

## SERENITY BDD con CUCUMBER

### 3

## Implementando pasos y objetos

(Tiempo ejecución 4 horas)

Ya contamos con nuestro archivo de características (**.feature**) listo, pero no hay ningún código en segundo plano para admitir estos pasos, por lo tanto, el trabajo está medio hecho.

Ahora debemos escribir el código necesario para que nuestros pasos sean ejecutados de forma automática.

#### *Recordemos....*

Un modelo BDD está compuesto por la siguiente estructura de conexión

**Feature:** cada línea de un escenario mapea con un método en la clase .definition

**Definition:** cada método mapea con unos métodos en la clase .steps

**Steps:** para la construcción de los pasos es necesaria la definición de objetos en .pageobjects

**Pageobjects:** contiene las clases con la definición de los objetos

Este es nuestro requisito:

Historia de Usuario: Verificar el diligenciamiento de la pantalla “Popup Validation”.

Criterios de Aceptación:

- Verificar diligenciamiento exitoso.
- Verificar mensaje de validación para cada campo

url de prueba: <https://colorlib.com/polygon/metis/login.html>

Pasos para la ejecución de la prueba.

1. Autenticacion en colorlib

- a. Abrir navegador con la url de prueba
- b. Ingresar usuario demo
- c. Ingresar password demo
- d. Click en botón Sign in
- e. Verificar la Autenticación (label en home)

2. Ingresar a Funcionalidad Popup Validation

- a. Clic en Menu “Forms”
- b. Clic en submenú “Form Validation”
- c. Verificación : se presenta pantalla de la funcionalidad con título Popup Validation

3. Diligenciar Formulario Popup Validation

- a. Diligenciar todos los campos del formulario
- b. Clic en botón Validate

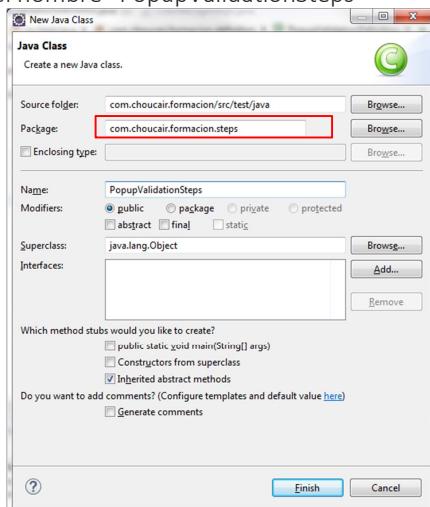
4. Verificar Respuesta Exitosa/Fallida



Objetivo:

Durante esta sesión vamos a enfocarnos en implementar el código requerido para la acción 1 (Autenticar en colorlib), la cual está compuesta por diferentes pasos, incluyendo abrir la url de prueba en un navegador.

Paso 1, Crear la clase **steps**, la cual contendrá el paso a paso de la **definition** seleccionada, en el paquete "com.choucair.formacion.steps" con el nombre "PopupValidationSteps"



Paso 2, agregar el siguiente código:

