

## ScreenPlay+BDD+Services – Rest & Soap

### Primeros Pasos

(Tiempo ejecución 4 horas)



Sea Bienvenido a la guía de automatización para consumo de Servicios Rest y/o SOAP. Vamos a iniciar con el proceso de creación de pruebas automatizadas para servicios con el marco Screenplay + BDD.

*Screenplay + BDD* nos permite crear pruebas que necesiten un consumo de servicios (Rest / SOAP), con todas las ventajas del patrón. Screenplay integra [Rest Assured](#) el cual nos permite realizar el consumo de los servicios, aunque según la [documentación oficial de Screenplay](#) solo se habla de servicios Rest, en la práctica es perfectamente posible realizar el consumo de servicios SOAP.

### Tabla de contenido

Prueba de REST con Serenity BDD .....	2
Prueba de SOAP con Serenity BDD .....	13



## ScreenPlay+BDD+Services – Rest & Soap

### Prueba de REST con Serenity BDD

Para nuestra automatización REST haremos uso del siguiente enlace <https://regres.in/> en el cual se realizará el consumo de un servicio para crear un usuario. A través de éste se realizara una solicitud y obtendremos una respuesta.

Request <i>/api/users</i>	Response <i>201</i>
<pre>{   "name": "morpheus",   "job": "leader" }</pre>	<pre>{   "name": "morpheus",   "job": "leader",   "id": "513",   "createdAt": "2020-02-23T15:58:02.529Z" }</pre>

Para nuestro ejercicio enviaremos el Request(solicitud)

```
{
  "name": "brandon",
  "job": "java developer"
}
```

Iniciaremos con la creación del proyecto, para esto usando el arquetipo de screenplay que lo podemos trabajar usando la Guía “**Serenity ScreenPlay 1.1 - Crear Proyecto Base en IntelliJ**”.

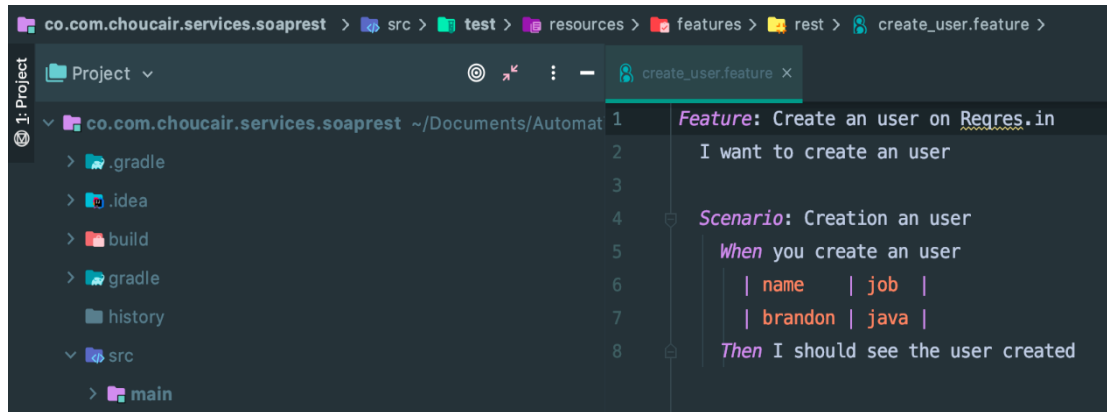
Antes de continuar, se debe agregar la dependencia al archivo **build.gradle** para poder realizar el consumo de algún servicio desde nuestra automatización:

```
implementation "net.serenity-bdd:serenity-screenplay-rest:1.9.49"
```



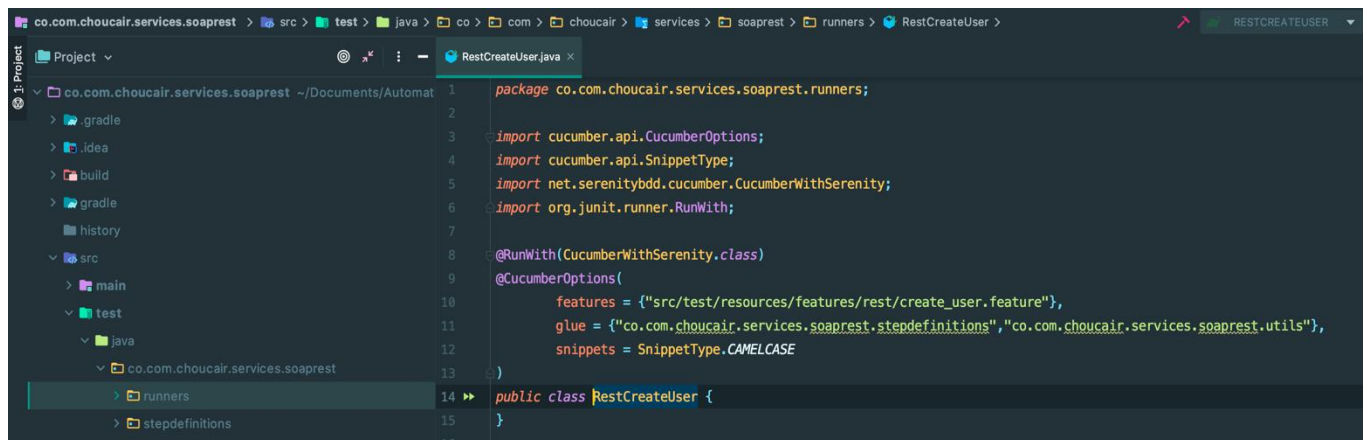
## ScreenPlay+BDD+Services – Rest & Soap

