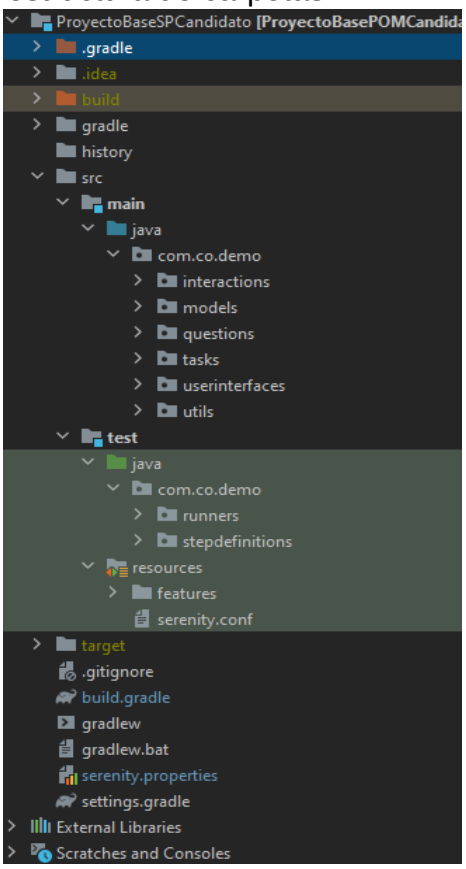
GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Creación de capas estructurales basadas en el patrón de diseño Screenplay

el proyecto debe tener la siguiente estructura de carpetas

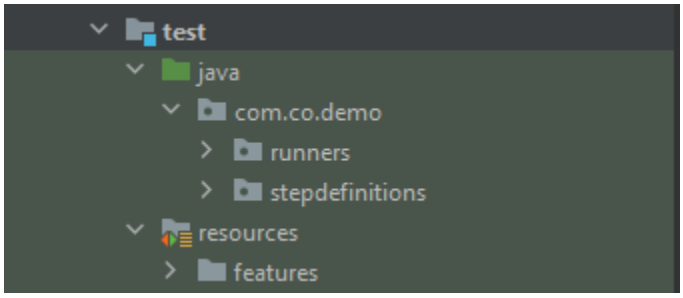


NOTA: observamos dos carpetas con nombre **main** y **test**. vamos a contextualizar y entender nuestra automatización inicialmente desde la carpeta **test** y luego la carpeta **main.**

el flujo de funcionamiento es el siguiente

1. creación del serenityLogin.feature
2. creación y ejecución de Runner.java
3. generación de métodos GIven When Then
4. Creación de StepDefinition.java con los pasos generados.

A continuación, veremos la creación y configuración de la carpeta test la cual contiene los runners, step definitions y archivos. feature con nuestras historias de usuario en lenguaje Gherkin y orientadas a BDD.



GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Features

En la carpeta **features** se van a incluir los archivos .feature que contienen las historias de usuario en lenguaje Gherkin, acá se definirían todos los escenarios de prueba y se van a incluir las anotaciones @Tags que son llamados desde el runner

serenityLogin.feature

```
#Autor: Michael Garzon Rodriguez
#language:en

@regression
Feature: Login on the website

  Background:
    Given the user is on the serenity demo page

  @successlogin
  Scenario Outline: testing the login module successfully
    When attempts to log in
      | user   | pass   |
      | <user> | <pass> |
    Then will validate the text on screen <message>
    Examples:
      | user  | pass     | message   |
      | admin | serenity | Dashboard |
```

Nota:

Usamos **@tag** para marcar un feature que pertenezca a una regresión o un caso más específico sería un escenario exitoso y uno fallido.

La palabra clave **Feature:** no puede duplicarse en el mismo **.feature**, El propósito de la misma es proporcionar uina descripción de alto nivel de una característica de software y agrupar escenarios relacionados.

la palabra clave **Background:** se usa para definir una precondición de la prueba. así no repetimos el mismo escenario en caso de múltiples escenarios.