- 2.1 instanciar una clase que agregaremos para los objetos del login, para el ejemplo la llamaremos "ColorlibLoginPage"
- 2.2 Crear el método que va a contener los pasos para lograr la autenticación, para efectos del ejemplo, copiemos el texto definido en el requisito y lo comentamos, así nos servirá de documentación.
- 2.3 Serenity, proporciona el método ".open()", que nos permitirá abrir la url en el navegador parametrizado de una forma sencilla, más adelante vemos la forma de suministrar la url.
- 2.4 Para los pasos de diligenciar el formulario de autenticación, agregamos el método "IngresarDatos", el cual recibirá los parámetros Usuario y Clave que vienen de la **definition**.
- 2.5 Por último agregamos un método para la verificación de un label existente en el Home de la aplicación que nos ayude a identificar el ingreso a la aplicación

```

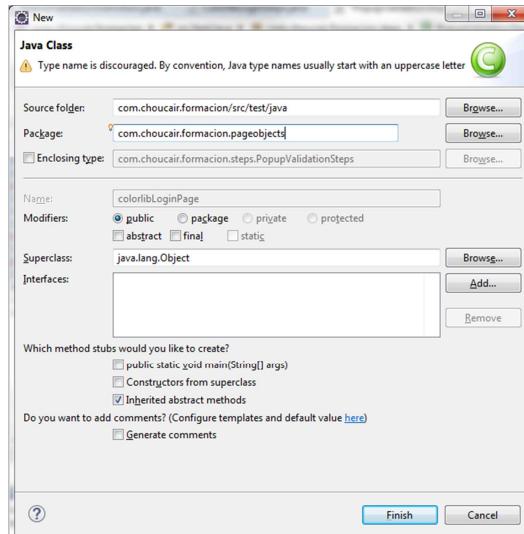
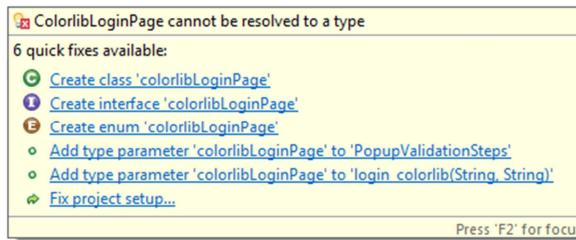
com.choucair.formacion > src/test/java > com.choucair.formacion.steps > PopupValidationSteps >
1 package com.choucair.formacion.steps;
2
3 import net.thucydides.core.annotations.Step;
4
5 public class PopupValidationSteps {
6
7     ColorlibLoginPage colorlibLoginPage; 2.1
8
9     @Step
10    public void login_colorlib(String strUsuario, String strPass) { 2.2
11        // a. Abrir navegador con la url de prueba
12        colorlibLoginPage.open(); 2.3
13        /*
14            b. Ingresar usuario demo
15            c. Ingresar password demo
16            d. Click en botón Sign in */
17        colorlibLoginPage.IngresarDatos(strUsuario, strPass); 2.4
18        /*
19            e. Verificar la Autenticación (label en home)
20            colorlibLoginPage.VerificaHome(); 2.5
21    }
}

```



Paso 3, Crear la clase objects, la cual contendrá la definición de los objetos requeridos para la ejecución de los pasos, en el paquete “com.choucair.formacion.pagobjects”, la cual vamos a llamar “ColorlibLoginPage”.

Como la clase ColorlibLoginPage no existe, se presentará subrayada, posicione el mouse y seleccione “Create class ....”



Paso 3.1, agregar la anotación `@DefaultUrl` con la url de prueba, la cual será utilizada por el método `.open`, que agregamos en los `steps`.

Paso 3.2, extender la clase a PageObject.

```
10 @DefaultUrl("https://colorlib.com/polygon/metis/login.html")
11 public class ColorlibLoginPage extends PageObject{
```

Paso 3.3, Definición de objetos.

Para cada objeto presente en la pantalla, debemos realizar la identificación por alguna de sus propiedades y asignarlas a un objeto, de la siguiente forma:

```
//Campo usuario
@FindBy(xpath="//*[@id='login']/form/input[1]")
public WebElementFacade txtUsername;
```

En este ejemplo estamos identificando el objeto por la propiedad xpath y le asignamos el nombre “txtUsername”

