Story)this.storyThreadLocal.get();  
 if (story == null) {  
 this.log.warn("Story has not been set!");  
 } else {  
 if (story.getName() != null && !story.getName().equals("BeforeStories") && !story.getName().equals("AfterStories")) {  
 this.reportAfterStory((Story)this.storyThreadLocal.get());  
 }  
  
 }  
 }  
 }  
  
 protected void reportAfterStory(Story story) {  
 String status = (Boolean)this.runningStoryStatus.get() ? " PASSED " : " FAILED ";  
 this.log.info("====================================================");  
 this.log.info("End Story: " + story.getName());  
 this.log.info("Status: " + status);  
 this.log.info("====================================================");  
 }  
  
 public void narrative(Narrative narrative) {  
 this.log.info(narrative);  
 }  
  
 public void lifecyle(Lifecycle lifecycle) {  
 this.log.info(lifecycle);  
 }  
  
 public void scenarioNotAllowed(Scenario scenario, String filter) {  
 this.log.info(String.*format*("%s (NOT ALLOWED [filter: %s])", scenario, filter));  
 }  
  
 public void beforeScenario(String scenarioTitle) {  
 this.log.info(scenarioTitle);  
 }  
  
 public void scenarioMeta(Meta meta) {  
 this.log.info(meta);  
 }  
  
 public void afterScenario() {  
 }  
  
 public void givenStories(GivenStories givenStories) {  
 }  
  
 public void givenStories(List<String> storyPaths) {  
 }  
  
 public void beforeExamples(List<String> steps, ExamplesTable table) {  
 }  
  
 public void example(Map<String, String> tableRow) {  
 }  
  
 public void afterExamples() {  
 }  
  
 public void beforeStep(String step) {  
 }  
  
 public void successful(String step) {  
 this.log.info(String.*format*("%s (SUCCESSFUL)", step));  
 }  
  
 public void ignorable(String step) {  
 this.log.info(String.*format*("%s (IGNORED)", step));  
 }  
  
 public void pending(String step) {  
 this.log.info(String.*format*("%s (PENDING)", step));  
 }  
  
 public void notPerformed(String step) {  
 this.log.info(String.*format*("%s (NOT PERFORMED)", step));  
 }  
  
 public void failed(String step, Throwable cause) {  
 this.runningStoryStatus.set(false);  
 String exceptionClass = "";  
 String exceptionMessage = "";  
 if (cause instanceof UUIDExceptionWrapper) {  
 exceptionClass = cause.getCause().getClass().getName();  
 exceptionMessage = cause.getCause().getMessage();  
 } else {  
 exceptionClass = cause.getClass().getName();  
 exceptionMessage = cause.getMessage();  
 }  
  
 this.log.error(String.*format*("%s (FAILED), cause was: %s:%s", step, exceptionClass, exceptionMessage));  
 if (Boolean.*parseBoolean*(System.*getProperty*("verbose", "false"))) {  
 cause.printStackTrace();  
 }  
  
 }  
  
 public void failedOutcomes(String step, OutcomesTable table) {  
 }  
  
 public void restarted(String step, Throwable cause) {  
 }  
  
 public void dryRun() {  
 }  
  
 public void pendingMethods(List<String> methods) {  
 }  
  
 public void comment(String arg0) {  
 }  
  
 public void restartedStory(Story arg0, Throwable arg1) {  
 }  
  
 public void afterGivenStories() {  
 }  
  
 public void beforeGivenStories() {  
 }  
  
 public void beforeScenario(Scenario arg0) {  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally;  
  
import com.google.common.collect.ImmutableList;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Iterator;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
import org.jbehave.core.model.Scenario;  
  
public class RallyReportScenario {  
 public static final String *TEST\_CASE\_ID* = "TestId";  
 public static final String *TITLE\_NOT\_DEFINED* = "FAILED TO GET SCENARIO";  
 private String title;  
 private Map<String, String> examples;  
 private List<String> steps;  
 private String cause;  
 private String screenshotPath;  
 private List<String> testCaseIDs;  
 private RallyReportScenario.Verdict verdict;  
 private Scenario scenario = null;  
  
 public RallyReportScenario(Scenario scenario) {  
 this.scenario = scenario;  
 if (scenario == null) {  
 this.title = "FAILED TO GET SCENARIO";  
 this.steps = ImmutableList.*of*();  
 this.testCaseIDs = new LinkedList();  
 } else {  
 this.title = scenario.getTitle();  
 this.steps = scenario.getSteps();  
 this.testCaseIDs = this.parseScenarioTestCaseIDs(scenario);  
 }  
  
 this.cause = null;  
 this.examples = null;  
 this.screenshotPath = null;  
 this.verdict = null;  
 }  
  
 private List<String> parseScenarioTestCaseIDs(Scenario scenario) {  
 String testcaseMeta = scenario.getMeta().getProperty("TestId");  
 return (List)(testcaseMeta.isEmpty() ? new ArrayList() : Arrays.*asList*(testcaseMeta.split(",")));  
 }  
  
 public Scenario getScenario() {  
 return this.scenario;  
 }  
  
 public String getTitle() {  
 return this.title;  
 }  
  
 public void setExamples(Map<String, String> examples) {  
 this.examples = examples;  
 }  
  
 public void setExamples(Map<String, String> examples, Set<String> metaTags) {  
 if (metaTags == null) {  
 this.setExamples(examples);  
 } else {  
 Map<String, String> filterTable = new HashMap();  
 Iterator var4 = examples.keySet().iterator();  
  
 while(var4.hasNext()) {  
 String key = (String)var4.next();  
 if (!metaTags.contains(key)) {  
 filterTable.put(key, examples.get(key));  
 }  
 }  
  
 this.examples = filterTable;  
 }  
 }  
  
 public Map<String, String> getExamples() {  
 return this.examples;  
 }  
  
 public void setCause(String cause) {  
 this.cause = cause;  
 }  
  
 public String getCause() {  
 return this.cause;  
 }  
  
 public List<String> getSteps() {  
 return this.steps;  
 }  
  
 public String getScreenshotPath() {  
 return this.screenshotPath;  
 }  
  
 public void setScreenshotPath(String screenshotPath) {  
 this.screenshotPath = screenshotPath;  
 }  
  
 public List<String> getTestCaseIDs() {  
 return this.testCaseIDs;  
 }  
  
 public void setTestCaseIDs(List<String> testCaseIDs) {  
 this.testCaseIDs = testCaseIDs;  
 }  
  
 public RallyReportScenario.Verdict getVerdict() {  
 return this.verdict;  
 }  
  
 public void setVerdict(RallyReportScenario.Verdict verdict) {  
 this.verdict = verdict;  
 }  
  
 public static enum Verdict {  
 *Blocked*,  
 *Error*,  
 *Fail*,  
 *Inconclusive*,  
 *Pass*;  
  
 private Verdict() {  
 }  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally;  
  
import java.util.Iterator;  
import java.util.LinkedList;  
import java.util.List;  
import org.jbehave.core.model.Narrative;  
import org.jbehave.core.model.Story;  
  
public class RallyReportStory {  
 private static final String *NL* = "<br>";  
 private static final String *BOLD* = "<b>";  
 private static final String *bold* = "</b>";  
 private static final String *TABLE* = "<table>";  
 private static final String *table* = "</table>";  
 private static final String *TR* = "<tr>";  
 private static final String *tr* = "</tr>";  
 private static final String *TH* = "<th>";  
 private static final String *th* = "</th>";  
 private static final String *SPACE* = " ";  
 private static final String *DOT* = ".";  
 private static final String *NARRATIVE* = "Narrative:";  
 private static final String *SCENARIO* = "Scenario";  
 private static final String *asA* = "As a ";  
 private static final String *iWantTo* = "I want to ";  
 private static final String *soThat* = "So that ";  
 private static final String *TA* = "[TA]";  
 public static final String *NAME\_PROPERTY* = "name";  
 public static final String *PROJECT\_PROPERTY* = "Project";  
 private String name;  
 private String projectKey = "";  
 private Narrative narrative;  
 private List<RallyReportScenario> failedScenarios;  
  
 public RallyReportStory(Story story) {  
 this.name = this.parseStoryName(story);  
 this.projectKey = this.getStoryProjectKey(story);  
 this.narrative = story.getNarrative();  
 this.failedScenarios = new LinkedList();  
 }  
  
 private String getStoryProjectKey(Story story) {  
 String projectKey = story.getMeta().getProperty("Project");  
 return projectKey;  
 }  
  
 private String parseStoryName(Story story) {  
 String name = story.getMeta().getProperty("name");  
 if (name.isEmpty()) {  
 name = story.getName();  
 }  
  
 StringBuilder str = new StringBuilder();  
 return str.append("[TA]").append(" ").append(name).toString();  
 }  
  
 public String getName() {  
 return this.name;  
 }  
  
 public String getStoryProjectKey() {  
 return this.projectKey;  
 }  
  
 public Narrative getNarrative() {  
 return this.narrative;  
 }  
  
 public List<RallyReportScenario> getFailedScenarios() {  
 return this.failedScenarios;  
 }  
  
 public RallyReportScenario getLastFailedScenario() {  
 return (RallyReportScenario)this.failedScenarios.get(this.failedScenarios.size() - 1);  
 }  
  
 public void addFailedScenario(RallyReportScenario scenario) {  
 this.failedScenarios.add(scenario);  
 }  
  
 public String getDescription() {  
 StringBuilder str = new StringBuilder();  
 Narrative narrative = this.getNarrative();  
 str.append("<b>").append("Narrative:").append("</b>").append("<br>");  
 str.append("As a ").append(narrative.asA()).append("<br>").append("I want to ").append(narrative.iWantTo()).append("<br>").append("So that ").append(narrative.soThat()).append("<br>").append("<br>");  
 int scenarioCont = 1;  
  
 for(Iterator var4 = this.failedScenarios.iterator(); var4.hasNext(); ++scenarioCont) {  
 RallyReportScenario scenario = (RallyReportScenario)var4.next();  
 str.append("<b>").append("Scenario").append(" ").append(scenarioCont).append(".").append(" ").append(scenario.getTitle()).append("</b>").append("<br>");  
 Iterator var6 = scenario.getSteps().iterator();  
  
 String value;  
 while(var6.hasNext()) {  
 value = (String)var6.next();  
 str.append(value).append("<br>");  
 }  
  
 str.append("<b>").append("Example:").append("</b>").append("<br>");  
 if (scenario.getExamples() != null) {  
 str.append("<table>");  
 str.append("<tr>");  
 var6 = scenario.getExamples().keySet().iterator();  
  
 while(var6.hasNext()) {  
 value = (String)var6.next();  
 str.append("<th>").append(value).append("</th>");  
 }  
  
 str.append("</tr>");  
 str.append("<tr>");  
 var6 = scenario.getExamples().values().iterator();  
  
 while(var6.hasNext()) {  
 value = (String)var6.next();  
 str.append("<th>").append(value).append("</th>");  
 }  
  
 str.append("</tr>");  
 str.append("</table>");  
 }  
  
 str.append("<br>");  
 }  
  
 return str.toString();  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.ui.screenshots;  
  
import java.util.UUID;  
import org.jbehave.core.annotations.AfterScenario;  
import org.jbehave.core.annotations.ScenarioType;  
import org.jbehave.core.annotations.AfterScenario.Outcome;  
import org.jbehave.core.failures.UUIDExceptionWrapper;  
import org.jbehave.web.selenium.WebDriverProvider;  
import org.jbehave.web.selenium.WebDriverScreenshotOnFailure;  
  
public class CustomWebDriverScreenshotOnFailure extends WebDriverScreenshotOnFailure {  
 private ThreadLocal<String> screenshotPath;  
 private WebDriverProvider driverProvider;  
  
 public CustomWebDriverScreenshotOnFailure(WebDriverProvider driverProvider) {  
 super(driverProvider);  
 this.driverProvider = driverProvider;  
 this.screenshotPath = new ThreadLocal();  
 }  
  
 public WebDriverProvider getDriverProvider() {  
 return this.driverProvider;  
 }  
  
 protected String screenshotPath(UUID uuid) {  
 String path = super.screenshotPath(uuid);  
 this.savePath(path);  
 return path;  
 }  
  
 public void savePath(String path) {  
 this.screenshotPath.set(path);  
 }  
  
 public String getScreenshotPath() {  
 return (String)this.screenshotPath.get();  
 }  
  
 @AfterScenario(  
 uponOutcome = Outcome.*FAILURE*,  
 uponType = ScenarioType.*ANY* )  
 public void afterScenarioFailure(UUIDExceptionWrapper uuidWrappedFailure) throws Exception {  
 super.afterScenarioFailure(uuidWrappedFailure);  
 }  
}

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
package com.mastercard.quality.engineering.mtaf.ui.screenshots;  
  
import java.io.File;  
import java.text.MessageFormat;  
import org.jbehave.core.reporters.StoryReporterBuilder;  
import org.jbehave.web.selenium.WebDriverProvider;  
import org.jbehave.web.selenium.RemoteWebDriverProvider.SauceLabsJobHasEnded;  
  
public class RallyScreenshotAfterScenario {  
 public static final String *DEFAULT\_SCREENSHOT\_NAME* = "TestCasePassedScreenshot";  
 public static final String *DEFAULT\_SCREENSHOT\_PATH\_PATTERN* = "{0}/screenshots/scenario-{1}.png";  
 protected final StoryReporterBuilder reporterBuilder;  
 protected final String screenshotPathPattern;  
 protected final WebDriverProvider driverProvider;  
  
 public RallyScreenshotAfterScenario(WebDriverProvider driverProvider) {  
 this.driverProvider = driverProvider;  
 this.reporterBuilder = new StoryReporterBuilder();  
 this.screenshotPathPattern = "{0}/screenshots/scenario-{1}.png";  
 }  
  
 public String saveScreenshot() {  
 return this.saveScreenshot("TestCasePassedScreenshot");  
 }  
  
 public String saveScreenshot(String name) {  
 String screenshotPath = MessageFormat.*format*(this.screenshotPathPattern, this.reporterBuilder.outputDirectory(), name);  
 String currentUrl = "[unknown page title]";  
  
 try {  
 currentUrl = this.driverProvider.get().getCurrentUrl();  
 } catch (Exception var10) {  
 }  
  
 boolean savedIt = false;  
  
 try {  
 savedIt = this.driverProvider.saveScreenshotTo(screenshotPath);  
 } catch (SauceLabsJobHasEnded var8) {  
 System.*err*.println("[Rally] Screenshot of page '" + currentUrl + "' has \*\*NOT\*\* been saved. The SauceLabs job has ended, possibly timing out on their end.");  
 return null;  
 } catch (Exception var9) {  
 System.*out*.println("[Rally] Screenshot of page '" + currentUrl + ". Will try again. Cause: " + var9.getMessage());  
  
 try {  
 savedIt = this.driverProvider.saveScreenshotTo(screenshotPath);  
 } catch (Exception var7) {  
 System.*err*.println("[Rally] Screenshot of page '" + currentUrl + "' has \*\*NOT\*\* been saved to '" + screenshotPath + "' because error '" + var9.getMessage() + "' encountered. Stack trace follows:");  
 var9.printStackTrace();  
 return null;  
 }  
 }  
  
 if (savedIt) {  
 System.*out*.println("[Rally] Screenshot of page '" + currentUrl + "' has been saved to '" + screenshotPath + "' with " + (new File(screenshotPath)).length() + " bytes");  
 } else {  
 System.*err*.println("[Rally] Screenshot of page '" + currentUrl + "' has \*\*NOT\*\* been saved. If there is no error, perhaps the WebDriver type you are using is not compatible with taking screenshots");  
 }  
  
 return screenshotPath;  
 }  
}

package com.mastercard.testing.gdp.ui.tests.stories;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.nio.charset.StandardCharsets;  
  
import org.apache.commons.io.IOUtils;  
import org.jbehave.core.io.InvalidStoryResource;  
import org.jbehave.core.io.LoadFromClasspath;  
  
public class GdpStoryLoader extends LoadFromClasspath {  
  
 @Override  
 public String loadResourceAsText(String resourcePath) {  
 InputStream stream = resourceAsStream(resourcePath);  
 try {  
 return IOUtils.*toString*(stream, StandardCharsets.*ISO\_8859\_1*);  
 } catch (IOException ioe) {  
 throw new InvalidStoryResource(resourcePath, stream, ioe);  
 }  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.stories;  
  
import com.epam.reportportal.jbehave.ReportPortalFormat;  
import com.github.valfirst.jbehave.junit.monitoring.JUnitReportingRunner;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.ALMService;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyIntegrationConfiguration;  
import com.mastercard.quality.engineering.mtaf.jbehave.reporters.alm.rally.RallyService;  
import com.mastercard.quality.engineering.mtaf.ui.configuration.MastercardUIStories;  
import com.mastercard.quality.engineering.mtaf.ui.reporters.RallySeleniumStoryReporter;  
import com.mastercard.quality.engineering.mtaf.ui.screenshots.CustomWebDriverScreenshotOnFailure;  
import com.mastercard.testing.gdp.ui.tests.configuration.TestConfiguration;  
import com.mastercard.testing.gdp.ui.tests.reporters.FailedStoriesReporter;  
import com.mastercard.testing.gdp.ui.tests.reporters.GdpRallySeleniumStoryReporter;  
import com.mastercard.testing.gdp.ui.tests.reporters.RPFreemarkerViewGenerator;  
import org.apache.commons.lang3.StringUtils;  
import org.jbehave.core.configuration.Configuration;  
import org.jbehave.core.configuration.MostUsefulConfiguration;  
import org.jbehave.core.embedder.StoryControls;  
import org.jbehave.core.io.CodeLocations;  
import org.jbehave.core.io.StoryFinder;  
import org.jbehave.core.reporters.CrossReference;  
import org.jbehave.core.reporters.Format;  
import org.jbehave.core.reporters.StoryReporter;  
import org.jbehave.core.reporters.StoryReporterBuilder;  
import org.jbehave.core.steps.ParameterControls;  
import org.junit.runner.RunWith;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
import java.net.URISyntaxException;  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Date;  
import java.util.List;  
  
@RunWith(JUnitReportingRunner.class)  
public class GdpExampleWebStories extends MastercardUIStories {  
  
 public GdpExampleWebStories() {  
 context = getAnnotatedApplicationContext();  
 }  
  
 @Override  
 public ApplicationContext getAnnotatedApplicationContext() {  
 return new AnnotationConfigApplicationContext(TestConfiguration.class);  
 }  
  
