



Test Automatisé

Présentée par :
Emna Ouni





TestNG

TestNG est un framework de test basé sur Java qui est conçu pour faciliter l'écriture et l'exécution de tests unitaires et d'intégration. Il est largement utilisé pour les tests automatisés, en particulier avec des applications développées en Java. TestNG offre des fonctionnalités avancées comme la gestion des dépendances, le regroupement des tests, la parallélisation,

Implémenté TestNG

```
16      <dependencies>
17          <dependency>
18              <groupId>org.seleniumhq.selenium</groupId>
19              <artifactId>selenium-java</artifactId>
20              <version>4.22.0</version>
21          </dependency>
22          <!-- https://mvnrepository.com/artifact/org.testng/testng -->
23          <dependency>
24              <groupId>org.testng</groupId>
25              <artifactId>testng</artifactId>
26              <version>7.10.2</version>
27              <scope>test</scope>
28          </dependency>
29      </dependencies>
30
```

Annotations principales de TestNG :

TestNG offre plusieurs annotations qui vous permettent de structurer et de contrôler vos tests de manière efficace.

@Test

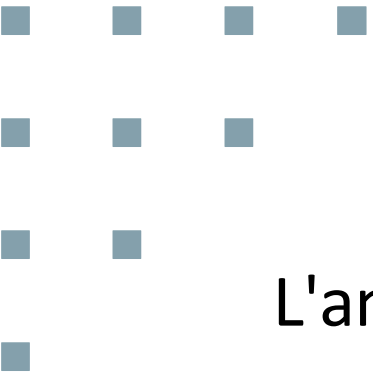
- Utilisée pour marquer une méthode comme un test à exécuter.
- Options : enabled, timeout, priority, dependsOnMethods, etc.

```
@Test
public void Test() {
    driver.get("https://www.saucedemo.com/");
    String title = driver.getTitle();
    String expected = "Swag Labs";
    Assert.assertEquals(expected, title);
}
```



Dans TestNG, pour désactiver un test, vous pouvez utiliser l'attribut **enabled=false** dans l'annotation `@Test`. Cela permet d'ignorer un test sans le supprimer du code.

```
@Test(enabled = false)
public void Test1() {
    driver.get("https://www.saucedemo.com/");
    String title = driver.getTitle();
    String expected = "Swag Labs";
    Assert.assertEquals(expected, title);
}
```



L'annotation **timeout** est utilisée pour définir une limite de temps d'exécution pour un test. Si le test prend plus de temps que le délai spécifié pour s'exécuter, il sera considéré comme échoué.

```
@Test (timeOut = 3000)
public void Test1() {
    driver.get("https://www.saucedemo.com/");
    String title = driver.getTitle();
    String expected = "Swag Labs";
    Assert.assertEquals(expected, title);
}
```


L'attribut **priority** est utilisé pour définir l'ordre d'exécution des tests dans une classe de tests. Les tests avec des valeurs de priorité inférieures seront exécutés avant ceux avec des valeurs plus élevées.

```
@Test(priority = 1)
public void test1() {
    System.out.println("first_test");
}

@Test(priority = 2)
public void test2() {
    System.out.println("second_test");
}

@Test(priority = 0)
public void test3() {
    System.out.println("third_test");
}

@Test
public void test4() {
    System.out.println("last_test");
}
```

✓ Default Suite	11 ms	✓ Tests passed: 4 of 4 tests – 11 ms
✓ Testng1	11 ms	third_test
✓ TESTTNG	11 ms	last_test
✓ test3	8 ms	first_test
✓ test4	1 ms	second_test
✓ test1	1 ms	
✓ test2	1 ms	=====

L'attribut **dependsOnMethods** permet de spécifier que l'exécution d'un test dépend de la réussite d'un ou plusieurs autres tests. Si le test dont dépend un autre échoue, le test dépendant sera ignoré.

```
@Test(priority = 1)
public void test1() {
    System.out.println("first_test");
}

@Test(priority = 2)
public void test2() {
    System.out.println("second_test");
}

@Test(priority = 0)
public void test3() {
    System.out.println("third_test");
}

@Test
public void test4() {
    System.out.println("last_test");
}

@Test (dependsOnMethods = {"test3"})
public void test5(){
    System.out.println("test fivee");
}
```