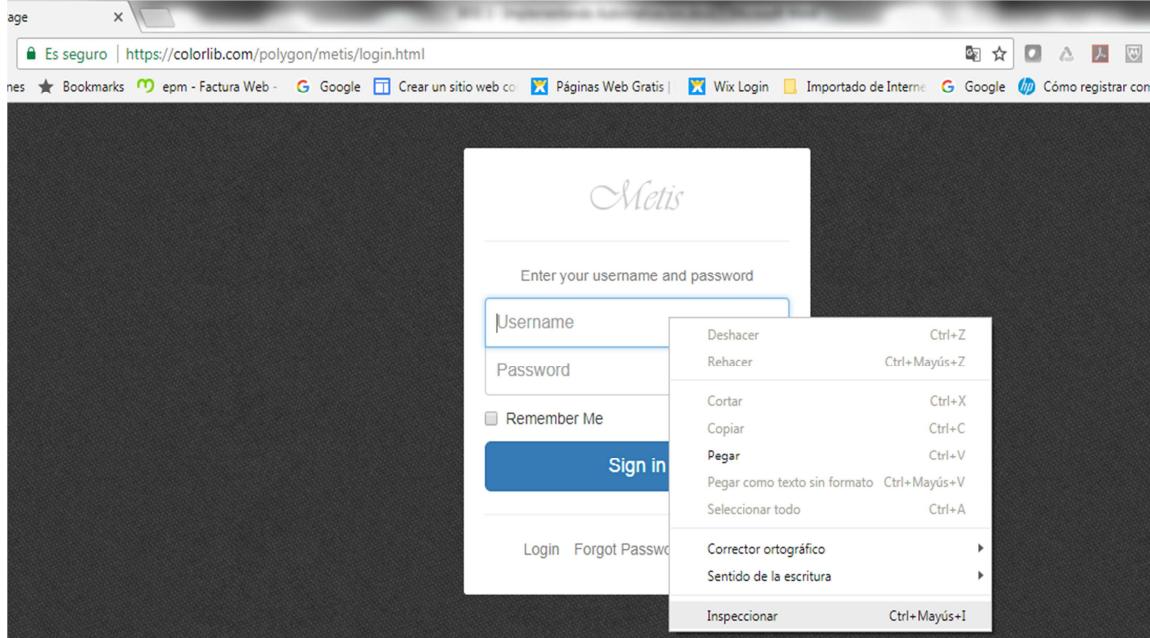


**Nota:**

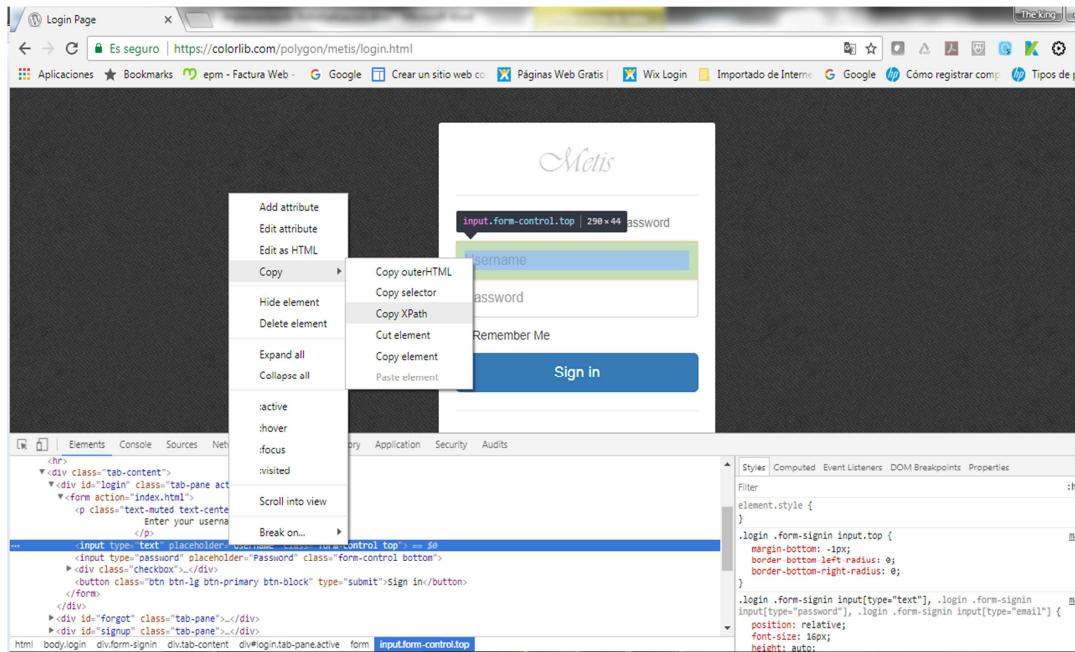
Existen muchas formas de inspeccionar las propiedades de un elemento, para este taller usaremos la siguiente:

Abrir la url en el navegador Chrome

Clic derecho sobre el objeto a inspeccionar > Inspeccionar



En la ventana de desarrollo, dar clic sobre la línea resaltada > Copy > Copy Xpath

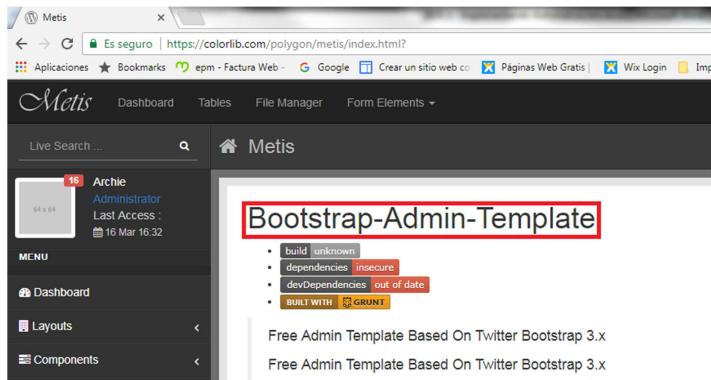


Pegue el xpath devuelto por Chrome **//\*[@id="login"]/form/input[1]** y reemplace las dobles comillas por comilla simple **//\*[@id='login']/form/input[1]**



Repetimos el procedimiento para los demás objetos.

El objeto a mapear en el home será el siguiente:



Al final tendremos algo así:

```
10 @DefaultUrl("https://colorlib.com/polygon/metis/login.html")
11 public class ColorlibLoginPage extends PageObject{
12 //Campo usuario
13@   @FindBy(xpath="//*[@id='login']/form/input[1]")
14   public WebElementFacade txtUsername;
15 // Campo password
16@   @FindBy(xpath="//*[@id='login']/form/input[2]")
17   public WebElementFacade txtPassword;
18 // Boton
19@   @FindBy(xpath="//*[@id='login']/form/button")
20   public WebElementFacade btnSignIn;
21 // label del home a verificar
22@   @FindBy(xpath="//*[@id='bootstrap-admin-template']")
23   public WebElementFacade lblHomePpal;
24
```

Paso 4, implementar el método “IngresarDatos”, el cual invocamos en la clase **steps**.

```
26@   public void IngresarDatos(String strUsuario, String strPass){
27     txtUsername.sendKeys(strUsuario);
28     txtPassword.sendKeys(strPass);
29     btnSignIn.click();
30 }
```

Como vemos, este método recibe los parámetros Usuario y Clave enviados desde los **steps**.

Para hacer referencia a los objetos, utilizamos el nombre de la variable instanciada previamente y al colocar el punto (.) el sistema nos propondrá las diferentes acciones disponibles.



Paso 5, implementar el método “VerificaHome”, el cual invocamos en la clase **steps**.

```
public void VerificaHome(){
    String labelv = "Bootstrap-Admin-Template";
    String strMensaje = lblHomePpal.getText();
    assertThat(strMensaje, containsString(labelv));
}
```

Definamos una variable con el texto que queremos verificar.

Asignamos el valor obtenido del objeto con el método `.getText()`

Y con la sentencia `assertThat`, comparamos si los dos valores son iguales.

Resumen...