 @Override  
 public List<String> storyPaths() {  
 List paths = null;  
 String storyName = System.*getProperty*("storyName");  
 if (StringUtils.*isNotEmpty*(storyName)) {  
 if (storyName.endsWith(".story") && !storyName.contains(",")) {  
 paths = (new StoryFinder()).findPaths(CodeLocations.*codeLocationFromClass*(this.getClass()),  
 "\*\*/stories/\*\*/" + storyName, "");  
 } else if (storyName.endsWith(".story") && storyName.contains(",")) {  
 List<String> stories = Arrays.*asList*(storyName.split(","));  
 List<String> includes = new ArrayList<>();  
 stories.stream().forEach(story -> includes.add("\*\*/stories/\*\*/" + story));  
 paths = (new StoryFinder()).findPaths(CodeLocations.*codeLocationFromClass*(this.getClass()), includes,  
 new ArrayList<>());  
 } else {  
 paths = (new StoryFinder()).findPaths(CodeLocations.*codeLocationFromClass*(this.getClass()),  
 Arrays.*asList*("\*\*/stories/folder".replace("folder", storyName + "/\*.story")),  
 Arrays.*asList*("\*\*/excluded/\*\*/\*.story", "\*\*/\*.txt"));  
 }  
 } else {  
 paths = (new StoryFinder()).findPaths(CodeLocations.*codeLocationFromClass*(this.getClass()),  
 "\*\*/stories/\*\*/\*.story", "\*\*/stories/exclude/\*\*/\*.story");  
 }  
 return paths;  
 }  
  
 @Override  
 public Configuration configuration() {  
 Configuration configuration;  
 configuration = new MostUsefulConfiguration()  
 .useStoryControls(  
 new StoryControls().doResetStateBeforeScenario(false).doSkipScenariosAfterFailure(true))  
 .useStoryLoader(new GdpStoryLoader())  
 .useStoryReporterBuilder(this.getStoryReporterBuilder())  
 .useViewGenerator(new RPFreemarkerViewGenerator())  
 .useParameterControls((new ParameterControls()).useDelimiterNamedParameters(true))  
 .useParameterConverters(getConverters());  
  
 return configuration;  
 }  
  
 private boolean isRPEnabled() {  
 return (System.getProperty("rp.enable") != null && Boolean.valueOf(System.getProperty("rp.enable")));  
 }  
  
 @Override  
 protected Format[] storyFormat() {  
 Format[] defaultFormats = super.storyFormat();  
 if (isRPEnabled()) {  
 List<Format> formatList = new ArrayList<>(Arrays.asList(defaultFormats));  
 formatList.add(ReportPortalFormat.INSTANCE);  
 return formatList.toArray(new Format[formatList.size()]);  
 } else {  
 return defaultFormats;  
 }  
 }  
  
 @Override  
 protected RallySeleniumStoryReporter getRallyReport() {  
 CustomWebDriverScreenshotOnFailure onFailureScreenshotter = context  
 .getBean(CustomWebDriverScreenshotOnFailure.class);  
 RallySeleniumStoryReporter.ScreenShotMode screenShotMode = context  
 .getBean(RallySeleniumStoryReporter.ScreenShotMode.class);  
 RallyIntegrationConfiguration rallyIntegrationConfiguration = context  
 .getBean(RallyIntegrationConfiguration.class);  
  
 String testSetDescription = getTestSetDescription(rallyIntegrationConfiguration);  
  
 rallyIntegrationConfiguration.withTestSetDesc(testSetDescription);  
 ALMService rallyALMService = null;  
 try {  
 rallyALMService = new RallyService(rallyIntegrationConfiguration);  
 } catch (URISyntaxException e) {  
 LOG.error("URI Syntax Error for Rally ALM Service", e);  
 }  
 return new GdpRallySeleniumStoryReporter(rallyALMService, onFailureScreenshotter, screenShotMode,  
 rallyIntegrationConfiguration.isDefectsCreationEnabled());  
 }  
  
 @Override  
 protected StoryReporterBuilder getStoryReporterBuilder() {  
 StoryReporterBuilder storyReporterBuilder;  
 storyReporterBuilder = ((new StoryReporterBuilder()).withFormats(this.storyFormat())  
 .withReporters(this.getReporters()).withFailureTraceCompression(true)  
 .withCrossReference(new CrossReference()));  
  
 return storyReporterBuilder;  
 }  
  
 private String getRallyTestSetDescription() {  
 String testSetName = System.getProperty("rally.testSetDesc");  
 if (testSetName != null)  
 return testSetName;  
 else  
 return "";  
 }  
  
 private String getTestSetDescription(RallyIntegrationConfiguration rallyIntegrationConfiguration) {  
 String testSetDescription = "Regression";  
 String runtimeTestSetDescription = this.getRallyTestSetDescription();  
 if (!runtimeTestSetDescription.isEmpty()) {  
 testSetDescription = runtimeTestSetDescription;  
 } else if (!rallyIntegrationConfiguration.getTestSetDesc().isEmpty()) {  
 testSetDescription = rallyIntegrationConfiguration.getTestSetDesc();  
 }  
  
 if (!testSetDescription.contains("\_")) {  
 String dateFormat = "MM-dd-yyyy";  
 SimpleDateFormat simpleDateFormat = new SimpleDateFormat(dateFormat);  
 String currentDateTime = simpleDateFormat.format(new Date());  
 testSetDescription = String.format("%s\_%s", testSetDescription, currentDateTime);  
 }  
 LOG.info("Test Set Name: " + testSetDescription);  
 return testSetDescription;  
 }  
  
 @Override  
 protected StoryReporter[] getReporters() {  
 StoryReporter[] storyReporters = super.getReporters();  
 List<StoryReporter> storyReportersList = new ArrayList<>(Arrays.asList(storyReporters));  
 storyReportersList.add(new FailedStoriesReporter());  
 return storyReportersList.toArray(new StoryReporter[storyReportersList.size()]);  
 }  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import com.fasterxml.jackson.core.JsonParseException;  
import com.fasterxml.jackson.databind.DeserializationFeature;  
import com.fasterxml.jackson.databind.JsonMappingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.mastercard.apie.axon.sdk.core.AxonMessage;  
import com.mastercard.quality.engineering.mtaf.jbehave.context.TestContextProvider;  
import com.mastercard.testing.gdp.ui.framework.GDPDriverProviderFactory;  
import com.mastercard.testing.gdp.ui.tests.constants.ContextConstants;  
import com.mastercard.testing.gdp.ui.tests.constants.LinkConstants;  
import com.mastercard.testing.gdp.ui.tests.constants.LinkConstantsB2C;  
import com.mastercard.testing.gdp.ui.tests.constants.UIConstants;  
import com.mastercard.testing.gdp.ui.tests.constants.UIStrings;  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
import com.mastercard.testing.gdp.ui.tests.helper.CustomWaiters;  
import io.restassured.builder.RequestSpecBuilder;  
import io.restassured.http.ContentType;  
import io.restassured.http.Cookies;  
import io.restassured.response.Response;  
import io.restassured.specification.RequestSpecification;  
import org.apache.commons.lang3.math.NumberUtils;  
import org.apache.commons.lang3.RandomStringUtils;  
import org.apache.commons.lang3.StringUtils;  
import org.apache.http.HttpResponse;  
import org.apache.http.client.HttpClient;  
import org.apache.http.client.methods.HttpPost;  
import org.apache.http.entity.StringEntity;  
import org.apache.http.impl.client.HttpClients;  
import org.openqa.selenium.By;  
import org.openqa.selenium.Cookie;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.ui.ExpectedConditions;  
import org.openqa.selenium.support.ui.WebDriverWait;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.core.env.Environment;  
import org.springframework.stereotype.Service;  
  
import java.io.File;  
import java.io.IOException;  
import java.math.BigInteger;  
import java.nio.charset.Charset;  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.sql.Timestamp;  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.time.\*;  
import java.time.format.DateTimeFormatter;  
import java.time.format.DateTimeParseException;  
import java.time.temporal.ChronoUnit;  
import java.util.\*;  
import java.util.regex.Pattern;  
import java.util.stream.Collectors;  
  
import static com.mastercard.testing.gdp.ui.tests.constants.ContextConstants.*RESPONSE*;  
import static com.mastercard.testing.gdp.ui.tests.constants.ContextConstants.*ALGORITHM*;  
import static com.mastercard.testing.gdp.ui.tests.constants.ContextConstants.*SALTSTRING*;  
import static com.mastercard.testing.gdp.ui.tests.constants.UIConstants.*BY*;  
import static io.restassured.RestAssured.*given*;  
import static org.assertj.core.api.Assertions.*assertThat*;  
  
@Service("commonFunctions")  
public class CommonFunctions {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(CommonFunctions.class);  
  
 private static final int *RADIX* = 16;  
 private static final int *SIGNUM* = 1;  
 private static final String *XSRF\_TOKEN\_HEADER\_KEY* = "X-XSRF-TOKEN";  
 protected String baseJSONUrl;  
 private static final String *APPLICATION\_JSON* = "application/json";  
 private static final String *HEADER\_NAME\_CONTENT\_TYPE* = "Content-Type";  
 private static final String HEADER\_NAME\_ACCCEPT = "Accept";  
 private static final String DATE\_FORMATTER\_WITH\_LOCALE\_CST = "dd MMM YYYY 'at' hh:mm a 'US CST'";  
 private static final String B2C\_CREATED\_DATE\_FORMATTER = "dd MMM yyyy 'at' hh:mm a 'US CST'";  
 private int year = Calendar.getInstance().get(Calendar.YEAR);  
 private String currentYear = Integer.toString(year);  
  
 private List<String> devCloundEnvironments = Arrays.asList("stablestage", "production");  
 private Random rand = new Random();  
  
 @Autowired  
 private Environment env;  
 @Autowired  
 private ObjectMapper mapper;  
 @Autowired  
 protected TestContextProvider testContextProvider;  
 @Autowired  
 private GDPDriverProviderFactory gdpDriverProviderfactory;  
 @Autowired  
 private CustomWaiters customWaiters;  
  
 @Value("${rest.response.unwraproot:false}")  
 private Boolean unwrapRootValue;  
  
 @Value("${gdp.manual.url}")  
 private String stageAobBaseURL;  
   
 @Value("${gdp.b2c.web.base.url}")  
 private String b2cBaseUrl;  
  
 @Value("${aker.proxy.per.request.timeout}")  
 private String akerProxyPerRequestTimeout;  
  
 private LocalDateTime timeBeforeTestBegin;  
 private LocalDateTime timeAfterTestEnd;  
 private HttpResponse httpResponse;  
  
 public void setBaseURL(String baseURL) {  
 baseJSONUrl = baseURL;  
 }  
  
 public RequestSpecification getReqSpecBuilder(String resPath) {  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl)).setBasePath(resPath)  
 .addHeader(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON)  
 .addHeader(HEADER\_NAME\_ACCCEPT, APPLICATION\_JSON)  
 .addHeader(LinkConstants.CORRELATION\_ID, getCurrentDate())  
 .addHeader(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout).build();  
 }  
  
 public RequestSpecification getReqSpecBuilderWithEmptyBody(String resPath) {  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl)).setBasePath(resPath).setBody("")  
 .addHeader(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON)  
 .addHeader(HEADER\_NAME\_ACCCEPT, APPLICATION\_JSON)  
 .addHeader(LinkConstants.CORRELATION\_ID, getCurrentDate())  
 .addHeader(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout).build();  
 }  
  
 public RequestSpecification getGetRequestBuilderWithQueryParams(String resourcePath, String reqParam,  
 Map<String, Object> queryParam) {  
 StringBuilder resPathSB = new StringBuilder(resourcePath);  
 String resPath;  
 String reqParameter = reqParam;  
 if (reqParameter == null || reqParameter.equals("")) {  
 resPath = resPathSB.toString();  
 } else {  
 resPath = resPathSB.append(reqParameter).toString();  
 }  
  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl)).setBasePath(resPath)  
 .addQueryParams(queryParam).build();  
 }  
  
 public RequestSpecification getGetRequestBuilder(String resourcePath, String reqParam) {  
 StringBuilder resPathSB = new StringBuilder(resourcePath);  
 String reqParameter = reqParam;  
 String resPath;  
 if (reqParameter == null || reqParameter.equals("")) {  
 resPath = resPathSB.toString();  
 } else {  
 resPath = resPathSB.append(reqParameter).toString();  
 }  
 return getReqSpecBuilder(resPath);  
 }  
  
 public RequestSpecification getGetRequestBuilderWithHeaders(String resourcePath, String reqParam,  
 Map<String, String> headers) {  
 StringBuilder resPathSB = new StringBuilder(resourcePath);  
 String reqParameter = reqParam;  
 String resPath;  
 if (reqParameter == null || reqParameter.equals("")) {  
 resPath = resPathSB.toString();  
 } else {  
 resPath = resPathSB.append(reqParameter).toString();  
 }  
 headers.put(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON);  
 headers.put(HEADER\_NAME\_ACCCEPT, APPLICATION\_JSON);  
 headers.put(LinkConstants.CORRELATION\_ID, getCurrentDate());  
 headers.put(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout);  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl)).addHeaders(headers)  
 .setBasePath(resPath).build();  
 }  
  
 public RequestSpecification getGetRequestBuilderWithDefaultAndGivenHeadersAndQueryParams(String resourcePath,  
 String reqParam, Map<String, String> headers, Map<String, Object> queryParam) {  
 StringBuilder resPathSB = new StringBuilder(resourcePath);  
 String resPath;  
 String reqParameter = reqParam;  
 if (reqParameter == null || reqParameter.equals("")) {  
 resPath = resPathSB.toString();  
 } else {  
 resPath = resPathSB.append(reqParameter).toString();  
 }  
 headers.put(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON);  
 headers.put(HEADER\_NAME\_ACCCEPT, APPLICATION\_JSON);  
 headers.put(LinkConstants.CORRELATION\_ID, getCurrentDate());  
 headers.put(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout);  
  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl)).setBasePath(resPath)  
 .addHeaders(headers).addQueryParams(queryParam).build();  
 }  
  
 public RequestSpecification getPostRequestBuilderWithCustomHeaders(String resourcePath, String requestBody,  
 String pathToUrl, Map<String, String> headers) {  
 RequestSpecBuilder specBuilder = getRequestSpecBuilder(resourcePath, null, pathToUrl);  
 return specBuilder.addHeaders(headers).setBody(requestBody).build();  
 }  
  
 public RequestSpecification getPostRequestBuilderWithCustomHeaders(String resourcePath, String requestBody,  
 String... reqHeader) {  
 Map<String, String> listOfHeaders = new HashMap<>();  
 for (int count = 0; count < reqHeader.length; count++) {  
 String key = reqHeader[count].split("=", 2)[0];  
 String value = reqHeader[count].split("=", 2)[1];  
 listOfHeaders.put(key, value);  
 }  
  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl)).setBasePath(resourcePath)  
 .setBody(requestBody).addHeaders(listOfHeaders)  
 .addHeader(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON)  
 .addHeader(LinkConstants.CORRELATION\_ID, getCurrentDate())  
 .addHeader(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout).build();  
  
 }  
  
 public RequestSpecification getPostRequestBuilder(String resourcePath, String requestBody, String pathToUrl) {  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(pathToUrl)).setBasePath(resourcePath)  
 .setBody(requestBody).addHeader(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON)  
 .addHeader(LinkConstants.CORRELATION\_ID, getCurrentDate())  
 .addHeader(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout).build();  
 }  
  
 public RequestSpecification getPutRequestBuilder(String resourcePath, String reqParam) {  
 return getDeleteRequestBuilder(resourcePath, reqParam);  
 }  
  
 public RequestSpecification getPutRequestBuilder(String resourcePath) {  
 return getReqSpecBuilderWithEmptyBody(resourcePath);  
 }  
  
 public RequestSpecification getPutRequestBuilder(String resourcePath, int reqParam, String requestBody) {  
 String resPath = resourcePath;  
 if (reqParam != 0) {  
 resPath = resourcePath + reqParam;  
 }  
 return new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl)).setBasePath(resPath)  
 .setBody(requestBody).addHeader(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON)  
 .addHeader(LinkConstants.CORRELATION\_ID, getCurrentDate())  
 .addHeader(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout).build();  
 }  
  
 public RequestSpecification getDeleteRequestBuilder(String resourcePath, String reqParam) {  
 StringBuilder resPathSB = new StringBuilder(resourcePath);  
 String resPath;  
 String reqParameter = reqParam;  
 if (reqParameter == null || reqParameter.equals("")) {  
 resPath = resPathSB.toString();  
 } else {  
 resPath = resPathSB.append(reqParameter).toString();  
 }  
 return getReqSpecBuilder(resPath);  
 }  
  
 public RequestSpecBuilder getRequestSpecBuilder(String resourcePath, String reqParam, String pathToUrl) {  
 StringBuilder resPathSB = new StringBuilder(resourcePath);  
 String resPath;  
 String reqParameter = reqParam;  
 if (reqParameter == null || reqParameter.equals("")) {  
 resPath = resPathSB.toString();  
 } else {  
 resPath = resPathSB.append(reqParameter).toString();  
 }  
  
 return new RequestSpecBuilder().setBaseUri(env.getProperty((pathToUrl==null)?baseJSONUrl:pathToUrl)).setBasePath(resPath)  
 .addHeader(HEADER\_NAME\_CONTENT\_TYPE, APPLICATION\_JSON)  
 .addHeader(HEADER\_NAME\_ACCCEPT, APPLICATION\_JSON)  
 .addHeader(LinkConstants.CORRELATION\_ID, getCurrentDate())  
 .addHeader(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout);  
 }  
  
