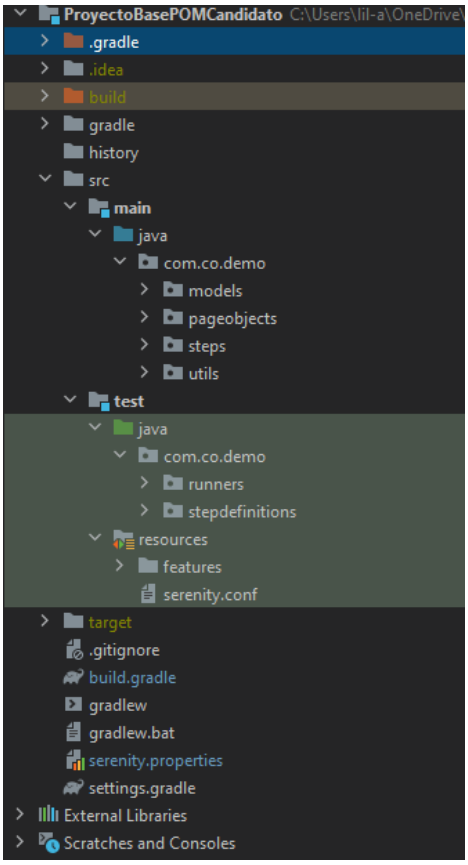
GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Creación de capas estructurales basadas en el patrón de diseño POM

el proyecto debe tener la siguiente estructura de carpetas

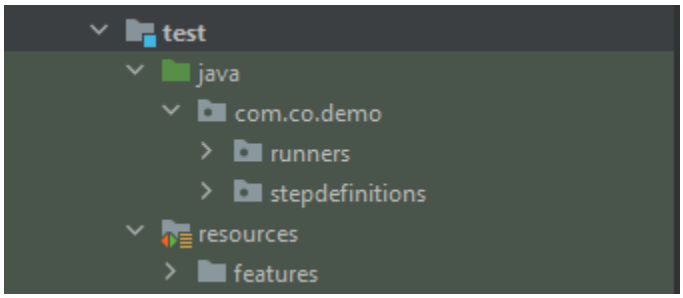


NOTA: observamos dos carpetas con nombre **main** y **test**. vamos a contextualizar y entender nuestra automatización inicialmente desde la carpeta **test** y luego la carpeta **main.**

el flujo de funcionamiento es el siguiente

1. creación del serenityLogin.feature
2. creacion y ejecucion 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)

