

# Aotomation design pattern

## : Automation Best Design Pattern

### مقدمة المشروع

مشروع الأتمتة هذا صُمم باستخدام أفضل الممارسات في عالم اختبار واجهات المستخدم مع تكامل Selenium WebDriver وTestNG مع Java 17 اعتمدنا على لغة (UI Testing). Log4j2, Apache POI, Java Faker, WebDriverManager, Allure Report, وExtentReports. قابل الهدف من المشروع هو بناء إطار عمل احترافي، لإعادة الاستخدام، ومن لإجراء اختبارات على واجهات ويب ديناميكية.

### : (Project Structure) شرح هيكل المشروع

```
src
├── main
│   ├── java
│   │   └── org.example.datatest
│   │       ├── pages      ← كائنات الصفحات (Page Object Model)
│   │       ├── utils      ← أدوات مساعدة مشتركة مثل Config و Excel و Faker
│   │       ├── screenShots ← تصوير الفشل TestNG Listener يحتوي على
│   │       └── utils      ← مجلد لحفظ لقطات الشاشة عند الفشل
│   └── resources
│       ├── data          ← file.properties و TestData.xlsx ملفات البيانات مثل
│       └── log4j2.xml     ← للتسجيل إعدادات Log4j2
├── test
│   ├── java
│   │   └── org.example.datatest.tests ← كلاس اختبار يحتوي على سيناريوهات الـ
└── تسجيل
    └── resources
        └── testng.xml     ← TestNG خطة تشغيل
```

### شرح مفصل لكل كلاس + كود كامل وتحليل في Pages

 **BasePage.java**

```

package org.example.datatest.pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.PageFactory;

public class BasePage {
    protected WebDriver driver;

    public BasePage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }
}

```

### شرح:

- كلاس أساسي لكل الصفحات.
- @FindBy لتهيئة كل العناصر باستخدام PageFactory يتم فيه استخدام
- جاهز driver يتم وراثته في كل صفحات الموقع لتوفير



### registePage.java

```

package org.example.datatest.pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class registePage extends BasePage {
    @FindBy(xpath = "(//button[@data-action='click→slideover-account#toggle'])[1]")
    private WebElement registerButton;

    public registePage(WebDriver driver) {
        super(driver);
    }
}

```

```
public void registerButtonClick() {
    registerButton.click();
}
}
```

شرح:

- يمثل صفحة تسجيل.
- XPath العنصر الأساسي هو زر التسجيل يتم تحديده به.
- دون الحاجة لتكرار الكود driver لتوفير BasePage يورث من.

## in utils

## Setup.java شرح كلاس

 الكود:

```
package org.example.datatest.utils;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.concurrent.TimeUnit;

public class Setup {
    private static final ThreadLocal<WebDriver> driver = new ThreadLocal<
    >();

    public static void setDriver(String browser) {
        if (browser.equalsIgnoreCase("chrome")) {
            WebDriverManager.chromedriver().setup();
            driver.set(new ChromeDriver());
        }
        driver.get().manage().window().maximize();
    }

    public static WebDriver getDriver() {
```

```

        return driver.get();
    }

    public static void quitDriver() {
        if (driver.get() != null) {
            driver.get().quit();
            driver.remove();
        }
    }
}

```

## 🧠 الشرح التفصيلي:

### ✅ ThreadLocal:

```
private static final ThreadLocal<WebDriver> driver = new ThreadLocal<>();
```

- Thread لكل `WebDriver` يسمح بتخزين نسخة مستقلة من
- TestNG (parallel execution) مهم جدًا لدعم التشغيل المتوازي

### 🔧 المشكلة التي يحلها:

وكنتم في TestNG، `parallel="tests"` (مثلاً) بتشغل في نفس الوقت Test لو عندك أكثر من static عادي كـ WebDriver بتستخدم

```

java
نسخة
public static WebDriver driver;


```

وده هيعمل → `driver` هيشاركوا نفس (الاختبارات) Threads كل الـ

- ❌ تداخل في الجلسات.
- ❌ تحكم في نفس المتصفح من أكثر من اختبار.
- ❌ اختبارات بتفشل بشكل عشوائي.

### 🔑 1. **private** :

من أي تعديل أو وصول مباشر من كلاس ثانية `driver` علشان نحمي متغير

يكون من خلال ميثودز ( `WebDriver` ) نخلي التحكم في السواق **Encapsulation مبدأ ال**    
بس `getDriver()` و `setDriver()` مثل

## 2. **static** :

Object مرتبط بالكلاس نفسها مش بال `driver` علشان نخلي متغير

يعني:

- في `new Setup()` محتاج تعمل `static` مش `driver` ومتغير `Setup` , لو عندك كلاس كل مرة.
- تقدر توصله من أي مكان كده `static` → لكن لما تخليه:

```
java
نسخته
Setup.getDriver();
```

أو Page Object وده مهم جدًا في الفريمورك عشان تقدر توصل للسواق بسهولة من كل Test Class.

## 3. **final** :

نفسه `ThreadLocal` معناها إنك مش هتغير "المرجع" الخاص بـ

يعني ما ينفعش تقول

```
java
نسخته
driver = new ThreadLocal<>(); // ❌ ده مش مسموح لو هو
```

: `.set()` و `.get()` لكن مسموح تعدل المحتوى جواه باستخدام

```
java
نسخته
driver.set(new ChromeDriver()); // ✅ تمام
```

## **setDriver:**

```
public static void setDriver(String browser) {
    if (browser.equalsIgnoreCase("chrome")) {
        WebDriverManager.chromedriver().setup();
        driver.set(new ChromeDriver());
    }
    driver.get().manage().window().maximize();
}
```

- `ChromeDriver` . يتم إعداد وتشغيل Chrome، يتحقق من اسم المتصفح، وإذا كان
- لتثبيت السائق المناسب تلقائيًا `WebDriverManager` يستخدم
- `ThreadLocal` . يتم تعيينه باستخدام
- ثم يقوم بتكبير نافذة المتصفح

### ✓ getDriver:

```
public static WebDriver getDriver() {
    return driver.get();
}
```

- الحالي Thread المرتبطة بـ WebDriver لإعادة النسخة الحالية من

### ✓ quitDriver:

```
public static void quitDriver() {
    if (driver.get() != null) {
        driver.get().quit();
        driver.remove();
    }
}
```

- يستخدم لإغلاق المتصفح بعد انتهاء الاختبار
- يتأكد من عدم وجود تسرب في الذاكرة بإزالة السائق من `ThreadLocal` .

### 📎 نصائح عملية:

- استخدم هذا الكلاس في جميع الاختبارات لتهيئة المتصفح
- بعد انتهاء الاختبار `quitDriver()` لا تنس دائمًا إغلاق السائق باستخدام

- `setDriver()` بداخل `else if` ببساطة بإضافة `Edge` أو `Firefox` يمكنك تطوير الكود لدعم .

