

# FRAMEWORK SERENITY BDD

# CONTENIDO

- ¿Qué es el mapeo de elementos Web?

---
- ¿Qué es Serenity BDD?

---
- Escribir pruebas con lenguaje Gherkin

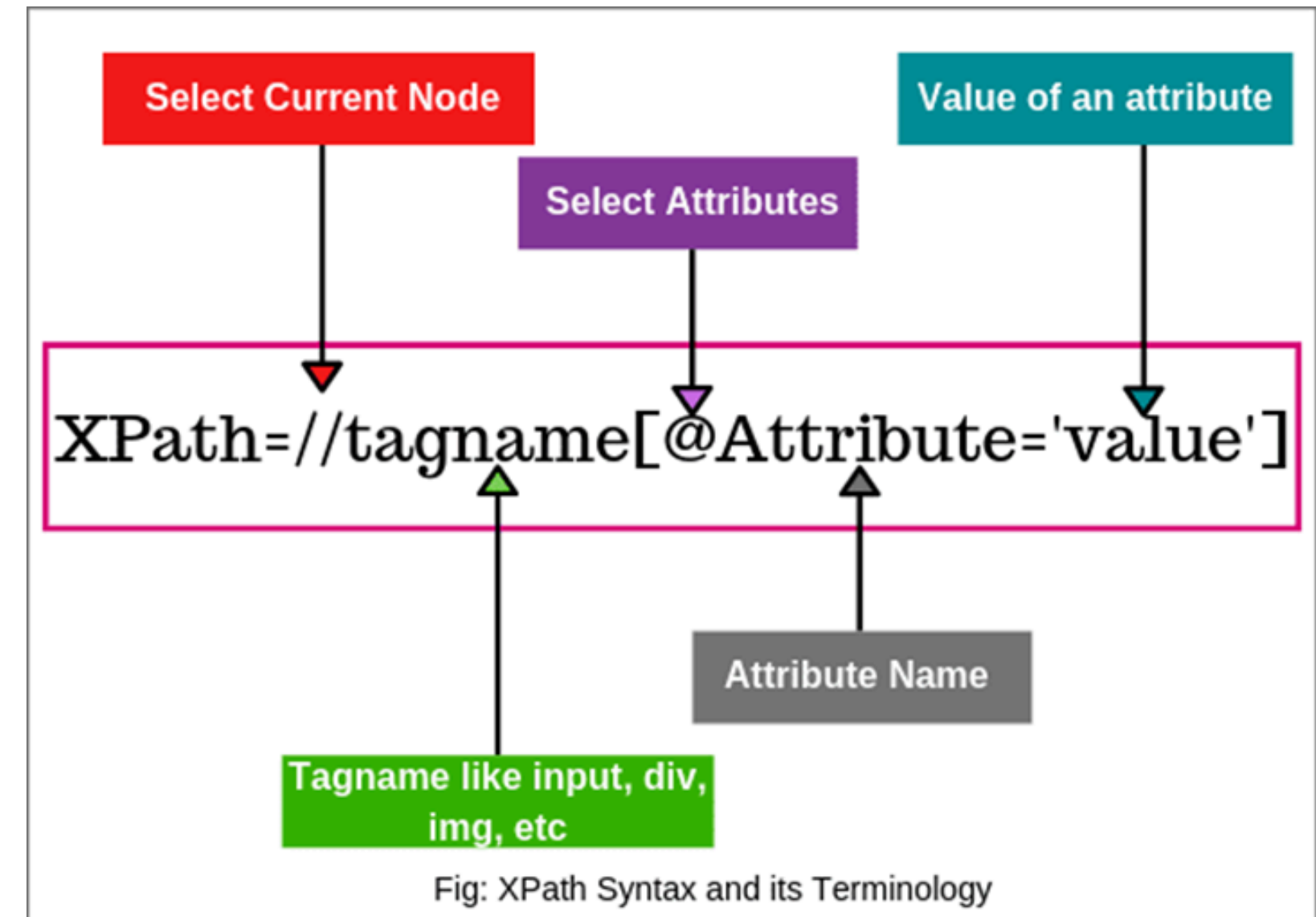
---
- Estructura del proyecto con patrón Screenplay

---
- Instalación y configuración del entorno

---

# ¿QUÉ ES EL MAPEO DE ELEMENTOS WEB?

- El mapeo de elementos es el proceso de localizar estos elementos para que puedan ser manipulados por un programa
- XPath es un lenguaje que se usa para localizar elementos en el código HTML de una página web.