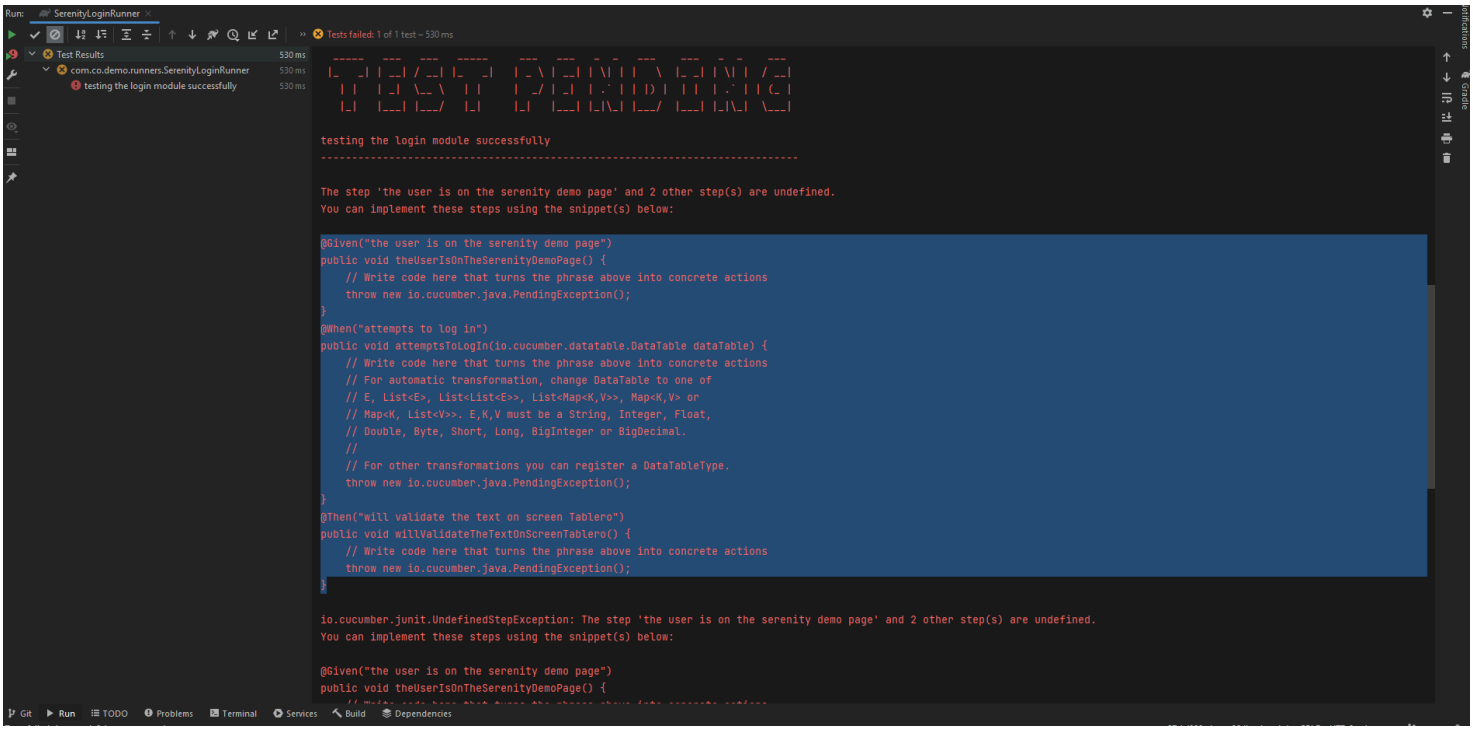
```java
package com.co.demo.runners;

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 plugins 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 sobreescribir 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 traves 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

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

import ...



5 usages  Michael Ferneein Garzon Rodriguez *
public class SerenityLoginStepDefinitions {

    no usages
    @Steps
    protected SerenityLoginSteps loginSteps;

    1 usage  Michael Ferneein Garzon Rodriguez *
    @Given("the user is on the serenity demo page")
    public void theUserIsOnTheSerenityDemoPage() {
        // Write code here that turns the phrase above into concrete actions
        throw new io.cucumber.java.PendingException();
    }
    1 usage  Michael Ferneein Garzon Rodriguez *
    @When("attempts to log in")
    public void attemptsToLogIn(io.cucumber.datatable.DataTable dataTable) {
        // Write code here that turns the phrase above into concrete actions
        // For automatic transformation, change DataTable to one of
        // E, List<E>, List<List<E>>, List<Map<K,V>>, Map<K,V> or
        // Map<K, List<V>>. E,K,V must be a String, Integer, Float,
        // Double, Byte, Short, Long, BigInteger or BigDecimal.
        //
        // For other transformations you can register a DataTableType.
        throw new io.cucumber.java.PendingException();
    }
    1 usage  Michael Ferneein Garzon Rodriguez *
    @Then("will validate the text on screen Tablero")
    public void willValidateTheTextOnScreenTablero() {
        // Write code here that turns the phrase above into concrete actions
        throw new io.cucumber.java.PendingException();
    }
}
```

Vamos a incluir la anotación **@Steps** para crear una variable del tipo de la clase que contiene todos nuestros step. en nuestro caso particular tenemos la clase **SerenityLoginSteps.java** que ya contiene los metodos definidos y que vamos a incluir en cada uno de los pasos **Given When Then**

ahora te preguntas. porque la palabra clave protected?

vamos a mantener el uso exclusivo para esta clase y no necesitamos acceder a esta variable desde otra clase. Esta propiedad es parte del paradigma de programación a objetos POO. puedes ver los conceptos en este enlace

Paradigma de la POO

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

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

```
1   package com.co.demo.stepdefinitions;
2
3   import ...
11
12
        3 usages   Michael Ferneein Garzon Rodriguez *
13  public class SerenityLoginStepDefinitions {
14
            3 usages
15      @Steps
16      protected SerenityLoginSteps loginSteps;
17
            1 usage   Michael Ferneein Garzon Rodriguez *
18      @Given("the user is on the serenity demo page")
19      public void theUserIsOnTheSerenityDemoPage(){
20          loginSteps.OpenTheWebSite(GlobalData.baseUrl);
21      }
22
            1 usage   Michael Ferneein Garzon Rodriguez *
23      @When("attempts to log in")
24      public void attemptsToLogIn(DataTable table) {
25          loginSteps.login(UserData.setData(table).get(0));
26      }
27
            1 usage   Michael Ferneein Garzon Rodriguez
28      @Then("^will validate the text on screen (.*)$")
29      public void willValidateTheTextOnScreenMessage(String text) {
30          loginSteps.validateTextOnScreen(text);
31      }
32  }
```

La anotación **@Steps** que se observa en la línea 15 hace referencia a la variable de la clase **SerenityLoginSteps** que contiene todos los **step** que van a ser usados dentro de cada uno de los métodos que define **Given**, **When** y **Then**.

