



2.3 – Pruebas de Servicios de Internet

Tema 6 - Pruebas de aceptación



Universidad
Rey Juan Carlos

Micael Gallego

micael.gallego@urjc.es
@micael_gallego

Michel Maes

michel.maes@urjc.es



- Introducción
- Gherkin
- Cucumber
- Karate

- **Introducción**
- Gherkin
- Cucumber
- Karate

- Según el **ISTQB** (*International Software Testing Qualifications Board*), las pruebas de aceptación (*acceptance tests*) son los tests formales para determinar si un sistema satisface los criterios de aceptación y permite a los usuarios determinar si aceptan el sistema o no.
- Son las pruebas realizadas por los usuarios en las **versiones beta** del software (**beta testing**)

- Según la **Agile Alliance**, una prueba de aceptación es una descripción formal del comportamiento de un software usando un **ejemplo o escenario**
- Existen diferentes notaciones para especificar los tests, pero siempre son de **alto nivel (no técnicas y que pueden usar por negocio)**, pero que se pueden **ejecutar**

- Ejemplo de prueba de aceptación

```
1 Feature: Login Page
2
3   As a user I want to be able to login and out of the system.
4
5 Background:
6   Given I am at the login page
7
8 Scenario: Login as a user
9   When I login with user 'janedoe' and password 'password'
10  Then I should see the home page
11  And I can logout
```

- Beneficios
 - Fomenta la colaboración entre desarrolladores con usuarios, clientes o expertos del dominio porque los **requisitos** se deben expresar como un **contrato no ambiguo**
 - Un producto que pasa las pruebas de aceptación será considerado **adecuado** (aunque los usuarios pueden refinar las pruebas/ejemplo o sugerir nuevos si es necesario)
 - Se limitan los nuevos **defectos y regresiones**

- **Error comunes**
 - **Incluir demasiadas definiciones técnicas en los escenarios**
 - Los usuarios y expertos del dominio entienden peor las pruebas de aceptación que contienen detalles técnicos
 - Para evitarlo, lo ideal es que los propios usuarios y expertos sean los que escriban las pruebas de aceptación

- **ATDD: Desarrollo guiado por pruebas de aceptación**
 - ***Acceptance Test Driven Development***
 - Es una metodología de desarrollo que involucra a los desarrolladores, testers y clientes para escribir las pruebas de aceptación antes de implementar la funcionalidad
 - Estos tests de aceptación representan el punto de vista del usuario y actúan como una forma de requisitos que describen cómo debería funcionar el sistema.
 - Cuando se ejecutan de forma automática, verifican el comportamiento del sistema

- **BDD: Desarrollo guiado por comportamiento**
 - *Behaviour Driven Development*
 - También conocido como Especificación mediante ejemplos (Specification by example)
 - En muchos contextos se usa como sinónimo de ATDD, pero algunos autores indican algunas diferencias entre ellos

<https://lizkeogh.com/2011/06/27/atdd-vs-bdd-and-a-potted-history-of-some-related-stuff/>

<https://gabo esquivel.com/blog/2014/differences-between-tdd-atdd-and-bdd/>

- Introducción
- **Gherkin**
- Cucumber
- Karate

- **Gherkin** es un lenguaje específico del dominio, entendible por negocio que te permite describir el comportamiento del software sin detallar cómo se implementa
- Sirve dos propósitos: **documentar** y las **pruebas automáticas**
- Se puede usar en **cualquier idioma** (aquí la usaremos en inglés)
- Gherkin en **ficheros**:
 - Extensión **.feature**
 - Cada fichero contiene una única **funcionalidad**
 - Pueden tener **varias especificaciones**

- **Sintaxis**
 - Basado en la indentación de las líneas (Python, YAML)
 - Comentarios con #