Continuaremos con la creación de nuestra HU (historia de usuario) en cucumber.



```
co.com.choucair.services.soaprest > src > test > resources > features > rest > create_user.feature >
Project
co.com.choucair.services.soaprest ~/Documents/Automat
> .gradle
> .idea
> build
> gradle
  history
  src
    main
1 Feature: Create an user on Regres.in
2 I want to create an user
3
4 Scenario: Creation an user
5 When you create an user
6   | name | job |
7   | brandon | java |
8 Then I should see the user created
```

Seguido crearemos nuestra clase runner la cual llamaremos **RestCreateUser**, en dicha clase se usará la anotación **@CucumberOptions** con los atributos (*features, glue y snippets*)



```
co.com.choucair.services.soaprest > src > test > java > co > com > choucair > services > soaprest > runners > RestCreateUser >
Project
co.com.choucair.services.soaprest ~/Documents/Automat
> .gradle
> .idea
> build
> gradle
  history
  src
    main
    test
      java
        co.com.choucair.services.soaprest
          runners
          stepdefinitions
1 package co.com.choucair.services.soaprest.runners;
2
3 import cucumber.api.CucumberOptions;
4 import cucumber.api.SnippetType;
5 import net.serenitybdd.cucumber.CucumberWithSerenity;
6 import org.junit.runner.RunWith;
7
8 @RunWith(CucumberWithSerenity.class)
9 @CucumberOptions(
10     features = {"src/test/resources/features/rest/create_user.feature"},
11     glue = {"co.com.choucair.services.soaprest.stepdefinitions","co.com.choucair.services.soaprest.utils"},
12     snippets = SnippetType.CAMEL_CASE
13 )
14 public class RestCreateUser {
15 }
16
```



## ScreenPlay+BDD+Services – Rest & Soap

Además, ejecutaremos nuestro runner para generar los métodos propuestos a implementar en los pasos

```
Ex Tests ignored: 1 of 1 test – 71 ms

1 Scenarios (1 undefined)
2 Steps (2 undefined)
0m0.335s

You can implement missing steps with the snippets below:

@When("^you create an user$")
public void youCreateAnUser(DataTable arg1) {
    // Write code here that turns the phrase above into concrete actions
    // For automatic transformation, change DataTable to one of
    // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.
    // E,K,V must be a scalar (String, Integer, Date, enum etc).
    // Field names for YourType must match the column names in
    // your feature file (except for spaces and capitalization).
    throw new PendingException();
}

@Then("^I should see the user created$")
public void iShouldSeeTheUserCreated() {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Los mismos se llevaran para implementarlos en nuestra clase stepdefinition la cual llamaremos **“StepDefinitionRestCreateUser”**

```
public class StepDefinitionRestCreateUser {
    @When("^you create an user$")
    public void youCreateAnUser(DataTable arg1) {
        // Write code here that turns the phrase above into concrete actions
        // For automatic transformation, change DataTable to one of
        // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.
        // E,K,V must be a scalar (String, Integer, Date, enum etc).
        // Field names for YourType must match the column names in
        // your feature file (except for spaces and capitalization).
        throw new PendingException();
    }