 public Response executeGetRequest(String link, String reqParam) {  
 RequestSpecification builder = getGetRequestBuilder(link, reqParam);  
 Response response = given().spec(builder).log().all().when().get();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executeGetRequestWithGivenHeadersAndQueryParams(String resourcePath, String reqParam,  
 Map<String, String> headers, Map<String, Object> queryParam) {  
 RequestSpecification builder = getGetRequestBuilderWithDefaultAndGivenHeadersAndQueryParams(resourcePath,  
 reqParam, headers, queryParam);  
 Response response = given().spec(builder).log().all().when().get();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executeGetRequestWithGivenHeaders(String resourcePath, String reqParam,  
 Map<String, String> headers) {  
 RequestSpecification builder = getGetRequestBuilderWithHeaders(resourcePath, reqParam, headers);  
 Response response = given().spec(builder).log().all().when().get();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executeGetRequestWithQueryParams(String link, String reqParam, Map<String, Object> queryParam) {  
 RequestSpecification builder = getGetRequestBuilderWithQueryParams(link, reqParam, queryParam);  
 Response response = given().spec(builder).log().all().when().get();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executeGetRequest(String link) {  
 RequestSpecification builder = new RequestSpecBuilder().setBaseUri(env.getProperty(baseJSONUrl))  
 .setBasePath(link).addHeader(LinkConstants.CORRELATION\_ID, getCurrentDate())  
 .addHeader(LinkConstants.AKER\_PROXY\_REQUEST\_TIMEOUT, akerProxyPerRequestTimeout).build();  
 Response response = given().spec(builder).log().all().when().get();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executePostRequestwithCustomHeadersArray(String link, String requestBody, String... reqHeader) {  
 RequestSpecification builder = getPostRequestBuilderWithCustomHeaders(link, requestBody, reqHeader);  
 Response response = given().spec(builder).log().all().when().post();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executePostRequestWithCustomHeaders(String link, String requestBody, Map<String, String> headers) {  
 RequestSpecification builder = getPostRequestBuilderWithCustomHeaders(link, requestBody, baseJSONUrl, headers);  
 Response response = given().spec(builder).log().all().post();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executePostRequest(String link, String requestBody) {  
 RequestSpecification builder = getPostRequestBuilder(link, requestBody, baseJSONUrl);  
 Response response = given().spec(builder).log().all().when().post();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executePutRequest(String resourcePath, String pathParameter) {  
 RequestSpecification builder = getPutRequestBuilder(resourcePath, pathParameter);  
 Response response = given().spec(builder).log().all().when().put();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executePutRequest(String resourcePath) {  
 RequestSpecification builder = getPutRequestBuilder(resourcePath);  
 Response response = given().spec(builder).log().all().when().put();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executePutRequestWithRequestBody(String resourcePath, int pathParameter, String requestBody) {  
 RequestSpecification builder = getPutRequestBuilder(resourcePath, pathParameter, requestBody);  
 Response response = given().spec(builder).log().all().when().put();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public Response executeDeleteRequest(String link, String reqParam) {  
 RequestSpecification builder = getDeleteRequestBuilder(link, reqParam);  
 Response response = given().spec(builder).log().all().when().delete();  
 printResponseBody(response);  
 testContextProvider.get().put(RESPONSE, response);  
 return response;  
 }  
  
 public int getActualStatusCodeFromContextProvider() {  
 Response response = testContextProvider.get().get(RESPONSE);  
 return response.getStatusCode();  
 }  
  
 public Response getResponseFromContextProvider() {  
 return testContextProvider.get().get(RESPONSE);  
 }  
  
 public void printResponseBody(Response response) {  
 if (logger.isInfoEnabled()) {  
 logger.info("API Response body is: {}", response.asString());  
 logger.info("API Response status code is: {}", response.getStatusCode());  
 }  
 testContextProvider.get().put(RESPONSE, response);  
 }  
  
 public <T> List<T> getActualPOJOClassesWithMessages(Class<T> returnType, List<AxonMessage> messageList) {  
 return messageList.stream().map(message -> parseResponse(message.getPayload(), returnType))  
 .collect(Collectors.toList());  
 }  
  
 public <T> T parseResponse(Object content, Class<T> returnType) {  
 this.mapper.configure(DeserializationFeature.UNWRAP\_ROOT\_VALUE, this.unwrapRootValue);  
 if (content == null) {  
 return null;  
 } else {  
 try {  
 String contentString = this.mapper.writeValueAsString(content);  
 return this.mapper.readValue(contentString, returnType);  
 } catch (JsonParseException jsonParseException) {  
 logger.error(String.format("JsonParseException: %s", jsonParseException.getMessage()));  
 throw new GDPUIException(jsonParseException.getMessage(), jsonParseException);  
 } catch (JsonMappingException jsonMappingException) {  
 logger.error(String.format("JsonMappingException: %s", jsonMappingException.getMessage()));  
 throw new GDPUIException(jsonMappingException.getMessage(), jsonMappingException);  
 } catch (IOException ioException) {  
 logger.error(String.format("IOException: %s", ioException.getMessage()));  
 throw new GDPUIException(ioException.getMessage(), ioException);  
 }  
 }  
 }  
  
 public <T> T parseResponse(String content, Class<T> returnType) {  
 this.mapper.configure(DeserializationFeature.ACCEPT\_SINGLE\_VALUE\_AS\_ARRAY, true);  
 this.mapper.configure(DeserializationFeature.UNWRAP\_ROOT\_VALUE, this.unwrapRootValue);  
 if (content == null) {  
 return null;  
 } else {  
 try {  
 return this.mapper.readValue(content, returnType);  
 } catch (JsonParseException jsonParseException) {  
 logger.error(String.format("JsonParseException: %s", jsonParseException.getMessage()));  
 throw new GDPUIException(jsonParseException.getMessage(), jsonParseException);  
 } catch (JsonMappingException jsonMappingException) {  
 logger.error(String.format("JsonMappingException: %s", jsonMappingException.getMessage()));  
 throw new GDPUIException(jsonMappingException.getMessage(), jsonMappingException);  
 } catch (IOException ioException) {  
 logger.error(String.format("IOException: %s", ioException.getMessage()));  
 throw new GDPUIException(ioException.getMessage(), ioException);  
 }  
 }  
 }  
  
 public String getErrorMessage() {  
 StringBuilder message = new StringBuilder();  
 return message.toString();  
 }  
  
 public String appendReqParamsToResourcePath(String resourcePath, String reqParam) {  
 StringBuilder resourcePathSB = new StringBuilder(resourcePath);  
 for (int i = 0; i < reqParam.split("&").length; i++) {  
 if (i == 0)  
 resourcePathSB.append("?").append(reqParam.split("&")[i]);  
 else  
 resourcePathSB.append("&").append(reqParam.split("&")[i]);  
 }  
 return resourcePathSB.toString();  
 }  
  
 public Response getResponse() {  
 return testContextProvider.get().get(ContextConstants.RESPONSE);  
 }  
  
 public String getSecurePassword(String stringToEncrypt) {  
 String encryptedString = null;  
 try {  
 MessageDigest md = MessageDigest.getInstance(ALGORITHM);  
 md.update(getStringBytes(SALTSTRING));  
 byte[] bytes = md.digest(getStringBytes(stringToEncrypt));  
 encryptedString = new BigInteger(SIGNUM, bytes).toString(RADIX);  
 } catch (NoSuchAlgorithmException e) {  
 logger.error(e.toString());  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return encryptedString;  
 }  
  
 public String modifyDbDatetoTargetDateFormat(String string) {  
  
 Timestamp ts = Timestamp.valueOf(string);  
 return getDateByFormat(ts);  
 }  
  
 public String getDateByFormat(Timestamp timestamp) {  
 try {  
 LocalDateTime convertedTime = timestamp.toLocalDateTime();  
 DateTimeFormatter createdTimeFormatter = DateTimeFormatter.ofPattern(UIConstants.DATE\_TIME\_FORMAT,  
 Locale.ENGLISH);  
 return convertedTime.format(createdTimeFormatter).toUpperCase();  
 } catch (DateTimeParseException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public int getActualStatusCode() {  
 Response response = testContextProvider.get().get(ContextConstants.RESPONSE);  
 return response.getStatusCode();  
 }  
  
 public String getCurrentDate() {  
 LocalDate date = LocalDate.now();  
 DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yMMdd");  
 return dtf.format(date);  
 }  
  
 public String encrptyKey(String stringToEncrypt) {  
 String encryptedString = null;  
 try {  
 MessageDigest md = MessageDigest.getInstance("MD5");  
 md.update(getStringBytes("GDPSECURESALTKEY"));  
 byte[] bytes = md.digest(getStringBytes(stringToEncrypt));  
 encryptedString = new BigInteger(1, bytes).toString(16);  
 } catch (NoSuchAlgorithmException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 return encryptedString;  
 }  
  
 private byte[] getStringBytes(String stringForBytes) {  
 Charset charSet = StandardCharsets.UTF\_8;  
 return stringForBytes.getBytes(charSet);  
 }  
  
 public <T> T getActualPojoClasses(Class<T> returnType) {  
 Response response = testContextProvider.get().get(RESPONSE);  
 return parseResponse(response.body().asString(), returnType);  
 }  
  
 public String getPositionOfElement(String positionValue) {  
 int position = (Integer.parseInt(positionValue) - NumberUtils.INTEGER\_ONE);  
 return Integer.toString(position);  
 }  
  
 public void refreshPage() {  
 gdpDriverProviderfactory.getDriverProvider().get().navigate().refresh();  
 customWaiters.waitForAngularToLoad();  
 }  
  
 public Boolean waitForElementToBeVisible(By by, long timeoutSec) {  
 customWaiters.waitForAngularToLoad();  
 WebElement elem = new WebDriverWait(gdpDriverProviderfactory.getDriverProvider().get(), timeoutSec)  
 .until(ExpectedConditions.visibilityOfElementLocated(by));  
  
 return (elem != null);  
 }  
  
 public Boolean waitForElementToBeVisible(WebElement element, long timeoutSec) {  
 customWaiters.waitForAngularToLoad();  
 WebElement elem = new WebDriverWait(gdpDriverProviderfactory.getDriverProvider().get(), timeoutSec)  
 .until(ExpectedConditions.visibilityOf(element));  
 return (elem != null);  
 }  
  
 public void verifyTheDateFormat(String dateformat, String value) {  
 SimpleDateFormat gdpDateFormat = new SimpleDateFormat(dateformat);  
 try {  
 gdpDateFormat.parse(value);  
 } catch (java.text.ParseException e) {  
 throw new GDPUIException("Date is not in the specified format " + dateformat);  
 }  
 }  
  
 public int generateRandomNumberInRange(int lastIndex) {  
 return rand.nextInt(lastIndex);  
 }  
  
 public String getReferenceIDFromURL(String url) {  
 if (logger.isDebugEnabled())  
 logger.info(String.format("URL to fetch Response Record ID : %s", url));  
 if (StringUtils.isNotEmpty(url)) {  
 int index = url.lastIndexOf('/');  
 url = url.substring(index + 1);  
 return url.split("\\?")[0];  
 }  
 return url;  
 }  
  
 public void compareJSONStrings(String actualPayload, String expectedPayload) throws IOException {  
 mapper = new ObjectMapper();  
 Map<String, Object> actualJSON = (mapper.readValue(actualPayload, Map.class));  
 Map<String, Object> expectedJSON = (mapper.readValue(expectedPayload, Map.class));  
 assertThat(actualJSON).containsAllEntriesOf(expectedJSON);  
 }  
  
 public String formatDate(String date, String s) {  
 Date d = null;  
 try {  
 d = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss").parse(date);  
 } catch (ParseException e) {  
 logger.error(String.format("Parse exception occurred %s", e));  
 }  
 SimpleDateFormat dateFormat = new SimpleDateFormat(s);  
 return dateFormat.format(d).toUpperCase().replace(currentYear, currentYear + " at");  
 }  
  
 public String formatDateWithUIDate(String date, String s) {  
 Date d = null;  
 try {  
 d = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss").parse(date);  
 } catch (ParseException e) {  
 logger.error(String.format("Parse exception occurred %s", e));  
 }  
 SimpleDateFormat dateFormat = new SimpleDateFormat(s);  
 return dateFormat.format(d).toUpperCase().replace(String.valueOf(dateFormat.getCalendar().getWeekYear()),  
 dateFormat.getCalendar().getWeekYear() + " at");  
 }  
  
 public String getUniqueEmail(String email) {  
 String[] arrEmail = email.split("@");  
 return arrEmail[0] + "\_" + System.currentTimeMillis() + "@" + arrEmail[1];  
 }  
  
 public String formatDateWithCST(Timestamp timestamp) {  
 LocalDateTime convertedTime = timestamp.toLocalDateTime();  
 DateTimeFormatter createdTimeFormatter = DateTimeFormatter.ofPattern(DATE\_FORMATTER\_WITH\_LOCALE\_CST);  
 return convertedTime.format(createdTimeFormatter);  
 }  
  
 public String getDateInRequiredFormat(Timestamp timeStamp, String formatter) {  
 try {  
 LocalDateTime convertedTime = timeStamp.toLocalDateTime();  
 DateTimeFormatter createdTimeFormatter = DateTimeFormatter.ofPattern(formatter);  
 return convertedTime.format(createdTimeFormatter).toUpperCase();  
 } catch (DateTimeParseException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public List<String> getWebElementsText(List<WebElement> elementsAsList) {  
 List<String> elementsAsStr = new ArrayList<>();  
 for (WebElement webElement : elementsAsList) {  
 elementsAsStr.add(webElement.getText());  
 }  
  
 return elementsAsStr;  
 }  
  
 public List<String> getWebElementsAttributeValue(List<WebElement> elementsAsList, String attribute) {  
 List<String> elementsAsStr = new ArrayList<>();  
 for (WebElement webElement : elementsAsList) {  
 elementsAsStr.add(webElement.getAttribute(attribute));  
 }  
  
 return elementsAsStr;  
 }  
  
 public List<Date> getDateFromDateStr(List<String> elementsAsText) {  
 SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy:HH:mm a");  
 List<Date> dateListFromUI = new ArrayList<>();  
 for (String d : elementsAsText) {  
 String[] dateArr = d.split(" ");  
 String dateFormat = dateArr[0] + "-" + dateArr[1] + "-" + dateArr[2] + ":" + dateArr[4] + " " + dateArr[5];  
 Date date = null;  
 try {  
 date = sdf.parse(dateFormat);  
 } catch (ParseException e) {  
 assertThat(false);  
 }  
 dateListFromUI.add(date);  
 }  
 return dateListFromUI;  
  
 }  
  
 public String modifyDbDatetoB2CDateSubmittedFormat(Timestamp timestamp) {  
 try {  
 LocalDateTime convertedTime = timestamp.toLocalDateTime();  
 DateTimeFormatter createdTimeFormatter = DateTimeFormatter.ofPattern(B2C\_CREATED\_DATE\_FORMATTER);  
 return convertedTime.format(createdTimeFormatter).toUpperCase();  
 } catch (DateTimeParseException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public String alterB2CUIDateSubmitted(String str) {  
 return StringUtils.remove(StringUtils.remove(str, "US"), "CST").trim();  
 }  
  
 public List<HashMap<String, String>> fetchRecordsWithAllColumns(int tableSize, List<String> columnHeaders,  
 String xpathToTable, int eachRowSize) {  
 List<HashMap<String, String>> dashBoardRecordFromUI = new ArrayList<>();  
 for (int i = 1; i <= tableSize; i++) {  
 HashMap<String, String> rowdata = new HashMap<>();  
 for (int j = 2; j <= eachRowSize; j++) {  
  
 String value = customWaiters  
 .getVisibleElement(By.xpath(xpathToTable + "[" + i + "]" + "/td" + "[" + j + "]")).getText();  
 rowdata.put(columnHeaders.get(j - 2), value);  
 }  
 dashBoardRecordFromUI.add(rowdata);  
 }  
 return dashBoardRecordFromUI;  
 }  
  
 public boolean validateDateFormatForB2CDashboard(String date) {  
 List<String> list = new ArrayList<>(Arrays.asList(date.split(" ")));  
 boolean b = false;  
  
 if (StringUtils.containsAny(list.get(5), "PM", "AM") && StringUtils.contains(list.get(6), "US")  
 && StringUtils.contains(list.get(7), "CST")) {  
 list.remove(6);  
 list.remove(6);  
 String dateStr = StringUtils.join(list, " ");  
 DateFormat f = new SimpleDateFormat(B2C\_CREATED\_DATE\_FORMATTER);  
 f.setLenient(false);  
 try {  
 f.parse(dateStr);  
 b = true;  
 } catch (ParseException e) {  
 b = false;  
 }  
 } else {  
 b = false;  
 }  
 return b;  
 }  
  
 public void setStartTime() {  
 timeBeforeTestBegin = getCurrentCSTLocalDateTime();  
 }  
  
 public void setEndTime() {  
 timeAfterTestEnd = getCurrentCSTLocalDateTime().plusSeconds(10);  
 }  
  
 public boolean verifyTimeStampIsInBetweenStartAndEndTimes(LocalDateTime time) {  
 return (timeBeforeTestBegin.isBefore(time) && timeAfterTestEnd.isAfter(time));  
 }  
  
 public LocalDateTime getCurrentCSTLocalDateTime() {  
 double offsetHours;  
 if (ZoneId.of("America/Chicago").getRules().isDaylightSavings(Instant.now())) {  
 offsetHours = -5;  
 } else {  
 offsetHours = -6;  
 }  
 ZoneOffset offset = ZoneOffset.ofHours((int) offsetHours);  
  
 return Instant.now().atOffset(offset).toLocalDateTime();  
 }  
  
 public String getRandomString(int length) {  
 return RandomStringUtils.randomAlphanumeric(length);  
 }  
  
 public String modifyDbDateToDDMMYYYYFormat(Timestamp timestamp) {  
 try {  
 LocalDateTime convertedTime = timestamp.toLocalDateTime();  
 DateTimeFormatter createdTimeFormatter = DateTimeFormatter.ofPattern(UIConstants.DATE\_TIME\_FORMAT);  
 return convertedTime.format(createdTimeFormatter);  
 } catch (DateTimeParseException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public static boolean isValidDateFormat(String date, String format) {  
 DateFormat formatter = new SimpleDateFormat(format, Locale.ENGLISH);  
 try {  
 formatter.parse(date);  
 return true;  
 } catch (ParseException e) {  
 return false;  
 }  
 }  
  
 public void waitForTextToBeVisible(By by, String value, long timeoutSec) {  
 new WebDriverWait(gdpDriverProviderfactory.getDriverProvider().get(), timeoutSec)  
 .until(ExpectedConditions.textMatches(by, Pattern.compile(value)));  
 }  
  
 public long getNoOfDays(Timestamp timeStamp) {  
 LocalDate currentDate = LocalDate.now();  
 LocalDate lastUpdatedDate = timeStamp.toLocalDateTime().toLocalDate();  
 return ChronoUnit.DAYS.between(lastUpdatedDate, currentDate);  
 }  
  
 public String modifyDbDateToDDMMYYYYFormat(Timestamp timestamp, String timeFormat) {  
 try {  
 LocalDateTime convertedTime = timestamp.toLocalDateTime().truncatedTo(ChronoUnit.MINUTES);  
 DateTimeFormatter createdTimeFormatter = DateTimeFormatter.ofPattern(timeFormat);  
 return convertedTime.format(createdTimeFormatter);  
 } catch (DateTimeParseException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 public String modifyDbDateToMMDDYYYYFormat(String time) {  
 try {  
 Timestamp ts = Timestamp.valueOf(time);  
 LocalDateTime convertedTime = ts.toLocalDateTime();  
 DateTimeFormatter createdTimeFormatter = DateTimeFormatter.ofPattern(UIConstants.MM\_DD\_YY\_FORMAT\_UI,  
 Locale.ENGLISH);  
 return convertedTime.format(createdTimeFormatter).toUpperCase();  
 } catch (DateTimeParseException e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
  