Para acceder a los métodos **step** de la clase **SerenityLoginSteps** solo debes usar la variable de la clase seguido de un punto para acceder a todos los steps disponibles de la clase.

Nota: para algunas versiones de Cucumber, las variables pueden cambiar, en este caso particular la línea 28 tiene esta particularidad **(.*)**, esto es una variable y para que sea identificada como una el paso **@Then**(“**^**will validate the text on screen **(.*)$**”) tiene después de la comilla doble inicial el símbolo **^** y antes de la comilla doble final tiene el símbolo **$**.

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Steps

En la carpeta **steps** se van a incluir las clases de Java que nos permitan crear métodos **Steps** y estos serán llamados desde las clases **StepDefinitions** que vamos a crear más adelante.

en el caso de **POM** encima de cada método se debe dejar la anotación **@Step** ya que esto se va a incluir como la casuística del reporte. Además estos métodos van a contener líneas de código puro de la librería **Selenium** y aparte se debe incluir la instancia de un objeto de la clase PageObject que contiene los selectores que pertenecen a la interacción en cuestión.

SerenityLoginSteps.java

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

import ...

5 usages  Michael Ferneein Garzon Rodriguez *
public class SerenityLoginSteps {
    6 usages
    SerenityLoginPage loginPage = new SerenityLoginPage();

    1 usage  Michael Ferneein Garzon Rodriguez
    @Step
    public void OpenTheWebSite(String url) {
        getDriver().get(url);
    }

    1 usage  Michael Ferneein Garzon Rodriguez *
    @Step
    public void login(UserData userData) {
        loginPage.getDriver().findElement(SerenityLoginPage.getTxtUser()).clear();
        loginPage.getDriver().findElement(SerenityLoginPage.getTxtPass()).clear();

        loginPage.getDriver().findElement(SerenityLoginPage.getTxtUser()).sendKeys(userData.getUser());
        loginPage.getDriver().findElement(SerenityLoginPage.getTxtPass()).sendKeys(userData.getPass());

        loginPage.getDriver().findElement(SerenityLoginPage.getBtnSubmit()).click();
    }

    1 usage  Michael Ferneein Garzon Rodriguez
    @Step
    public void validateTextOnScreen(String text) {
        assert loginPage.getDriver().findElement(SerenityDashBoardPage.getTxtValidation()).getText().contains(text);
    }
}
```

Referencias:
https://www.programiz.com/java-programming - Aprende Java
https://www.selenium.dev/documentation/webdriver/ - Aprende a usar WebDriver de Selenium
https://www.browserstack.com/guide/verify-and-assert-in-selenium - Aprende a validar con Assert

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

PageObjects

En la carpeta **pageobjects** 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 o el de su preferencia y sus respectivos métodos Getter para acceder a estos selectores desde otras clases.

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

private static final By ELEMENTO_WEB_NOMBRE = By.xpath("//elemento[@atributo='valor']")

SerenityLoginPage.java

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

import ...

8 usages  Michael Ferneein Garzon Rodriguez *
public class SerenityLoginPage extends PageObject {

    1 usage
    private static final By TXT_USER = By.xpath( xpathExpression: "//input[@id='LoginPanel0_Username']");
    1 usage
    private static final By TXT_PASS = By.xpath( xpathExpression: "//input[@id='LoginPanel0_Password']");
    1 usage
    private static final By BTN_SUBMIT = By.xpath( xpathExpression: "//button[@id='LoginPanel0_LoginButton']");


    2 usages  Michael Ferneein Garzon Rodriguez
    public static By getTxtUser() { return TXT_USER; }

    2 usages  Michael Ferneein Garzon Rodriguez
    public static By getTxtPass() { return TXT_PASS; }

    1 usage  Michael Ferneein Garzon Rodriguez
    public static By getBtnSubmit() { return BTN_SUBMIT; }
}
```

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.