# TIPOS DE XPATH

## ESTÁTICOS Y DINÁMICOS

### XPATH ESTÁTICOS

- Es una ruta fija que apunta directamente a un elemento web

Ejemplo: `/html/body/div[2]/div[1]/form/input[1]`

### CARACTERÍSTICAS

- No cambia mientras la estructura del sitio no cambie.
- Muy preciso, pero frágil si se modifica el diseño de la página.
- Útil para sitios con estructura estable.

# TIPOS DE XPATH

## XPATH DINÁMICO

- Es una ruta que utiliza expresiones para adaptarse a elementos que pueden cambiar de posición o atributos.

Ejemplo: `//input[@name='username']`

## CARACTERÍSTICAS

.

- Más flexible y resistente a cambio en la estructura de la pagina.
- Se enfoca en atributos como id,name,class,text(),etc.
- Ideal para paginas web dinámicas o con elementos generados por javascript

## ¿QUÉ ES SERENITY BDD?

- Serenity BDD es una herramienta que ayuda a automatizar pruebas de software de forma comprensible para todos.
- Su objetivo es conectar a los programadores con los usuarios del negocio a través de pruebas fáciles de leer.

## ¿PARA QUÉ SIRVE?

- Ayuda a verificar que el software funciona como se espera.
- Permite escribir escenarios de prueba en lenguaje natural.
- Genera reportes detallados sobre cómo se comporta el sistema.

## ¿QUÉ HERRAMIENTAS USA SERENITY BDD?

- Cucumber: permite escribir pruebas en lenguaje natural.
- JUnit o Selenium: se conectan para ejecutar las pruebas reales.
- Reportes automáticos: muestra qué pruebas pasaron y cuáles fallaron.

## ¿QUIÉN PUEDE BENEFICIARSE DE SERENITY BDD?

- Clientes o usuarios del negocio: porque pueden entender y validar pruebas.
- Analistas funcionales: pueden escribir pruebas sin programar.
- Desarrolladores: implementan la lógica técnica detrás de las pruebas.
- Testers (QA): automatizan pruebas sin necesidad de código complejo.

## VENTAJAS PRINCIPALES

- Comunicación clara entre todos los equipos.
- Documentación automática de requisitos.
- Detección rápida de errores.
- Pruebas reutilizables y mantenibles.

# INTERACCIONES COMUNES EN SERENITY BDD

## 1. `Open.url("...")`

Qué hace: Abre una URL en el navegador.

**Ejemplo:** `Open.url("https://example.com")`

## 2. `Click.on(Target)`

Qué hace: Hace clic sobre un elemento.

**Ejemplo:** `Click.on(LoginPage.LOGIN_BUTTON)`

## 3. `Enter.theValue("...").into(Target)`

Qué hace: Escribe un valor en un campo de entrada.

**Ejemplo:** `Enter.theValue("usuario").into(LoginPage.USERNAME_FIELD)`

## 4. `SelectFromOptions.byVisibleText("...").from(Target)`

Qué hace: Selecciona una opción de un menú desplegable por el texto visible.

**Ejemplo:**

`SelectFromOptions.byVisibleText("Colombia").from(FormPage.COUNTRY_DROPDOWN)`



## 5. MoveMouse.to(Target)

Qué hace: Mueve el mouse sobre un elemento (útil para hover).

**Ejemplo:** MoveMouse.to(Menu.MY\_ACCOUNT)

## 6. WaitUntil.the(Target, Condition).forNoMoreThan(Duration)

Qué hace: Espera a que un elemento cumpla una condición (visible, habilitado, etc.).

**Ejemplo:** WaitUntil.the(LoginPage.LOGIN\_BUTTON, isVisible()).forNoMoreThan(10).seconds()

## 7. Scroll.to(Target)

Qué hace: Desplaza la vista hasta un elemento.