✓ Testng1	11 ms	third_test
✓ TESTTNG	11 ms	last_test
✓ test3	7 ms	test fivee
✓ test4	1 ms	first_test
✓ test5	1 ms	second_test
✓ test1	1 ms	
✓ test2	1 ms	=====
		Default Suite

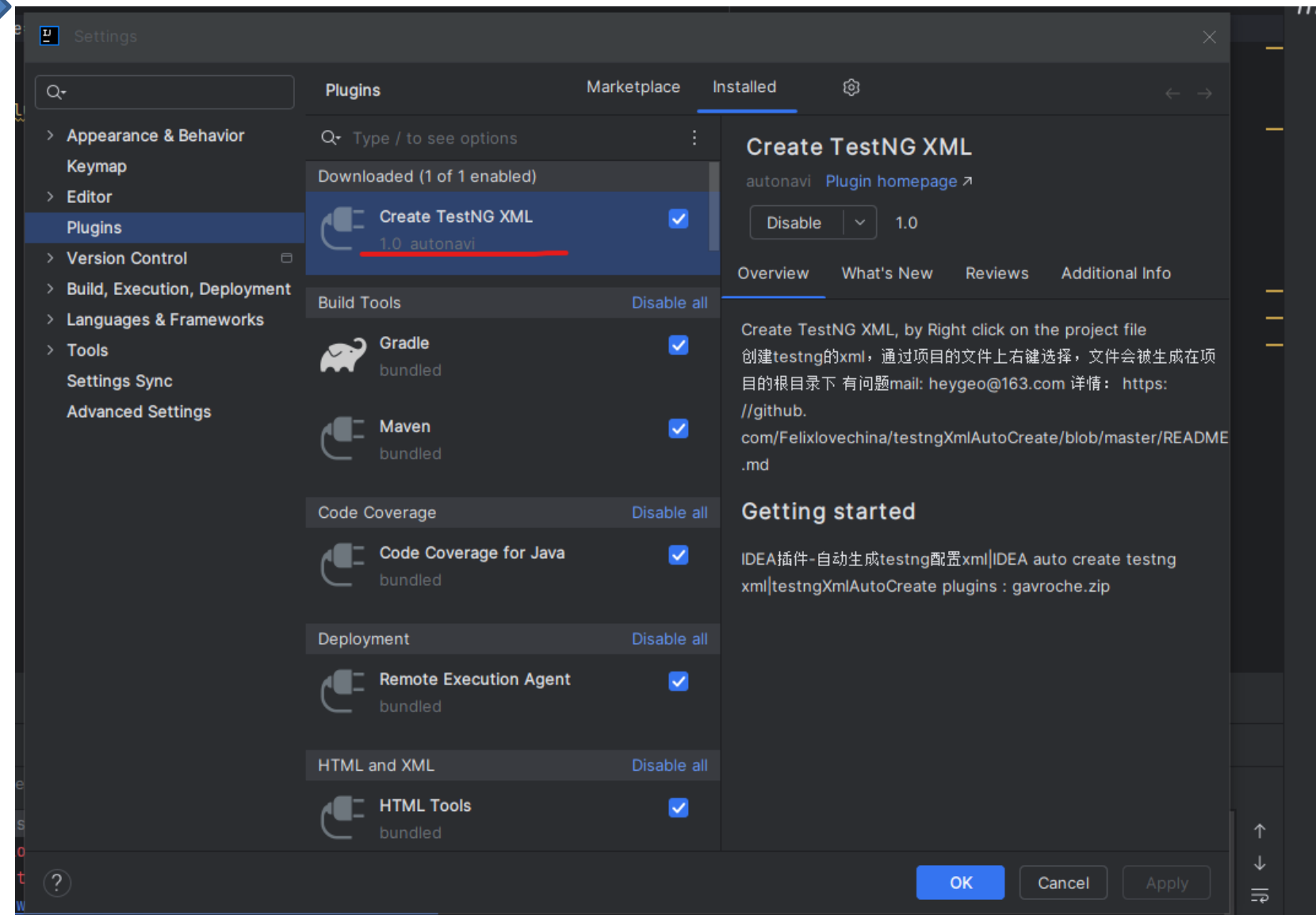
L'annotation **@DataProvider** dans TestNG permet de fournir des données à vos tests sous forme de tableau. Cela vous permet d'exécuter le même test avec plusieurs ensembles de données, facilitant ainsi les tests paramétrés.

```
@DataProvider(name = "Login")
public Object[][] Login() {
    return new Object[][]{
        {"standard_user", "secret_sauce"}, // Cas 1 : Connexion réussie
        {"standard_user", "secret_sau"}, // Cas 2 : Mot de passe incorrect
        {"AMEL", "secret_sauce"}, // Cas 3 : Nom d'utilisateur incorrect
        {"", "secret_sauce"}, // Cas 4 : Nom d'utilisateur vide
        {"AMEL", ""}, // Cas 5 : Mot de passe vide
    };
}
```