    @Then("^I should see the user created$")
    public void iShouldSeeTheUserCreated() {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }
}
```



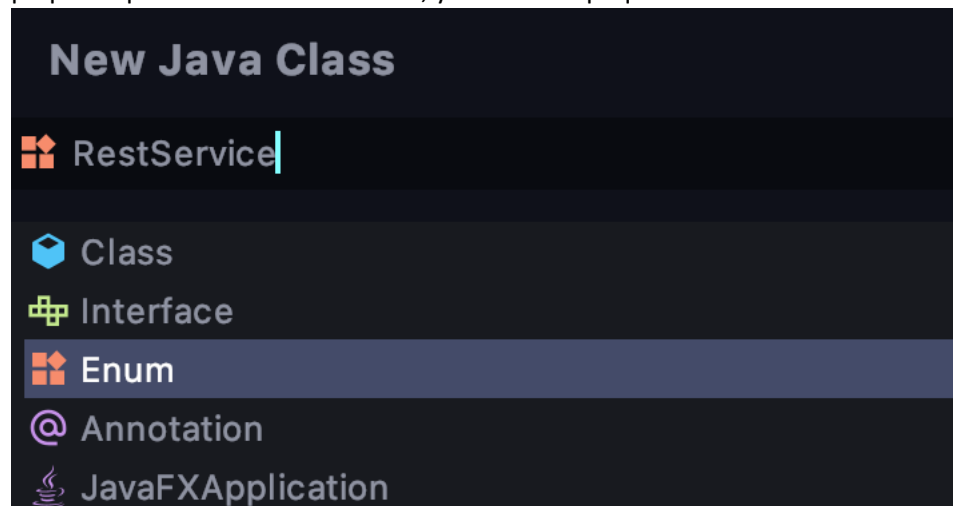
## ScreenPlay+BDD+Services – Rest & Soap

Como siempre es necesario prepara un escenario y crear un actor. Para ello vamos a hacer uso del paquete “**utils**” donde crearemos una clase llamada “**BeforeHook**” con el método “**prepareStage()**”. Hacemos uso de la anotación “**@Before**” para que nuestra ejecución inicie desde este punto.

```
@Before
public void prepareStage(){
    OnStage.setTheStage(new OnlineCast());
    theActorCalled("brandon").whoCan(CallAnApi.at(BASE_URL.toString()));
}
```

Aquí hacemos uso de “**OnStage**” para preparar el escenario. Además, vamos a declarar el actor y haremos uso de la habilidad “**whoCan**” a la cual le pasaremos la clase “**CallAnApi**” y el método “**at**” y al método le asignaremos la URL base.

**Nota:** la “**BASE\_URL**” la crearemos en un **enum**. Para implementarlo dentro del paquete “**utils**” crearemos otro paquete que llamaremos “**enums**”, y dentro del paquete crearemos una clase llamada “**RestService**”



## ScreenPlay+BDD+Services – Rest & Soap

Muchos se preguntarán por qué un **enum** y por qué no una clase que contenga constantes, y es porque los enum son algo más que constantes, la clase del tipo *enum* puede definir métodos y otras propiedades.

Nuestra clase debe quedar así:

```
package co.com.choucair.services.soaprest.utils.enums;

public enum RestService {

    BASE_URL("https://reqres.in"),
    CREATE_USER("/api/users");

    private String uri;

    RestService(String uri) {
        this.uri = uri;
    }

    @Override
    public String toString() {
        return uri;
    }
}
```

Ahora volveremos a nuestra clase **StepDefinitions** donde implementaremos los métodos planteados por cucumber.

```
public class StepDefinitionRestCreateUser {

    @When("^you create an user$")
    public void youCreateAnUser(DataTable arg1) {
        // Write code here that turns the phrase above into concrete actions
        // For automatic transformation, change DataTable to one of
        // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.
        // E,K,V must be a scalar (String, Integer, Date, enum etc).
        // Field names for YourType must match the column names in
        // your feature file (except for spaces and capitalization).
        throw new PendingException();
    }

    @Then("^I should see the user created$")
    public void iShouldSeeTheUserCreated() {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }
}
```



## ScreenPlay+BDD+Services – Rest & Soap

Limpiaremos los métodos “**public void**” de tal forma que no queden comentarios y excepciones, además iniciaremos con la implementación de nuestro método

```
@When("^you create an user$")
public void youCreateAnUser(List<ModelCreateUserRest> modelCreateUserRests) {
}
```

Dicho método es modificado, primero el dato de entrada dejara de ser “**DataTable**” y lo cambiaremos por un tipo “**List**” para crear un modelo de datos.

Ahora vamos a crear una clase llamada “**ModelCreateUserRest**” en el paquete “**models**”, dicha clase debe quedar así:

```
public class ModelCreateUserRest {
    private String name;
    private String job;

    public ModelCreateUserRest(String name, String job) {
        this.name = name;
        this.job = job;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getJob() {
        return job;
    }

    public void setJob(String job) {
        this.job = job;
    }

    @Override
    public String toString() {
        return "{" +
            "\"name\": " + "\"" + name + "\"" +
            ", \"job\": " + "\"" + job + "\"" +
            "}";
    }
}
```

## ScreenPlay+BDD+Services – Rest & Soap

En esta clase se declaran dos variables (name, job) a las cuales se les crea su respectivo get y set como su constructor, además creamos un método **“toString()”** el cual tendrá el JSON con el cual vamos a hacer la petición.

**Nota:** importante validar el formato correcto del [JSON](#).

Volvemos nuevamente a nuestra clase **“StepDefinitionRestCreateUser”** y agregamos la siguiente línea.

