//  
// 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.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.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.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);  
 }  
}

<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.mastercard.gdp.testing</groupId>  
 <artifactId>gdp-mtaf-ui</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>gdp-mtaf-ui</name>  
 <description>Created for UI automation</description>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <jbehave.core.version>4.6</jbehave.core.version>  
 <jbehave.execution.threads>1</jbehave.execution.threads>  
 <aspectj.version>1.9.4</aspectj.version>  
 <mtaf.version>19.3.2.0-SNAPSHOT</mtaf.version>  
 <axon-sdk.version>1.4.0</axon-sdk.version>  
 <lombok.version>1.18.10</lombok.version>  
 <commons-lang.version>3.9</commons-lang.version>  
 <jackson.version>2.10.1</jackson.version>  
 <commons-logging.version>1.2</commons-logging.version>  
 <slf4j.version>1.7.30</slf4j.version>  
 <log4j.version>2.13.0</log4j.version>  
 <assertj.version>3.14.0</assertj.version>  
 <postgresql.version>42.2.9</postgresql.version>  
 <apache.commons.version>2.7.0</apache.commons.version>  
 <common.dbutils.version>1.7</common.dbutils.version>  
 <junit.version>5.5.2</junit.version>  
 <json-path.version>2.4.0</json-path.version>  
 <spring.version>5.1.8.RELEASE</spring.version>  
 <oracle.version>12.2.0.1</oracle.version>  
 <metaFilters></metaFilters>  
 <sonar.exclusions>pom.xml,  
 src/main/java/com/mastercard/testing/gdp/ui/tests/domain/\*\*/\*,  
 src/main/java/com/mastercard/testing/gdp/ui/tests/configuration/TestConfiguration.java  
 </sonar.exclusions>  
 <sonar.coverage.exclusions>  
 src/\*\*/\*  
 </sonar.coverage.exclusions>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>com.mastercard.quality.engineering</groupId>  
 <artifactId>mtaf-jbehave-tools</artifactId>  
 <version>${mtaf.version}</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.codehaus.jackson/jackson-mapper-asl -->  
 <dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>${jackson.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.commons</groupId>  
 <artifactId>commons-lang3</artifactId>  
 <version>${commons-lang.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>${lombok.version}</version>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <groupId>com.mastercard.quality.engineering</groupId>  
 <artifactId>mtaf-ui-tools</artifactId>  
 <version>${mtaf.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>com.ibatis</groupId>  
 <artifactId>ibatis2-common</artifactId>  
 <version>2.1.7.597</version>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>${junit.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.jbehave</groupId>  
 <artifactId>jbehave-core</artifactId>  
 <version>${jbehave.core.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.jbehave</groupId>  
 <artifactId>jbehave-core</artifactId>  
 <version>${jbehave.core.version}</version>  
 <classifier>resources</classifier>  
 <type>zip</type>  
 </dependency>  
 <dependency>  
 <groupId>org.jbehave</groupId>  
 <artifactId>jbehave-spring</artifactId>  
 <version>${jbehave.core.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>commons-logging</groupId>  
 <artifactId>commons-logging</artifactId>  
 <version>${commons-logging.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-core</artifactId>  
 <version>${log4j.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>${slf4j.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.assertj</groupId>  
 <artifactId>assertj-core</artifactId>  
 <version>${assertj.version}</version>  
 </dependency>  
 <!-- AspectJ -->  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjrt</artifactId>  
 <version>${aspectj.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>${aspectj.version}</version>  
 </dependency>  
 <!-- DB Utils -->  
 <dependency>  
 <groupId>commons-dbutils</groupId>  
 <artifactId>commons-dbutils</artifactId>  
 <version>${common.dbutils.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.commons</groupId>  
 <artifactId>commons-dbcp2</artifactId>  