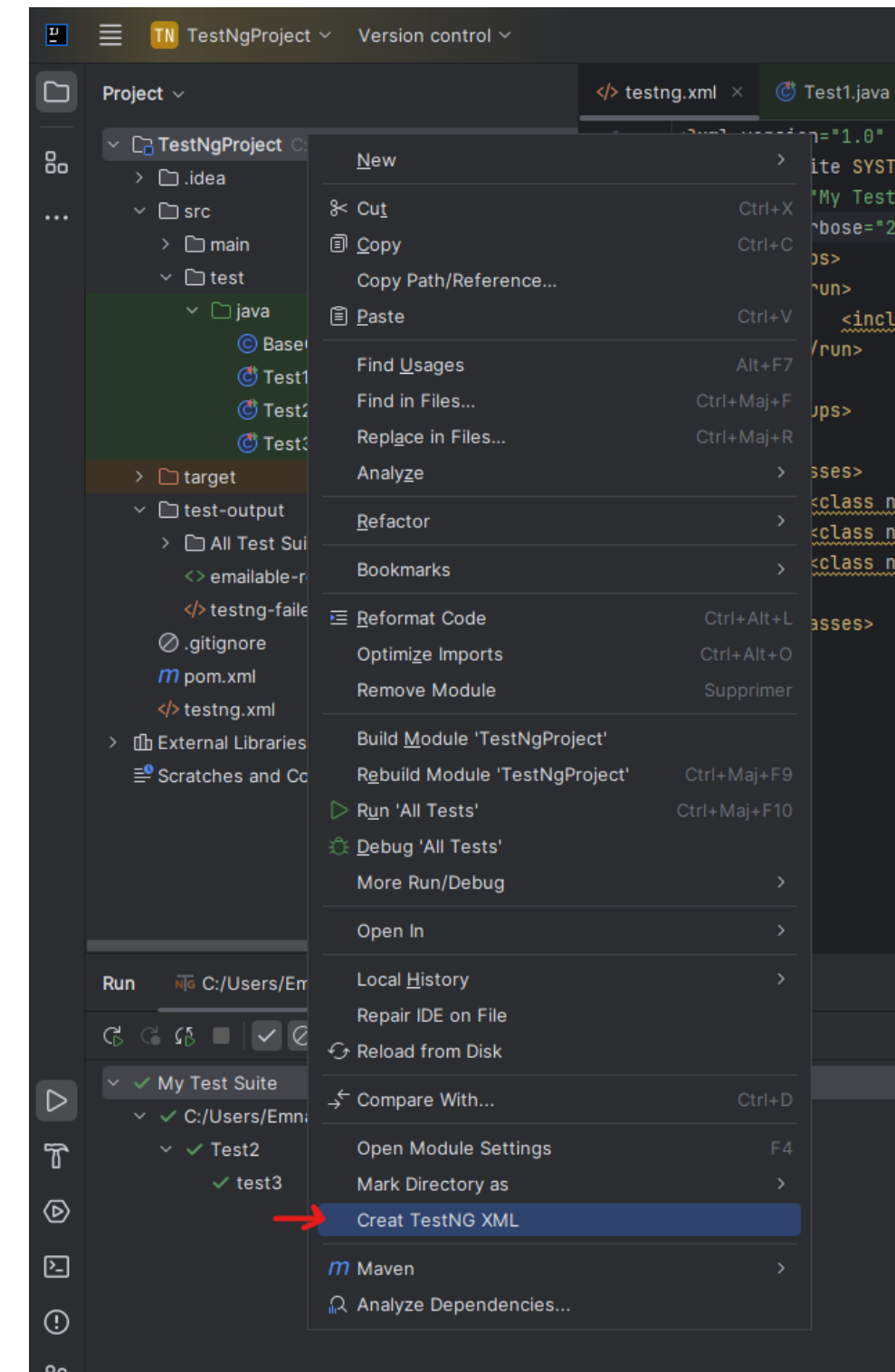
```
@Test(dataProvider = "Login")
public void Test2(String Username, String Password) {
    driver.get("https://www.saucedemo.com/");
    String title = driver.getTitle();
    String expected = "Swag Labs";
    Assert.assertEquals(expected, title);
    WebElement username = driver.findElement(By.xpath(xpathExpression: "//*[@id='user-name']"));
    username.sendKeys(Username);
    WebElement password = driver.findElement(By.xpath(xpathExpression: "//*[@id='password']"));
    password.sendKeys(Password);
    WebElement login = driver.findElement(By.xpath(xpathExpression: "//*[@id='login-button']"));
    login.click();
}
```