```
@When("^you create an user$")
public void youCreateAnUser(List<ModelCreateUserRest> modelCreateUserRests) {
    theActorInTheSpotlight().attemptsTo(CreateUserRest.with(modelCreateUserRests));
}
```

Dentro del paquete **“Task”** creamos una clase a la que llamaremos **“CreateUserRest”** y un método llamado **“with()”**. Nuestra clase debería ir así:

```
public class CreateUserRest implements Task {
    private List<ModelCreateUserRest> modelCreateUserRest;

    public CreateUserRest(List<ModelCreateUserRest> modelCreateUserRest) {
        this.modelCreateUserRest = modelCreateUserRest;
    }

    public static CreateUserRest with(List<ModelCreateUserRest> modelCreateUserRests) {
        return Tasks.instrumented(CreateUserRest.class, modelCreateUserRests);
    }

    @Override
    public <T extends Actor> void performAs(T actor) {

    }
}
```

Continuamos haciendo uso del actor con la habilidad de intentar **“attemptsTo”**.

```
@Override
public <T extends Actor> void performAs(T actor) {
    actor.attemptsTo(ConsumeService.withPost(modelCreateUserRest.get(0).toString()));
}
```

**Nota:** Recordemos que una **interacción** es una actividad de bajo nivel que ejerce directamente el Actor con su capacidad para interactuar con una interfaz externa específica del sistema como lo son: sitio web, una aplicación móvil o un servicio web.



## ScreenPlay+BDD+Services – Rest & Soap

Ahora crearemos una clase en el paquete *"interactions"*. Con la finalidad de tener un código mas limpio y desacoplado, dicha clase la llamaremos *"ConsumeService"* que puede contener (1 ó n) cantidad de métodos.

Para el ejemplo crearemos un método publico estático , donde *"WithPost"* será una clase y *"withPost"* será un método de la misma clase y la cual le pasaremos por parámetro un *String*. El cual nos retornara la clase y el parámetro.

```
public class ConsumeService {  
    private ConsumeService() {  
    }  
    public static WithPost withPost(String body){  
        return Tasks.instrumented(WithPost.class,body);  
    }  
}
```

Como lo habíamos expresado anteriormente es necesario crear la clase *"WithPost"* con el método *"withPost"* en el mismo paquete *"interactions"*. Esta interacción será la encargada de enviar la solicitud *"post"* para crear el usuario.

Nuestra clase *"WithPost"* debería quedar así:

En el *@Override* vamos a hacer uso de la clase *"Post"* para enviar la petición para crear el usuario.

```
package co.com.choucair.services.soaprest.interactions;  
  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Interaction;  
import net.serenitybdd.screenplay.rest.interactions.Post;  
import static co.com.choucair.services.soaprest.utils.enums.RestService.CREATE_USER;  
  
public class WithPost implements Interaction {  
    private String body;  
  
    public WithPost(String body) {  
        this.body = body;  
    }  
  
    @Override  
    public <T extends Actor> void performAs(T actor) {  
  
        actor.attemptsTo(Post.to(CREATE_USER.toString())  
            .with(requestSpecification -> requestSpecification  
                .headers("Content-Type", "application/json")  
                .body(body)));  
    }  
}
```

## ScreenPlay+BDD+Services – Rest & Soap

Antes de dar continuidad con la validación, vamos a crear una clase en el paquete *utils* que llamaremos “*constant*”, aquí se crea una contante que llamaremos “*VALUE*” y que contendrá el código esperado para nuestra validación.

```
package co.com.choucair.services.soaprest.utils;  
  
public class Constant {  
    public static final int VALUE = 201;  
}
```



## ScreenPlay+BDD+Services – Rest & Soap

Ahora si es momento de continuar con la validación, es decir, vamos a realizar la construcción de la questions. Por tal razón vamos a volver a nuestra clase “**StepDefinitionRestCreateUser**” y vamos a hacer la implementación del **@Then**.

```
@Then("^I should see the user created$")
public void iShouldSeeTheUserCreated() {
    theActorInTheSpotlight().should(seeThat(LastResponseStatusCode.is(VALUE)));
}
```

Haciendo uso del actor en la escena, le vamos a dar la habilidad de ver que la ultima respuesta sea “201”. Para ello es necesario crear la clase “**LastResponseStatusCode**” y el método “**is**” en el paquete “**questions**”.

Vamos a implementar la validación como lo hemos aprendido en las guías anteriores (Web,Mobiles,Escritorio).