Variables ***user*** y ***pass*** las cuales tienen métodos Getter 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.

UserData.java

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

import ...

7 usages  Michael Ferneein Garzon Rodriguez *
public class UserData {

    1 usage
    String user;
    1 usage
    String pass;

    1 usage  Michael Ferneein Garzon Rodriguez *
    public static List<UserData> setData(DataTable table){
        List<UserData> data = new ArrayList<>();
        List<Map<String, String>> info = table.asMaps();
        for (Map<String,String> pointer : info) {
            data.add(new ObjectMapper().convertValue(pointer, UserData.class));
        }
        return data;
    }
    1 usage  Michael Ferneein Garzon Rodriguez
    public String getUser() { return user; }

    1 usage  Michael Ferneein Garzon Rodriguez
    public String getPass() { return pass; }
}
```

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Utils

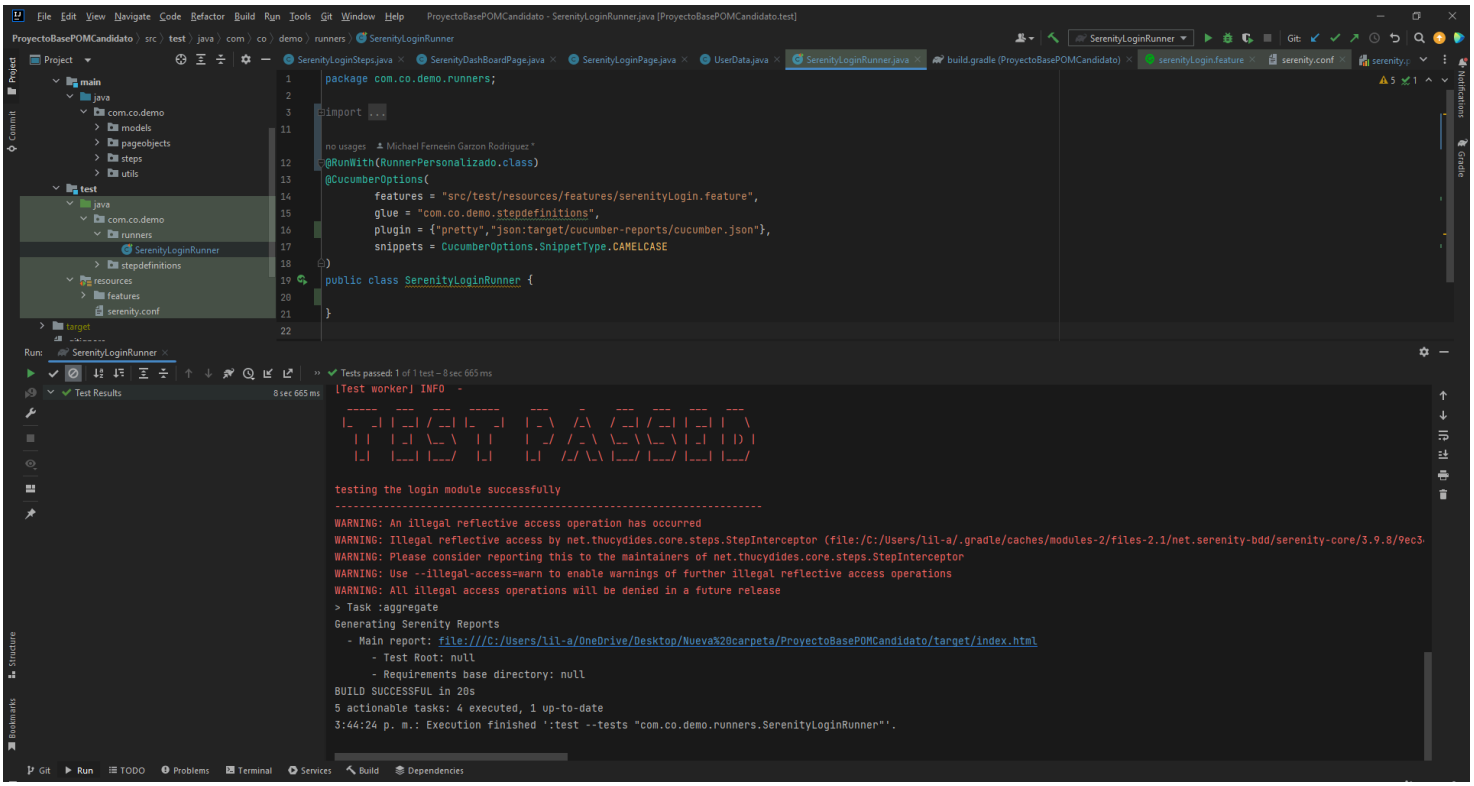
En la carpeta utils se van a incluir las clases de utilidad que nos permitan ejecutar ciertas acciones en nuestra prueba como por ejemplo. Una clase que nos ayude a traer datos desde un excel, una clase que nos genere un número entero aleatorio, una clase que nos permita almacenar constantes que van a usarse en el proyecto. ejemplo:

GlobalData.java

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

import net.thucydides.core.util.EnvironmentVariables;

2 usages  Michael Ferneein Garzon Rodriguez *
public class GlobalData {
    no usages
    public static EnvironmentVariables environmentVariables;
    1 usage
    public static final String baseUrl = "https://demo.serenity.is/Account/Login/?ReturnUrl=%2F";

}
```