Feature: Serve coffee

Coffee should not be served until paid for

Coffee should not be served until the button has been pressed

If there is no coffee left then money should be refunded

Scenario: Buy last coffee

Given there are 1 coffees left in the machine

And I have deposited 1\$

When I press the coffee button

Then I should be served a coffee

- **Pasos: Given / When / Then**
- Se usan con el mismo significado que hemos visto en los tests unitarios
 - **Given:** Definición del estado de partida del sistema
 - **When:** Acción realizada para ejercitar el SUT
 - **Then:** Resultado esperado

<https://cucumber.io/docs/gherkin/>

- **Pasos: And / But**

- Se usan para mejorar la legibilidad de las especificaciones

Scenario: Multiple Givens

Given one thing

Given another thing

Given yet another thing

When I open my eyes

Then I see something

Then I don't see something else



Scenario: Multiple Givens

Given one thing

And another thing

And yet another thing

When I open my eyes

Then I see something

But I don't see something else

- Esquemas de escenarios (scenario outlines)
- Cuando los escenarios son parecidos, se pueden usar plantillas y los datos en tablas

```
Scenario: Eat 5 out of 12
  Given there are 12 cucumbers
  When I eat 5 cucumbers
  Then I should have 7 cucumbers
```

```
Scenario: Eat 5 out of 20
  Given there are 20 cucumbers
  When I eat 5 cucumbers
  Then I should have 15 cucumbers
```



```
Scenario Outline: Eating
  Given there are <start> cucumbers
  When I eat <eat> cucumbers
  Then I should have <left> cucumbers
```

Examples:

start	eat	left	
12	5	7	
20	5	15	



- **Editores**

- Para facilitar la escritura de features con Gherkin existen múltiples editores
- El más sencillo de instalar es Tidy Gherkin, una extensión de Google Chrome



Gherkin

TIDY! FILE + EDIT + INSERT + SUPPORT + ABOUT + **SETTINGS** Pg Try Pretty Gherki

rows columns **INSERT TABLE**  

```
12 Scenario Outline: Eating
13
14     Given that there are <start> cucumbers
15     When I eat <eat> cucumbers
16     Then I should have <left> cucumbers
17
18     | start | eat | left |
19     | 20    | 10  | 10   |
20     | 15    | 3   | 12   |
```

TIDY PREVIEW **JAVA STEPS** RUBY STEPS

```
1 package my.package.name
2
3 import cucumber.api.PendingException;
4 import cucumber.api.java.en.Given;
5 import cucumber.api.java.en.When;
6 import cucumber.api.java.en.Then;
7 import cucumber.api.junit.Cucumber;
8 import org.junit.runner.RunWith;
9
10 @RunWith(Cucumber.class)
11 public class MyStepDefinitions {
12
13     @Given("^there are 12 cucumbers$")
14     public void there_are_12_cucumbers() throws Throwable {
15         throw new PendingException();
16     }
17 }
```

- Escribe los tests de la clase de números complejos como features Gherkin usando Tidy Gherkin
 - Cuando cualquier número complejo es sumado a $(0+0i)$ el resultado es el mismo número
 - Dado un número, cada una de sus partes puede ser obtenida de forma individual

- Introducción
- Gherkin
- **Cucumber**
- Karate

- Es un framework para ejecutar especificaciones **Gherkin**
- Originalmente escrito en **Ruby**, actualmente tiene versiones para **Java** y **JavaScript**

cucumber 

- Para **ejecutar** una **funcionalidad Gherkin** (*feature*) es necesario implementar **código Java de pegamento** (*glue code*) que “interprete” los pasos usando clases del código
- Ese código Java **leerá** los ficheros **.feature** para cargar los **valores** y los **ejemplos** de las plantillas
- Las especificaciones se ejecutan como tests de **Junit** que pasarán sólo si el software se comporta como se espera