Hasta el momento tenemos una clase **steps** que invoca unos métodos creados en una clase **page**, y la clase **page** contiene la definición de los objetos a utilizar, sólo nos falta enlazar la clase **definition** con la clase **steps**.

Paso 6, Agregar el paso creado a la **definition**, así:

Instanciamos la clase de **steps** creada previamente:

```
@Steps
PopupValidationSteps popupValidationSteps;
```

Al interior del método `Given`, agregamos el llamado al método implementado en la clase **steps**, con los pasos requeridos para realizar la autenticación.

```
@Given("^Autentico en colorlib con usuario \"([^\"]*)\" y clave \"([^\"]*)\"$")
public void autentico_en_colorlib_con_usuario_y_clave(String Usuario, String Clave) {
    popupValidationSteps.login_colorlib(Usuario, Clave);
}
```

Como vemos, los parámetros `Usuario` y `Clave`, recibidos desde la feature se pasan como parámetros al método `login_colorlib`.



A continuación veamos como la secuencia de las clases queda finalmente implementada:

Clase: PopupValidationDefinition.java

```
1 package com.choucair.formacion.definition;
2
3 import com.choucair.formacion.steps.PopupValidationSteps;
4
5 import cucumber.api.java.en.Given;
6 import cucumber.api.java.en.Then;
7 import cucumber.api.java.en.When;
8 import net.thucydides.core.annotations.Steps;
9
10 public class PopupValidationDefinition {
11
12     @Steps
13     PopupValidationSteps popupValidationSteps;
14
15
16     @Given("^Autentico en colorlib con usuario \"([^\"]*)\" y clave \"([^\"]*)\"$")
17     public void autentico_en_colorlib_con_usuario_y_clave(String Usuario, String Clave) {
18         popupValidationSteps.login_colorlib(Usuario, Clave);
19     }
20
21     @Given("^Ingreso a la funcionalidad Forms Validation$")
22     public void ingreso_a_la_funcionalidad_Forms_Validation() {
23
24     }
25
26     @When("^Diligencio Formulario Popup Validation$")
27     public void diligencio_Formulario_Popup_Validation() {
28
29     }
30
31     @Then("^Verifico ingreso exitoso$")
32     public void verifco_ingreso_exitoso() {
33
34     }
35
36
37 }
```



### Clase: PopupValidationSteps.java

```

1 package com.choucair.formacion.steps;
2
3 import com.choucair.formacion.pageobjects.ColorlibLoginPage;
4
5 import net.thucydides.core.annotations.Step;
6
7 public class PopupValidationSteps {
8
9     ColorlibLoginPage colorlibLoginPage;
10
11    @Step
12    public void login_colorlib(String strUsuario, String strPass) {
13        // a. Abrir navegador con la url de prueba
14        colorlibLoginPage.open();
15        /* b. Ingresar usuario demo
16        c. Ingresar password demo
17        d. Click en botón Sign in */
18        colorlibLoginPage.IngresarDatos(strUsuario, strPass);
19        // e. Verificar la Autenticación (label en home)
20        colorlibLoginPage.VerificaHome();
21    }
22}