## ConfigReader.java شرح كلاس

### الكود:

```
package org.example.datatest.utils;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class ConfigReader {
    private static volatile ConfigReader instance;
    private static Properties properties ;

    private ConfigReader() {
        try {
            InputStream file = ConfigReader.class.getClassLoader().getResourceAsStream("data/file.properties");
            properties = new Properties();
            properties.load(file);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load configuration file!");
        }
    }

    public static ConfigReader getInstance(){
        if (instance==null){
            synchronized (ConfigReader.class){
                if (instance==null){
                    instance=new ConfigReader();
                }
            }
        }
        return instance;
    }
}
```

```
public String getProperty(String key){
    return properties.getProperty(key);
}
}
```

## 🧠 الشرح التفصيلي الكامل للكلاس + المفاهيم المعقدة:

### ✅ الهدف:

هو كلاس وظيفته الأساسية هي قراءة القيم الموجودة داخل ملف `ConfigReader` `file.properties` ، وتوفيرها لكافة أجزاء المشروع التي تحتاجها.

## 🧱 Singleton Design Pattern + Thread Safety

### 💠 ما هو Singleton؟

طوال وقت تشغيل (Instance) هو نمط تصميم ييمنع إنشاء أكثر من نسخة من الكلاس التطبيق. نحتاجه في الكلاسات التي مش محتاجين نكرر تحميلها أو إنشاؤها.

### 💠 لماذا استخدمناه هنا؟

لأننا لا نريد أن نحمل ملف الإعدادات كل مرة نحتاج فيها إلى قيمة، بل مرة واحدة فقط ويتم إعادة استخدام الكائن المحمل.

## 🔍 تفصيل كل جزء مهم في الكود:

### 💠 `private static volatile ConfigReader instance;`

- Singleton هذا هو كائن الـ.
- `static` ( `getInstance` ) لأننا نحتاج استخدامه من دالة `static`.
- `volatile` Threads، سيتم الوصول إليه من عدة `instance` أن المتغير JVM نوضح بها للـ `volatile` ، ويُجبر على القراءة من Thread لكل CPU الـ cache وبالتالي لا يتم حفظه مؤقتاً في الذاكرة الرئيسية دائماً.

### 💠 `private ConfigReader()` (الكونستركتور):



- حتى لا يُمكن لأي كود خارجي أن ينشئ نسخة من هذا الكلاس، ويتم `private` جعلناه
- إنشأؤه فقط من الداخل
- في هذا الكونستركتور نستخدم كود تحميل ملف الإعدادات:

```
InputStream file = ConfigReader.class.getClassLoader().getResourceAsStream("data/file.properties");
properties = new Properties();
properties.load(file);
```

- معناها استخدم محمل الكلاس للبحث عن الملف داخل `ConfigReader.class.getClassLoader()` مسار المشروع
- `properties` . ثم نقوم بتحميل البيانات إلى كائن

## 🔒 Double-Checked Locking باستخدام Singleton والتأكد من أنه Checked Locking:

```
if (instance == null) {
    synchronized (ConfigReader.class) {
        if (instance == null) {
            instance = new ConfigReader();
        }
    }
}
```

### 👉 شرح هذا الجزء سطر بسطر:

#### ◆ الأولي `if (instance == null)`:

- إلا إذا كان لازم `synchronized` الهدف منها تحسين الأداء: لا ندخل على

#### ◆ `synchronized (ConfigReader.class)` :

- نفسه `Class` هذا يعني أننا نغلق الوصول إلى هذا الب্লوك على كائن الـ
- لذلك نستخدم اسم الكلاس نفسه ، `this` فلا يمكن استخدام `static` لأننا داخل دالة `ConfigReader.class` كمعرّف للقفل
- آخر ينشئ نسخة أثناء الإنشاء `Thread` هذا يضمن أن لا

## ◆ الثانية (instance == null) if:

- Thread آخر ممكن يكون دخل وأنشأ الكائن قبل دخول هذا Thread نعيد التحقق لأن

## 🔄 مقارنة مهمة:

التقنية	الوظيفة
<code>volatile</code>	يشوفوا Threads من تخزين نسخة قديمة وتضمن أن جميع الـ JVM تمنع الـ تحديث قيمة للكائن
<code>synchronized</code>	من إنشاء نفس الكائن في نفس الوقت Thread تمنع أكثر من
<code>ConfigReader.class</code>	static في دالة <code>this</code> نستخدمه كقفل لأنه لا يمكننا استخدام

## 🔧 مثال استخدام الكلاس:

```
String browser = ConfigReader.getInstance().getProperty("browser");
```

## كيف تعمل:

- `getInstance()` تنشئ نسخة واحدة فقط إن لم تكن موجودة
- من الملف `browser` تقرأ قيمة المفتاح `getProperty("browser")` ثم

## 📁 (file.properties) شكل ملف الإعدادات:

```
baseUrl=https://demo.spreecommerce.org/
username=testuser
password=test123
browser=chrome
```

يجب أن يكون داخل المسار التالي:

```
src/main/resources/data/file.properties
```

## ✅ لماذا قمنا بكل هذا؟

- حتى نستطيع في المستقبل تغيير البيانات من ملف خارجي بدون الحاجة إلى الدخول وتعديل الكود.

- ونجعل المشروع أكثر مرونة وقابلية للإدارة والتوسعة.
- أيضًا الطريقة دي بتخلي الكود منظم ومحمي من المشاكل في البيئات متعددة الـ Threads.

## ملخص سريع:

العنصر	التفسير
Singleton	كائن واحد فقط لكل المشروع
volatile	Threads لتأكيد الرؤية الموحدة بين كل
synchronized	حماية من تزامن الدخول للإنشاء
ConfigReader.class	static في الدوال class-level قفل
Properties	لحفظ وقراءة البيانات من ملف خارجي
InputStream	classpath لتحميل ملف الخصائص من

إذا فهمت هذا الكلاس كويس، فأنت أصبحت مستعدًا لبناء أي نظام إعدادات مرن وآمن لأي مشروع أوتوميشن كبير!

## ExcelUtils.java شرح كلاس

## TestNG + Excel باستخدام Excel شرح التعامل مع DataProvider

### الهدف:

وتشغيلها تلقائيًا على اختبارات باستخدام Excel ربط بيانات اختبار مخزنة في ملف

@DataProvider من TestNG.