```
public class LastResponseStatusCode implements Question<Boolean> {
    private int cod;

    public LastResponseStatusCode(int cod) {
        this.cod = cod;
    }

    public static LastResponseStatusCode is(int cod) {
        return new LastResponseStatusCode(cod);
    }

    @Override
    public Boolean answeredBy(Actor actor) {
        int response = lastResponse().statusCode();
        return response == cod;
    }
}
```

En este caso haciendo uso `lastResponse().statusCode()`; podemos obtener la ultima respuesta del codigo de estado que realizo el servicio la cual la comparamos que sea igual a 201.

**¡Hasta aquí termina el ejemplo basico de usa solicitud por rest haciendo uso del patron de Screenplay;**



# ScreenPlay+BDD+Services – Rest & Soap

Después de ejecutar el runner podrás validar el reporte y veras algo así:

The screenshot displays the Serenity BDD report interface. At the top, the navigation bar includes links for 'Overall Test Results', 'Requirements', 'Capabilities', and 'Features'. The main content area shows a test suite 'Create An User On Regres.In' with a 'Cast' button. Below this, a table lists the test steps and their outcomes:

Steps	Outcome	Duration
When you create an user	SUCCESS	4.01s
Brandon create user rest	SUCCESS	3.61s
Brandon with post	SUCCESS	3.59s
brandon executes a POST on the resource /api/users	SUCCESS	3.54s
Then I should see the user created	SUCCESS	0.09s
Then last response status code	SUCCESS	0.01s
	SUCCESS	4.27s

The detailed view of the 'brandon executes a POST on the resource /api/users' step shows the following information:

- REST Query:** POST https://regres.in/api/users
- Request Headers:** Accept=application/json, Content-Type=application/json; charset=UTF-8
- Content Body:** {"name": "brandon", "job": "java"}
- Response Headers:** Date: Tue, 24 Mar 2020 02:59:51 GMT, Content-Type: application/json; charset=utf-8, Content-Length: 81, Connection: keep-alive, Set-Cookie: \_\_cfduid=da7bbe3ecd7a1eee950a3a9e6ce6c4cf5158501f, X-Powered-By: Express, Access-Control-Allow-Origin: \*, Etag: W/"51-kZjp8ISGA6WptiszrjZXiCR4BE", Via: 1.1 vegur, CF-Cache-Status: DYNAMIC, Expect-CT: max-age=604800, report-uri="https://report-uri.cld", Server: cloudflare, CF-RAY: 578d23776f5af25f-B0G
- Response Body:** {"name": "brandon", "job": "java", "id": "644", "createdAt": "2020-03-24T02:59:51.869Z"}
- Response Cookies:** \_\_cfduid=da7bbe3ecd7a1eee950a3a9e6ce6c4cf51585018791; Path=/; I

Serenity BDD version 2.0.90



## ScreenPlay+BDD+Services – Rest & Soap

### Prueba de SOAP con Serenity BDD

Para nuestra automatización SOAP haremos uso del siguiente enlace <http://www.dneonline.com/calculator.asmx> en el cual se hará el consumo del servicio para sumar dos números, donde haremos una solicitud y obtendremos una respuesta.

En la imagen logramos apreciar la solicitud

#### Calculator

Click [here](#) for a complete list of operations.

##### Add

Adds two integers. This is a test Webservice. ©DNE Online

##### Test

The test form is only available for requests from the local machine.

##### SOAP 1.1

The following is a sample SOAP 1.1 request and response. The [placeholders](#) shown need to be replaced with actual values.

```
POST /calculator.asmx HTTP/1.1
Host: www.dneonline.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/Add"
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://tempuri.org/">
      <intA>int</intA>
      <intB>int</intB>
    </Add>
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddResponse xmlns="http://tempuri.org/">
      <AddResult>int</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```

Request

Response



## ScreenPlay+BDD+Services – Rest & Soap

Iniciaremos con la creación de nuestra HU(historia de usuario) con cucumber.

```
add_two_numbers.feature x
1  Feature: Add two number on dneonline service
2      I as a user want to add two numbers
3
4  Scenario: Add two number
5      When you add two number
6          | 2 | 4 |
7      Then I should see the response of the service is 200
```