**Ejemplo:** Scroll.to(CheckoutPage.CONFIRM\_BUTTON)

## 8. Hit.the(Keys.ENTER).into(Target)

Qué hace: Simula pulsar una tecla (como ENTER, TAB, etc.) dentro de un campo.

**Ejemplo:** Hit.the(Keys.ENTER).into(SearchBox.FIELD)

## 9. DoubleClick.on(Target)

Qué hace: Hace doble clic en un elemento.

**Ejemplo:** DoubleClick.on(ItemsList.FIRST\_ITEM)

## 10. JavaScriptClick.on(Target)

Qué hace: Hace clic usando JavaScript (útil cuando Click.on() falla por problemas de visibilidad).

**Ejemplo:** JavaScriptClick.on(SpecialButton.NO\_DOM\_BUTTON)

## 11. Clear.field(Target)

Qué hace: Limpia el contenido de un campo de texto.

**Ejemplo:** Clear.field(FormPage.EMAIL\_FIELD)

# ESCRIBIR PRUEBAS CON LENGUAJE GHERKIN

## ¿QUÉ ES EL LENGUAJE GHERKIN?

Gherkin es un lenguaje muy sencillo que sirve para escribir casos de prueba o escenarios de comportamiento de un sistema o aplicación.

Se usa mucho para que tanto programadores como personas no técnicas puedan entender y colaborar en cómo debe funcionar un sistema.

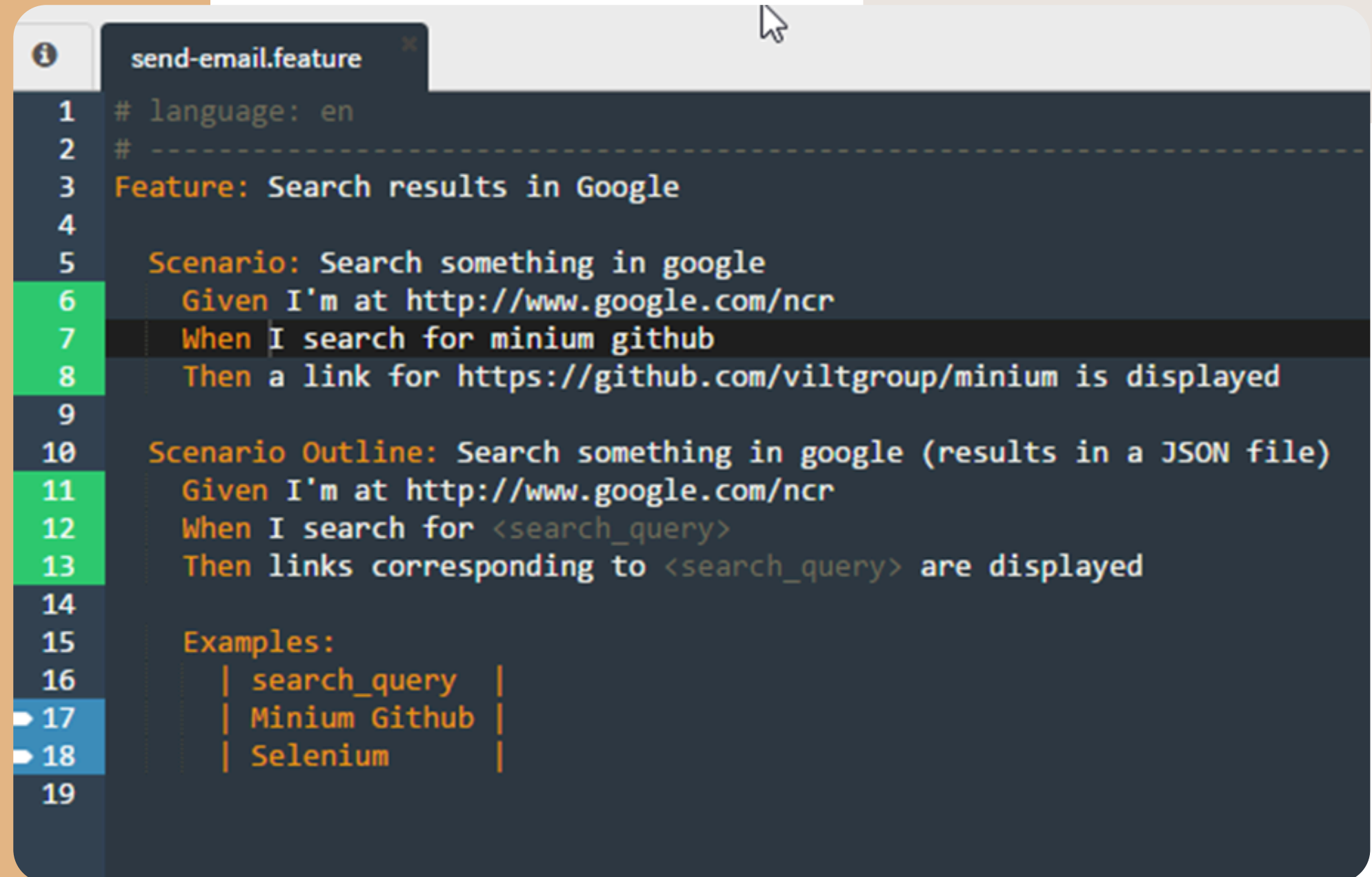
## ¿PARA QUÉ SIRVE?