La palabra clave **Scenario Outline:** se puede utilizar para ejecutar el mismo escenario varias veces, con diferentes combinaciones de valores usando la palabra clave **Examples** y una tabla con datos.

la palabra clave **Given:** se utiliza para describir el contexto inicial del sistema, es decir, la situación en la que se encuentra el escenario. Normalmente, esto es algo que ocurrió en el pasado.

La palabra clave **When:** se utiliza para describir un evento o una acción. Esto puede ser una persona interactuando con el sistema, o puede ser un evento desencadenado por otro sistema.

La palabra clave **Then:** se utiliza para describir un resultado esperado o un resultado.

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Runners

En la carpeta **runners** se van a incluir las clases de Java que nos permitan invocar a través de anotaciones de **JUnit** y **Cucumber Junit** para la ejecución de nuestras pruebas con Java y Cucumber.

aquí indicaremos el tipo de Runner que vamos a utilizar, tal sea, un runner personalizado o el runner de Cucumber con SerenityBDD.

SerenityLoginRunner.java (CucumberWithSerenityRunner.class)

```java
package com.co.demo.runners;

import ...

no usages  Michael Ferneein Garzon Rodriguez *
@RunWith(CucumberWithSerenity.class)
@CucumberOptions(
        features = "src/test/resources/features/serenityLogin.feature",
        glue = "com.co.demo.stepdefinitions",
        tags = "@@successlogin",
        plugin = {"pretty","json:target/cucumber-reports/cucumber.json"},
        snippets = CucumberOptions.SnippetType.CAMELCASE
)
public class SerenityLoginRunner {
}
```

Nota: esta sería la forma de construir un runner básico con la clase CucumberWithSerenity

SerenityLoginRunner.java (RunnerPersonalizado.class)