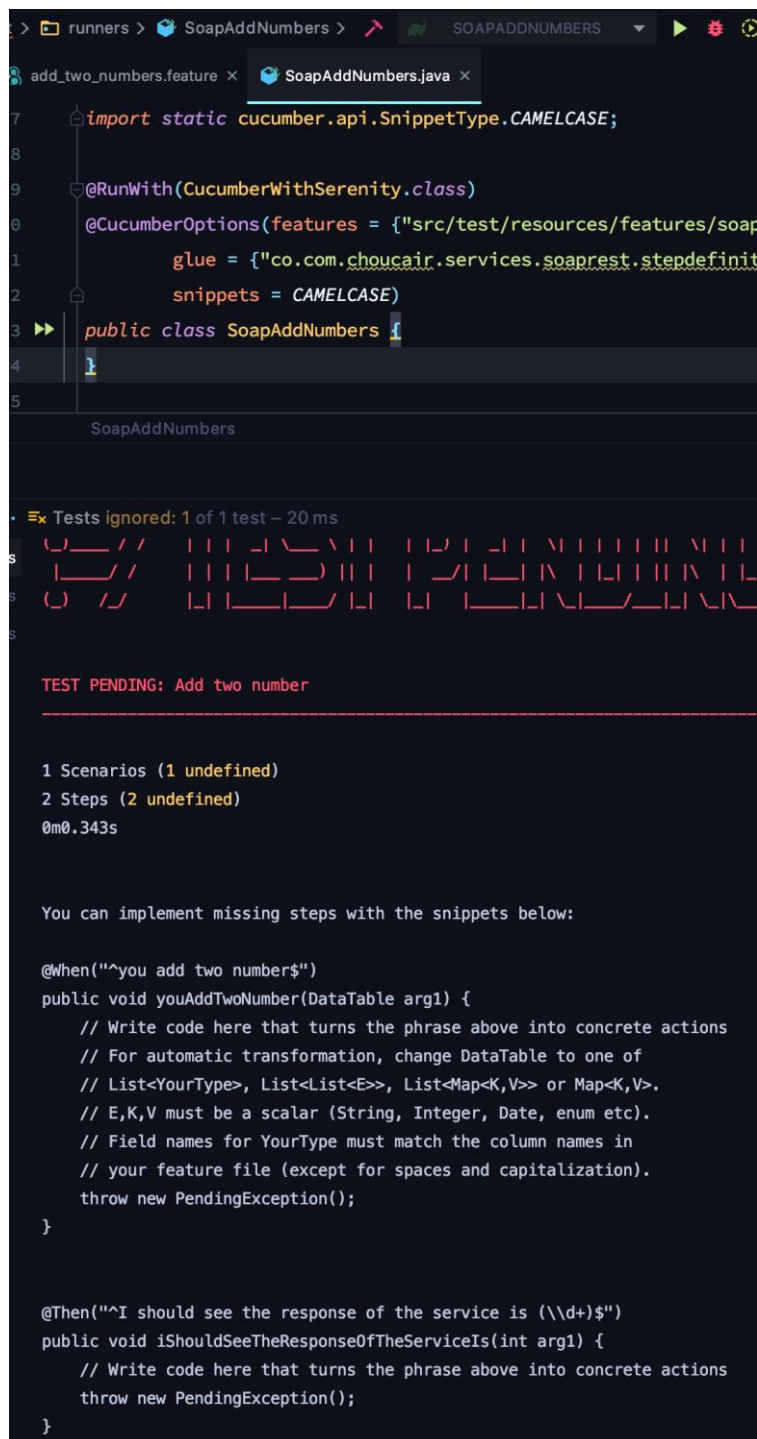
Seguido crearemos nuestra clase runner la cual llamaremos “**SoapAddNumbers**”, en dicha clase se usará la anotación **@CucumberOptions** con los atributos (*features*, *glue* y *snippets*)

```
SoapAddNumbers.java x
1  package co.com.choucair.services.soaprest.runners;
2
3  import cucumber.api.CucumberOptions;
4  import net.serenitybdd.cucumber.CucumberWithSerenity;
5  import org.junit.runner.RunWith;
6
7  import static cucumber.api.SnippetType.CAMELCASE;
8
9  @RunWith(CucumberWithSerenity.class)
10  @CucumberOptions(features = {"src/test/resources/features/soap/add_two_numbers"},
11      glue = {"co.com.choucair.services.soaprest.stepdefinitions",
12          "co.com.choucair.services.soaprest.utils"},
13      snippets = CAMELCASE)
14  public class SoapAddNumbers {
15  }
```

Ejecutaremos nuestro runner para generar los métodos propuestos a implementar en los pasos