1. Especificar cómo debe comportarse un sistema, en lenguaje claro.
2. Asegurarse de que todo lo que se construye cumpla con lo que se necesita.
3. Ayudar a que todos (clientes, testers, desarrolladores) hablen el mismo idioma.

# ¿CÓMO SE VE GHERKIN?

SIEMPRE USA PALABRAS CLAVE COMO:

- Feature: qué funcionalidad se está describiendo.
- Scenario: un caso concreto de prueba.
- Given (Dado): el contexto inicial.
- When (Cuando): una acción.
- And (Y) adicionar una acción
- Then (Entonces): el resultado esperado.



The screenshot shows a code editor with a file named 'send-email.feature'. The code is written in Gherkin syntax and includes a feature description, two scenarios with steps, and an examples table. Line numbers 1 through 19 are visible on the left side of the editor.

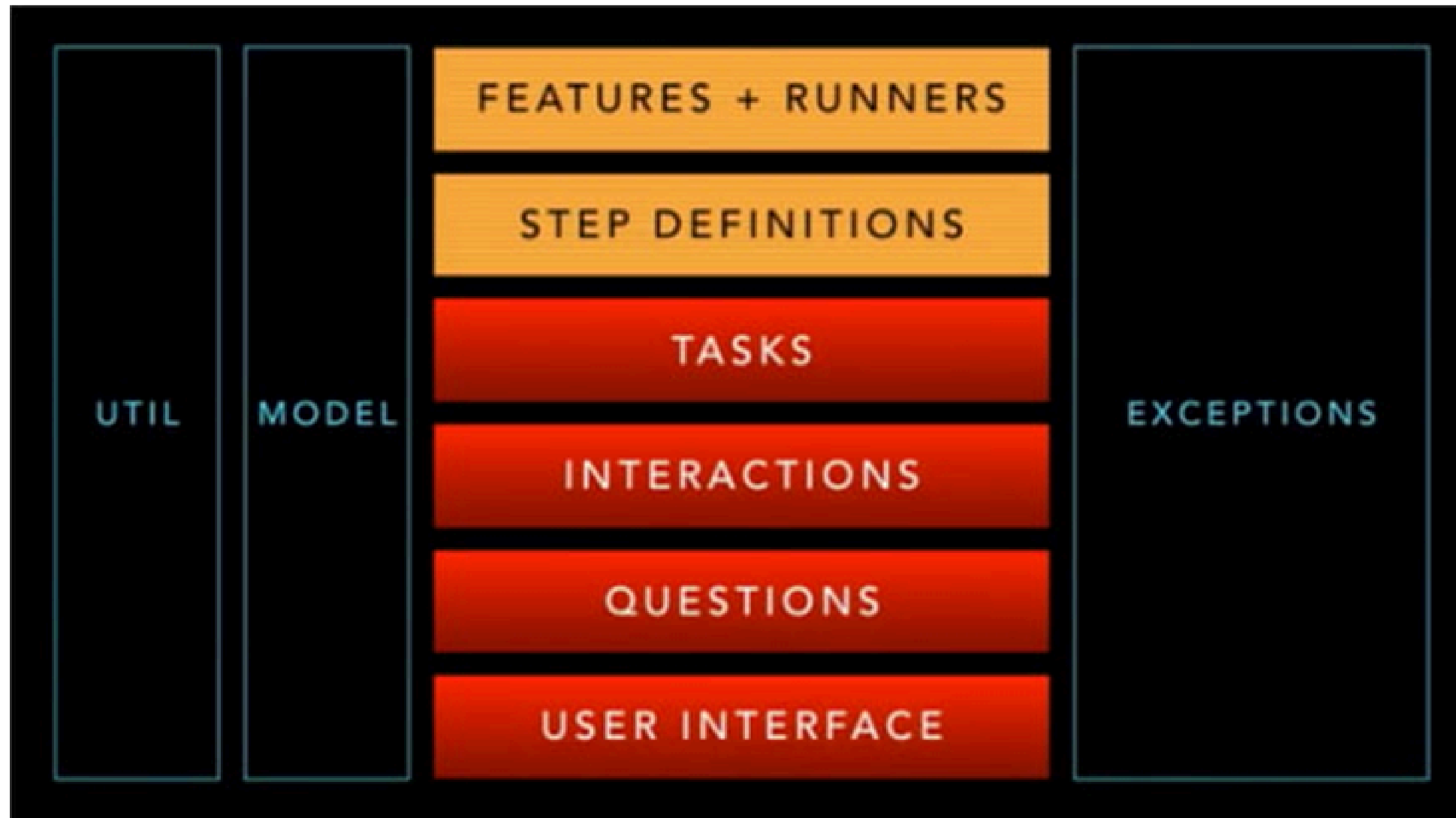
```
1 # language: en
2 # -----
3 Feature: Search results in Google
4
5 Scenario: Search something in google
6   Given I'm at http://www.google.com/ncr
7   When I search for minium github
8   Then a link for https://github.com/viltgroup/minium is displayed
9
10 Scenario Outline: Search something in google (results in a JSON file)
11   Given I'm at http://www.google.com/ncr
12   When I search for <search_query>
13   Then links corresponding to <search_query> are displayed
14
15 Examples:
16   | search_query |
17   | Minium Github |
18   | Selenium |
19
```

# ESTRUCTURA DEL PROYECTO CON PATRÓN SCREENPLAY



Screenplay es un patrón de diseño de software enfocado a pruebas. Define conceptos como actor, habilidades, preguntas, tareas e interacciones. A través del uso de estos conceptos podemos llevar a cabo la modelación de los diferentes componentes que vamos a usar en nuestros tests, permitiendo definir código que favorece el principio de responsabilidad única en el software.