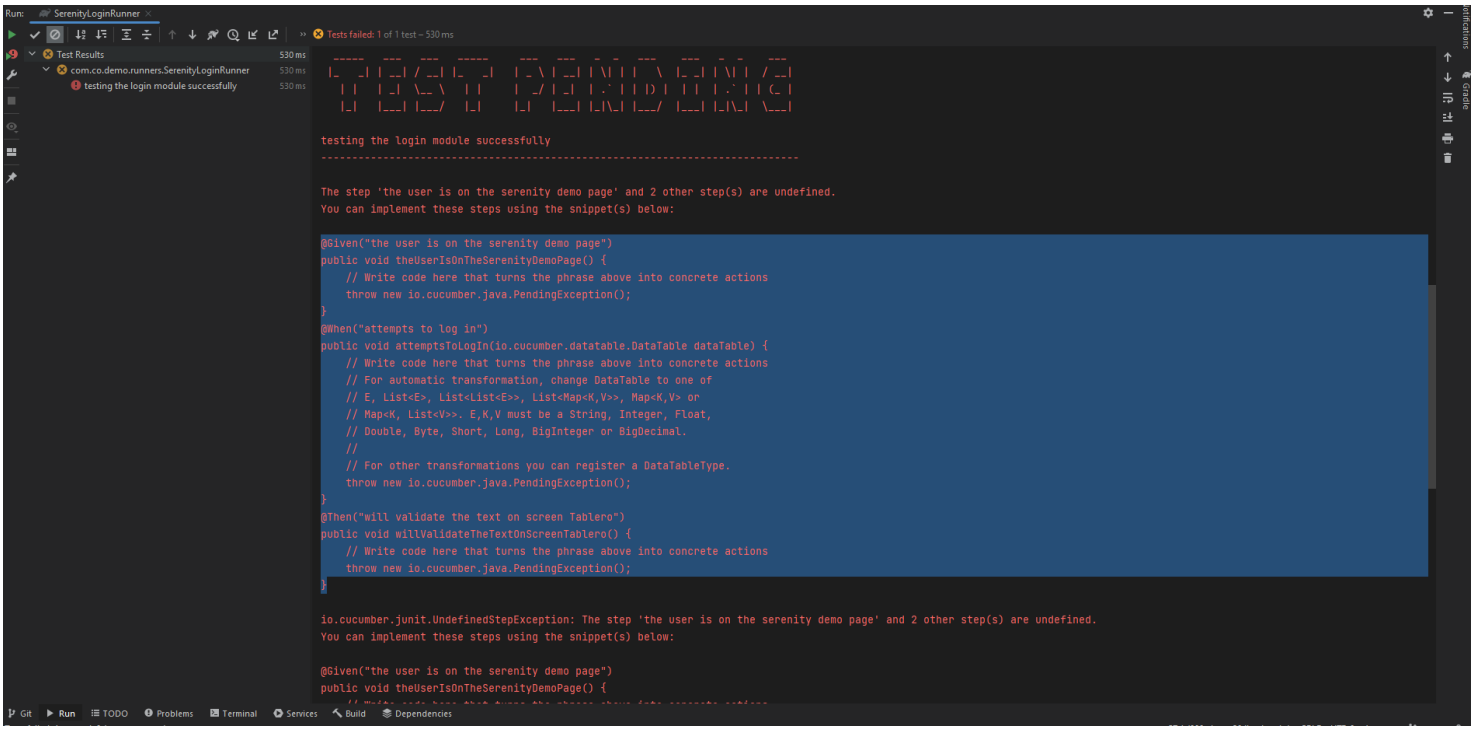
```java
package com.co.demo.runners;

import ...

no usages  Michael Ferneein Garzon Rodriguez *
@RunWith(RunnerPersonalizado.class)
@CucumberOptions(
        features = "src/test/resources/features/serenityLogin.feature",
        glue = "com.co.demo.stepdefinitions",
        tags = "@successLogin",
        plugin = {"pretty","json:target/cucumber-reports/cucumber.json"},
        snippets = CucumberOptions.SnippetType.CAMELCASE
)
public class SerenityLoginRunner {
    no usages  new *
    @BeforeSuite
    public static void test() throws IOException, InvalidFormatException {
        DataToFeature.overrideFeatureFiles( featuresDirectoryPath: "src/test/resources/features/serenityLogin.feature");
    }
}
```

Nota: podemos observar la opción de cucumber de plugin para reportes de cucumber y dentro de la clase podemos ver una anotación customizada que indica que el método **test()** debe ejecutarse antes que la suite de pruebas y así sobrescribir el **.feature** indicado.

seguimos a ejecutar el runner para generar los métodos o snippets que vamos a pegar en la clase StepDefinitions.java o a través de consola con el comando

gradle clean test aggregate -Dtags=@caso1

donde @caso1 sería el tag que queremos correr primero.



al ejecutar la prueba por primera vez la prueba va a “fallar” pero en realidad nos lanzó los snippets que aún no se encuentran definidos en los **stepdefinitions.** Los elementos resaltados en la imagen se copian y pegan en el step definitions, una vez los tengamos en el stepDefinitions solo debemos eliminar todos los comentarios y las excepciones que ya no van a ser necesarias para nuestra prueba.

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

StepDefinitions

En la carpeta **StepDefinitions** se van a incluir las clases de Java que nos permitan definir los pasos que vamos a seguir para ejecutar nuestra prueba. Aquí es donde se van a pegar los snippets o métodos generados en consola después de ejecutar el runner por primera vez. Estos métodos son resultado de traducir el archivo .feature al lenguaje de programación Java.