Création testng.xml

1



2



L'annotation **@Groups** est utilisée pour regrouper les tests et les exécuter par groupe.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="My Test Suite">
  <test verbose="2" preserve-order="true" name="C:/Users/Emna/IdeaProjects/TestNgProject">
    <groups>
      <run>
        <include name="regression"></include>
      </run>
    </groups>
    <classes>
      <class name="Test1"></class>
      <class name="Test2"></class>
      <class name="Test3"></class>
    </classes>
  </test>
</suite>
```

Run C:/Users/Emna/IdeaProjects/TestNgProject/testng.xml

✓ My Test Suite 10 ms

✓ Tests passed: 1 of 1 test – 10 ms

✓ C:/Users/Emna/IdeaProjects/TestNgProject 10 ms

✓ Test2 10 ms

✓ test3 0 ms

C:\Users\Emna\jdk\openjdk-23.0.1\bin\java.exe ...

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder"

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See <http://www.slf4j.org/codes.html#StaticLoggerBinder> for further details

Before Groups

Methode3_Groupe2

After Groups

=====

My Test Suite

Total tests run: 1, Passes: 1, Failures: 0, Skips: 0

=====

Process finished with exit code 0

```
import ...

public class Test2 {
  @BeforeGroups("regression")
  public void beforeGroups() { System.out.println("Before Groups"); }

  @Test(groups = "Groupe1")
  public void test1() {
    System.out.println("Methode1_Groupe1");
  }

  @Test(groups = "Groupe1")
  public void test2() {
    System.out.println("Methode2_Groupe1");
  }

  @Test(groups = {"regression"})
  public void test3() {
    System.out.println("Methode3_Groupe2");
  }

  @AfterGroups("regression")
  public void afterGroups() {
    System.out.println("After Groups");
  }
}
```

Le parallélisme

Le parallélisme dans TestNG permet d'exécuter des tests simultanément sur plusieurs threads, ce qui réduit le temps d'exécution et optimise les performances des tests.

Un **thread** (ou fil d'exécution) est une unité d'exécution au sein d'un programme. Un programme peut avoir plusieurs threads qui s'exécutent **en parallèle** les uns des autres.

👉 Exemple :

- Un navigateur web (**processus**) peut ouvrir plusieurs onglets (threads).
- Un test Selenium peut exécuter plusieurs cas de test en parallèle sur plusieurs navigateurs (**threads**).

Parallélisme au niveau des classes, méthodes, tests ou instances

◆ Exécution parallèle au niveau des méthodes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="methods" thread-count="3">
  <test name="Test1">
    <classes>
      <class name="SeleniumTest"/>
    </classes>
  </test>
</suite>
```

☞ Chaque méthode de test sera exécutée dans un thread distinct.

◆ Exécution parallèle au niveau des classes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="classes" thread-count="2">
  <test name="Test1">
    <classes>
      <class name="Test1"/>
      <class name="Test2"/>
    </classes>
  </test>
</suite>
```

☞ Chaque classe sera exécutée dans un thread distinct.



■ ■ ◆ Exécution parallèle au niveau des tests (dans <suite>)



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="tests" thread-count="2">
  <test name="Test1">
    <classes>
      <class name="Test1"/>
    </classes>
  </test>
  <test name="Test2">
    <classes>
      <class name="Test2"/>
    </classes>
  </test>
</suite>
```

👉 Chaque test <test> sera exécuté dans un thread distinct.