# ARQUITECTURA DE SCREENPLAY



# FEATURES

En esta capa se escriben los archivos .feature, estos describen las características y escenarios a probar haciendo uso del lenguaje Gherkin, además se pueden entender como las historias de usuario se van a desarrollar.

Ejemplo:

```
1  >>  Feature: Prueba Login
2
3  >>  Scenario: Prueba login Datos correctos
4      Given Que ingreso a la pagina de prueba
5      When Diligencio el usuario "usuario" y password "password"
6      Then inicio sesion de manera exitosa
7
```

# RUNNERS

Contiene los archivos runner encargados de ejecutar los escenarios de los feature. Se usan las anotaciones `@RunWith()` `@CucumberOptions()` para especificar qué ejecutar y cómo hacerlo.

## Ejemplo

```
package runners;

import io.cucumber.junit.CucumberOptions;
import net.serenitybdd.cucumber.CucumberWithSerenity;
import org.junit.runner.RunWith;

@RunWith(CucumberWithSerenity.class)
@CucumberOptions(
    features = "src/test/resources/features/Login.feature",
    glue = "stepDefinitions",
    //tags = "",
    snippets = CucumberOptions.SnippetType.CAMELCASE
)

public class runnerLogin {
}
```



# STEP DEFINITION

Gestiona los fragmentos de código definidos en el Given, Then, When de los escenarios de los archivos .feature. Los fragmentos de código son conocidos como Code Snippets los cuales son métodos que mapean las instrucciones de los escenarios.