SerenityLoginStepDefinitions.java

```
package com.co.choucair.stepdefinitions;

import ...

Michael Ferneein Garzon Rodriguez *
public class SerenityLoginStepDefinitions {

    1 usage   Michael Ferneein Garzon Rodriguez
    @Given("the user is on the serenity demo page")
    public void theUserIsOnTheSerenityDemoPage() {
        OnStage.theActorCalled(ACTOR).wasAbleTo(Open.url(URL));
    }
    1 usage   Michael Ferneein Garzon Rodriguez
    @When("attempts to log in")
    public void attemptsToLogIn(DataTable dataTable) {
        OnStage.theActorInTheSpotlight().attemptsTo(
                Login.onTheSite(UserLoombokData.setData(dataTable).get(0))
        );
    }
    1 usage   Michael Ferneein Garzon Rodriguez *
    @Then("^validate the text on screen (.*)$")
    public void validateTheTextOnScreenDashboard(String text) {
        OnStage.theActorInTheSpotlight().should(seeThat(ValidateText.of(TXT_VALIDATION), containsString(text)));
    }

}
```

En este caso especial de Screenplay tenemos que crear una clase llamada **Hooks.java** la cual va a contener el método setTheStage() con la anotación @**Before** el cual se va a ejecutar antes que las pruebas y nos va a configurar el escenario para el actor y un método que le dirá al WebDriver que cierre el navegador después de la prueba.

A continuación, un ejemplo de la clase **Hooks.java**

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Hooks.java

```java
package com.co.choucair.stepdefinitions;

import ...

Michael Ferneein Garzon Rodriguez *
public class Hooks {
    Michael Ferneein Garzon Rodriguez
    @Before
    public void setup() {
        OnStage.setTheStage(new OnlineCast());
    }
    Michael Ferneein Garzon Rodriguez
    @After
    public void close() {
        getDriver().close();
    }
}
```

En esta clase podemos incluir funciones que podrían ejecutarse antes de las pruebas esto con el fin de controlar el inicio y el fin de la prueba.

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Tasks

En la carpeta **tasks** se van a incluir las clases de Java que nos permitan usar la interfaz **Task** y esta al integrarse con la palabra clave implements para poder invocar los métodos abstractos de la interfaz y poder usar el método **performAs()** el cual es un método polimorfo debido a la anotación @**Override** la cual sobreescribe el método cada vez que se ejecuta, las clases de tipo **Task** se deben nombrar de tal manera que conjugen una acción, por ejemplo, Create, Add, Authenticate… seguida de un . y el método para acceder a la clase de tal manera que la sintaxis tenga un acercamiento a lenguaje natural, por ejemplo, Create.userOn(), Add.newUser(), Authenticate.onTheSite() …

SerenityLoginSteps.java

```
1   package com.co.choucair.tasks;
2
3   import ...
14
    Michael Ferneein Garzon Rodriguez *
15  public class Login implements Task {
        3 usages
16      UserLoombokData userLoombokData;
        Michael Ferneein Garzon Rodriguez
17      public Login(UserLoombokData userLoombokData) { this.userLoombokData = userLoombokData; }
20
        Michael Ferneein Garzon Rodriguez
21      @Override
22      public <T extends Actor> void performAs(T actor) {
23          actor.attemptsTo(
24                  WaitUntil.the(SerenityLoginPage.TXT_USER, isVisible()).forNoMoreThan( amount: 10).seconds(),
25                  Enter.theValue(userLoombokData.getUser()).into(SerenityLoginPage.TXT_USER),
26                  Enter.theValue(userLoombokData.getPass()).into(SerenityLoginPage.TXT_PASS),
27                  JavaScriptClick.on(SerenityLoginPage.BTN_SUBMIT),
28                  WaitUntil.the(SerenityLoginPage.TXT_VALIDATION, isVisible()).forNoMoreThan( amount: 10).seconds()
29          );
30      }
31
        1 usage   Michael Ferneein Garzon Rodriguez
32      public static Login onTheSite(UserLoombokData userLoombokData){
33          return Instrumented.instanceOf(Login.class).withProperties(userLoombokData);
34      }
35  }
```

Vamos por partes:

- **Línea 15:** después del nombre de la clase se escribe **implements Task** y este se marcará en error pidiendo que se integre un método abstracto el cual observamos en la **línea 22**.
- **Linea 16:** se crea una variable de tipo **UserLoombokData** la cual hace parte de una clase modelo.
- **Linea 17:** constructor de la clase para asignar los valores a la variable **userLoombokData**
- **Linea 23:** el uso del actor para definir las interacciones con los elementos web como Enter, Click.
- **Linea 32:** Método **Perfomable** el cual nos permite instanciar la clase a través del método OnTheSite() con las propiedades que se le asignan por parámetro.