```

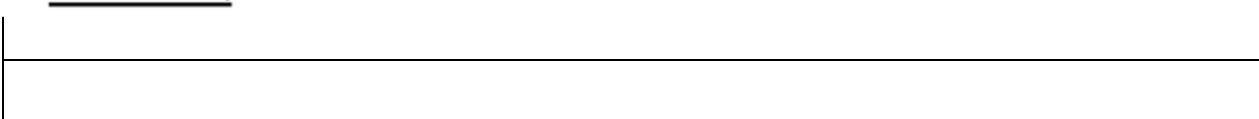
### Clase: ColorlibLoginPage.java

```

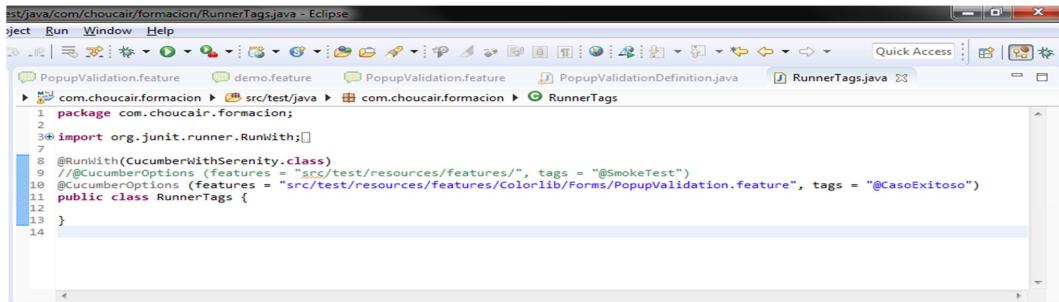
1 package com.choucair.formacion.pageobjects;
2
3 import net.serenitybdd.core.annotations.findby.FindBy;
4 import net.serenitybdd.core.pages.PageObject;
5 import net.serenitybdd.core.pages.WebElementFacade;
6 import net.thucydides.core.annotations.DefaultUrl;
7 import static org.hamcrest.MatcherAssert.assertThat;
8 import static org.hamcrest.Matchers.containsString;
9
10 @DefaultUrl("https://colorlib.com/polygon/metis/login.html")
11 public class ColorlibLoginPage extends PageObject{
12 //Campo usuario
13    @FindBy(xpath="//*[@id='login']/form/input[1]")
14    public WebElementFacade txtUsername;
15 // Campo password
16    @FindBy(xpath="//*[@id='login']/form/input[2]")
17    public WebElementFacade txtPassword;
18 // Boton
19    @FindBy(xpath="//*[@id='login']/form/button")
20    public WebElementFacade btnSignIn;
21 // label del home a verificar
22    @FindBy(xpath="//*[@id='bootstrap-admin-template']")
23    public WebElementFacade lblHomePpal;
24
25
26    public void IngresarDatos(String strUsuario, String strPass){
27        txtUsername.sendKeys(strUsuario);
28        txtPassword.sendKeys(strPass);
29        btnSignIn.click();
30    }
31
32    public void VerificaHome(){
33        String labelv = "Bootstrap-Admin-Template";
34        String strMensaje = lblHomePpal.getText();
35        assertThat(strMensaje, containsString(labelv));
36    }
37
38
39}
40