 }  
  
 public boolean isTheDateValid(String dateToValidate, String dateFormat) {  
 if (dateToValidate == null)  
 return false;  
  
 SimpleDateFormat simpleDateFormat = new SimpleDateFormat(dateFormat);  
 simpleDateFormat.setLenient(false);  
 try {  
 simpleDateFormat.parse(dateToValidate);  
 } catch (ParseException ex) {  
 logger.error("Unable to parse the date provided", ex);  
 return false;  
 }  
 return true;  
 }  
  
 public HttpResponse executePostMethod(String link, String requestBody) throws IOException {  
 HttpClient client = HttpClients.custom().build();  
 HttpPost post = new HttpPost(env.getProperty(baseJSONUrl) + link);  
 StringEntity entity = new StringEntity(requestBody);  
 entity.setContentType(UIConstants.APPLICATION\_JSON);  
 post.setEntity(entity);  
 httpResponse = client.execute(post);  
 return httpResponse;  
 }  
  
 public int getStatusCodeFromHttpResponse() {  
 return httpResponse.getStatusLine().getStatusCode();  
 }  
  
 public long verifyTimeStampDifference() {  
 long minutes = ChronoUnit.SECONDS.between(timeBeforeTestBegin, timeAfterTestEnd);  
 return (minutes);  
 }  
  
 public boolean isElementDisplayed(WebElement element) {  
 try {  
 WebDriverWait wait = new WebDriverWait(customWaiters.getDriver(), customWaiters.getElementWaitTime());  
 wait.until(ExpectedConditions.visibilityOf(element));  
 return element.isDisplayed();  
 } catch (org.openqa.selenium.NoSuchElementException | org.openqa.selenium.StaleElementReferenceException  
 | org.openqa.selenium.TimeoutException e) {  
 return false;  
 }  
 }  
  
 public Date getDateValueFromText(String dateValueInText) {  
 Date date = null;  
 DateFormat formatter = new SimpleDateFormat(UIConstants.DATE\_TIME\_FORMAT, Locale.US);  
 if (dateValueInText.contains(BY.trim())) {  
 try {  
 date = formatter.parse(getDateValueWithOutUser(dateValueInText));  
 } catch (ParseException e) {  
 logger.info(String.format("Unable to parse date %s", e));  
 }  
 } else {  
 try {  
 date = formatter.parse(dateValueInText);  
 } catch (ParseException e) {  
 logger.error(String.format("Error occurred while parsing %s", e));  
 }  
 }  
 return date;  
 }  
  
 private String getDateValueWithOutUser(String dateWithUser) {  
 return dateWithUser.substring(0, dateWithUser.indexOf(BY.trim()) - 1);  
 }  
  
 public String getURLWithCredentials(String username, String password, String url) {  
 if (checkIfEnvIsDevCloud(devCloundEnvironments)) {  
 return url.replace(UIStrings.HTTP\_PROTOCOL,  
 UIStrings.HTTP\_PROTOCOL + username + UIStrings.COLON + password + UIStrings.AT);  
 } else {  
 return url.replace(UIStrings.HTTPS\_PROTOCOL,  
 UIStrings.HTTPS\_PROTOCOL + username + UIStrings.COLON + password + UIStrings.AT);  
 }  
 }  
  
 private boolean checkIfEnvIsDevCloud(List<String> devCloundEnvironments) {  
 return devCloundEnvironments.stream().parallel().anyMatch(System.getProperty("env")::contains);  
 }  
  
 public List<String> getDevCloudEnvironments() {  
 return devCloundEnvironments;  
 }  
  
 public RequestSpecification getRequestSpecification() {  
 RequestSpecBuilder requestSpecBuilder = new RequestSpecBuilder();  
 requestSpecBuilder.setBaseUri(stageAobBaseURL);  
 requestSpecBuilder.setBasePath(LinkConstants.GDP\_AOB\_APP\_API);  
 requestSpecBuilder.setContentType(ContentType.JSON);  
 requestSpecBuilder.addHeader(XSRF\_TOKEN\_HEADER\_KEY, this.getXsrfToken());  
 requestSpecBuilder.addCookies(this.prepareCookies(this.getCookiesFromBrowserSession()));  
 return requestSpecBuilder.build();  
 }  
  
 public StringBuilder getUrlBasedOnEnvironment() {  
 StringBuilder productDetailsPage = new StringBuilder();  
 return productDetailsPage.append(b2cBaseUrl).append(LinkConstantsB2C.B2C\_SECURE\_URL\_PART)  
 .append(LinkConstantsB2C.B2C\_WEB).append(LinkConstantsB2C.DEFAULT\_LOCALE\_CODE\_EN\_US).append("#");  
 }  
  
 private Set<Cookie> getCookiesFromBrowserSession() {  
 return customWaiters.getDriver().manage().getCookies();  
 }  
  
 private String getXsrfToken() {  
 String xsrfTokenFromBrowser = customWaiters.getDriver().manage()  
 .getCookieNamed("AMWEBJCT!%2Fjct\_gdpindda\_test!XSRF-TOKEN").toString();  
 return xsrfTokenFromBrowser.split("=")[1];  
 }  
  
 private Cookies prepareCookies(Set<Cookie> cookiesFromSelenium) {  
 List<io.restassured.http.Cookie> cookiesForRA = new ArrayList<>();  
 cookiesFromSelenium.forEach(ck -> cookiesForRA.add(buildCookie(ck)));  
 return new Cookies(cookiesForRA);  
 }  
  
 private io.restassured.http.Cookie buildCookie(Cookie webDriverCookie) {  
 io.restassured.http.Cookie.Builder builder = new io.restassured.http.Cookie.Builder(webDriverCookie.getName(),  
 webDriverCookie.getValue()).setDomain(webDriverCookie.getDomain()).setPath(webDriverCookie.getPath())  
 .setHttpOnly(webDriverCookie.isHttpOnly()).setSecured(webDriverCookie.isSecure());  
  
 Optional<Date> dateOptional = Optional.ofNullable(webDriverCookie.getExpiry());  
 dateOptional.ifPresent(builder::setExpiryDate);  
  
 return builder.build();  
 }  
  
 public void checkFieldInMap(Map<WebElement, WebElement> fields, String key, String value) {  
 if (key.equalsIgnoreCase(UIConstants.CARD\_NUMBER\_MANUAL\_RES)) {  
 value = value.replace("•", ".");  
 }  
 Set<WebElement> keys = fields.keySet();  
 Boolean result = false;  
  
 for (WebElement keyField : keys) {  
 if (keyField.getText().equals(key) && fields.get(keyField).getText().equals(value)) {  
 result = true;  
 break;  
 }  
 }  
 assertThat(result).isTrue();  
 }  
  
 public String getAbsoluteFileName(String fileName) {  
 return System.getProperty("user.dir") + File.separator + "src" + File.separator + "main" + File.separator  
 + "resources" + File.separator + "config" + File.separator + "TestData" + File.separator + fileName;  
 }  
 public String extractDigitsFromString(String strValue){  
 String str = strValue.trim();  
 String digits="";  
 for (int i = 0; i < str.length(); i++) {  
 char chrs = str.charAt(i);   
 if (Character.isDigit(chrs))  
 digits = digits+chrs;  
 }  
 return digits;  
}  
   
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import java.util.Random;  
  
*/\*\*  
 \* A credit card number generator.  
 \*/*public class CreditCardNumberGenerator {  
  
 private static final String *MASTERCARD\_BIN\_NUMBER* = "5";  
 private static final int *CARD\_LENGTH* = 16;  
 private Random random = new Random(System.*currentTimeMillis*());  
  
 */\*\*  
 \* Generates a random valid credit card number. For more information about the  
 \* credit card number generation algorithms and credit card numbers refer to  
 \* <a href="http://euro.ecom.cmu.edu/resources/elibrary/everycc.htm">Everything  
 \* you ever wanted to know about CC's</a>,  
 \* <a href="http://www.darkcoding.net/credit-card/">Graham King's blog</a>, and  
 \* <a href=  
 \* "http://codytaylor.org/2009/11/this-is-how-credit-card-numbers-are-generated.html"  
 \* >This is How Credit Card Numbers Are Generated</a>  
 \*  
 \** ***@param*** *bin The bank identification number, a set digits at the start of  
 \* the credit card number, used to identify the bank that is  
 \* issuing the credit card.  
 \** ***@param*** *length The total length (i.e. including the BIN) of the credit card  
 \* number.  
 \** ***@return*** *A randomly generated, valid, credit card number.  
 \*/* public String generate(String bin, int length) {  
  
 // The number of random digits that we need to generate is equal to the  
 // total length of the card number minus the start digits given by the  
 // user, minus the check digit at the end.  
 int randomNumberLength = length - (bin.length() + 1);  
  
 StringBuilder builder = new StringBuilder(bin);  
 for (int i = 0; i < randomNumberLength; i++) {  
 int digit = this.random.nextInt(10);  
 builder.append(digit);  
 }  
  
 // Do the Luhn algorithm to generate the check digit.  
 int checkDigit = this.getCheckDigit(builder.toString());  
 builder.append(checkDigit);  
  
 return builder.toString();  
 }  
  
 */\*\*  
 \* Generates the check digit required to make the given credit card number valid  
 \* (i.e. pass the Luhn check)  
 \*  
 \** ***@param*** *number The credit card number for which to generate the check digit.  
 \** ***@return*** *The check digit required to make the given credit card number valid.  
 \*/* private int getCheckDigit(String number) {  
  
 // Get the sum of all the digits, however we need to replace the value  
 // of the first digit, and every other digit, with the same digit  
 // multiplied by 2. If this multiplication yields a number greater  
 // than 9, then add the two digits together to get a single digit  
 // number.  
 //  
 // The digits we need to replace will be those in an even position for  
 // card numbers whose length is an even number, or those is an odd  
 // position for card numbers whose length is an odd number. This is  
 // because the Luhn algorithm reverses the card number, and doubles  
 // every other number starting from the second number from the last  
 // position.  
 int sum = 0;  
 for (int i = 0; i < number.length(); i++) {  
  
 // Get the digit at the current position.  
 int digit = Integer.*parseInt*(number.substring(i, (i + 1)));  
  
 if ((i % 2) == 0) {  
 digit = digit \* 2;  
 if (digit > 9) {  
 digit = (digit / 10) + (digit % 10);  
 }  
 }  
 sum += digit;  
 }  
  
 // The check digit is the number required to make the sum a multiple of  
 // 10.  
 int mod = sum % 10;  
 return ((mod == 0) ? 0 : 10 - mod);  
 }  
  
 public static String generateValidMasterCardNumber() {  
 return new CreditCardNumberGenerator().generate(*MASTERCARD\_BIN\_NUMBER*, *CARD\_LENGTH*);  
 }  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class DynamicWebElement {  
 private String name;  
 private String type;  
 private String subtype;  
 private String label;  
 private String xpath;  
 private String labelXpath;  
 private String parentXpath;  
 private Integer order;  
 private Integer viewId;  
 private String value;  
 private List<DynamicWebElement> childElements;  
 private List<DynamicWebElement> values;  
 private List<List<DynamicWebElement>> childElementGroup;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 public String getSubtype() {  
 return subtype;  
 }  
  
 public void setSubtype(String subtype) {  
 this.subtype = subtype;  
 }  
  
 public String getLabel() {  
 return label;  
 }  
  
 public void setLabel(String label) {  
 this.label = label;  
 }  
  
 public String getXpath() {  
 return xpath;  
 }  
  
 public void setXpath(String xpath) {  
 this.xpath = xpath;  
 }  
  
 public String getParentXpath() {  
 return parentXpath;  
 }  
  
 public void setParentXpath(String parentXpath) {  
 this.parentXpath = parentXpath;  
 }  
  
 public Integer getOrder() {  
 return order;  
 }  
  
 public void setOrder(Integer order) {  
 this.order = order;  
 }  
  
 public String getValue() {  
 return value;  
 }  
  
 public void setValue(String value) {  
 this.value = value;  
 }  
  
 public List<DynamicWebElement> getChildElements() {  
 return childElements;  
 }  
  
 public void setChildElements(List<DynamicWebElement> childElements) {  
 this.childElements = childElements;  
 }  
  
 public String getLabelXpath() {  
 return labelXpath;  
 }  
  
 public void setLabelXpath(String labelXpath) {  
 this.labelXpath = labelXpath;  
 }  
  
 public List<DynamicWebElement> getValues() {  
 return values;  
 }  
  
 public void setValues(List<DynamicWebElement> values) {  
 this.values = values;  
 }  
  
 public void addValue(DynamicWebElement value) {  
 if (null == this.values) {  
 this.values = new ArrayList<>();  
 }  
 this.values.add(value);  
 }  
  
 public void addChildElement(DynamicWebElement child) {  
 if (null == this.childElements) {  
 this.childElements = new ArrayList<>();  
 }  
 this.childElements.add(child);  
 }  
  
 public List<List<DynamicWebElement>> getChildElementGroup() {  
 return childElementGroup;  
 }  
  
 public void setChildElementGroup(List<List<DynamicWebElement>> childElementGroup) {  
 this.childElementGroup = childElementGroup;  
 }  
  
 public void addChildElementGroup(List<DynamicWebElement> childElements) {  
 if (null == this.childElementGroup) {  
 this.childElementGroup = new ArrayList<>();  
 }  
 this.childElementGroup.add(childElements);  
 }  
  
 public Integer getViewId() {  
 return this.viewId;  
 }  
  
 public void setViewId(Integer viewId) {  
 this.viewId = viewId;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
import org.apache.log4j.Logger;  
import org.json.JSONArray;  
import org.json.JSONObject;  
import org.springframework.stereotype.Component;  
import org.springframework.util.CollectionUtils;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
  
@Component  
public class DynamicWebElementUtil {  
 ObjectMapper om = new ObjectMapper();  
 private static final String *UNDERSCORE* = "\_";  
 private static final String *XPATH\_PREFIX\_ID* = "//\*[@id='";  
 private static final String *TYPE\_MOBILE* = "mobile";  
 private static final String *TYPE\_PHONE* = "phone";  
 private static final String *KEY\_ORDER\_MAP* = "orderMap";  
 private static final String *KEY\_TYPE\_MAP* = "typeMap";  
 private static final String *KEY\_ELEMENT\_DISPLAY\_ORDER* = "elementDisplayOrder";  
 private static final String *KEY\_ELEMENT\_TYPE\_CODE* = "elementTypeCode";  
 private static final String *KEY\_ELEMENT\_TAG\_TEXT* = "elementTagText";  
 private static final String *TYPE\_ARRAY* = "Array";  
 private static final String *TYPE\_OBJECT* = "Object";  
 private static final String *TYPE\_JSON\_ARRAY* = "JSONArray";  
 private static final String *TYPE\_JSON\_OBJECT* = "JSONObject";  
 private static final String *XPATH\_EMPTY* = "";  
 private static final String *RECORD\_XPATH* = "//Record";  
 private static final String *RECORD* = "Record";  
  
 private final Logger logger = Logger.*getLogger*(DynamicWebElementUtil.class);  
  
 public DynamicWebElement processJSONResponse(String jsonResponse, String jsonLables, String jsonDataElements) {  
 DynamicWebElementViewCounter viewCounter = new DynamicWebElementViewCounter();  
 Map<String, String> labelMap = this.prepareLabelMap(new JSONObject(jsonLables));  
 Map<String, Object> groupMap = this.prepareMap(new JSONArray(jsonDataElements));  
 Map<String, Integer> orderMap = (Map<String, Integer>) groupMap.get(*KEY\_ORDER\_MAP*);  
 Map<String, String> typeMap = (Map<String, String>) groupMap.get(*KEY\_TYPE\_MAP*);  
  