Nota: una vez la prueba haya finalizado se puede acceder a los reportes dando click al enlace que nos muestra en la consola o se puede dirigir a la carpeta target y allí ubicar el archivo index.html

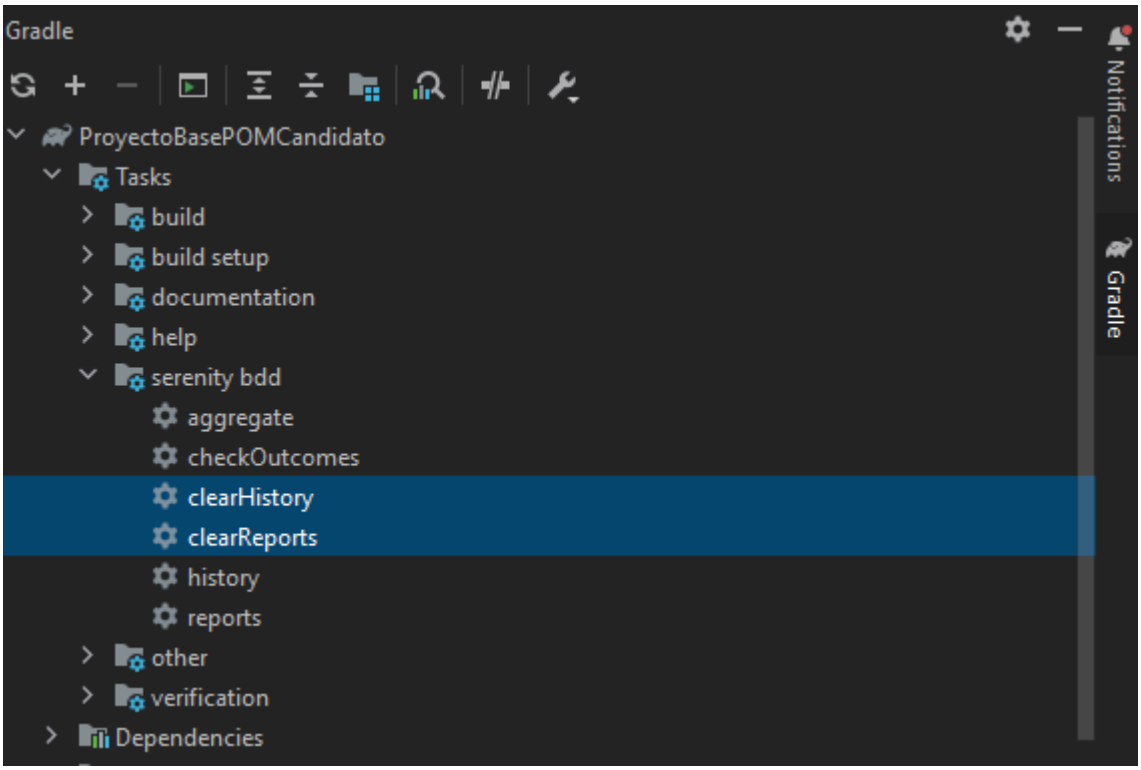
file:///C:/Users/usuario/ruta/proyecto/ProyectoBasePOMCandidato/target/index.html



GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

Este sería el reporte. al haber varias ejecuciones nos mostrará gráficas de acuerdo a 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 clickear las dos opciones resaltadas y es todo!

GUIA PARA ELABORAR UNA AUTOMATIZACIÓN WEB

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