Parallélisme avec des tests de données (@DataProvider)

- Si un test utilise **@DataProvider**, on peut exécuter les tests en parallèle en ajoutant **parallel = true** :

```
@Test(dataProvider = "Login" )
public void Test2(String Username, String Password , String cas) {
    driver.get("https://www.saucedemo.com/");
    String title = driver.getTitle();
    String expected = "Swag Labs";
    Assert.assertEquals(expected, title);
    WebElement username = driver.findElement(By.xpath( xpathExpression: "//*[@id='user-name']"));
    username.sendKeys(Username);
    WebElement password = driver.findElement(By.xpath( xpathExpression: "//*[@id='password']"));
    password.sendKeys>Password);
    WebElement login = driver.findElement(By.xpath( xpathExpression: "//*[@id='login-button']"));
    login.click();
    System.out.println("Executing " + cas + " on thread: " + Thread.currentThread().getId());
}

@DataProvider(name = "Login", parallel = true)
public Object[][] Login() {
    return new Object[][]{
        {"standard_user", "secret_sauce" , "cas1"}, // Cas 1 : Connexion réussie
        {"standard_user", "secret_sau" , "cas2"}, // Cas 2 : Mot de passe incorrect
        {"AMEL", "secret_sauce" , "cas3"}, // Cas 3 : Nom d'utilisateur incorrect
        {"", "secret_sauce" , "cas4"}, // Cas 4 : Nom d'utilisateur vide
        {"AMEL", "" , "cas5"}, // Cas 5 : Mot de passe vide
    };
}
```

✓ Tests passed: 5 of 5 tests – 10sec 328 ms

SuperClass

Executing cas5 on thread: 50

Executing cas4 on thread: 49

Executing cas3 on thread: 48

Executing cas1 on thread: 46

Executing cas2 on thread: 47

AfterClass

=====

Default Suite

Total tests run: 5, Passes: 5, Failures: 0, Skips: 0

=====

👉 Chaque jeu de données est exécuté dans un thread distinct.

Utilisation de l'annotation @Test(threadPoolSize, invocationCount)

L'annotation @Test(invocationCount, threadPoolSize) permet d'exécuter un test plusieurs fois et de répartir son exécution sur plusieurs threads.

```
public class Parallelisme {  
    @Test(invocationCount = 6, threadPoolSize = 2)  
    public void testMethod() {  
        System.out.println("Executing on thread: " + Thread.currentThread().getId());  
    }  
}
```

- **invocationCount** = 6 → Le test sera exécuté 6 fois.
- **threadPoolSize** = 2 → TestNG va utiliser 2 threads pour exécuter ces 6 itérations en parallèle.

✓ Tests passed: 6 of 6 tests - 20 ms

```
Executing on thread: 29  
Executing on thread: 28  
Executing on thread: 29  
Executing on thread: 28  
Executing on thread: 29  
Executing on thread: 28
```

```
=====  
Default Suite  
Total tests run: 6, Passes: 6, Failures: 0, Skips: 0  
=====
```

Utilisation de parallel="true" dans testng.xml pour Selenium WebDriver

```
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class SeleniumTest {
    WebDriver driver; 5 usages

    @Parameters("browser")
    @BeforeMethod
    public void setup(String browser) {
        if (browser.equalsIgnoreCase("chrome")) {
            driver = new ChromeDriver();
        } else if (browser.equalsIgnoreCase("edge")) {
            driver = new EdgeDriver();
        }
    }

    @Test
    public void testGoogle() {
        driver.get("https://www.google.com");
        System.out.println("Running test on: " + driver.getTitle() + " | Thread ID: " + Thread.currentThread().getId());
    }

    @AfterMethod
    public void tearDown() {
        driver.quit();
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="My Test Suite">
    <test name="ChromeTest">
        <parameter name="browser" value="chrome"/>
        <classes>
            <class name="SeleniumTest"/>
        </classes>
    </test>
    <test name="EdgeTest">
        <parameter name="browser" value="edge"/>
        <classes>
            <class name="SeleniumTest"/>
        </classes>
    </test>
</suite>
```

👉 Cela permet d'exécuter les tests en parallèle sur **Chrome** et **Edge**.

Création rapport Testng