### المتطلبات:

1. Excel للتعامل مع ملفات Apache POI مكتبة ( .xlsx )
2. Excel لقراءة Util فئة مثل ExcelUtils
3. TestNG فئة مزود بيانات مثل ExcelDataProvider
4. في الاختبارات @DataProvider استخدام

## 1 كود ExcelUtils (Excel لقراءة بيانات):

```

public class ExcelUtils {
    public static Object[][] getExcelData(String sheetName) {
        try {
            FileInputStream file = new FileInputStream("src/main/resources/data/TestData.xlsx");
            Workbook workbook = new XSSFWorkbook(file);
            Sheet sheet = workbook.getSheet(sheetName);
            int totalRows = sheet.getPhysicalNumberOfRows();
            int totalCols = sheet.getRow(0).getPhysicalNumberOfCells();

            Object[][] data = new Object[totalRows - 1][totalCols];
            for (int i = 1; i < totalRows; i++) {
                Row row = sheet.getRow(i);
                for (int j = 0; j < totalCols; j++) {
                    DataFormatter formatter = new DataFormatter();
                    data[i - 1][j] = formatter.formatCellValue(row.getCell(j));
                }
            }
            workbook.close();
            file.close();
            return data;
        } catch (IOException e) {
            e.printStackTrace();
            return new Object[0][0];
        }
    }
}

```

## 🔍 شرح أهم الأجزاء:

🟡 **getPhysicalNumberOfRows()** :

تحسب عدد الصفوف المستخدمة فعليًا في الشيت (تشمل رأس العمود + البيانات).

🟡 **sheet.getRow(0).getPhysicalNumberOfCells()** :

تحسب عدد الأعمدة (الخلايا) في أول صف، لأننا نعتبره نموذجًا لباقي الصفوف.

🟡 **المصفوفة:**

```
Object[][] data = new Object[totalRows - 1][totalCols];
```

- نحذف الصف الأول لأنه رأس الجدول (عنوان الأعمدة).
- لتخزين القيم فقط (index 1) نبدأ من الصف 1.

## DataFormatter:

```
DataFormatter formatter = new DataFormatter();  
data[i - 1][j] = formatter.formatCellValue(row.getCell(j));
```

- مهما كان نوعها (رقم، نص، تاريخ، إلخ) String يحوّل كل قيمة في الخلية إلى.

## كود مزود البيانات **ExcelDataProvider.java** :

```
public class ExcelDataProvider {  
    @DataProvider(name = "LoginData")  
    public static Object[][] loginData() {  
        return ExcelUtils.getExcelData("Sheet0");  
    }  
}
```

### شرح:

- `@DataProvider(name = "LoginData")` : اسم Provider للاختبار مع الذي هتربطه.
- `Object[][]` : لاستخدامه كقائمة بيانات لكل اختبار TestNG النوع المتوقع من :

## كود الاختبار:

```
@Test(dataProvider = "LoginData", dataProviderClass = ExcelDataProvider.  
class)  
@Description("Register with different credentials from Excel")  
public void registerWithExcelData(String username, String password) {  
    System.out.println("Username: " + username + " - Password: " + password);  
}
```

## 📌 كيف بيشتغل:

- (بدون الصف الأول) Excel يشغل الاختبار بعدد الصفوف الموجودة في ملف TestNG.
- (username , password). إلى test method arguments كل صف يتم تعريبه ك

## ❓ أسئلة مهمة تم توضيحها:

### ◆ "هو يفهم مين؟"

- @DataProvider(name = "LoginData") اللي مكتوب في LoginData من الاسم
- @Test(dataProvider = "LoginData") ومرتبطة بنفس الاسم في

### ◆ "هو يربط تلقائي؟"

- لأ، لازم تحدد:
  - اسم ال Provider.
  - dataProviderClass كلاس البيانات باستخدام

### ◆ "؟ pom.xml محتاج أضيف حاجة في"

- Apache POI لازم تضيف مكتبة

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>5.2.3</version>
</dependency>
```

## ✅ الخلاصة:

العنصر	الشرح
ExcelUtils	كمصفوفة Excel كلاس يقرأ بيانات من ملف
DataProvider	يربط بين البيانات والاختبار باستخدام اسم
@Test	Excel ينفذ الاختبار مرة لكل صف من
DataFormatter	String يحول القيمة في الخلية إلى

💡 لو حابب أعملك تمثيل بصري (رسم بياني لتدفق البيانات)، بلغني

## استخدام @DataProvider شرح خاصية dataProviderClass

### ما هي @DataProvider ؟

نُستخدم لتوفير بيانات متعددة لحالات الاختبار. هذا يجعل اختبارك TestNG هي طريقة من يعتمد على البيانات ويسهل اختبار أكثر من سيناريو ببيانات مختلفة.

### الشكل الأساسي:

```
@DataProvider(name = "LoginData")
public Object[][] loginData() {
    return ExcelUtils.getExcelData("Sheet1");
}
```

### داخل كلاس الاختبار:

```
@Test(dataProvider = "LoginData", dataProviderClass = ExcelDataProvider.
class)
public void registerWithExcelData(String username, String password) {
    // Test logic
}
```

### الفرق بين:

#### بدون dataProviderClass :

```
@Test(dataProvider = "LoginData")
```

- تكون موجودة في نفس الكلاس @DataProvider لازم

#### مع dataProviderClass :

```
@Test(dataProvider = "LoginData", dataProviderClass = ExcelDataProvider.
class)
```

- تقدر تستخدم كلاس خارجي لتنظيم البيانات زي `ExcelDataProvider.java`

## **dataProviderClass** فائدة استخدام

- فصل البيانات عن كود الاختبار.
- إعادة استخدام البيانات.
- تنظيم المشروع بشكل نظيف وقابل للتوسع.

### ملاحظات:

- `@Test` و `@DataProvider` يجب أن يتطابق في `DataProvider` اسم.
  - تمثل البيانات: كل صف = اختبار واحد، كل عمود = باراميتر `Object[][]`.
- مثلاً لو عندك 3 صفوف في الإكسل → الاختبار يتنفذ 3 مرات

## **FakeDataUtils.java** شرح كلاس

### الكود:

```
package org.example.datatest.utils;

import com.github.javafaker.Faker;

public class FakeDataUtils {
    private static final Faker faker = new Faker();

    public static String getFullName() {
        return faker.name().fullName();
    }

    public static String getEmail() {
        return faker.internet().emailAddress();
    }

    public static String getPassword() {
        return faker.internet().password();
    }
}
```



## الشرح التفصيلي:

### ✓ الهدف من الكلاس:

لإستخدامها (Fake Test Data) هو كلاس مسؤول عن توليد بيانات وهمية `FakeDataUtils` في اختبارات التسجيل أو الدخول أو أي سيناريو يتطلب إدخال بيانات ديناميكية.

### ✓ Faker لماذا نستخدم؟

- لتجنب تكرار البيانات.
- لإنتاج بيانات ديناميكية واقعية مثل أسماء، إيميلات، كلمات مرور.
- لتجربة اختبارات بأكثر قدر من السيناريوهات الممكنة.

### ✓ محتوى الكلاس:

- `Faker faker = new Faker();` → مكتبة تقوم بإنشاء بيانات عشوائية.
- `getFullName()` → يُرجع اسم كامل مثل "Ahmed Mostafa".
- `getEmail()` → يُرجع إيميل عشوائي مثل "[ahmed.mostafa@gmail.com](mailto:ahmed.mostafa@gmail.com)".
- `getPassword()` → يُرجع كلمة مرور مثل "p@ssw0rd123".

### أمثلة عملية على الاستخدام:

```
String name = FakeDataUtils.getFullName();
String email = FakeDataUtils.getEmail();
String password = FakeDataUtils.getPassword();
```

### داخل اختبار:

```
@Test
public void registerWithFakeData() {
    String name = FakeDataUtils.getFullName();
    String email = FakeDataUtils.getEmail();
    String password = FakeDataUtils.getPassword();
    System.out.println("Registering with: " + name + " | " + email + " | " + password);
}
```

### ملاحظة:

- تأكد من وجود مكتبة `javafaker` في `pom.xml`:

```
<dependency>
  <groupId>com.github.javafaker</groupId>
  <artifactId>javafaker</artifactId>
  <version>1.0.2</version>
</dependency>
```

- لو عاوز تولد بيانات بلغة أو منطقة محددة، ممكن تعمل كده:

```
java
نسخة
Faker faker = new Faker(new Locale("ar-EG")); // اللغة العربية مصر
```

- لأن بعض القيم بتبقى Regex في بعض البيانات (زي التليفون)، ممكن تحتاج تعديل أجنبية.

## screenshotUtil.java شرح كلاس

### الكود:

```
package org.example.datatest.utils;

import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.commons.io.FileUtils;

public class screenshotUtil {
    public static void TakescreenShot(String testName) {
        WebDriver driver = Setup.getDriver();
        String Time = new SimpleDateFormat("yyyyMMdd_HH:mm:ss").format
```

```
(new Date());
    String filePath = "src/main/java/org/example/datatest/screenshots/" +
    testName + "_" + Time + ".png";

    File SrcFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType
    e.FILE);
    File destFile = new File(filePath);
    try {
        FileUtils.copyFile(SrcFile, destFile);
    } catch (IOException e) {
        System.out.println("Failed to save screenshot: " + e.getMessage());
    }
}
```

## 🧠 الشرح التفصيلي:

### ✅ الهدف من الكلاس:

هو كلاس يُستخدم لالتقاط لقطة شاشة تلقائيًا عند فشل اختبار أو عند الحاجة، وحفظها في مجلد مخصص داخل المشروع مع اسم فريد لكل صورة.

### ✅ لماذا نستخدمه؟

- لتوثيق الأخطاء أثناء التشغيل التلقائي.
- Allure أو Extent يُسجّل عملية تتبع الخطأ لاحقًا في التقارير مثل

### ✅ شرح أهم النقاط:

- `Setup.getDriver()` → الحالي WebDriver للحصول على
- `SimpleDateFormat` + `new Date()` → إنشاء اسم فريد للصورة باستخدام الوقت
- `((TakesScreenshot) driver).getScreenshotAs(...)` → تحويل المتصفح إلى كائن يمكنه التقاط الشاشة
- `FileUtils.copyFile(...)` → حفظ الصورة في المسار المحدد

## 📌 مثال عملي:

مثل `TestListener` عادةً ما يتم استدعاء هذا الكلاس من

```
screenshotUtil.TakescreenShot("testLogin");
```

## TestListener.java شرح كلاس

### الكود:

```
package org.example.datatest.utils;

import io.qameta.allure.Allure;
import org.testng.ITestListener;
import org.testng.ITestResult;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class TestListener implements ITestListener {

    @Override
    public void onTestFailure(ITestResult result) {
        String testName = result.getMethod().getMethodName();
        screenshotUtil.TakescreenShot(testName); // تلتقط السكرين شوت وتحفظها

        // ابعت السكرين شوت دي للتقرير
        String path = "src/main/java/org/example/datatest/screenshots/" + testName + ".png";
        File file = new File(path);

        try {
            Allure.addAttachment("Screenshot on Failure", new FileInputStream(file));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

## 1. س `TestListener.java`

- يرث من `ITestListener`.
- بيناديهـا تلقائيـًا TestNG وهي واحدة من الدوال اللي `onTestFailure()` بنستخدم دالة `Test` عند فشل أي.

### معلومة مهمة:

يحتوي على مجموعة دوال TestNG في Interface هو `ITestListener` مثل `onTestStart`, `onTestSuccess`, `onTestFailure`, `onTestSkipped`, ...  
عشان نلتقط لقطة عند `onTestFailure` إحنا هنا استخدمنا بس `الفشل`.

## 1. `testng.xml` باستخدام TestNG بـ Listener ربط الـ

الخاص بينا Listener إنه يشغل TestNG بنضيف الكود ده عشان نبلاغ ، `testng.xml` في ملف

```
<listeners>
  <listener class-name="org.example.datatest.utils.TestListener"/>
</listeners>
```

هيبقى فاهم إنه كل ما يحصل فشل في اختبار، ينفذ الدالة TestNG كده `onTestFailure()` اللي كتبناها في `TestListener.java`.

## دورة العمل بالتفصيل:

1. من ملف Test المستخدم يشغل `testng.xml`.
2. فشل لأي سبب Test لو:
  - `TestListener` من كلاس `onTestFailure()` ينادي أوتوماتيكيًا TestNG.
  - `screenshotUtil.TakescreenShot()` يستدعي `onTestFailure()` الكود داخل.
  - بالاسم المناسب `screenshots` ويحفظ في مجلد Screenshot يتلقط.

## مكان الصور:

- الصور بتتحفظ في: `src/main/java/org/example/datatest/screenshots/`
- اسم الصورة يكون: `اسم_الاختبار_والتاريخ.png`

مثال:

registerTest\_20250728\_164015.png

## ✓ فوائد الطريقة دي:

الشرح	الفائدة
بتقدر تشوف سكرين شوت بالضبط وقت الفشل	التتبع البصري
بدل ما تحاول تفهم من اللوج بس، تشوف الصورة مباشرة	Debug تسريع الـ
الصور بتكون دليلك، أو Jenkins أو GitHub Actions لما الاختبارات تتشغل في	CI/CD مفيدة في

## 📌 نقاط لازم تنتبه لها:

- موجود فعلاً في المشروع `screenshots` تأكد إن مجلد
- بشكل صحيح ThreadLocal driver بيرجع الـ `getDriver()` فيه `Setup.java` تأكد إن كلاس
- مش في الكود فقط ، `testng.xml` في Listener لازم تضيف الـ
- فاحرص على التمييز، Test لو بتستخدم عدة متصفحات، الصور هتتسمى بنفس اسم الـ بينهم لو لازم

## ✏️ مثال عملي:

لو عندك اختبار زي ده

```
@Test
public void testLoginFailure() {
    Assert.assertEquals("actual", "expected"); // هتفشل
}
```

هتلاقى صورة اتسجلت تلقائياً بعد الفشل.