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

UserInterfaces

En la carpeta **userinterfaces** se van a incluir las clases de Java que van a ser las que nos permitan crear los selectores que van a contener el selector de tipo Xpath, CssSelector.

usamos la clase **Target** para poder localizar los elementos xpath de esta manera

private static final Target ELEMENTO = Target.the(“elemento web”).locatedBy(“xpath o cssSelector”)

SerenityLoginPage.java

```
package com.co.choucair.userinterfaces;

import ...

7 usages  Michael Ferneein Garzon Rodriguez
public class SerenityLoginPage {
    2 usages
    public static final Target TXT_USER = Target.the( targetElementName: "txt")
            .located(By.id("LoginPanel0_Username"));
    1 usage
    public static final Target TXT_PASS = Target.the( targetElementName: "txt")
            .located(By.xpath( xpathExpression: "//input[@id='LoginPanel0_Password']"));
    1 usage
    public static final Target BTN_SUBMIT = Target.the( targetElementName: "txt")
            .located(By.xpath( xpathExpression: "//button[@id='LoginPanel0_LoginButton']"));
    3 usages
    public static final Target TXT_VALIDATION = Target.the( targetElementName: "txt")
            .located(By.xpath( xpathExpression: "//h1"));

}
```

Nota: si necesita practicar la creación de los Xpath o CSSselector se recomienda practicar en estos sitios tomando apuntes con el fin de que se pueda aprender las distintas maneras que puede usar para crear un selector efectivo y funcional. (***NO SE RECOMIENDA*** copiar y pegar el xpath directamente del sitio ya que puede copiar un selector absoluto y esto es considerado mala práctica)

Referencias**:**

https://topswagcode.com/xpath/ : Aprende xpath gamificado

https://flukeout.github.io/ : Aprende CSSSelector gamificado

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Models

En la carpeta **models** se van a incluir las clases de Java que van a ser las que nos permitan manejar la data de prueba que vamos a traer desde el archivo .feature que contiene las variables que vamos a usar. ejemplo: usuario y contraseña.

UserLoombokData.java

```java
package com.co.choucair.models;

import ...
9 usages  Michael Ferneein Garzon Rodriguez *
@Data
public class UserLoombokData {

    String user;
    String pass;

    1 usage  Michael Ferneein Garzon Rodriguez
    public static List<UserLoombokData> setData(DataTable table){
        List<UserLoombokData> data = new ArrayList<>();
        List<Map<String, String>> mapList = table.asMaps();
        for (Map<String, String> map : mapList) {
            data.add(new ObjectMapper().convertValue(map, UserLoombokData.class));
        }
        return data;
    }
}
```

Justo encima del nombre de la clase vamos a dejar la anotación **@Data** de Loombook la cual nos generará implícitamente getters para todos los campos, un método toString y un método HashCode.

Variables ***user*** y ***pass*** las cuales tienen métodos Getter implicitos que nos permiten llamar los valores de las variables desde otras clases.

Además, tenemos el método ***setData(DataTable table)*** el cual nos va a permitir recibir un argumento DataTable el cual va a ser usado para transformar la data del .feature a un ArrayList() para luego instanciar la clase con los datos que vienen desde el archivo .feature.

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Questions

En la carpeta **questions** se van a incluir las clases de tipo Question<ANSWER> que nos permitan crear aserciones mas dinámicas y fáciles de entender y mantener.