 DynamicWebElement webElement = this.prepareDynamicWebElement(new JSONObject(jsonResponse), typeMap, labelMap,  
 orderMap);  
 this.prepareViewIds(webElement, viewCounter);  
 ObjectMapper mapper = new ObjectMapper();  
 this.prepareXpath(webElement, labelMap, *XPATH\_EMPTY*, 0, 1);  
 logger.info("viewId:::" + viewCounter.getViewId());  
  
 try {  
 logger.info(mapper.writeValueAsString(webElement));  
 } catch (JsonProcessingException e) {  
 throw new GDPUIException("Exception while processing the JSON!", e);  
 }  
  
 return webElement;  
 }  
  
 private void prepareViewIds(DynamicWebElement webElement, DynamicWebElementViewCounter viewCounter) {  
 updateViewId(webElement, viewCounter);  
 if (*RECORD*.equals(webElement.getSubtype()) || *TYPE\_OBJECT*.equals(webElement.getType())) {  
 webElement.setViewId(viewCounter.getViewId());  
 List<DynamicWebElement> childElements = webElement.getChildElements();  
 this.prepareViewIds(childElements, viewCounter);  
 } else if (*TYPE\_ARRAY*.equals(webElement.getType())  
 && (*RECORD*.equals(webElement.getSubtype()) || isComplexSubType(webElement.getSubtype()))  
 && isPrimitiveGroup(webElement.getChildElementGroup())) {  
 prepareViewIdsGroupForArrayAndObject(webElement, viewCounter);  
 } else if (*TYPE\_ARRAY*.equals(webElement.getType()) && !isComplexSubType(webElement.getSubtype())) {  
 webElement.setViewId(viewCounter.getViewId());  
 prepareViewIdList(webElement.getValues(), viewCounter);  
 } else if (*TYPE\_JSON\_OBJECT*.equals(webElement.getSubtype())) {  
 prepareViewIdsGroupForArrayAndObject(webElement, viewCounter);  
 } else {  
 webElement.setViewId(viewCounter.getViewId());  
 }  
 updateViewId(webElement, viewCounter);  
 }  
  
 private void prepareViewIdsGroupForArrayAndObject(DynamicWebElement webElement,  
 DynamicWebElementViewCounter viewCounter) {  
 webElement.setViewId(viewCounter.getViewId());  
 prepareViewIdGroup(webElement.getChildElementGroup(), viewCounter);  
 }  
  
 private void updateViewId(DynamicWebElement webElement, DynamicWebElementViewCounter viewCounter) {  
 if (*TYPE\_OBJECT*.equals(webElement.getType()) || *TYPE\_ARRAY*.equals(webElement.getType())  
 || *RECORD*.equals(webElement.getType()) || checkForElementSubtype(webElement)) {  
 viewCounter.updateViewId();  
 }  
 }  
  
 private boolean checkForElementSubtype(DynamicWebElement webElement) {  
 return *TYPE\_JSON\_OBJECT*.equals(webElement.getSubtype()) || *TYPE\_JSON\_ARRAY*.equals(webElement.getSubtype());  
 }  
  
 private boolean isNotPrimitiveGroup(List<DynamicWebElement> webElements) {  
 if (webElements == null) {  
 return false;  
 }  
 return webElements.stream().anyMatch(element -> *isComplexTpe*(element.getType()));  
 }  
  
 private boolean isPrimitiveGroup(List<List<DynamicWebElement>> webelementsGroup) {  
 if (webelementsGroup == null) {  
 return false;  
 }  
 return webelementsGroup.stream()  
 .anyMatch(elements -> elements.stream().anyMatch(element -> *isComplexTpe*(element.getType())));  
 }  
  
 private void prepareViewIds(List<DynamicWebElement> childElements, DynamicWebElementViewCounter viewCounter) {  
 childElements.forEach(element -> this.prepareViewIds(element, viewCounter));  
 }  
  
 private void prepareViewIdList(List<DynamicWebElement> webElements, DynamicWebElementViewCounter viewCounter) {  
 webElements.forEach(webElement -> this.prepareViewIds(webElement, viewCounter));  
 if (isNotPrimitiveGroup(webElements)) {  
 viewCounter.updateViewId();  
 }  
 }  
  
 private void prepareViewIdGroup(List<List<DynamicWebElement>> webElementGroup,  
 DynamicWebElementViewCounter viewCounter) {  
 webElementGroup.forEach(webElements -> prepareViewIdList(webElements, viewCounter));  
 }  
  
 private DynamicWebElement prepareDynamicWebElement(JSONObject json, Map<String, String> typeMap,  
 Map<String, String> labelMap, Map<String, Integer> orderMap) {  
 DynamicWebElement parentElement = new DynamicWebElement();  
 List<DynamicWebElement> childElements = new ArrayList<>();  
 DynamicWebElement webElement = null;  
  
 parentElement.setLabel(*RECORD*);  
 parentElement.setName(*RECORD*);  
 parentElement.setOrder(1);  
 parentElement.setParentXpath(*RECORD\_XPATH*);  
 parentElement.setSubtype(*RECORD*);  
 parentElement.setType(*RECORD*);  
 parentElement.setValue(*RECORD*);  
 parentElement.setXpath(*RECORD\_XPATH*);  
  
 for (String key : json.keySet()) {  
 webElement = new DynamicWebElement();  
 webElement.setLabel(labelMap.get(key));  
 webElement.setName(key);  
 webElement.setOrder(orderMap.get(key));  
 Object value = json.get(key);  
 webElement.setType(typeMap.get(key));  
 if (!isComplexObject(value)) {  
 webElement.setSubtype(value.getClass().getSimpleName());  
 webElement.setValue(value.toString());  
 } else if (value instanceof JSONObject) {  
 webElement.setSubtype(value.getClass().getSimpleName());  
 webElement.setChildElements(  
 prepareDynamicWebElement((JSONObject) value, typeMap, labelMap, orderMap).getChildElements());  
 } else if (value instanceof JSONArray) {  
 JSONArray array = (JSONArray) value;  
 if (array.length() > 0) {  
 prepareDynamicElementForJsonArray(typeMap, labelMap, orderMap, webElement, key, array);  
 }  
 }  
  
 childElements.add(webElement);  
 }  
  
 childElements.sort(this::compare);  
 parentElement.setChildElements(childElements);  
 return parentElement;  
 }  
  
 private void prepareDynamicElementForJsonArray(Map<String, String> typeMap, Map<String, String> labelMap,  
 Map<String, Integer> orderMap, DynamicWebElement webElement, String key, JSONArray array) {  
 webElement.setSubtype(array.get(0).getClass().getSimpleName());  
 if (!isComplexObject(array.get(0))) {  
 webElement.setValues(prepareDynamicWebElement(array, typeMap, labelMap, orderMap, key).getChildElements());  
 } else {  
 webElement.setChildElementGroup(  
 prepareDynamicWebElement(array, typeMap, labelMap, orderMap, key).getChildElementGroup());  
 }  
 }  
  
 private DynamicWebElement prepareDynamicWebElement(JSONArray jsonArray, Map<String, String> typeMap,  
 Map<String, String> labelMap, Map<String, Integer> orderMap, String name) {  
 DynamicWebElement rootElement = new DynamicWebElement();  
 DynamicWebElement webElement = null;  
 for (Object value : jsonArray) {  
 webElement = new DynamicWebElement();  
 webElement.setLabel(labelMap.get(name));  
 webElement.setName(name);  
 webElement.setOrder(orderMap.get(name));  
 webElement.setType(typeMap.get(name));  
 webElement.setSubtype(value.getClass().getSimpleName());  
 if (!isComplexObject(value)) {  
 webElement.setOrder(orderMap.get(name));  
 webElement.setValue(value.toString());  
 webElement.setType(value.getClass().getSimpleName());  
 rootElement.addChildElement(webElement);  
 } else {  
 List<DynamicWebElement> childElements = prepareDynamicWebElement((JSONObject) value, typeMap, labelMap,  
 orderMap).getChildElements();  
 rootElement.addChildElementGroup(childElements);  
  
 }  
 }  
 if (!CollectionUtils.*isEmpty*(rootElement.getChildElements())) {  
 rootElement.getChildElements().sort(this::compare);  
 }  
 return rootElement;  
 }  
  
 private boolean isComplexObject(Object object) {  
 return (object instanceof List || object instanceof JSONObject || object instanceof JSONArray);  
 }  
  
 private boolean isComplexSubType(String subType) {  
 return (*TYPE\_JSON\_OBJECT*.equals(subType) || *TYPE\_JSON\_ARRAY*.equals(subType));  
 }  
  
 public static boolean isComplexTpe(String type) {  
 return (*TYPE\_OBJECT*.equals(type) || *TYPE\_ARRAY*.equals(type));  
 }  
  
 public static boolean isPhoneType(String name) {  
 return name.contains(*TYPE\_PHONE*) || name.contains(*TYPE\_MOBILE*);  
 }  
  
 private Map<String, Object> prepareMap(JSONArray elementsArray) {  
 Map<String, Object> groupMap = new HashMap<>();  
 Map<String, Integer> orderMap = new HashMap<>();  
 Map<String, String> typeMap = new HashMap<>();  
 for (Object object : elementsArray) {  
 JSONObject json = (JSONObject) object;  
 typeMap.put(json.getString(*KEY\_ELEMENT\_TAG\_TEXT*), json.getString(*KEY\_ELEMENT\_TYPE\_CODE*));  
 orderMap.put(json.getString(*KEY\_ELEMENT\_TAG\_TEXT*), json.getInt(*KEY\_ELEMENT\_DISPLAY\_ORDER*));  
 }  
 groupMap.put(*KEY\_TYPE\_MAP*, typeMap);  
 groupMap.put(*KEY\_ORDER\_MAP*, orderMap);  
  
 return groupMap;  
 }  
  
 private Map<String, String> prepareLabelMap(JSONObject labelObject) {  
 Map<String, String> labelMap = new HashMap<>();  
 for (String key : labelObject.keySet()) {  
 labelMap.put(key, labelObject.getString(key));  
 }  
 return labelMap;  
 }  
  
 private DynamicWebElement prepareXpath(DynamicWebElement dynamicWebElement, Map<String, String> labelMap,  
 String groupName, Integer groupIndex, Integer index) {  
  
 String name = (null != groupName ? groupName : dynamicWebElement.getName());  
  
 dynamicWebElement.setLabelXpath(*XPATH\_PREFIX\_ID* + name + *UNDERSCORE* + dynamicWebElement.getViewId() + *UNDERSCORE* + groupIndex + *UNDERSCORE* + index + "\_label']");  
  
 dynamicWebElement.setXpath(*XPATH\_PREFIX\_ID* + name + *UNDERSCORE* + dynamicWebElement.getViewId() + *UNDERSCORE* + groupIndex + *UNDERSCORE* + index + "\_value']");  
  
 dynamicWebElement  
 .setParentXpath(*XPATH\_PREFIX\_ID* + "combo-list-group-" + dynamicWebElement.getViewId() + "-panel']");  
  
 processInnerDynamicElements(dynamicWebElement, labelMap);  
 return dynamicWebElement;  
 }  
  
 private void processInnerDynamicElements(DynamicWebElement dynamicWebElement, Map<String, String> labelMap) {  
 Integer innerGroupIndex = 0;  
 processInnerChildElemets(dynamicWebElement, labelMap, innerGroupIndex);  
 processInnerElementValues(dynamicWebElement, labelMap, innerGroupIndex);  
 processInnerChildElementGroup(dynamicWebElement, labelMap);  
 }  
  
 private void processInnerChildElemets(DynamicWebElement dynamicWebElement, Map<String, String> labelMap,  
 Integer innerGroupIndex) {  
 Integer innerIndex = 1;  
 Integer prev = -1;  
 if (!CollectionUtils.*isEmpty*(dynamicWebElement.getChildElements())) {  
 for (DynamicWebElement element : dynamicWebElement.getChildElements()) {  
 if (!(prev.equals(element.getViewId()))) {  
 innerIndex = 1;  
 }  
 prepareXpath(element, labelMap, null, innerGroupIndex, innerIndex++);  
 prev = element.getViewId();  
 }  
 }  
 }  
  
 private void processInnerElementValues(DynamicWebElement dynamicWebElement, Map<String, String> labelMap,  
 Integer innerGroupIndex) {  
 Integer innerIndex = 1;  
 if (!CollectionUtils.*isEmpty*(dynamicWebElement.getValues())) {  
 for (DynamicWebElement element : dynamicWebElement.getValues()) {  
 prepareXpath(element, labelMap, dynamicWebElement.getName(), innerGroupIndex, innerIndex++);  
 }  
 }  
 }  
  
 private void processInnerChildElementGroup(DynamicWebElement dynamicWebElement, Map<String, String> labelMap) {  
 Integer innerGroupIndex = 0;  
 if (!CollectionUtils.*isEmpty*(dynamicWebElement.getChildElementGroup())) {  
 for (List<DynamicWebElement> elementGroup : dynamicWebElement.getChildElementGroup()) {  
 if (!CollectionUtils.*isEmpty*(elementGroup)) {  
 prepareXpathForElementGroup(labelMap, innerGroupIndex, elementGroup);  
 }  
 innerGroupIndex++;  
 }  
 }  
 }  
  
 private void prepareXpathForElementGroup(Map<String, String> labelMap, Integer innerGroupIndex,  
 List<DynamicWebElement> elementGroup) {  
 Integer innerIndex;  
 innerIndex = 1;  
 for (DynamicWebElement element : elementGroup) {  
 prepareXpath(element, labelMap, null, innerGroupIndex, innerIndex);  
 innerIndex++;  
 }  
 }  
  
 private int compare(DynamicWebElement leftElement, DynamicWebElement rightElement) {  
 return leftElement.getOrder().compareTo(rightElement.getOrder());  
 }  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
public class DynamicWebElementViewCounter {  
 private Integer viewId = 0;  
  
 public Integer getViewId() {  
 return this.viewId;  
 }  
  
 public void updateViewId() {  
 this.viewId = this.getViewId() + 1;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import com.mastercard.testing.gdp.ui.tests.constants.UIConstants;  
import com.mastercard.testing.gdp.ui.tests.domain.CountryLanguage;  
import org.apache.poi.hssf.usermodel.HSSFRow;  
import org.apache.poi.hssf.usermodel.HSSFSheet;  
import org.apache.poi.hssf.usermodel.HSSFWorkbook;  
import org.apache.poi.ss.usermodel.Row;  
import org.springframework.stereotype.Component;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.\*;  
  
@Component  
public class ExcelReader {  
 public Map<String, List<CountryLanguage>> getData() throws IOException {  
 String fileName = System.*getProperty*("user.dir") + UIConstants.*REGIONS\_COUNTRY\_LIST*;  
 InputStream excelFileToRead = new FileInputStream(fileName);  
 HSSFWorkbook wb = new HSSFWorkbook(excelFileToRead);  
 HSSFSheet sheet = wb.getSheetAt(0);  
 Iterator<Row> rows = sheet.rowIterator();  
  
 Map<String, List<CountryLanguage>> regionToCountries = new HashMap<>();  
 // Ignoring first row  
 rows.next();  
 HSSFRow row;  
 while (rows.hasNext()) {  
 row = (HSSFRow) rows.next();  
 String regionValue = row.getCell(0).getStringCellValue().trim();  
 String countryValue = row.getCell(1).getStringCellValue().trim();  
 String languageValue = row.getCell(4).getStringCellValue().trim();  
 String localeValue = row.getCell(7).getStringCellValue().trim();  
 String languageValueInEnglish = row.getCell(2).getStringCellValue().trim().toLowerCase();  
 List<CountryLanguage> countryLanguages = regionToCountries.get(regionValue);  
 if (countryLanguages == null) {  
 countryLanguages = new ArrayList<>();  
 }  
 countryLanguages.add(new CountryLanguage(countryValue, languageValue, localeValue, languageValueInEnglish));  
 regionToCountries.put(regionValue, countryLanguages);  
 }  
 wb.close();  
 return regionToCountries;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import org.apache.commons.lang3.StringUtils;  
import org.springframework.stereotype.Component;  
  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
@Component  
public class MaskCreditCardUtils {  
 public static final String *CARD\_REGEX* = "(.)";  
 public static final String *MASK* = ".";  
 public static final String *CARD\_SPACE\_REGEX* = ".{4}";  
  
 public String maskCardNumber(String cardValue) {  
 if (StringUtils.*isEmpty*(cardValue)) {  
 return StringUtils.*EMPTY*;  
 }  
 int cardLength = cardValue.length();  
 String lastdigits = StringUtils.*substring*(cardValue, cardLength - 4, cardLength);  
 String beforeDigits = StringUtils.*substring*(cardValue, 0, cardLength - 4);  
 String afterAddingSpaces = addSpacesIntoCardNumber(beforeDigits.replaceAll(*CARD\_REGEX*, *MASK*));  
 return afterAddingSpaces + lastdigits;  
 }  
  
 private String addSpacesIntoCardNumber(String cardNumber) {  
 StringBuilder builder = new StringBuilder();  
 Pattern p = Pattern.*compile*(*CARD\_SPACE\_REGEX*);  
 Matcher m = p.matcher(cardNumber);  
  
 while (m.find()) {  
 builder.append(m.group());  
 builder.append(StringUtils.*SPACE*);  
 }  
 return builder.toString();  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import java.security.SecureRandom;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
import java.util.concurrent.ThreadLocalRandom;  
  
import org.apache.commons.collections4.MapUtils;  
import org.apache.commons.lang3.RandomStringUtils;  
import org.jbehave.core.annotations.\*;  
import org.jbehave.core.model.ExamplesTable;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.stereotype.Component;  
  
import com.mastercard.testing.gdp.ui.tests.aspect.ContextVariableTestData;  
import com.mastercard.testing.gdp.ui.tests.constants.QueryConstants;  
import com.mastercard.testing.gdp.ui.tests.domain.entities.LabelTranslationEntity;  
import com.mastercard.testing.gdp.ui.tests.domain.entities.handlers.LabelTranslationHandler;  
import com.mastercard.testing.gdp.ui.tests.domain.enums.Context;  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
import com.mastercard.testing.gdp.ui.tests.steps.common.BaseSteps;  
  
@Component  
public class QueryUtil extends BaseSteps {  
  
 @Autowired  
 private ContextVariableTestData contextVariablesSteps;  
  
 @Value("${env:local}")  
 private String environment;  
  
 @Value("${b2c.app.user2}")  
 private String userId;  
  
 @Value("${b2c.app.testUser}")  
 private String testUserId;  
  
 private static final String *INSERTION\_COLUMN\_NAMES* = "\tprvcy\_rqst\_id, rqst\_type\_cd, rqst\_cntxt\_cd, rqst\_email\_addr, rqst\_dt, rqst\_stat\_cd, crte\_user\_id, crte\_ts, lst\_updt\_user\_id, lst\_updt\_ts, first\_nam, lst\_nam, rqstr\_first\_nam, rqstr\_lst\_nam, srch\_key\_txt, rqst\_cmplt\_ts, vw\_rqst\_id)\n";  
 private static final String *INSERT\_INTO\_PRVCY\_RQST* = "INSERT INTO prvcy\_rqst(\n";  
 private StringBuilder whereConditions = new StringBuilder(" WHERE ");  
 private StringBuilder orderByCondition = new StringBuilder();  
 private StringBuilder offsetValue = new StringBuilder();  
 private StringBuilder limitValue = new StringBuilder();  
 private StringBuilder sortWay = new StringBuilder();  
 private StringBuilder prvcyRqstListStr = new StringBuilder();  
 private List<Map<String, String>> prvcyRqstIdsInServFuncPrvcyRqst = new ArrayList<>();  
 static String *randomStringChars* = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz-";  
 static SecureRandom *rnd* = new SecureRandom();  
  