## ScreenPlay+BDD+Services – Rest & Soap



```
runners > SoapAddNumbers > SOAPADDDNUMBERS
add_two_numbers.feature x SoapAddNumbers.java x
import static cucumber.api.SnippetType.CAMEL_CASE;
@RunWith(CucumberWithSerenity.class)
@CucumberOptions(features = {"src/test/resources/features/soap"},
    glue = {"co.com.choucair.services.soaprest.stepdefinitions"},
    snippets = CAMEL_CASE)
public class SoapAddNumbers {
    // ...
}

SoapAddNumbers

Ex Tests ignored: 1 of 1 test – 20 ms
TEST PENDING: Add two number

1 Scenarios (1 undefined)
2 Steps (2 undefined)
0m0.343s

You can implement missing steps with the snippets below:

@When("^you add two number$")
public void youAddTwoNumber(DataTable arg1) {
    // Write code here that turns the phrase above into concrete actions
    // For automatic transformation, change DataTable to one of
    // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.
    // E,K,V must be a scalar (String, Integer, Date, enum etc).
    // Field names for YourType must match the column names in
    // your feature file (except for spaces and capitalization).
    throw new PendingException();
}

@Then("^I should see the response of the service is (\\d+)$")
public void iShouldSeeTheResponseOfTheServiceIs(int arg1) {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```



## ScreenPlay+BDD+Services – Rest & Soap

Pasos que se llevaran a implementar en el stepdefinition donde crearemos la clase

“**StepDefinitionsSoapAddNumbers**”

```
add_two_numbers.feature x SoapAddNumbers.java x StepDefinitionsSoapAddNumbers.java x
1 package co.com.choucair.services.soaprest.stepdefinitions;
2
3 public class StepDefinitionsSoapAddNumbers {
4     @When("^you add two number$")
5     public void youAddTwoNumber(DataTable arg1) {
6         // Write code here that turns the phrase above into concrete actions
7         // For automatic transformation, change DataTable to one of
8         // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.
9         // E,K,V must be a scalar (String, Integer, Date, enum etc).
10        // Field names for YourType must match the column names in
11        // your feature file (except for spaces and capitalization).
12        throw new PendingException();
13    }
14
15
16    @Then("^I should see the response of the service is (\\d+)$")
17    public void iShouldSeeTheResponseOfTheServiceIs(int arg1) {
18        // Write code here that turns the phrase above into concrete actions
19        throw new PendingException();
20    }
21 }
22
```





## ScreenPlay+BDD+Services – Rest & Soap

Nuevamente como lo hicimos para el caso del ejemplo de rest vamos a prepara un escenario y crear un actor. Para ello vamos a hacer uso del paquete “*utils*” donde crearemos una clase llamada “*BeforeHookSoap*” con el método “*prepareStage* ()”. Hacemos uso de la anotación “@Before” para que nuestra ejecución inicie desde este punto. Además crearemos una clase [enum](#) a la que llamaremos “*SoapService*” dentro del paquete “*enums*” que se encuentra dentro del paquete “*utils*”.

Nuestra clase *BeforeHookSoap* quedara así:

```
BeforeHookSoap.java x SoapService.java x
1 package co.com.choucair.services.soaprest.utils;
2
3 import cucumber.api.java.Before;
4 import net.serenitybdd.screenplay.actors.OnStage;
5 import net.serenitybdd.screenplay.actors.OnlineCast;
6 import net.serenitybdd.screenplay.rest.abilities.CallAnApi;
7
8 import static co.com.choucair.services.soaprest.utils.enums.SoapService.BASE_URL;
9 import static net.serenitybdd.screenplay.actors.OnStage.theActorCalled;
10
11 public class BeforeHookSoap {
12     @Before
13     public void prepareStage(){
14         OnStage.setTheStage(new OnlineCast());
15         theActorCalled( requiredActor: "brandon").whoCan(CallAnApi.at(BASE_URL.toString()));
16     }
17 }
```



## ScreenPlay+BDD+Services – Rest & Soap

Y nuestra clase *enum* “*SoapService*” quedara así:

```
reHookSoap.java x SoapService.java x
package co.com.choucair.services.soaprest.utils.enums;

public enum SoapService {
    BASE_URL( uri: "http://www.dneonline.com/"),
    ADD_NUMBERS( uri: "calculator.asmx");

    private String uri;

    SoapService(String uri) {
        this.uri = uri;
    }

    @Override
    public String toString() { return uri; }
}
```

Volvemos a nuestra clase “*StepDefinitionsSoapAddNumbers*” donde implementaremos los métodos planteados por cucumber. Limpiaremos los métodos “public void” de tal forma que no queden comentarios y excepciones, además iniciaremos con la implementación de nuestro método.

```
@When("^you add two number$")
public void youAddTwoNumber(DataTable arg1) {
```

Método que modificaremos en el tipo de dato que recibe por parámetro por una lista de string.

```
@When("^you add two number$")
public void youAddTwoNumber(List<String>values) {
```

Seguidamente hacemos uso del actor para que este atento a agregar números con los valores, donde *AddNumbers* será la clase y *with* será el método de la clase.