La interfaz Question<ANSWER> recibe un tipo de dato que sería el que la clase va a devolver, un ejemplo sería reemplazar ANSWER por Boolean para validar si un elemento es visible o no. También podría reemplazar ANSWER con String para que nuestra clase pueda extraer un texto y compararlo.

ValidateText.java

```
1   package com.co.choucair.questions;
2
3   > import ...
7
    3 usages  Michael Ferneein Garzon Rodriguez *
8   public class ValidateText implements Question<String> {
        2 usages
9       Target element;
10
        1 usage  Michael Ferneein Garzon Rodriguez
11      public ValidateText(Target element) {
12          this.element = element;
13      }
        Michael Ferneein Garzon Rodriguez
14      @Override
15      public String answeredBy(Actor actor) {
16          return element.resolveFor(actor).getText();
17      }
18
        1 usage  Michael Ferneein Garzon Rodriguez *
19  @   public static Question<String> of(Target element){
20          return new ValidateText(element);
21      }
22  }
```

Vamos por partes:

- **Linea 8:** después del nombre de la clase se escribe **implements Question<String>** y este se marcará en error pidiendo que se integre un método abstracto el cual observamos en la **línea 15**
- **Linea 9:** variable de tipo **Target** la cual va a contener el selector que se le asigne a la case a través del método perfomable
- **Linea 11:** constructor de la clase.
- **Linea 16:** retorna el texto en este caso, el elemento debe ser resuelto para el actor y dependiendo de el tipo de interfaz que usamos podemos ver sugerencias del tipo de validación que necesitamos hacer.
- **Linea 19:** Método **Perfomable** el cual nos permite instanciar la clase a través del método OnTheSite() con las propiedades que se le asignan por parámetro

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Interactions

En la carpeta **interactions** se van a incluir las clases de tipo Interaction la cual es idéntica a la clase de tipo Task con la diferencia que podemos crear nuestras propias interacciones con la librería Selenium y cualquier otra utilidad.

Podemos diseñar interacciones como una lista desplegable, un Drag&Drop, un Thread,sleep() etc.