 List<String> firstNames = Arrays.*asList*("Andy", "Brad", "Chester", "David", "Ethan", "Freddy", "Gordon",  
 "Hendricks", "Jamie", "Hafthor", "AJ");  
 List<String> lastNames = Arrays.*asList*("Harper", "Ford", "Elaine", "Lewis", "Cooper", "Wolowitz", "Brady",  
 "Bjornsson", "Pedersen", "Almindinger");  
  
 private static String getRandomName(int length) {  
 StringBuilder sb = new StringBuilder(length);  
 for (int i = 0; i < length; i++)  
 sb.append(RandomStringUtils.*randomAlphanumeric*(1));  
 return sb.toString();  
 }  
  
 public void insertPrivacyRequest(@Named("$prvcy\_rqst\_id") int privacyRequestId,  
 @Named("$rqst\_type\_cd") String requestTypeCode, @Named("$rqst\_cntxt\_cd") String requestContextCode,  
 @Named("$rqst\_stat\_cd") String requestStatusCode) {  
 String query = String.format(INSERT\_QUERY, privacyRequestId, requestTypeCode, requestContextCode,  
 requestStatusCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 public void insertPrivacyRequestWithTypeCode(@Named("$prvcy\_rqst\_id") int privacyRequestId,  
 @Named("$rqst\_type\_cd") String requestTypeCode, @Named("$rqst\_cntxt\_cd") String requestContextCode,  
 @Named("$rqst\_stat\_cd") String requestStatusCode, @Named("$portalTypeCode") String portalTypeCode) {  
 String query = String.format(INSERT\_QUERY\_WITH\_TYPE\_CODE, privacyRequestId, requestTypeCode, requestContextCode,  
 requestStatusCode, portalTypeCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 */\*\*\*  
 \*  
 \** ***@param*** *localWhereConditionHash - a hashmap of where conditions, fieldName as  
 \* key and fieldValue as value  
 \** ***@param*** *localSingleConditionHash - a hashmap of other conditions. Possible key  
 \* Values can be "ORDERBY"(comma seperated  
 \* values), "OFFSET", "LIMIT", "SORTWAY"(asc,  
 \* desc)  
 \** ***@return*** *\*/* public String getSelectQuerybyConditions(String portalType, Map<String, String> localWhereConditionHash,  
 Map<String, String> localSingleConditionHash, String... query) {  
 String queryToReturn = null;  
 // Construct Query in the right order of conditions. Where followed by Orderby  
 // followed by Limit, Offset  
 if (MapUtils.*isEmpty*(localWhereConditionHash)) {  
 localWhereConditionHash.put("1", "1");  
 }  
 Integer numOfConditions = localWhereConditionHash.size();  
 Integer currentKey = 1;  
 Set<String> whereKeys = localWhereConditionHash.keySet();  
 // Fetch Where conditions.  
 for (String field : whereKeys) {  
 if (currentKey.equals(numOfConditions)) {  
 updateLocalWhereConditions(localWhereConditionHash, field);  
 } else {  
 appendLocalWhereConditions(localWhereConditionHash, field);  
 }  
 currentKey += 1;  
 }  
  
 if (!localSingleConditionHash.isEmpty()) {  
 appendLocalSingleWhereConditions(localSingleConditionHash);  
 }  
  
 if (0 == query.length) {  
 String basicPrivacyRequestQuery;  
 if (Context.*B2C*.name().equalsIgnoreCase(portalType)) {  
 basicPrivacyRequestQuery = B2C\_BASE\_SELECT\_PRIVACY\_REQS\_QUERY;  
 } else {  
 basicPrivacyRequestQuery = BASE\_SELECT\_PRIVACY\_REQS\_QUERY;  
 }  
 queryToReturn = basicPrivacyRequestQuery + whereConditions + orderByCondition + sortWay;  
 queryToReturn = queryToReturn + offsetValue + limitValue;  
 } else {  
 queryToReturn = query[0] + whereConditions + orderByCondition;  
 }  
  
 localWhereConditionHash.clear();  
 localSingleConditionHash.clear();  
 resetStringValues();  
 return queryToReturn;  
 }  
  
 private void appendLocalSingleWhereConditions(Map<String, String> localSingleConditionHash) {  
 Set<String> singleConditionKeys = localSingleConditionHash.keySet();  
 for (String condition : singleConditionKeys) {  
 switch (condition.toUpperCase()) {  
 case "ORDERBY":  
 orderByCondition = orderByCondition.append(" ORDER BY ").append(localSingleConditionHash.get(condition))  
 .append(" ");  
 break;  
 case "OFFSET":  
 offsetValue = offsetValue.append(" OFFSET ")  
 .append(Integer.*parseInt*(localSingleConditionHash.get(condition))  
 \* Integer.*parseInt*(localSingleConditionHash.get("LIMIT")))  
 .append(" ROWS ");  
 break;  
 case "LIMIT":  
 limitValue = limitValue.append(" FETCH NEXT ").append(localSingleConditionHash.get(condition))  
 .append(" ROWS ONLY");  
 break;  
 case "SORTWAY":  
 sortWay = sortWay.append(" ").append(localSingleConditionHash.get(condition));  
 break;  
 default: // default clause should be the last one  
 break;  
 }  
 }  
 }  
  
 private void appendLocalWhereConditions(Map<String, String> localWhereConditionHash, String field) {  
 if (localWhereConditionHash.get(field).isEmpty()) {  
 whereConditions = whereConditions.append(field).append(" AND ");  
 } else {  
 if (field.contains("like")) {  
 whereConditions = whereConditions.append(field).append("'").append(localWhereConditionHash.get(field))  
 .append("' AND ");  
 } else {  
 whereConditions = whereConditions.append(field).append("='").append(localWhereConditionHash.get(field))  
 .append("' AND ");  
 }  
 }  
 }  
  
 private void updateLocalWhereConditions(Map<String, String> localWhereConditionHash, String field) {  
 if (localWhereConditionHash.get(field).isEmpty()) {  
 whereConditions = whereConditions.append(field);  
 } else {  
 if (field.contains("like")) {  
 whereConditions = whereConditions.append(field).append("'").append(localWhereConditionHash.get(field))  
 .append("'");  
 } else {  
 whereConditions = whereConditions.append(field).append("='").append(localWhereConditionHash.get(field))  
 .append("'");  
 }  
 }  
 }  
  
 private void resetStringValues() {  
 whereConditions.setLength(0);  
 whereConditions.append(" WHERE ");  
 orderByCondition.setLength(0);  
 offsetValue.setLength(0);  
 limitValue.setLength(0);  
 sortWay.setLength(0);  
  
 }  
  
 public String insertServFuncPrvcyReq(String reqId, String servFuncCode, String searchKey) {  
 Integer servFuncPrvReqId = ThreadLocalRandom.*current*().nextInt(800000, 900000 + 1);  
 dBOperationUtils.executeInsertStatement(  
 "insert into gdp\_owner.serv\_func\_prvcy\_rqst (serv\_func\_prvcy\_rqst\_id, prvcy\_rqst\_id, serv\_func\_cd, prvcy\_rqst\_pyld\_txt, prvcy\_resp\_pyld\_txt, rqst\_send\_ts, resp\_rcv\_ts, resp\_rvw\_ts,prvcy\_resp\_disp\_txt) values("  
 + servFuncPrvReqId + ",'" + reqId + "','" + servFuncCode + "','" + searchKey  
 + "',null,(SELECT CURRENT\_TIMESTAMP::timestamp without time zone),null,null,null)");  
 return servFuncPrvReqId.toString();  
 }  
  
 public void deleteRecordsFromDBTable(String table, String condition) {  
 dBOperationUtils.executeDeleteStatement("delete from " + table + " where " + condition);  
 }  
  
 @Given("there exists a product $prdct\_cd with name $prdct\_nam")  
 public void insertProduct(@Named("prdct\_cd") String productCode, @Named("prdct\_nam") String productName) {  
 String query = String.*format*(INSERT\_PRODUCT, productCode, productName);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a service definition with code $serv\_cd with product code $prdct\_cd and service name $serv\_nam")  
 public void insertService(@Named("$serv\_cd") String serviceCode, @Named("$prdct\_cd") String productCode,  
 @Named("$serv\_nam") String serviceName) {  
 String query = String.*format*(INSERT\_SERV\_FUNC, serviceCode, productCode, serviceName);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a service function definition $serv\_func\_cd with service code $serv\_cd, description $serv\_func\_desc, "  
 + "type code $serv\_func\_type\_cd, owner email address $ownr\_email\_addr, business context $busn\_cntxt\_cd")  
 public void insertServiceFunctionDef(@Named("$serv\_func\_cd") String serviceFunctionCode,  
 @Named("$serv\_cd") String serviceCode, @Named("$serv\_func\_desc") String serviceFunctionDesc,  
 @Named("$serv\_func\_type\_cd") String serviceFunctionTypeCode,  
 @Named("$ownr\_email\_addr") String ownerEmailAddress, @Named("$busn\_cntxt\_cd") String businessContextCode) {  
 String query = String.*format*(INSERT\_SERV\_FUNC\_DEF, serviceFunctionCode, serviceCode, serviceFunctionDesc,  
 serviceFunctionTypeCode, ownerEmailAddress, businessContextCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a service function integration record with details $serv\_func\_integration\_id, $serv\_func\_cd, $gdp\_func\_cd, $serv\_integration\_type\_cd")  
 public void insertServiceFunctionIntegrationDetails(  
 @Named("$serv\_func\_integration\_id") int servicceFunctionIntegrationId,  
 @Named("$serv\_func\_cd") String serviceFunctionCode, @Named("$gdp\_func\_cd") String gdpFunctionCode,  
 @Named("$serv\_integration\_type\_cd") String serviceIntegrationTypeCode) {  
 String query = String.*format*(INSERT\_SERV\_FUNC\_INTG, servicceFunctionIntegrationId, serviceFunctionCode,  
 gdpFunctionCode, serviceIntegrationTypeCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a data element with details $elmt\_tag\_txt, $elmt\_desc, $elmt\_type\_cd, $hide\_sw and $editable\_sw")  
 public void insertProduct(@Named("elmt\_tag\_txt") String elementTagText,  
 @Named("elmt\_desc") String elementDescription, @Named("elmt\_type\_cd") String elementTypecode,  
 @Named("hide\_sw") boolean hideSW, @Named("editable\_sw") boolean editableSW) {  
 String query = String.*format*(INSERT\_DATA\_ELEM, elementTagText, elementDescription, elementTypecode, hideSW,  
 editableSW);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a service function data element table with details $serv\_func\_data\_elmt\_id, $serv\_func\_cd, $key\_sw, $elmt\_tag\_txt, $del\_key\_sw, $disp\_order\_num")  
 public void insertServiceFunctionDataElement(@Named("serv\_func\_data\_elmt\_id") int serviceFunctionDataElementId,  
 @Named("serv\_func\_cd") String serviceFunctionCode, @Named("key\_sw") boolean keySW,  
 @Named("elmt\_tag\_txt") String elementTagText, @Named("del\_key\_sw") boolean deleteKeySW,  
 @Named("disp\_order\_num") int displayOrderNumber) {  
  
 String query = String.*format*(INSERT\_SERV\_FUNC\_DATA\_ELEM, serviceFunctionDataElementId, serviceFunctionCode,  
 keySW, elementTagText, deleteKeySW, displayOrderNumber);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 public String createPrvcyReqsParentAndChild(String context, String reqTypeCd, String portalTypeCode) {  
  
 Integer reqId = ThreadLocalRandom.*current*().nextInt(800000, 900000 + 1);  
 insertPrivacyRequestWithTypeCode(reqId, reqTypeCd, context, "I", portalTypeCode);  
  
 return reqId.toString();  
 }  
  
 @Given("DELETE records from prvcy\_rqst table")  
 public void deleteFromPrvcyRqstTable() {  
 if (!prvcyRqstIdsInServFuncPrvcyRqst.isEmpty()) {  
 dBOperationUtils  
 .executeDeleteStatement("DELETE from prvcy\_rqst WHERE prvcy\_rqst\_id in (" + prvcyRqstListStr + ")");  
 prvcyRqstListStr.setLength(0);  
 }  
 }  
  
 @Given("there exists a product level privacy request with details $prdct\_prvcy\_rqst\_id, $prvcy\_rqst\_id, $prdct\_cd, $prdct\_rqst\_stat\_cd")  
 public void insertProductPrvcyRqst(@Named("$prdct\_prvcy\_rqst\_id") int productPrivacyRequestID,  
 @Named("$prvcy\_rqst\_id") int privacyRequestID, @Named("$$prdct\_cd") String productCode,  
 @Named("$prdct\_rqst\_stat\_cd") String productRequestStatusCode) {  
 String query = String.*format*(INSERT\_PRDCT\_PRVCY\_RQST, productPrivacyRequestID, privacyRequestID, productCode,  
 productRequestStatusCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a service function privacy request with details $serv\_func\_prvcy\_rqst\_id, $prvcy\_rqst\_id, $serv\_func\_cd, $prvcy\_rqst\_pyld\_txt, $prdct\_prvcy\_rqst\_id, $prdct\_cd, $serv\_func\_rqst\_stat\_cd")  
 public void insertServiceFunctionPrivacyRequest(  
 @Named("$serv\_func\_prvcy\_rqst\_id") int serviceFunctionPrivacyRequestId,  
 @Named("$prvcy\_rqst\_id") int privacyRequestId, @Named("$serv\_func\_cd") String serviceFunctionCode,  
 @Named("$prvcy\_rqst\_pyld\_txt") String privacyRequestPayloadText,  
 @Named("$prdct\_prvcy\_rqst\_id") int productPrivacyRequestId, @Named("$prdct\_cd") String productCode,  
 @Named("$serv\_func\_rqst\_stat\_cd") String serviceFunctionRequestStatusCode) {  
 String query = String.*format*(INSERT\_SERV\_FUNC\_PRVCY\_RQST, serviceFunctionPrivacyRequestId, privacyRequestId,  
 serviceFunctionCode, privacyRequestPayloadText, productPrivacyRequestId, productCode,  
 serviceFunctionRequestStatusCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 public String insertRecordsIntoPrdcPrvcyRqstTable(String prvcyRqstId, String context, String servFuncCode,  
 String payloadText) {  
 try {  
 String query = String.*format*(EXTRACT\_PRDCT\_CD, context, servFuncCode);  
 String prdctCode = dBOperationUtils.executeSelectStmt(query).get(0).get("prdct\_cd");  
 Integer prdctPrvcyReqId = ThreadLocalRandom.*current*().nextInt(800000, 900000 + 1);  
 this.insertProductPrvcyRqst(prdctPrvcyReqId, Integer.*parseInt*(prvcyRqstId), prdctCode, "I");  
  
 // insert records into serv\_func\_prvcy\_rqst  
 Integer servFuncPrvReqId = ThreadLocalRandom.*current*().nextInt(800000, 900000 + 1);  
  
 insertServiceFunctionPrivacyRequest(servFuncPrvReqId, Integer.*parseInt*(prvcyRqstId), servFuncCode,  
 payloadText, prdctPrvcyReqId, prdctCode, "I");  
  
 return servFuncPrvReqId.toString();  
 } catch (Exception e) {  
 throw new GDPUIException(e.getMessage(), e);  
 }  
 }  
  
 @Given("there exist a product record with legal note $prdct\_cd with name $prdct\_name and $legal\_note\_cd")  
 public void insertProductWithLegalNote(@Named("prdct\_cd") String productCode,  
 @Named("prdct\_nam") String productName, @Named("legal\_note\_cd") String legalNote) {  
 String query = String.*format*(INSERT\_PRODUCT\_QUERY\_LEGALNOTE, productCode, productName, legalNote);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a note translation record $note\_cd, $loc\_cd, $note\_trnslt\_txt")  
 public void insertNoteTranslation(@Named("note\_cd") String noteCode, @Named("loc\_cd") String locationCode,  
 @Named("note\_trnslt\_txt") String noteTranslationText) {  
 String query = String.*format*(INSERT\_QUERY\_NOTE\_TRANSLATION, noteCode, locationCode, noteTranslationText);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a privacy request with valid names details $prvcy\_rqst\_id, $rqst\_type\_cd, $rqst\_cntxt\_cd, $rqst\_stat\_cd, $srch\_key\_txt, $vw\_rqst\_id")  
 public void insertRequestDetailsForDomain(@Named("$prvcy\_rqst\_id") int privacyRequestId,  
 @Named("$rqst\_type\_cd") String requestTypeCode, @Named("$rqst\_cntxt\_cd") String requestContextCode,  
 @Named("$rqst\_stat\_cd") String requestStatusCode, @Named("$srch\_key\_txt") String searchKeyText,  
 @Named("$vw\_rqst\_id") String viewRequestId) {  
  
 String firstName = firstNames.get(commonFunctions.generateRandomNumberInRange(firstNames.size() - 1));  
 String lastName = lastNames.get(commonFunctions.generateRandomNumberInRange(lastNames.size() - 1));  
 String query = String.*format*(INSERT\_REQUEST\_B2B, privacyRequestId, requestTypeCode, requestContextCode,  
 requestStatusCode, firstName, lastName, searchKeyText, viewRequestId);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a response record with disclosure switch and details $resp\_rec\_id, $serv\_func\_prvcy\_rqst\_id, $prdct\_prvcy\_rqst\_id, $resp\_rec\_nam, $resp\_data\_txt, $resp\_rec\_stat\_cd,$dsclsr\_sw")  
 public void insertServiceFunctionPrivacyRequest(@Named("$resp\_rec\_id") int responseRecordId,  
 @Named("$serv\_func\_prvcy\_rqst\_id") int serviceFunctionPrvcyRqstId,  
 @Named("$prdct\_prvcy\_rqst\_id") int prdctPrvcyRqstId, @Named("$resp\_rec\_nam") String responseRecordName,  
 @Named("$resp\_data\_txt") String responseDataText, @Named("$resp\_rec\_stat\_cd") String responseRecordStatCode,  
 @Named("$dsclsr\_sw") boolean disclosureSwitch) {  
 String query = String.*format*(INSERT\_RESPONSE\_RECORDS, responseRecordId, serviceFunctionPrvcyRqstId,  
 prdctPrvcyRqstId, responseRecordName, responseDataText, responseRecordStatCode, disclosureSwitch);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Then("update response record table with delete privacy request id $deleteProductPrivacyRequestId for the product privacy request id $productPrivacyRequestId")  
 public void insertDeleteRecord(@Named("$deleteProductPrivacyRequestId") int deleteProductPrivacyRequestId,  
 @Named("$productPrivacyRequestId") int productPrivacyRequestId) {  
 String query = String.*format*(UPDATE\_RESPONSE\_RECORD\_WITH\_DELETE\_ID, deleteProductPrivacyRequestId,  
 productPrivacyRequestId);  
 dBOperationUtils.executeUpdateStatement(query);  
 }  
  