```
@When("^you add two number$")
public void youAddTwoNumber(List<String>values) {
    theActorInTheSpotLight().attemptsTo(AddNumbers.with(values));
}
```



## ScreenPlay+BDD+Services – Rest & Soap

Recordemos la clase **AddNumbers** se debe crear en el paquete **Task** y debe verse así

```
AddNumbers.java x
1  package co.com.choucair.services.soaprest.tasks;
2
3  import ...
11
12  public class AddNumbers implements Task {
13      private List<String> values;
14
15      public AddNumbers(List<String> values) { this.values = values; }
16
17
18
19      @Override
20      public <T extends Actor> void performAs(T actor) {
21          actor.attemptsTo(
22              Post.to(ADD_NUMBERS.toString())
23                  .with(request -> request
24                      .header( headerName: "Content-Type", headerValue: "text/xml")
25                      .body("<?xml version='1.0' encoding='utf-8'?'>\n" +
26                          "<soap:Envelope\n" +
27                          "    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'\n" +
28                          "    xmlns:xsd='http://www.w3.org/2001/XMLSchema'\n" +
29                          "    xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>\n" +
30                          "      <soap:Body>\n" +
31                          "        <Add\n" +
32                          "          xmlns='http://tempuri.org/'>\n" +
33                          "            <intA>" + values.get(0) + "</intA>\n" +
34                          "            <intB>" + values.get(1) + "</intB>\n" +
35                          "          </Add>\n" +
36                          "        </soap:Body>\n" +
37                          "      </soap:Envelope>")
38          );
39      }
40
41
42      public static AddNumbers with(List<String> values) { return instrumented(AddNumbers.class, values); }
43
44  }
45
46
```

Te invito a que realices una interacción para mejorarlo  
**!intentalo!**

## ScreenPlay+BDD+Services – Rest & Soap

Regresamos a nuestra clase “*StepDefinitionsSoapAddNumbers*” para implementar la validación.

```
@Then("^I should see the response of the service is (\\d+)$")
public void iShouldSeeTheResponseOfTheServiceIs(int code) {
    theActorInTheSpotlight().should(
        seeThat("last response status code is 200",
        LastResponseStatus.isEqualsTo(code))
    );
}
```

Crearemos la clase *LastResponseStatus* en el paquete *questions* y dicha clase tendrá el método *isEqualsTo*.


```
LastResponseStatus.java x
1      package co.com.choucair.services.soaprest.questions;
2
3      import net.serenitybdd.screenplay.Actor;
4      import net.serenitybdd.screenplay.Question;
5
6      import static net.serenitybdd.rest.SerenityRest.lastResponse;
7
8      public class LastResponseStatus implements Question<Boolean> {
9          private int code;
10
11         public LastResponseStatus(int code) { this.code = code; }
12
13
14
15         @ public static LastResponseStatus isEqualsTo(int code) { return new LastResponseStatus(code); }
16
17
18
19         @Override
20         public Boolean answeredBy(Actor actor) { return lastResponse().statusCode() == code; }
21
22     }
23 }
```

¡Hasta aquí termina el ejemplo basico de usa solicitud por Soap haciendo uso del patron de Screenplay;




# ScreenPlay+BDD+Services – Rest & Soap


Después de ejecutar el runner podrás validar el reporte y veras algo así:


 Serenity BDD

Home > Soap > Add Two Number On Dneonline Service > Add two number

☒ Overall Test Results ☒ Requirements ☒ Capabilities ☒ Features Report generated 23-03-2020 20:46

 Add Two Number On Dneonline Service

 Add two number

 Cast

Steps	Outcome	
When you add two number	SUCCESS	2.88s
<div> <div>2</div> <div>4</div> </div>		
Brandon add numbers	SUCCESS	2.53s
<div> <div>brandon executes a POST on the resource /calculator.asmx</div> <div>POST http://www.dneonline.com/calculator.asmx</div> <div>REST Query</div> <div>Response</div> <div>Status code: 200</div> <div>Content Type: application/xml</div> <div>Request Headers</div> <div>Accept= */*</div> <div>Content-Type=text/xml; charset=ISO-8859-1</div> <div>Content Body</div> <div>&lt;soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"&gt;&lt;soap:Body&gt;&lt;Add xmlns="http://tempuri.org/"&gt;&lt;intA&gt;2&lt;/intA&gt;&lt;intB&gt;4&lt;/intB&gt;&lt;/Add&gt;&lt;/soap:Body&gt;&lt;/soap:Envelope&gt;</div> </div>	SUCCESS	0s
<div> <div>Then I should see the response of the service is 200</div> <div>Then last response status code is 200</div> </div>	SUCCESS	0.06s
	SUCCESS	0s
	SUCCESS	3.09s

Serenity BDD version 2.0.90