DropDownList.java

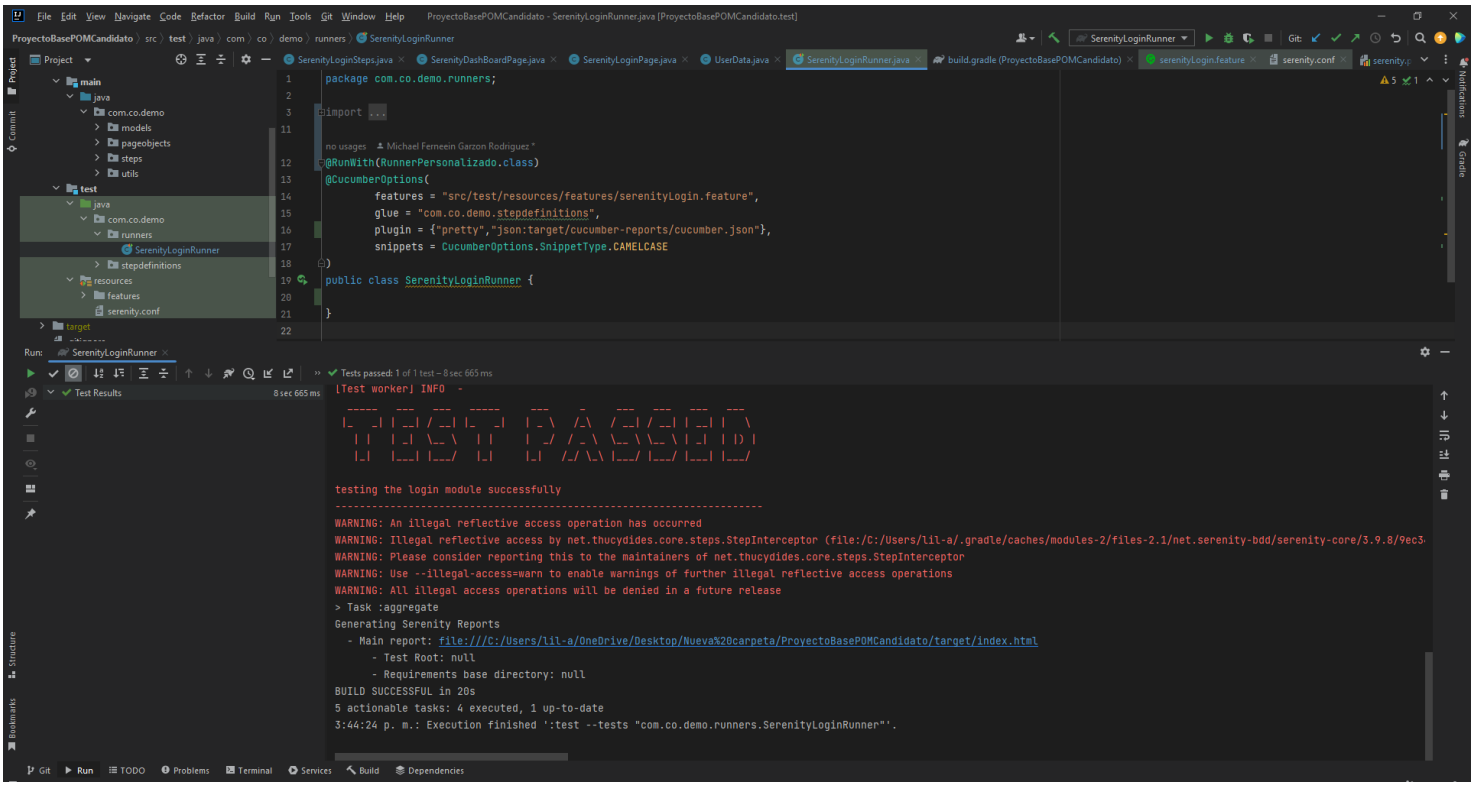
```java
public class InteractionDemo implements Interaction {

    2 usages
    Target element;
    2 usages
    String text;
    new *
    public InteractionDemo(Target element, String text) {
        this.element = element;
        this.text = text;
    }
    Michael Ferneein Garzon Rodriguez *
    @Override
    public <T extends Actor> void performAs(T actor) {
        WebElement father = element.resolveFor(actor);
        List<WebElement> childs = father.findElements(By.tagName("li"));
        for (WebElement i : childs) {
            if (i.getText().contains(text)){
                i.click();
                break;
            }
        }
    }
    Michael Ferneein Garzon Rodriguez *
    public static InteractionDemo on(Target element, String text) {
        return Instrumented.instanceOf(InteractionDemo.class).withProperties(element, text);
    }
}
```

En esta interacción podemos observar que en el método abstracto performAs() estamos usando la interfaz **WebElement** para capturar un elemento y resolverlo para el actor. Seguido tenemos una lista de **java.util** de tipo **WebElement** llamada **childs** y esta tomará la variable **father** y va a buscar entre todos sus hijos la etiqueta **li** y la va a almacenar en la lista.

Seguido tenemos un ciclo **foreach** el cual usará una variable **i** de tipo **WebElement** para recorrer la lista **childs** y por cada elemento va a ir preguntando si el texto del elemento inspeccionado por la variable **i** en la lista **childs** contiene el texto de la opción que queremos, si es así entonces daría un click a la opción y cerraría el ciclo con la palabra clave **break.**