 @Then("update response record table with delete privacy request id $deletePrivacyRequestId for the response record id $responseRecordId")  
 public void insertPartialDeletedRecord(@Named("deletePrivacyRequestId") int deleteProductPrivacyRequestId,  
 @Named("responseRecordId") int responseRecordId) {  
 String query = String.*format*(DELETE\_PARTIAL\_RECORDS, deleteProductPrivacyRequestId, responseRecordId);  
 dBOperationUtils.executeUpdateStatement(query);  
 }  
  
 @Given("there exists a service function definition $serv\_func\_cd with service code $serv\_cd, description $serv\_func\_desc, "  
 + "type code $serv\_func\_type\_cd, business context $busn\_cntxt\_cd, $legal\_note\_cd")  
 public void insertServiceFunctionWithLegalNote(@Named("$serv\_func\_cd") String serviceFunctionCode,  
 @Named("$serv\_cd") String serviceCode, @Named("$serv\_func\_desc") String serviceFunctionDesc,  
 @Named("$serv\_func\_type\_cd") String serviceFunctionTypeCode,  
 @Named("$busn\_cntxt\_cd") String businessContextCode, @Named("$legal\_note\_cd") String legalNoteCode) {  
 String query = String.*format*(INSERT\_SERV\_FUNC\_LEGAL\_NOTE, serviceFunctionCode, serviceCode, serviceFunctionDesc,  
 serviceFunctionTypeCode, businessContextCode, legalNoteCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a label translate record with details $label\_trnslt\_id, $label\_cd, $loc\_cd, $label\_trnslt\_cd")  
 public void insertLabelTranslateElement(@Named("label\_trnslt\_id") int labelTranslateId,  
 @Named("label\_cd") String labelCode, @Named("loc\_cd") String locationCode,  
 @Named("label\_trnslt\_cd") String labelTranslateCode) {  
 String queryToExecute = String.*format*(QueryConstants.*GET\_LABEL\_TRANSLATION\_RECORD*, locationCode, labelCode);  
 List<LabelTranslationEntity> labelTrnsltList = dBOperationUtils.getCompleteRowData(queryToExecute,  
 LabelTranslationHandler.class);  
 if (labelTrnsltList.isEmpty()) {  
 String query = String.*format*(INSERT\_QUERY\_LABEL\_TRANSLATE, labelTranslateId, labelCode, locationCode,  
 labelTranslateCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
 }  
  
 @Given("the translated labels available for service function code $serviceFunctionCode in $langulage language with default values and with display order")  
 public void readLablesForServiceFunction(@Named("serviceFunctionCode") String serviceFunctionCode,  
 @Named("langulage") String langulage) {  
  
 String query = String.*format*(SELECT\_QUERY\_LABEL\_TRANSLATE, serviceFunctionCode, langulage);  
 String defaultQuery = String.*format*(SELECT\_QUERY\_LABEL\_TRANSLATE, serviceFunctionCode, "en");  
 List<Map<String, String>> result1 = dBOperationUtils.executeSelectStmt(query);  
 List<Map<String, String>> result2 = dBOperationUtils.executeSelectStmt(defaultQuery);  
 // TO-DO: we need to refine the results so that we can the content of Englisg if  
 // any given language keys are missing.  
 // TO-DO: prepare the map of key and translation.  
  
 result1.addAll(result2);  
 }  
  
 @Given("there exists a service function definition $serv\_func\_cd with service code $serv\_cd, description $serv\_func\_desc, type code $serv\_func\_type\_cd, owner email address $ownr\_email\_addr,legal note cd $legal\_note\_cd,business context $busn\_cntxt\_cd")  
 public void insertServiceFunctionDefWithLegalNote(@Named("$serv\_func\_cd") String serviceFunctionCode,  
 @Named("$serv\_cd") String serviceCode, @Named("$serv\_func\_desc") String serviceFunctionDesc,  
 @Named("$serv\_func\_type\_cd") String serviceFunctionTypeCode,  
 @Named("$ownr\_email\_addr") String ownerEmailAddress, @Named("$legal\_note\_cd") String legalNoteCode,  
 @Named("$busn\_cntxt\_cd") String businessContextCode) {  
 String query = String.*format*(INSERT\_SERV\_FUNC\_DEF\_WITH\_LEGALNOTE, serviceFunctionCode, serviceCode,  
 serviceFunctionDesc, serviceFunctionTypeCode, ownerEmailAddress, legalNoteCode, businessContextCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("the records are deleted from serv func privacy for $requestID and product code $productCode")  
 @Alias("the records are deleted from service function privacy request for $requestID and product code $productCode")  
 public void deleteOtherRecords(@Named("$requestID") String requestID, @Named("$productCode") String productCode) {  
 dBOperationUtils.executeDeleteStatement("DELETE FROM serv\_func\_prvcy\_rqst where prdct\_cd not like '%"  
 + productCode + "%' AND prvcy\_rqst\_id = " + requestID + ";");  
 dBOperationUtils.executeDeleteStatement("delete from prdct\_prvcy\_rqst where prdct\_cd not like '%" + productCode  
 + "%' AND prvcy\_rqst\_id = " + requestID + " ;");  
 }  
  
 @Given("there exists a privacy request with details $prvcy\_rqst\_id, $rqst\_type\_cd, $rqst\_cntxt\_cd, $rqst\_stat\_cd, $rqst\_email\_addr, $srch\_key\_txt, $vw\_rqst\_id")  
 public void insertRequestDetailsForB2CDomain(@Named("prvcy\_rqst\_id") int privacyRequestId,  
 @Named("rqst\_type\_cd") String requestTypeCode, @Named("rqst\_email\_addr") String requesterEmailAddress,  
 @Named("rqst\_cntxt\_cd") String requestContextCode, @Named("rqst\_stat\_cd") String requestStatusCode,  
 @Named("srch\_key\_txt") String searchKeyText, @Named("vw\_rqst\_id") String viewRequestId) {  
  
 String firstName = firstNames.get(commonFunctions.generateRandomNumberInRange(firstNames.size() - 1));  
 String lastName = lastNames.get(commonFunctions.generateRandomNumberInRange(lastNames.size() - 1));  
 String query = String.*format*(INSERT\_REQUEST\_B2C, privacyRequestId, requestTypeCode, requestContextCode,  
 requesterEmailAddress, requestStatusCode, firstName, lastName, searchKeyText, viewRequestId);  
 dBOperationUtils.executeInsertStatement(query);  
  
 }  
  
 @Given("there exists a privacy request for B2C dashboard with valid details $prvcy\_rqst\_id, $rqst\_type\_cd, $rqst\_stat\_cd, $srch\_key\_txt")  
 public void insertRequestDetailsForB2C(@Named("$prvcy\_rqst\_id") int privacyRequestId,  
 @Named("$rqst\_type\_cd") String requestTypeCode, @Named("$rqst\_stat\_cd") String requestStatusCode,  
 @Named("$srch\_key\_txt") String searchKeyText) {  
  
 String firstName = "seahawks\_firstname";  
 String lastName = "seahawks\_lastname";  
 String query = String.*format*(INSERT\_REQUESTB2C, privacyRequestId, requestTypeCode, requestStatusCode, firstName,  
 lastName, searchKeyText, null);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("create a privacy request for B2C dashboard with valid details and time interval $prvcy\_rqst\_id, $rqst\_type\_cd, $rqst\_stat\_cd, $srch\_key\_txt, $interval\_minute, $tst\_rqst\_sw, $initiator\_type\_cd")  
 public void insertRequestDetailsForB2CWithInterval(@Named("$prvcy\_rqst\_id") int privacyRequestId,  
 @Named("$rqst\_type\_cd") String requestTypeCode, @Named("$rqst\_stat\_cd") String requestStatusCode,  
 @Named("$srch\_key\_txt") String searchKeyText, @Named("interval\_minute") int interval,  
 @Named("$tst\_rqst\_sw") boolean testRequestSwitch, @Named("$initiator\_type\_cd") String initiatorTypeCode) {  
 String firstName = "b2c";  
 String lastName = "user";  
 String requestorUserId = testRequestSwitch ? testUserId : userId;  
 String requestorEmailId = testRequestSwitch ? "b2cvalidator1@mastercard.com" : "b2c@test.com";  
 int timeInterval = Math.*abs*(interval);  
 String finalQuery = INSERT\_REQUEST\_B2C\_WITH\_INTERVAL.replace("time\_interval", Integer.*toString*(timeInterval));  
 String query = String.*format*(finalQuery, privacyRequestId, requestTypeCode, requestorEmailId, requestStatusCode,  
 requestorUserId, requestorUserId, firstName, lastName, searchKeyText, null, testRequestSwitch,  
 initiatorTypeCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a product level privacy request with timestamp $prdct\_prvcy\_rqst\_id, $prvcy\_rqst\_id, $prdct\_cd, $prdct\_rqst\_stat\_cd, $prdct\_ver\_stat\_cd")  
 public void insertProductWithTime(@Named("$prdct\_prvcy\_rqst\_id") int productPrivacyRequestID,  
 @Named("$prvcy\_rqst\_id") int privacyRequestID, @Named("$prdct\_cd") String productCode,  
 @Named("$prdct\_rqst\_stat\_cd") String productRequestStatusCode,  
 @Named("$prdct\_ver\_stat\_cd") String productVerStatusCode) {  
 String query = String.*format*(INSERT\_QUERY\_TIMESTAMP, productPrivacyRequestID, privacyRequestID, productCode,  
 productRequestStatusCode, productVerStatusCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a privacy request with requster email $prvcy\_rqst\_id, $rqst\_type\_cd, $rqst\_cntxt\_cd,$rqst\_email\_addr, $rqst\_stat\_cd, $vw\_rqst\_id")  
 public void insertPrivacyRequestWithRequsterEmaiAddress(@Named("$prvcy\_rqst\_id") int privacyRequestId,  
 @Named("$rqst\_type\_cd") String requestTypeCode, @Named("$rqst\_cntxt\_cd") String requestContextCode,  
 @Named("$rqst\_email\_addr") String requesterEmailAddress, @Named("$rqst\_stat\_cd") String requestStatusCode,  
 @Named("$vw\_rqst\_id") String viewRequestID) {  
 String query = String.*format*(PRVY\_INSERT\_QUERY, privacyRequestId, requestTypeCode, requestContextCode,  
 requesterEmailAddress, requestStatusCode, viewRequestID);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a product level privacy request with b2c status $prdct\_prvcy\_rqst\_id, $prvcy\_rqst\_id, $prdct\_cd, $prdct\_rqst\_stat\_cd, $prdct\_ver\_stat\_cd")  
 public void insertProductPrvcyRqstWithVersionStatus(@Named("$prdct\_prvcy\_rqst\_id") int productPrivacyRequestID,  
 @Named("$prvcy\_rqst\_id") int privacyRequestID, @Named("$$prdct\_cd") String productCode,  
 @Named("$prdct\_rqst\_stat\_cd") String productRequestStatusCode,  
 @Named("$prdct\_ver\_stat\_cd") String productVerStatusCode) {  
 String query = String.*format*(*INSERT\_PRDCT\_PRVCY\_RQST\_WITH\_LATEST\_STATUS*, productPrivacyRequestID,  
 privacyRequestID, productCode, productRequestStatusCode, productVerStatusCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("there exists a product level privacy request with interval of days timestamp $prdct\_prvcy\_rqst\_id, $prvcy\_rqst\_id, $prdct\_cd, $prdct\_rqst\_stat\_cd, $prdct\_ver\_stat\_cd, $daysOfInterval")  
 public void insertProductWithTimeInterval(@Named("$prdct\_prvcy\_rqst\_id") int productPrivacyRequestID,  
 @Named("$prvcy\_rqst\_id") int privacyRequestID, @Named("$prdct\_cd") String productCode,  
 @Named("$prdct\_rqst\_stat\_cd") String productRequestStatusCode,  
 @Named("$prdct\_ver\_stat\_cd") String productVerStatusCode, @Named("daysOfInterval") String daysOfInterval) {  
 String query = String.*format*(*INSERT\_QUERY\_TIMESTAMP\_INTERVAL*, productPrivacyRequestID, privacyRequestID,  
 productCode, productRequestStatusCode, daysOfInterval, productVerStatusCode);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("SQL script $sqlFilePath is executed to insert the test data in database")  
 @Alias("SQL script $sqlFilePath is executed to delete the test data in database")  
 public void executeSqlScript(@Named("sqlFilePath") String sqlFilePath) throws SQLException {  
 String sqlFile = null;  
 if (sqlFilePath != null && !sqlFilePath.isEmpty()) {  
 if (new ClassPathResource("/config/" + environment + sqlFilePath).exists()) {  
 sqlFile = "/config/" + environment + sqlFilePath;  
 } else if (new ClassPathResource("/config/testdatafiles" + sqlFilePath).exists()) {  
 sqlFile = "/config/testdatafiles" + sqlFilePath;  
 }  
 dBOperationUtils.executeSqlScript(sqlFile);  
 contextVariablesSteps.setContextVariable("sqlFile", sqlFile);  
 }  
 }  
  
 @Given("SQL script $sqlFilePath is executed to insert the test data in Oracle Database")  
 public void executeSqlScriptInOracleDB(@Named("sqlFilePath") String sqlFilePath) throws SQLException {  
 String sqlFile = null;  
 if (sqlFilePath != null && !sqlFilePath.isEmpty()) {  
 if (new ClassPathResource("/config/" + environment + sqlFilePath).exists()) {  
 sqlFile = "/config/" + environment + sqlFilePath;  
 } else if (new ClassPathResource("/config/testdatafiles" + sqlFilePath).exists()) {  
 sqlFile = "/config/testdatafiles" + sqlFilePath;  
 }  
 dbOperationOracleUtils.executeSqlScript(sqlFile);  
 contextVariablesSteps.setContextVariable("sqlFileOracle", sqlFile);  
 }  
 }  
  
 @Given("update existing requests for the user with details $emailToBeUpdated, $createdUserIdToBeUpdated, $createdUserId")  
 public void updatePrivacyRequestsUpdatedQuery(@Named("$emailToBeUpdated") String emailToBeUpdated,  
 @Named("$createdUserIdToBeUpdated") String createdUserIdToBeUpdated,  
 @Named("$createdUserId") String createdUserId) {  
 dBOperationUtils  
 .executeUpdateStatement(String.*format*(QueryConstants.*UPDATE\_PRIVACY\_REQUEST\_DETAILS\_FOR\_B2C\_UPDATED*,  
 emailToBeUpdated, createdUserIdToBeUpdated, createdUserId));  
 }  
  
 @When("request details requester email, portal type code, createUserId and locale code are updated as $requesterEmailAddress, $portalTypeCode, $createUserId, $localeCode for requestID $requestID")  
 public void updateRequestAsB2CPortal(@Named("requesterEmailAddress") String requesterEmailAddress,  
 @Named("portalTypeCode") String portalTypeCode, @Named("createUserId") String createUserId,  
 @Named("localeCode") String localeCode, @Named("requestID") String requestID) {  
 String updateRequesterEmailQuery = String.*format*(QueryConstants.*UPDATE\_REQUESTER\_EMAIL*, requesterEmailAddress,  
 requestID);  
 String updateCreateUserIdQuery = String.*format*(QueryConstants.*UPDATE\_CREATE\_USER*, createUserId, requestID);  
 String updatePortalCodeQuery = String.*format*(QueryConstants.*UPDATE\_PORTAL\_TYPE\_CODE*, portalTypeCode, requestID);  
 String updateLocaleCode = String.*format*(QueryConstants.*UPDATE\_REQUEST\_LOCALE*, localeCode, requestID);  
 dBOperationUtils.executeUpdateStatement(updateRequesterEmailQuery);  
 dBOperationUtils.executeUpdateStatement(updatePortalCodeQuery);  
 dBOperationUtils.executeUpdateStatement(updateLocaleCode);  
 dBOperationUtils.executeUpdateStatement(updateCreateUserIdQuery);  
 }  
  
 public void insertRecordsIntoPrivacyRequestAction(Integer prvcyRqstActnId, Integer privacyRequestID,  
 String internalComments, String messageToRequester, String createdUserId, String createdUserName,  
 String prvcyRqstActnCd) {  
 String query = String.*format*(*INSERT\_RECORD\_INTO\_PRVCY\_RQST\_ACTN*, prvcyRqstActnId, privacyRequestID,  
 internalComments, messageToRequester, createdUserId, createdUserName, prvcyRqstActnCd);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 @Given("Auto complete flag for request $requestId is updated to $flag in prvcy\_rqst\_table")  
 public void updateAutoCompleteSwitchToTrueOrFalse(@Named("requestId") String requestId,  
 @Named("flag") String flag) {  
 dBOperationUtils.executeUpdateStatement(  
 String.*format*(QueryConstants.*UPDATE\_AUTO\_COMPLETE\_SWITCH*, Boolean.*valueOf*(flag), requestId));  
 }  
  
 public void insertRecordsIntoAudHist(int prvcyRqstAudHistId, String audTypeCode, String audDesc, String audUserId,  
 int requestId) {  
 String query = String.*format*(*INSERT\_RECORD\_INTO\_AUD\_HIS*, prvcyRqstAudHistId, audTypeCode, audDesc, audUserId,  
 requestId);  
 dBOperationUtils.executeInsertStatement(query);  
 }  
  
 public void updatePrvcyRqstIdProof(String revUserId, String revUserName, boolean apprSw, int requestId) {  
 String query = String.*format*(*UPDATE\_PRVCY\_RQST\_ID\_PROOF*, revUserId, revUserName, apprSw, requestId);  
 dBOperationUtils.executeUpdateStatement(query);  
 }  
  
 public String getLocalCodeForPrivacyRequest(String requestId) {  
 return dBOperationUtils.executeSelectStmt(String.*format*(*LOCAL\_CODE\_FOR\_PRVCY\_REQUEST*, requestId)).get(0)  
 .get("type\_nam");  
 }  
  
 public void updateServiceFunctionPrvcyRqst(String columnName, String value, int requestId) {  
 String query = String.*format*(*UPDATE\_SERVICE\_FUNC\_PRVCY\_RQST*, columnName, value, requestId);  
 dBOperationUtils.executeUpdateStatement(query);  
 }  
  