## 🧠 خلاصة:

- `screenshotUtil` = ينفذ عملية الالتقاط
- `TestListener` = بيراقب كل الاختبارات
- `onTestFailure` = بيشتغل أوتوماتيكي عند الفشل
- `testng.xml` = Listener بكلاس الـ TestNG بيربط

وبكده تكون أتممت نظام التقاط الصور التلقائي وقت فشل الاختبار بشكل احترافي 🔥



### WaitUtils.java

```
package org.example.datatest.utils;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.Select;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

import static org.example.datatest.utils.Setup.getDriver;

public class WaitUtils {
    private static final int TimeOut =10;

    public static void waitForVisibility(WebElement element){
        WebDriverWait wait = new WebDriverWait(getDriver(), Duration.ofSeconds(TimeOut));
        wait.until(ExpectedConditions.visibilityOf(element));
    }
    public static void waitForClickability(WebElement element){
        WebDriverWait wait = new WebDriverWait(getDriver(), Duration.ofSeconds(TimeOut));
        wait.until(ExpectedConditions.elementToBeClickable(element));
    }
    public static void waitForInvisibility(WebElement element){
        WebDriverWait wait = new WebDriverWait(getDriver(), Duration.ofSeconds(TimeOut));
        wait.until(ExpectedConditions.invisibilityOf(element));
    }
    public static boolean waitForOptionsToBePresent(WebDriver driver, WebElement selectElement) {
        try {
            WebDriverWait wait = new WebDriverWait(driver, Duration.ofSecond
```

```
s(Timeout));
    wait.until(driver1 → {
        Select select = new Select(selectElement);
        return select.getOptions().size() > 0;
    });
    return true;
} catch (Exception e) {
    return false;
}
}
}
```

شرح:

- لتفادي الأخطاء الناتجة عن محاولة `waitForClickability` و `waitForVisibility` يحتوي على دوال التفاعل مع عناصر غير جاهزة.
- Selenium يعزز استقرار اختبارات.

## ✓ شرح ميثود `waitForOptionsToBePresent`

 التعريف:

```
java
نسخة
public static boolean waitForOptionsToBePresent(WebDriver driver, WebElement selectElement) {
    try {
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(Timeout));
        wait.until(driver1 → {
            Select select = new Select(selectElement);
            return select.getOptions().size() > 0;
        });
        return true;
    } catch (Exception e) {
        return false;
    }
}
```



```
}  
}
```

## 🧠 الفكرة العامة:

على صفحة الويب. (قائمة منسدلة) `<select>` الميثود دي بتستخدم في حالة وجود عنصر بتظهر بعد وقت معين، فبنستخدم الانتظار `<select>` أحياناً العناصر (الخيارات) داخل الـ `<select>` علشان (Explicit Wait):

- (option) **نستنى** لحد ما القائمة المنسدلة يكون فيها على الأقل اختيار واحد.
- `true` . لو حصل ده خلال الوقت المسموح → ترجع.
- `false` . لو الوقت خلص ولسه مفيش حاجة → ترجع.

## 🔍 شرح كل سطر:

### ✅ 1. إنشاء WebDriverWait

```
java  
نسخته  
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(Time  
Out));
```

- (عدد ثواني) `TimeOut` بنحدد إننا هنستنى لفترة.
- `driver` هو اللي هيتم الانتظار عليه.
- (`Constants` أو `WaitUtils` مثلاً) هو متغير ثابت غالباً معرف في كلاس خارجي `TimeOut`.

### ✅ 2. استخدام `wait.until(...)` مع Lambda

```
java  
نسخته  
wait.until(driver1 → {  
    Select select = new Select(selectElement);  
    return select.getOptions().size() > 0;  
});
```

```
});
```

دي أهم وأذكى حقة في الميثود، ودي معناها:

- استنى لحد ما الشرط يتحقق:
  - يكون فيه اختيارات موجودة (القائمة المنسدلة) `selectElement` أن العنصر
  - الشرط ده بيتحقق لما: `select.getOptions().size() > 0`.

## ⚡ شرح اللامبدا (Lambda Expression):

```
java
نسخة
driver1 → {
    Select select = new Select(selectElement);
    return select.getOptions().size() > 0;
}
```

### ✓ معناها:

- ده شكل مختصر للدالة اللي بنمررها لـ `wait.until(...)`.
- بدل ما نكتب كود كتير، بنستخدم لامبدا لتعبر عن الشرط بسرعة.

### 👁️ driver1 مين هو ؟

- **WebDriver** ده مجرد اسم متغير محلي بيمثل
- مش فارق، المهم المعنى `driverX` أو `d` ممكن تسميه.

### 🎯 الشرط:

- معناها إن في اختيارات في القائمة → `select.getOptions().size() > 0`.

### 🔄 باقي الكود:

### ✓ لو الشرط اتحقق بنجاح:

```
java
نسخة
```

```
return true;
```

مثلاً العنصر مش لاقية أو مفيش اختيارات بعد) Exception لو حصل أي **✗** (الانتظار):

```
java  
نسخة تحرير  
return false;
```

### ✓ ملخص مبسط:

العنصر	الشرح
<code>WebDriverWait</code>	لعمل انتظار صريح حتى يتحقق شرط معين
<code>wait.until(...)</code>	الشرط اللي هنتأكد منه
<code>Select select = new Select(...)</code>	لإنشاء عنصر قائمة منسدلة
<code>getOptions().size() &gt; 0</code>	شرط إن يكون فيه اختيارات
Lambda	طريقة مختصرة لكتابة الشرط داخل <code>wait.until()</code>
<code>return true/false</code>	نرجع الحالة النهائية حسب نجاح أو فشل الشرط

### 📌 ملاحظات مهمة:

- Exception وإلا هيحصل، **DOM** يكون موجود بالفعل في الـ `selectElement` لازم
- مباشرة `driver` لو مش بتمرير `Setup.getDriver()` مع `ThreadLocal<WebDriver>` بنستخدم
- مخصص لو حبيت **Exception** أو رمي **Logs** ممكن تحسن الكود بإضافة

### ✓ `WebActions.java`

```
package org.example.datatest.utils;  
  
import org.openqa.selenium.JavascriptExecutor;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.interactions.Actions;
```

```

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

import static org.example.datatest.utils.Setup.getDriver;

public class WebActions {

    public static void click(WebElement element) {
        WaitUtils.waitForClickability(element);
        element.click();
    }

    public static void type(WebElement element, String text) {
        WaitUtils.waitForVisibility(element);
        element.clear();
        element.sendKeys(text);
    }

    public static void hover(WebElement element) {
        WaitUtils.waitForVisibility(element);
        Actions action = new Actions(getDriver());
        action.moveToElement(element).perform();
    }

    public static void selectByVisibleText(WebElement element, String text) {
        WaitUtils.waitForInvisibility(element);
        Select select = new Select(element);
        select.selectByVisibleText(text);
    }

    public static void scrollToElement(WebElement element) {
        ((JavascriptExecutor) getDriver()).executeScript("arguments[0].scrollI
ntoView({behavior: 'smooth', block: 'center'});", element);
    }

    public static void scrollByPixels(int x, int y) {
        ((JavascriptExecutor) getDriver()).executeScript(