- Ejemplo

```
public class loginStepDefinition
{
    @Before
    public void setup() { setTheStage(new OnlineCast()); }
    1 usage
    @Given("Que ingreso a la pagina de prueba")
    public void queIngresoALaPaginaDePrueba() {

        theActorCalled( requiredActor: "usuario").wasAbleTo(Open.browserOn(new InterfazLogin()));
    }
    1 usage
    @When("Diligencio el usuario {string} y password {string}")
    public void diligencioElUsuarioYPassword(String usuario, String password) {
        theActorInTheSpotlight().attemptsTo(iniciarSesion.credenciales(usuario,password));
    }
    1 usage
    @Then("inicio sesion de manera exitosa")
    public void inicioSesionDeManeraExitosa() {
        theActorInTheSpotlight().should(seeThat(validarInicioSesionExitoso.prueba(),equalTo( operand: true)));
    }
}
```

# TASKS

Esta paquete contiene las clases que son el corazón del patrón de screenplay=Las tareas.

Las tareas son acciones de alto nivel que definen un conjunto de interacciones en el lenguaje de negocio.

- Ejemplo

```
4 usages
public class iniciarSesion implements Task {

    2 usages
    public String usuario;
    2 usages
    public String password;

    public iniciarSesion (String usuario, String password){
        this.usuario=usuario;
        this.password=password;
    }

    @Override
    public <T extends Actor> void performAs(T actor) {
        actor.attemptsTo(Enter.theValue(usuario).into(INPUT_USUARIO),
            Enter.theValue(password).into(INPUT_PASSWORD),
            Click.on(BUTTON_LOGIN));
    }

    1 usage
    public static iniciarSesion credenciales (String usuario, String password){
        return Tasks.instrumented(iniciarSesion.class,usuario,password);
    }
}
```

## INTERACTIONS

Se encargan de las interacciones con la interfaz de usuario. Muchas interacciones ya fueron desarrolladas y se encuentran disponibles gracias a las librerías de screenplay y de serenity BDD. Ejemplo: Open, Click, Enter, Hit, SelectFromOptions.

## QUESTIONS

Esta capa gestiona los Asserts o verificaciones de las pruebas las cuales son el fin último de las mismas, ya que con estas capturamos la información de la interfaz que puede ser usada. Text, Value, Visibility.

## USER INTERFACE

Almacena las clases en las cuales se mapean los elementos de la interfaz de usuario, inputs o botones.

# CAPAS TRANSVERSALES

## MODELS

- Los objetos de negocio son una abstracción de un ente real o virtual, modelado mediante atributos y métodos.

## UTILS

- Contiene clases con los métodos unitarios que pueden ser utilizados libremente por otras clases del proyecto como: Conversores, Waits, verificadores.

## EXCEPTIONS

- Esta es una capa muy importante que tiene como objetivo dar manejo a los errores que pueden presentarse en la aplicación. Es una capa sensible al negocio y particular a cada proyecto.

# BIBLIOGRAFÍA

## Fuentes web:

Pragma. (s.f.). ¿Cómo empezar con Serenity BDD Screenplay? Pragma. de

<https://www.pragma.com.co/academia/lecciones/como-empezar-con-serenity-bdd-screenplay>

Pragma. (s.f.). Conceptos básicos para una ejecución en Screenplay. Pragma. de

<https://www.pragma.com.co/academia/lecciones/conceptos-basicos-para-una-ejecucion-en-screenplay>

## Redes sociales:

Mesa, J. [Julian Mesa Consultant]. Screenplay es un patrón de diseño de software enfocado a pruebas. Define conceptos claros, reutilizables y mantenibles para automatizar pruebas. [Publicación en Facebook].

<https://www.facebook.com/julianmesaconsultant/posts/screenplay-es-un-patr%C3%B3n-de-dise%C3%B1o-de-software-enfocado-a-pruebas-define-concepto/2953321321554095/>

## Canal de YouTube:

Mesa, J. [@JulianMesaAutomation]. (s.f.). Julian Mesa Automation [Canal de YouTube]. YouTube. de

<https://www.youtube.com/@JulianMesaAutomation>