 @Given("the required test data is set-up using query: $queryTable")  
 public void insertTestData(@Named("queryTable") ExamplesTable queryTable) {  
 queryTable.getRows().forEach(value -> dBOperationUtils.executeInsertStatement(value.get("query")));  
 }  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Properties;  
  
import org.apache.log4j.Logger;  
import org.springframework.stereotype.Component;  
  
import com.mastercard.testing.gdp.ui.tests.exception.GDPUIException;  
  
@Component  
public class ReadExternalPropertyFile {  
  
 private static Logger *logger* = Logger.*getLogger*(ReadExternalPropertyFile.class);  
  
 private ReadExternalPropertyFile() {  
 }  
  
 public static String getPropertyValue(String fieldName) {  
 String versionString = null;  
 try {  
 Properties mainProperties = new Properties();  
 FileInputStream file;  
 String path = "./conf/userdetails.properties";  
 file = new FileInputStream(path);  
 mainProperties.load(file);  
 file.close();  
 versionString = mainProperties.getProperty(fieldName);  
 *logger*.info("Value fetched from file is " + versionString);  
 } catch (IOException ioException) {  
 *logger*.error("File not found !!!" + ioException);  
 throw new GDPUIException("Unable to find the property file.");  
 }  
 return versionString;  
 }  
  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import org.apache.log4j.Logger;  
  
import java.util.concurrent.TimeUnit;  
  
public final class SleepUtils {  
  
 private static Logger *logger* = Logger.*getLogger*(SleepUtils.class);  
  
 private SleepUtils() {  
 }  
  
 public static void sleepForSeconds(long seconds) {  
 *sleepForMilliseconds*(TimeUnit.*SECONDS*.toMillis(seconds));  
 }  
  
 public static void sleepForMilliseconds(long milliseconds) {  
 *logger*.info(String.*format*("Sleeping for %s milliseconds", milliseconds));  
 try {  
 Thread.*sleep*(milliseconds);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 *logger*.warn(e.getMessage(), e.getCause());  
 }  
 }  
}

package com.mastercard.testing.gdp.ui.tests.utils;  
  
import java.util.List;  
import java.util.Set;  
import java.util.concurrent.TimeUnit;  
import java.util.stream.Collectors;  
  
import org.apache.log4j.Logger;  
import org.openqa.selenium.By;  
import org.openqa.selenium.JavascriptExecutor;  
import org.openqa.selenium.Keys;  
import org.openqa.selenium.NoSuchElementException;  
import org.openqa.selenium.StaleElementReferenceException;  
import org.openqa.selenium.TimeoutException;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.interactions.Actions;  
import org.openqa.selenium.support.ui.ExpectedConditions;  
import org.openqa.selenium.support.ui.Select;  
import org.openqa.selenium.support.ui.WebDriverWait;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import com.mastercard.testing.gdp.ui.framework.GDPDriverProviderFactory;  
import com.mastercard.testing.gdp.ui.tests.constants.UIConstants;  
import com.mastercard.testing.gdp.ui.tests.helper.CustomWaiters;  
  
@Component  
public class WebDriverUtils {  
 protected static Logger *logger* = Logger.getLogger(WebDriverUtils.class);  
  
 @Autowired  
 private GDPDriverProviderFactory gdpDriverProviderfactory;  
  
 @Autowired  
 private CustomWaiters customWaiters;  
   
  
 @Autowired  
 private FormAppUrls formAppUrls;  
  
 private WebDriverUtils() {  
 }  
  
 private WebDriver getDriver() {  
 return gdpDriverProviderfactory.getDriverProvider().get();  
 }  
  
 private JavascriptExecutor getJavascriptExecutor() {  
 return ((JavascriptExecutor) getDriver());  
 }  
  
 public String getText(WebElement webElement) {  
 return webElement.getText();  
 }  
  
 public String getAttribute(WebElement webElement, String attribute) {  
 return webElement.getAttribute(attribute);  
 }  
 public String getValueAttributeByJS(WebElement element) {  
 return (getJavascriptExecutor().executeScript("return arguments[0].value;", element)).toString();  
 }  
  
 public void clearTextUsingJavaScriptExecutor(WebElement element) {  
 getJavascriptExecutor().executeScript("arguments[0].value ='';", element);  
 }  
  
 public void clearAndSendTextUsingJavaScriptExecutor(WebElement element,String text) {  
 getJavascriptExecutor().executeScript("arguments[0].value ='';", element);  
 getJavascriptExecutor().executeScript("arguments[0].value='" + text + "';", element);  
 }  
  
 public void click(WebElement webElement) {  
 new WebDriverWait(getDriver(), customWaiters.getElementWaitTime())  
 .ignoring(StaleElementReferenceException.class)  
 .until(ExpectedConditions.elementToBeClickable(webElement));  
 webElement.click();  
 }  
  
 public Boolean isDisplayed(WebElement webElement) {  
 return webElement.isDisplayed();  
 }  
  
 public Boolean isEnabled(WebElement webElement) {  
 return webElement.isEnabled();  
 }  
  
 public Boolean isSelected(WebElement webElement) {  
 return webElement.isSelected();  
 }  
  
 public void switchToGDPApplication() {  
 getDriver().manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);  
 // switch to the latest window.  
 for (String windowHandle : getDriver().getWindowHandles()) {  
 getDriver().switchTo().window(windowHandle);  
 }  
 }  
   
 public void switchToAppFrame() {  
 WebDriver webDriver = getDriver();  
 WebElement appFrame = webDriver.findElement(By.id("app-iframe"));  
 webDriver.switchTo().frame(appFrame);  
 }  
  
 public void clickUsingAction(By by) {  
 WebElement element = getDriver().findElement(by);  
 clickUsingAction(element);  
 }  
  
 public void clickUsingAction(WebElement element) {  
 Actions actions = new Actions(getDriver());  
 actions.moveToElement(element).click().build().perform();  
 }  
  
 public void mouseHover(WebElement element) {  
 Actions actions = new Actions(getDriver());  
 actions.moveToElement(element).build().perform();  
 }  
  
 public void clickWithJavaScriptExecutor(WebElement element) {  
 getJavascriptExecutor().executeScript("arguments[0].click();", element);  
 }  
  
 public void sendTextUsingJavaScriptExecutor(WebElement element, String text) {  
 getJavascriptExecutor().executeScript("arguments[0].value='" + text + "';", element);  
 }  
  
 public void sendTextUsingActionsClass(WebElement element, String text) {  
 Actions actions = new Actions(gdpDriverProviderfactory.getDriverProvider().get());  
 actions.moveToElement(element).click().sendKeys(element, text).build().perform();  
 }  
  
 public void clear(WebElement webElement) {  
 webElement.clear();  
 }  
  
 public void sendKeys(WebElement publishModalTextArea, String message) {  
 publishModalTextArea.sendKeys(message);  
 }  
   
 public void selectByVisibleText(WebElement webElement, String visibleText) {  
 Select dropdown = new Select(webElement);  
 dropdown.selectByVisibleText(visibleText);  
 }  
  
 public void selectByValue(WebElement webElement, String value) {  
 Select dropdown = new Select(webElement);  
 dropdown.selectByValue(value);  
 }  
  
 public void selectByValue(WebElement webElement, int index) {  
 Select dropdown = new Select(webElement);  
 dropdown.selectByIndex(index);  
 }  
  
 public void waitForAngularToLoad() {  
 waitForPageToLoad();  
 waitForAngularToLoad(customWaiters.getElementWaitTime());  
 }  
  
 public void waitForAngularToLoad(int timeOutInSeconds) {  
 String waitForAngularScript = "if (typeof window.getAllAngularTestabilities === \"function\") { var ta = window.getAllAngularTestabilities()[0]; return ta.\_isZoneStable && ta.\_pendingCount == 0;} else { return false;}";  
 WebDriverWait wait = new WebDriverWait(getDriver(), timeOutInSeconds);  
 wait.until(driver -> {  
 Boolean angularIsReadyForTests = (Boolean) ((JavascriptExecutor) driver)  
 .executeScript(waitForAngularScript);  
 logger.info("Angular is ready = " + angularIsReadyForTests);  
 return angularIsReadyForTests;  
 });  
 }  
  
 public void waitForAngularToFinishAllHttpRequests() {  
 String angularAllHttpRequestCompletedScript = "return angular.element(document).injector().get('$http').pendingRequests.length === 0;";  
 WebDriverWait wait = new WebDriverWait(getDriver(), 5);  
 wait.until(d -> {  
 Boolean requestsIsFinished = (Boolean) ((JavascriptExecutor) d)  
 .executeScript(angularAllHttpRequestCompletedScript);  
 logger.info("All http requests is finished = " + requestsIsFinished);  
 return requestsIsFinished;  
 });  
 }  
  
 public void forceAngularToRetrieveUserId() {  
 // This is workaround to go throw Consumer Auth,  
 // because js make two calls to get user id  
 // and sometimes response for first request came later than first one  
 String angularRetrieveCapIdScript = "return angular.element(document.getElementsByName('loginForm')).scope().retrieveCapId();";  
 getJavascriptExecutor().executeScript(angularRetrieveCapIdScript);  
 logger.info("Force retrieve user id for Consumer Auth");  
 }  
  
 public void jsClick(WebElement elementToJsClick) {  
 new WebDriverWait(getDriver(), customWaiters.getElementWaitTime())  
 .ignoring(StaleElementReferenceException.class)  
 .until(ExpectedConditions.elementToBeClickable(elementToJsClick));  
 getJavascriptExecutor().executeScript(UIConstants.JS\_EXECUTOR\_TO\_CLICK, elementToJsClick);  
 }  
 public void jsClickWithoutWait(WebElement elementToJsClick) {  
 getJavascriptExecutor().executeScript(UIConstants.JS\_EXECUTOR\_TO\_CLICK, elementToJsClick);  
 }  
  
  
 public Boolean isDisabled(WebElement element) {  
 return (boolean) getJavascriptExecutor().executeScript("return arguments[0].disabled;", element);  
 }  
  
 public void navigateBack() {  
   
 getDriver().navigate().back();  
 waitForAngularToLoad();  
 waitForAngularToLoad();  
   
 }  
  
 public void clearTextUsingSendKeys(WebElement webElement) {  
 webElement.click();  
 webElement.sendKeys(Keys.CONTROL + "a");  
 webElement.sendKeys(Keys.BACK\_SPACE);  
 }  
  
 public void replaceTextUsingSendKeys(WebElement webElement, String newValue) {  
 scrollToTheElement(webElement);  
 isDisplayed(webElement);  
 clearTextUsingSendKeys(webElement);  
 webElement.sendKeys(newValue);  
 webElement.sendKeys(Keys.TAB);  
 }  
  
 public boolean isElementAvailable(By locator, int sec) {  
 WebDriverWait wait = new WebDriverWait(getDriver(), sec);  
 try {  
 wait.until(ExpectedConditions.visibilityOfElementLocated(locator));  
 logger.info("Object is visible on webpage");  
 } catch (TimeoutException e) {  
 logger.info("object located by " + locator.toString() + " is not available on screen", e);  
 return false;  
 }  
 return true;  
 }  
  
 public boolean isElementPresent(By by) {  
 try {  
 getDriver().findElement(by);  
 return true;  
 } catch (NoSuchElementException e) {  
 logger.info(e);  
 return false;  
 }  
 }  
  
 public List<String> getListOfTextForElements(List<WebElement> elements) {  
 return elements.stream().map(WebElement::getText).collect(Collectors.toList());  
 }  
  
 public WebElement getChildElementByXpath(WebElement element, String locator) {  
 return element.findElement(By.xpath(locator));  
 }  
  
 public List<WebElement> getChildElementsByXpath(WebElement element, String locator) {  
 return element.findElements(By.xpath(locator));  
 }  
  
 public WebElement getChildElementById(WebElement element, String locator) {  
 return element.findElement(By.id(locator));  
 }  
  
 public String getCurrentUrl() {  
 return getDriver().getCurrentUrl();  
 }  
  
 public void scrollToTheElement(WebElement element) {  
 getJavascriptExecutor().executeScript("arguments[0].scrollIntoView(true);", element);  
 }  
  
 public void scrollToTopOfThePage() {  
 getJavascriptExecutor().executeScript("window.scrollTo(0, -document.body.scrollHeight)");  
 }  
   
 public void scrollToBottomOfThePage() {  
 getJavascriptExecutor().executeScript("window.scrollTo(0, document.body.scrollHeight)");  
 }  
  
 public void doubliClickUsingAction(WebElement element) {  
 Actions actions = new Actions(getDriver());  
 actions.moveToElement(element).doubleClick(element).click().build().perform();  
 }  
  
 public void selectCheckbox(WebElement checkboxElement) {  
 if (!checkboxElement.isSelected()) {  
 jsClick(checkboxElement);  
 } else {  
 logger.info("Element [" + checkboxElement + "] is already checked");  
 }  
 }  
  
 public void uncheck(WebElement checkboxElement) {  
 if (checkboxElement.isSelected()) {  
 clickWithJavaScriptExecutor(checkboxElement);  
 } else {  
 logger.info("Element [" + checkboxElement + "] is already unchecked");  
 }  
 }  
  
 public String getTagName(WebElement webElement) {  
 return webElement.getTagName();  
 }  
  
 public void switchToWindowByIndex(int windowIndex) {  
 Object[] windowHandles = getDriver().getWindowHandles().toArray();  
 if (windowIndex >= 0) {  
 getDriver().switchTo().window((String) windowHandles[windowIndex]);  
 logger.info("Window selected based on Index" + windowIndex);  
 getDriver().manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);  
 } else {  
 logger.info("Window index should be '>=0'");  
 }  
 }  
  
 public void closeCurrentWindow() {  
 getDriver().close();  
 }  
  
 public String getBackGroundColor(WebElement element, String color) {  
 return element.getCssValue(color);  
 }  
  
 public Boolean verifyPageScroll() {  
 String execScript = "return document.documentElement.scrollHeight>document.documentElement.clientHeight;";  
 return (Boolean) (getJavascriptExecutor().executeScript(execScript));  
  
 }  
  
 public Boolean verifyPageScrollImage(String fileType) {  
 String execScript = "return document.querySelector('.idv" + fileType  
 + "div').scrollHeight>document.querySelector('.idv" + fileType + "div').clientHeight;";  
 return (Boolean) (getJavascriptExecutor().executeScript(execScript));  
  
 }  
  
 public int getOpenWindowSize() {  
 Set<String> openWindow = getDriver().getWindowHandles();  
 return openWindow.size();  
 }  
  
 public List<WebElement> getOptions(WebElement webElement) {  
 Select dropdown = new Select(webElement);  
 return dropdown.getOptions();  
 }  
  
 private void waitForPageToLoad() {  
 String pageLoadStatus;  
 do {  
 pageLoadStatus = (String) getJavascriptExecutor().executeScript("return document.readyState");  
 logger.info("still loading....");  
 } while (!"complete".equals(pageLoadStatus));  
 logger.info("Page Loaded.");  
 }  
  
 public WebElement getFirstSelectedElement(WebElement webElement) {  
 Select dropdown = new Select(webElement);  
 return dropdown.getFirstSelectedOption();  
 }  
  
 public void refreshPage() {  
 gdpDriverProviderfactory.getDriverProvider().get().navigate().refresh();  
 customWaiters.waitForAngularToLoad();  
 }  
  
 public void navigateForward() {  
 getDriver().navigate().forward();  
 }  
  
 public void openLinkAndMaximizeWindow(String brokenEmailVerificationLink) {  
 WebDriver webDriver = getDriver();  
 webDriver.manage().window().maximize();  
 webDriver.navigate().to(brokenEmailVerificationLink);  
 }  
  
 public boolean isClickable(WebElement elem) {  
 try {  
 WebDriverWait wait = new WebDriverWait(customWaiters.getDriver(), customWaiters.getElementWaitTime());  
 wait.until(ExpectedConditions.elementToBeClickable(elem));  
 return true;  
 } catch (Exception e) {  
 logger.error(e.getMessage());  
 return false;  
 }  
 }  
  
 public boolean isElementNotAvailable(WebElement elem, int sec) {  
 try {  
 WebDriverWait wait = new WebDriverWait(customWaiters.getDriver(), sec);  
 wait.until(ExpectedConditions.invisibilityOf(elem));  
 } catch (TimeoutException e) {  
 logger.info("object is still available in page ", e);  
 return false;  
 }  
 return true;  
 }  
  
 public boolean isElementAvailable(WebElement elem, int sec) {  
 try {  
 WebDriverWait wait = new WebDriverWait(customWaiters.getDriver(), sec);  
 wait.until(ExpectedConditions.visibilityOf(elem));  
 } catch (TimeoutException e) {  
 logger.info("object is not available in page ", e);  
 return false;  
 }  
 return true;  
 }  
  
 public boolean isElementNotAvailableByLocator(By locator, int sec) {  
 try {  
 WebDriverWait wait = new WebDriverWait(customWaiters.getDriver(), sec);  
 wait.until(ExpectedConditions.invisibilityOfElementLocated(locator));  
 } catch (TimeoutException e) {  
 logger.info("locator is still available in page ", e);  
 return false;  
 }  
 return true;  
 }  
  
 public Boolean isChecked(WebElement element) {  
 return (boolean) getJavascriptExecutor().executeScript("return arguments[0].checked ;", element);  
 }  
  
 public void waitForElementToBeSelected(WebElement element, Boolean selectionState) {  
 WebDriverWait wait = new WebDriverWait(customWaiters.getDriver(), customWaiters.getElementWaitTime());  
 wait.until(ExpectedConditions.elementSelectionStateToBe(element, selectionState));  
 }  
  
 public String getFirstSelectedDropDownValue(WebElement webElement) {  
 return getFirstSelectedElement(webElement).getText();  
 }  
  
 public boolean isDropdownSuportsMultiSelect(WebElement webElement) {  
 Select dropdown = new Select(webElement);  
 return dropdown.isMultiple();  
 }  
  
 public List<String> getOptionsInDropdown(WebElement webElement) {  
 Select dropdown = new Select(webElement);  
 return dropdown.getOptions().stream().map(element -> element.getText()).collect(Collectors.toList());  
 }  
  
 public void maxmizeBrowser() {  
 customWaiters.getDriver().manage().window().maximize();  
 }  
   
 public String getBeforePseudoContent() {  
 String script = "return window.getComputedStyle( document.querySelector('input[type=radio]:checked+label.radio-label'),':before').getPropertyValue('color')";  
   
 JavascriptExecutor js = (JavascriptExecutor)getDriver();  
 return (String) js.executeScript(script);  
 }  
  
}