```

```

        "window.scrollTo(arguments[0], arguments[1]);", x, y);
    }
}

```

شرح:

- دوال جاهزة للنقر والكتابة.
- يضمن أن العنصر مرئي أو قابل للنقر قبل أي تفاعل.
- داخليًا WaitUtils يستخدم.

 in resources main

 شرح تفصيلي لإعدادات تسجيل الأحداث – log4j2.xml

 Log4j2 الغرض من:

قوي وحديث يستخدم لتتبع أحداث التنفيذ مثل Logging هو إطار عمل

- (Errors) تتبع أخطاء النظام
- (Info) كتابة سجلات التشغيل
- (Warnings) مراقبة العمليات المهمة

 الشكل العام للملف:

```

<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>

    <File name="FileLogger" fileName="logs/test-log.log">
      <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n"/>
    </File>
  </Appenders>

  <Loggers>

```

```

<Root level="info">
  <AppenderRef ref="Console"/>
  <AppenderRef ref="FileLogger"/>
</Root>
</Loggers>
</Configuration>

```

## 🧠 شرح المكونات الأساسية:



- `status="WARN"` يتم طباعته (XML خطأ في) معناها إذا حدث خطأ في التهيئة نفسها `status="WARN"` أو أعلى `WARNING` إذا كان مستوى الخطأ



- الجزء المسؤول عن تحديد أين تُطبع الرسائل:
  - Console: يطبع في التيرمينال
  - File: يكتب إلى ملف

## ✅ Console Appender:

```

<Console name="Console" target="SYSTEM_OUT">
  <PatternLayout pattern="%d{HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"/>
</Console>

```

- `%d{HH:mm:ss}` → وقت تنفيذ الرسالة
- `[%t]` → اسم الـ thread
- `%-5level` → مستوى الحدث (INFO, ERROR...)
- `%logger{36}` → اسم الكلاس
- `%msg` → الرسالة التي تسجل

## ✅ File Appender:

```

<File name="FileLogger" fileName="logs/test-log.log">
  <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -

```

```
%m%n"/>
</File>
```

- `fileName="logs/test-log.log"` → مكان حفظ السجلات
- `%c{1}` → اسم الكلاس فقط بدون الباكج
- `%L` → رقم السطر



```
<Loggers>
  <Root level="info">
    <AppenderRef ref="Console"/>
    <AppenderRef ref="FileLogger"/>
  </Root>
</Loggers>
```


- `level="info"` → أو أعلى info فقط, debug logs لا يتم طباعة
- `AppenderRef` → يشير إلى الكونسول والملف

### مثال عملي على استخدامه:

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Example {
    private static final Logger logger = LogManager.getLogger(Example.class);

    public void runTest() {
        logger.info("Starting test");
        logger.warn("This might be an issue");
        logger.error("Test failed!");
    }
}
```

 سيتم طباعة كل هذه الرسائل في التيرمينال وفي ملف `logs/test-log.log`.

### نصيحة:

- في مشروعك `logs` تأكد من وجود مجلد.
- إذا لم يوجد، لن يتم إنشاء الملف إلا يدوياً أو بالكود.

هل تحب أضيفه للملف كامل كجزء رسمي؟

### in test:

#### `registerTest.java`

```
package org.example.datatest.tests;

import org.testng.annotations.*;
import org.example.datatest.pages.registePage;
import org.example.datatest.utils.*;

public class registerTest {
    private registePage registePage;

    @Parameters("browser")
    @BeforeTest
    public void statrup(String browser) {
        String baseUrl = ConfigReader.getInstance().getProperty("baseUrl");
        Setup.setDriver(browser);
        Setup.getDriver().get(baseUrl);
        registePage = new registePage(Setup.getDriver());
    }

    @Test(description = "Click on Register Button")
    public void register() {
        registePage.registerButtonClick();
    }

    @Test(dataProvider = "LoginData", dataProviderClass = ExcelDataProvider.class)
    public void registerWithExcelData(String username, String password) {
        System.out.println("Username: " + username + " - Password: " + password);
    }
}
```



```

    }

    @Test(description = "Register using fake data")
    public void registerWithFakeData() {
        String name = FakeDataUtils.getFullName();
        String email = FakeDataUtils.getEmail();
        String password = FakeDataUtils.getPassword();
        System.out.println("Registering with: " + name + " | " + email + " | " +
password);
    }

    @AfterTest
    public void teardown() {
        Setup.quitDriver();
    }
}

```

### شرح:

- يحتوي على 3 اختبارات رئيسية.
- Excel وFaker يعتمد على بيانات من
- WebDriver على تهيئة وإغلاق لل `@BeforeTest` و `@AfterTest`.

### POM.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<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://mav
en.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>spreecommerceAtomationTest</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>

```

```

    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding
>
</properties>

<dependencies>
  <!-- Selenium WebDriver →
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.15.0</version>
  </dependency>

  <!-- WebDriver Manager →
  <dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>5.5.3</version>
  </dependency>

  <!-- TestNG →
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.10.2</version>
  </dependency>

  <!-- Apache POI for Excel Handling →
  <dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>5.2.3</version>
  </dependency>

  <!--for fake data→
  <dependency>
    <groupId>com.github.javafaker</groupId>
    <artifactId>javafaker</artifactId>

```

```

        <version>1.0.2</version>
    </dependency>

    <!--allure for Html report→
    <dependency>
        <groupId>io.qameta.allure</groupId>
        <artifactId>allure-testng</artifactId>
        <version>2.24.0</version>
    </dependency>

    <!-- SLF4J for Logging →
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>2.0.7</version>
    </dependency>

    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.20.0</version>
    </dependency>

    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.20.0</version>
    </dependency>

    <!-- Excel reader→
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>5.2.3</version>
    </dependency>

</dependencies>

```

```

<build>
  <plugins>
    <!-- Maven Compiler Plugin →
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>17</source>
        <target>17</target>
      </configuration>
    </plugin>

    <!-- Surefire Plugin for Running TestNG Tests →
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.0.0-M5</version>
      <configuration>
        <suiteXmlFiles>
          <suiteXmlFile>src/test/resources/testng.xml</suiteXmlFile>
        </suiteXmlFiles>
      </configuration>
    </plugin>

    <plugin>
      <groupId>io.qameta.allure</groupId>
      <artifactId>allure-maven</artifactId>
      <version>2.11.2</version>
    </plugin>

  </plugins>
</build>
</project>

```

## Testng.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SpreecommerceAutomationTest" parallel="tests" thread-count="2">

    <listeners>
        <listener class-name="org.example.datatest.utils.TestListener"/>
    </listeners>

    <test name="Chrome Test">
        <parameter name="browser" value="chrome"/>
        <classes>
            <class name="org.example.datatest.tests.registerTest"/>
        </classes>
    </test>