 <version>${apache.commons.version}</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->  
 <dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <version>${postgresql.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>com.mastercard.apie.axon</groupId>  
 <artifactId>axon-sdk</artifactId>  
 <version>${axon-sdk.version}</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.json/json -->  
 <dependency>  
 <groupId>org.json</groupId>  
 <artifactId>json</artifactId>  
 <version>20180813</version>  
 </dependency>  
 <dependency>  
 <groupId>com.jayway.jsonpath</groupId>  
 <artifactId>json-path</artifactId>  
 <version>${json-path.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.seleniumhq.selenium</groupId>  
 <artifactId>selenium-api</artifactId>  
 <version>3.141.59</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-beans</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <!-- RP Portal -->  
 <dependency>  
 <groupId>com.epam.reportportal</groupId>  
 <artifactId>agent-java-jbehave</artifactId>  
 <version>5.0.0-BETA-4</version>  
 </dependency>  
 <dependency>  
 <groupId>com.epam.reportportal</groupId>  
 <artifactId>httpclient-repacked</artifactId>  
 <version>1.0.2</version>  
 </dependency>  
 <dependency>  
 <groupId>com.epam.reportportal</groupId>  
 <artifactId>client-java-core</artifactId>  
 <version>2.7.1</version>  
 </dependency>  
 <dependency>  
 <groupId>com.epam.reportportal</groupId>  
 <artifactId>logger-java-log4j</artifactId>  
 <version>2.6.1</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/com.google.guava/guava -->  
 <dependency>  
 <groupId>com.google.guava</groupId>  
 <artifactId>guava</artifactId>  
 <version>28.1-jre</version>  
 </dependency>  
 <dependency>  
 <groupId>io.rest-assured</groupId>  
 <artifactId>rest-assured</artifactId>  
 <version>4.2.0</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->  
 <dependency>  
 <groupId>org.apache.poi</groupId>  
 <artifactId>poi</artifactId>  
 <version>4.1.1</version>  
 </dependency>  
 <dependency>  
 <groupId>com.codeborne</groupId>  
 <artifactId>selenide</artifactId>  
 <version>5.11.1</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>com.codeborne</groupId>  
 <artifactId>selenide</artifactId>  
 <version>5.11.1</version>  
 <scope>compile</scope>  
 </dependency>  
 <dependency>  
 <groupId>com.browserstack</groupId>  
 <artifactId>browserstack-local-java</artifactId>  
 <version>1.0.3</version>  
 </dependency>  
 <dependency>  
 <groupId>com.oracle</groupId>  
 <artifactId>ojdbc8</artifactId>  
 <version>${oracle.version}</version>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <compilerVersion>3.3</compilerVersion>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-jar-plugin</artifactId>  
 <configuration>  
 <excludes>  
 <!--DO NOT ADD ANY FILE TO ARTIFACT AS IT IS NO WHERE USED -->  
 <exclude>\*\*/\*.\*</exclude>  
 </excludes>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.jbehave</groupId>  
 <artifactId>jbehave-maven-plugin</artifactId>  
 <version>${jbehave.core.version}</version>  
 <executions>  
 <execution>  
 <id>run-stories-as-embeddables</id>  
 <phase>integration-test</phase>  
 <configuration>  
 <storyTimeoutInSecs>1200</storyTimeoutInSecs>  
 <threads>${jbehave.execution.threads}</threads>  
 <includes>  
 <include>\*\*/GdpExampleWebStories.java</include>  
 </includes>  
 <metaFilters>  
 <metaFilter>${metaFilters}</metaFilter>  
 </metaFilters>  
 <systemProperties>  
 <property>  
 <name>java.awt.headless</name>  
 <value>true</value>  
 </property>  
 </systemProperties>  
 <ignoreFailureInStories>true</ignoreFailureInStories>  
 <ignoreFailureInView>false</ignoreFailureInView>  
 </configuration>  
 <goals>  
 <goal>unpack-view-resources</goal>  
 <goal>run-stories-as-embeddables</goal>  
 </goals>  
 </execution>  
 </executions>  
 <dependencies>  
 <dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>1.2.17</version>  
 </dependency>  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-log4j12</artifactId>  
 <version>1.7.5</version>  
 </dependency>  
 </dependencies>  
 </plugin>  
 </plugins>  
 </build>  
</project>