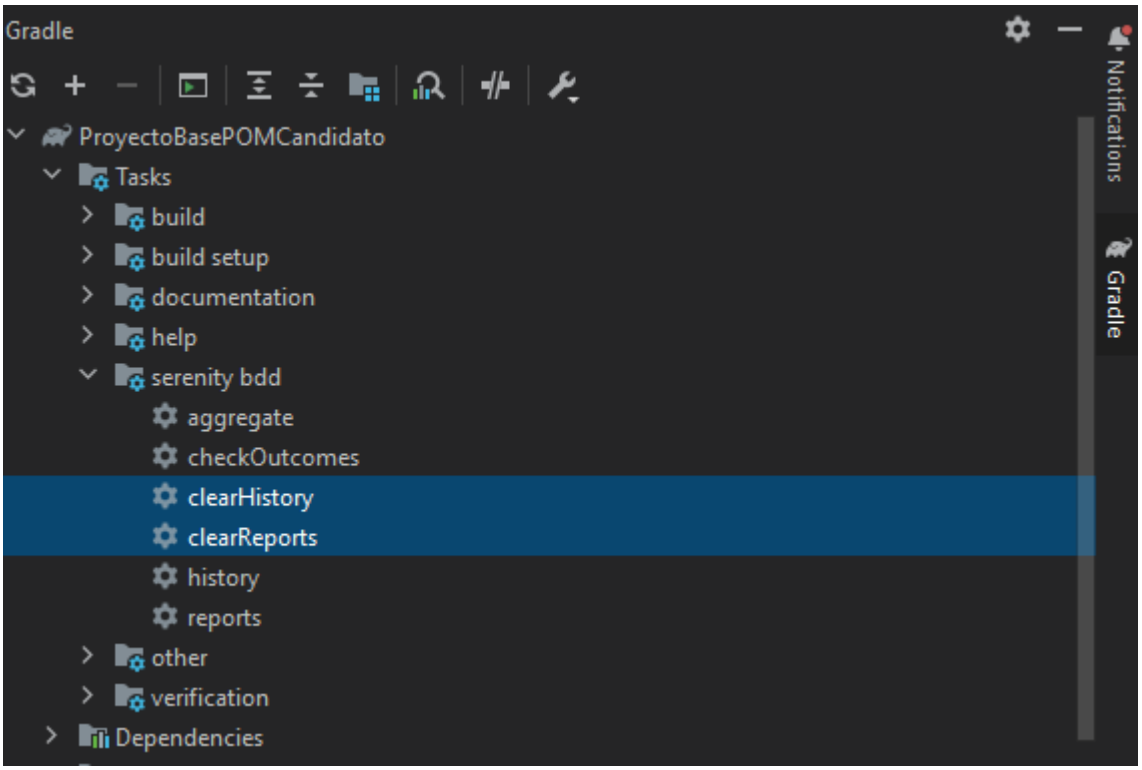
GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Ejecutamos la prueba a través del runner y vamos a observar la prueba exitosa.



Este sería el reporte. al haber varias ejecuciones nos mostrará gráficas de acuerdo con cuantas ejecuciones tenemos guardada en la carpeta **target.**

si desea eliminar todo rastro de pruebas y que empiece de nuevo puede acceder al plugin de Gradle ubicado en la pestaña lateral derecha



¡Solo debe doble clicar las dos opciones resaltadas y es todo!

Esta sería la estructura y definición de las capas del patrón de diseño POM. En el caso que necesite soporte y preguntas no dude en escribir al correo mgarzonr@choucairtesting.com o preguntar a algún automatizador del producto que esté asignado al soporte.

Material de ayuda:

- https://www.programiz.com/java-programming - Aprende Java
- https://cucumber.io/docs/guides/- Aprende Cucumber
- https://topswagcode.com/xpath/ - Aprende a crear Xpath jugando
- https://flukeout.github.io/ Aprende a crear CSSselector jugando
- https://www.selenium.dev/documentation/webdriver/ - Aprende a usar WebDriver de Selenium
- https://www.browserstack.com/guide/verify-and-assert-in-selenium - Assert

- https://portalacademico.cch.unam.mx/cibernetica1/algoritmos-y-codificacion/caracteristicas-POO#tab2