    <!-- <test name="Firefox Test">→
    <!--     <parameter name="browser" value="firefox"/>→
    <!--     <classes>→
    <!--         <class name="org.example.datatest.tests.registerTest"/>→
    <!--     </classes>→
    <!-- </test>→

    <!-- <test name="Edge Test">→
    <!--     <parameter name="browser" value="Edge"/>→
    <!--     <classes>→
    <!--         <class name="org.example.datatest.tests.registerTest"/>→
    <!--     </classes>→
    <!-- </test>→

</suite>
```

 **ماہ Log4j2؟**

يعني بدل Java في تطبيقات (logging) خاص بـ تسجيل الأحداث framework هو Log4j2 ما تستخدم System.out.println() ، التحذيرات بشكل ، ما تستخدم احترافي ومنظم.

## ✓ System.out.println بدل Log4j2 ليه نستخدم

الميزة	Log4j2	println
تسجيل مخصص (INFO, ERROR...)	✓	✗
حفظ في ملف	✓	✗
تحكم بالتنسيق	✓	✗
Threads دعم تعدد الـ	✓	✗
سرعة الأداء	✓ (أفضل بكثير)	✗

## 🔧 في مشروعك Log4j2 خطوات إعداد

### 1. في pom.xml Dependency أضف الـ :

```
xml
نسخة
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.20.0</version>
</dependency>

<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.20.0</version>
</dependency>
```

### 2. src/main/resources داخل log4j2.xml أنشئ ملف

```
xml
نسخة
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <!-- Console Logging -->
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>

    <!-- File Logging -->
    <File name="FileLogger" fileName="logs/test-log.log">
      <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n"/>
    </File>
  </Appenders>

  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
      <AppenderRef ref="FileLogger"/>
    </Root>
  </Loggers>
</Configuration>
```

#### ◆ شرح الإعدادات:

- **Appenders** : (Console أو File) فين تسجل الرسائل؟
- **PatternLayout** : تنسيق الرسالة
- **Loggers** : (INFO , ERROR ...) مين اللي هيسجل وبأي مستوى

### 3. استخدام Java في كود

```
java
نسخة
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

```

public class MyClass {
    private static final Logger logger = LogManager.getLogger(MyClass.class);

    public void testSomething() {
        logger.info("✅ بدأ الاختبار");
        logger.warn("⚠️ ده تحذير");
        logger.error("❌ حصل خطأ");
        logger.debug("📄 debug دي معلومات");
    }
}

```

## مستويات التسجيل (Log Levels)

المستوى	المعنى
FATAL	خطأ قاتل يوقف البرنامج
ERROR	خطأ عادي
WARN	تحذير
INFO	معلومات عامة (بدأ، انتهى، إلخ)
DEBUG	معلومات تفصيلية (للتصحيح)
TRACE	(نادر الاستخدام) debug معلومات أدق من

## مثال عملي:

```

java
نسخة
public class LoginTest {
    private static final Logger logger = LogManager.getLogger(LoginTest.class);

    public void runTest() {
        logger.info("بدء اختبار الدخول");
        try {
            // كود تسجيل الدخول

```



```

    } catch (Exception e) {
        logger.error("فشل الاختبار: " + e.getMessage());
    }
}
}

```

## 📁 أين يتم حفظ الملف؟

في الإعدادات أعلاه:

```

xml
نسخة
<fileName>logs/test-log.log</fileName>

```

بجانب المشروع `logs/` هتلاقى ملفات اللوج داخل مجلد.

## ✅ ملحوظة مهمة:

- حسب `error` أو `debug` إلى `<Root level="info">` تقدر تتحكم في مستوى التفاصيل بتغيير ما تحب.
- تقدر توصله تلقائيًا Java عشان `src/main/resources` يكون في `log4j2.xml` لازم الملف.

## ✅ شرح شامل - Allure Report

### 💡 ما هو Allure Report؟

هو نظام توليد تقارير اختبار احترافية. يستخدمه المحترفون لأنه بيقدّم Allure

- منظمة HTML واجهة.
- Attachments (Screenshots, Logs, Videos) يدعم.
- وغيرها، JUnit، TestNG، يدمج بسهولة مع.
- يعرض التقارير بشكل رسومي: الوقت، الحالة، التغطية، السيناريوهات.

# 🔧 Maven + Allure Report خطوات إعداد TestNG

## 1. ✅ إلى Dependencies أضف الـ `pom.xml`

```
xml
نسخة تحرير
<!-- Allure Core Integration -->
<dependency>
  <groupId>io.qameta.allure</groupId>
  <artifactId>allure-testng</artifactId>
  <version>2.24.0</version>
</dependency>
```

## 2. 🔄 إلى Allure أضف `testng.xml` كـ Listener

```
xml
نسخة تحرير
<listeners>
  <listener class-name="io.qameta.allure.testng.AllureTestNg"/>
</listeners>
```

## 3. 🖋️ في Test Cases استخدام Allure Annotations

```
java
نسخة تحرير
import io.qameta.allure.*;

@Test(description = "Click on Register Button")
@Epic("User Registration")
@Feature("Register Button Functionality")
@Story("Click register button to navigate to sign up form")
@Severity(SeverityLevel.CRITICAL)
@TestDescription("Test to verify if the register button works correctly")
```

```
public void register() {
    registrePage.registerButtonClick();
}
```

## 📌 Allure المستخدمة في Annotations شرح

Annotation	معناها
@Epic("...")	القسم الكبير (مثلاً: User Management)
@Feature("...")	Epic ميزة فرعية داخل الـ
@Story("...")	قصة الاستخدام أو سيناريو مستخدم معين
@Severity(...)	مدى أهمية الاختبار (BLOCKER, CRITICAL, NORMAL, MINOR, TRIVIAL)
@Description("...")	وصف مفصل لما يفعله هذا الاختبار
@Step("...")	تسجيل خطوة في التقرير (لو استخدمتها داخل الميثود)

## 📸 تلقائياً عند الفشل في Screenshot إرفاق Allure

في كلاس `TestListener.java`:

```
java
نسخة
@Override
public void onTestFailure(ITestResult result) {
    String testName = result.getMethod().getMethodName();
    screenshotUtil.TakescreenShot(testName); // يحفظها على الجهاز

    try {
        Path content = Paths.get("src/main/java/org/example/datatest/screenshots/" + testName + ".png");
        Allure.addAttachment("Screenshot", Files.newInputStream(content));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

## ⚙️ أوامر تشغيل التقرير

```
bash
نسخة
# بعد تنفيذ الاختبارات (mvn clean test)
allure serve allure-results
```

🚀 سيفتح المتصفح تلقائيًا على تقرير تفاعلي.

## 📁 أماكن الملفات

نوع الملف	الموقع
allure-results	يتم توليده بعد الاختبار تلقائيًا
screenshots	أنت تنشئ هذا وتخزن فيه الصور
allure-report	يتم إنشاؤه تلقائيًا عند تشغيل allure serve

## 🔧 متطلبات التشغيل

- على جهازك تثبيت Allure:

```
bash
نسخة
scoop install allure
```

أو

```
bash
نسخة
choco install allure
```

## 💬 Allure و ExtentReports و Log4j2 مقارنة سريعة بين

المقارنة	Allure	ExtentReports	Log4j2
نوع الأداة	تقرير اختبار تفاعلي	HTML (تقرير اختبار ثابت فقط)	نظام تسجيل (Logging)
واجهة رسومية	✓ تفاعلية جدًا	✓ لكنها ثابتة	✗ Logs فقط
دعم Screenshots	✓ بسهولة	✓ مع خطوات إضافية	✗
BDD / سيناريوهات يدعم	✓ @Story , @Feature , إلخ	✗	✗
داخلي Logging	✗	✗	✓ (مخصص لذلك)
سهل الدمج مع GitHub	✓	✓	✗
الأفضل للاختبارات	✓	جيد جدًا	✗ Debug فقط لا

## 🎯 الخلاصة:

الحالة	الأداة الأنسب
تريد تقارير تفاعلية غنية	🏆 Allure
فقط HTML تريد تقارير عادية و	ExtentReports
Debug تريد تسجيل الأخطاء والـ	Log4j2

## 📌 هل لازم أستخدام الكل مع بعض؟

لا، الأفضل:

- للتقارير Allure استخدم.
- لتسجيل الأخطاء والتفاصيل التقنية Log4j2 استخدم.
- ممكن تستخدمه لتقرير إضافي داخلي فقط , ExtentReports لو بتحب شكل.

## 📦 ExtentReports أولاً: ما هو؟

مميزة ومفصلة لنتائج HTML تُستخدم لإنشاء تقارير Java هو مكتبة ExtentReports أو أي إطار اختبار آخر JUnit أو TestNG اختباراتك في.

### ✓ المميزات:

- واجهة مستخدم جميلة بتقسيمات واضحة.

- يدعم تسجيل الخطوات (log/pass/fail/skip/warn).
- يدعم スクリーンショット.
- يدعم تصنيفات مثل author, category, device.
- TestNG سهل الدمج مع.

## ◆ (Maven) ثانيًا: طريقة إضافته للمشروع

دي في dependency ضيف الـ `pom.xml`:

```
xml
نسخة
<dependency>
  <groupId>com.aventstack</groupId>
  <artifactId>extentreports</artifactId>
  <version>5.1.0</version>
</dependency>
```

## ◆ ثالثًا: إنشاء ملف مدير التقرير ExtentReportManager.java

```
java
نسخة
public class ExtentReportManager {
    private static ExtentReports extent;

    public static ExtentReports getInstance() {
        if (extent == null) {
            ExtentSparkReporter spark = new ExtentSparkReporter("reports/ext
            ent-report.html");
            spark.config().setReportName("Automation Test Report");
            spark.config().setDocumentTitle("Extent Report");

            extent = new ExtentReports();
            extent.attachReporter(spark);
        }
        return extent;
    }
}
```

```
}  
}
```

## ◆ ربط التقرير بالـ `ExtentTestListener.java` رابعًا: إنشاء TestNG

```
java  
نسخة تحرير  
public class ExtentTestListener implements ITestListener {  
    private static ExtentReports extent = ExtentReportManager.getInstance  
();  
    private static ThreadLocal<ExtentTest> test = new ThreadLocal<>();  
  
    @Override  
    public void onTestStart(ITestResult result) {  
        ExtentTest extentTest = extent.createTest(result.getMethod().getMeth  
odName());  
        test.set(extentTest);  
    }  
  
    @Override  
    public void onTestSuccess(ITestResult result) {  
        test.get().log(Status.PASS, "Test Passed");  
    }  
  
    @Override  
    public void onTestFailure(ITestResult result) {  
        test.get().log(Status.FAIL, "Test Failed");  
        test.get().log(Status.FAIL, result.getThrowable());  
    }  
  
    @Override  
    public void onTestSkipped(ITestResult result) {  
        test.get().log(Status.SKIP, "Test Skipped");  
    }  
}
```

```
@Override
public void onFinish(ITestContext context) {
    extent.flush(); // يولد التقرير النهائي
}
}
```

## ◆ خامسًا: إضافة ExtentTestListener في ملف testng.xml

```
xml
نسخة
<listeners>
  <listener class-name="org.example.datatest.utils.ExtentTestListener"/>
</listeners>
```

## ◆ سادسًا: تشغيل المشروع

هيتولد ملف TestNG Suite أو أي `mvn clean test` بعد ما تشغل

```
bash
نسخة
/reports/extent-report.html
```

افتحه بأي متصفح وشوف تقرير أنيق جدًا ومفصل

## vs مقارنة سريعة: ExtentReports vs Allure

المقارنة	ExtentReports	Allure
الشكل الجمالي	ممتاز	احترافي أكثر
سهولة التثبيت	سهل جدًا	إعداد إضافي + CLI يحتاج
دعم الأنوتيشن	لا	نعم (Epic, Story, etc.)
دعم GitHub Actions	نعم	نعم
الأفضل للمبتدئين	✓	✗ (أصعب شوية)



المقارنة	ExtentReports	Allure
الأفضل للشركات الكبرى	✗	✓

## ✓ خلاصة

- ExtentReports. لو عاوز تقرير بسيط وجميل وسهل: استخدم.
- **Allure** روح لـ Enterprise أو مشاريع CI/CD لو شغال في بيئة احترافية أو.



## pom.xml التبعيات الرئيسية في ملف (شرح فقط)

- `selenium-java`: WebDriver الأساس لتشغيل.
- `testng`: إطار إدارة وتشغيل الاختبارات.
- `log4j-core` + `log4j-api`: تسجيل الأحداث ونتائج التنفيذ.
- `poi-ooxml`: Excel التعامل مع ملفات.
- `javafaker`: بيانات وهمية تلقائية.
- `webdrivermanager`: تحميل السائق المناسب تلقائيًا.
- `extent-reports`: تفاعلية HTML تقارير.
- `allure-testng`: تقارير مفصلة بنمط مهني.

## ✓ دليل عملي لإنشاء مشروع مماثل:

1. IntelliJ جديد في Maven أنشئ مشروع.
2. `pom.xml` أضف التبعيات في.
3. `pages`, `utils`, `tests`, `listeners`, `screenshots`: أنشئ الباكيجات.
4. `log4j2.xml`, `TestData.xlsx`, `file.properties`: أضف ملفات.
5. POM Pages صمّم.
6. `Setup.java` أنشئ WebDriver لإدارة.
7. `ConfigReader`, `ExcelUtils`, `FakeDataUtils`, `TestListener`, `screenshotUtil`: أضف.
8. `registerTest.java`: اكتب حالات اختبار داخل.
9. `testng.xml` أنشئ ملف.

## 10. شغل المشروع.

---