```



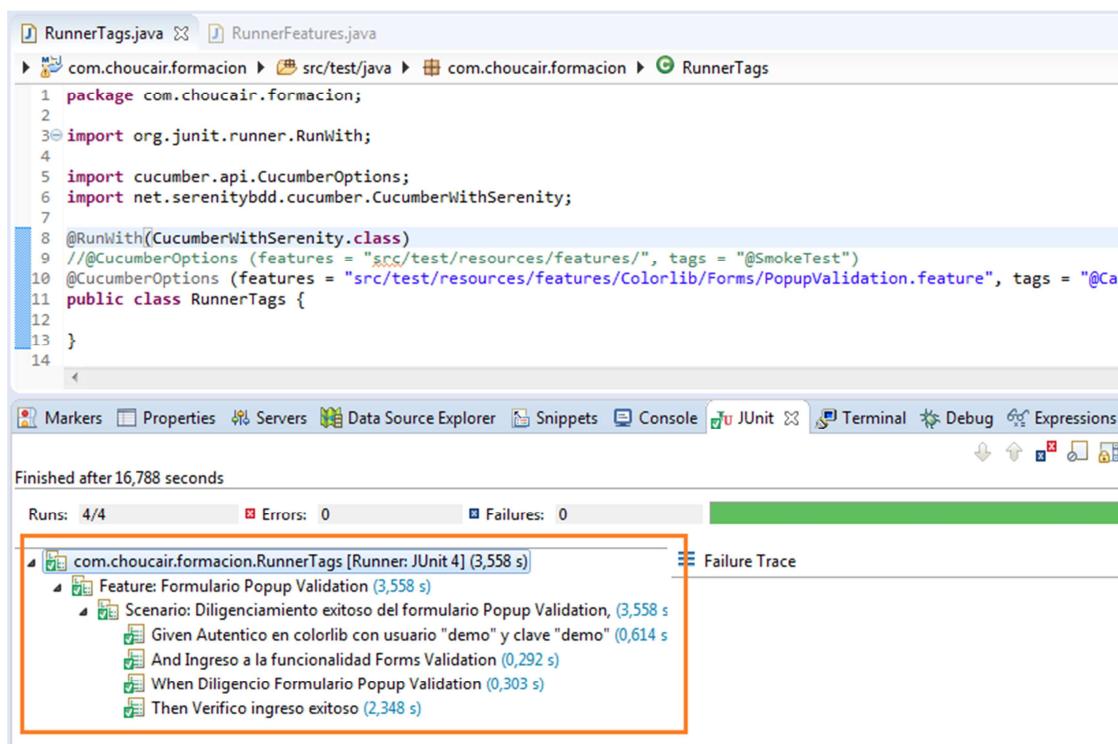


Paso 7, Ejecutar, abrimos la clase "RunnerTags" clic derecho sobre el runner > Run As > JUnit Test



El sistema deberá abrir el navegador Chrome e ingresar a la aplicación colorlib y por último cerrar el navegador.

Dar clic sobre la pestaña Junit, en esta se visualiza el resultado de la ejecución para cada paso:



## Paso 8, generar evidencia

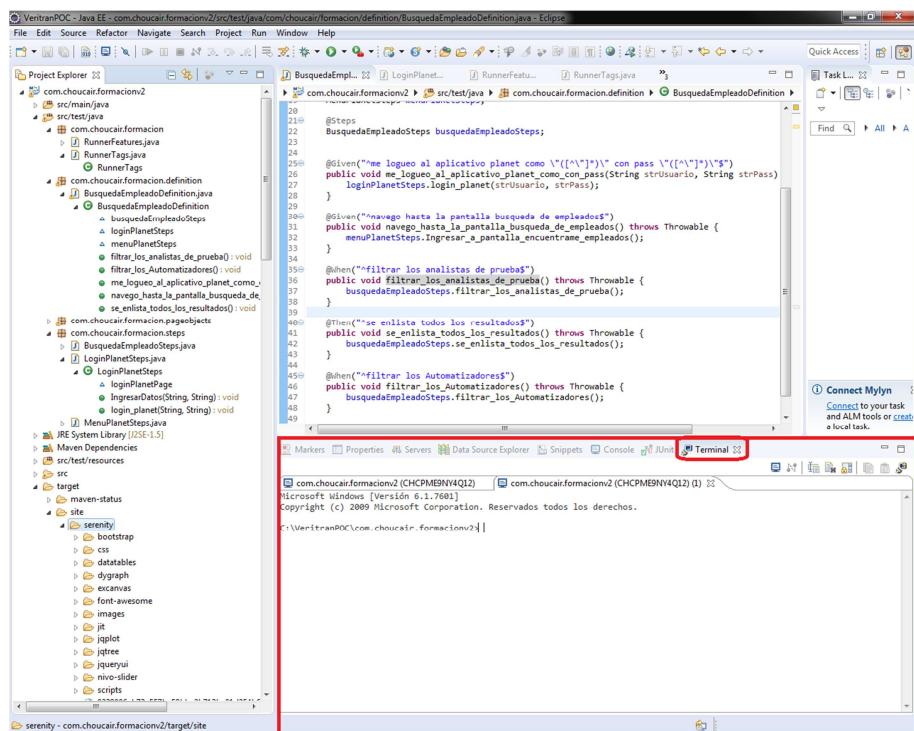
## **Generar Reporte de Ejecución**

Click derecho sobre el proyecto > Show In > Terminal

Se activará la consola Terminal, así:

En esta se debe escribir el siguiente comando:

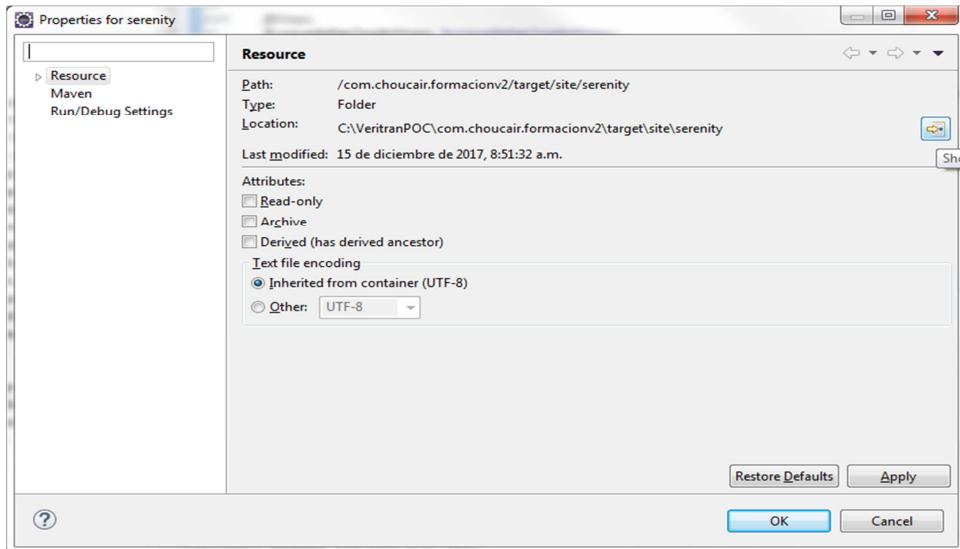
*mvn serenity:aggregate*



Una vez generado el reporte damos clic derecho al folder:

/target/site/serenity > Properties > Dar clic al botón “Show in system explorer”





Ubicar en la carpeta el archive index.html y dar doble click.

Distribution of tests (including rows in data-driven tests) by test result.

Test Type	Total	Pass	Fail	Pending	Ignore/Skip
Automated	1	1 (100%)	0 (0%)	0 (0%)	0 (0%)
Manual	0	0 (0%)	0 (0%)	0 (0%)	0 (0%)
<b>Total</b>	<b>1</b>	<b>1 (100%)</b>	<b>0 (0%)</b>	<b>0 (0%)</b>	<b>0 (0%)</b>
<i>Total Duration</i>					15 seconds

Related Tags:

Category	% Passed	Test count
Capabilities	100%	1
Features	100%	1
Tags	100%	1
Themes	100%	1

Tests:

Show	entries	Search:
2 Tests	5 Steps	Started 15:02:51 Took (secs) 14.51

Diligenciamiento exitoso del formulario Popup Validation,

Dar clic al test, se desplegarán los pasos ejecutados.

Formulario Popup Validation

El usuario debe poder ingresar al formulario los datos requeridos. Cada campo del formulario realiza validaciones de obligatoriedad, longitud y formato, el sistema debe presentar las validaciones respectivas para cada campo a través un globo informativo.

Diligenciamiento Exitoso Del Formulario Popup Validation,

no se presenta ningún mensaje de validación.

Steps	Screenshot	Outcome	Duration
Given Autentico en colorib con usuario "demo" y clave "demo"		SUCCESS	13.4s
When Login colorib: demo, demo		SUCCESS	12.09s
And Ingreso a la funcionalidad Forms Validation		SUCCESS	0.29s
When Diligencio Formulario Popup Validation		SUCCESS	0.3s
Then Verifico ingreso exitoso		SUCCESS	0.3s

Serenity BDD version 1.5.1