- **Feature**

/src/test/resources/es/codeurjc/test/cucumber/calcul.feature

Feature: Calculator

As a user

I want to use a calculator to add numbers

So that I don't need to add myself

Scenario: Add two numbers -2 & 3

Given I have a calculator

When I add -2 and 3

Then the result should be 1

Scenario: Add two numbers 10 & 15

Given I have a calculator

When I add 10 and 15

Then the result should be 25

- Ejecución de la feature como JUnit

/src/test/java/es/codeurjc/test/cucumber/CalculatorTest.java

```
package es.codeurjc.test.cucumber;  
  
import org.junit.runner.RunWith;  
  
import cucumber.api.junit.Cucumber;  
  
@RunWith(Cucumber.class)  
public class CalculatorTest {  
  
}
```


- **pom.xml**

```
<dependencies>
  <dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>1.2.5</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>1.2.5</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-library</artifactId>
    <version>1.3</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- Ejecución de los tests

The screenshot displays the Cucumber test runner interface. At the top, a summary bar shows 'Runs: 6/6 (8 skipped)', 'Errors: 0', and 'Failures: 0' next to a green progress bar. Below this, a tree view shows the test structure. The root node is 'es.codeurjc.test.cucumber.CalculatorTest [Runner: ...]'. It has two children: 'Feature: Calculator (0.001 s)' and 'Scenario: Add two numbers 10 & 15 (0.000 s)'. The 'Feature: Calculator' node is expanded, showing two scenarios: 'Scenario: Add two numbers -2 & 3 (0.000 s)' and 'Scenario: Add two numbers 10 & 15 (0.000 s)'. Each scenario has three steps: 'Given I have a calculator (0.000 s)', 'When I add -2 and 3 (0.000 s)' (or '10 and 15'), and 'Then the result should be 1 (0.000 s)' (or '25'). All steps are marked with a green checkmark icon, indicating they passed. To the right of the tree view, there is a 'Failure Trace' section with a hamburger menu icon and some small icons. A green callout box points to the 'Scenario: Add two numbers 10 & 15' and its steps, containing the text 'Test ignorados porque no tienen "pegamento"'.

Runs: 6/6 (8 skipped) ✖ Errors: 0 ✖ Failures: 0

es.codeurjc.test.cucumber.CalculatorTest [Runner: ...] Failure Trace

Feature: Calculator (0.001 s)

- Scenario: Add two numbers -2 & 3 (0.000 s)
 - Given I have a calculator (0.000 s)
 - When I add -2 and 3 (0.000 s)
 - Then the result should be 1 (0.000 s)
- Scenario: Add two numbers 10 & 15 (0.000 s)
 - Given I have a calculator (0.000 s)
 - When I add 10 and 15 (0.000 s)
 - Then the result should be 25 (0.000 s)

Test ignorados porque no tienen "pegamento"

```
2 Scenarios
6 Steps
0m0.000s
```

You can implement missing steps with the snippets below:

```
@Given("^I have a calculator$")
public void i_have_a_calculator() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I add -(\\d+) and (\\d+)$")
public void i_add_and(int arg1, int arg2) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^the result should be (\\d+)$")
public void the_result_should_be(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I add (\\d+) and (\\d+)$")
public void i_add_and(int arg1, int arg2) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

- Para ejecutar las pruebas de aceptación hay que implementar el **código de pegamento** (*glue code*)
- El test de JUnit:
 - Busca el código de pegamento en su **mismo paquete** (pero es configurable)
 - Busca los .feature en la carpeta con la **misma ruta del paquete** (pero es configurable)

```
@RunWith(Cucumber.class)
@CucumberOptions(
    plugin = {"pretty"},
    features = { "classpath:es/codeurjc/test/cucumber" },
    glue = {"es.codeurjc.test.cucumber" })
public class CalculatorTest {}
```

ejem2

- Código pegamento

/src/test/java/es/codeurjc/test/cucumber/CalculatorRunSteps.java

```
public class CalculatorRunSteps {  
  
    private int total;  
  
    private Calculator calculator;  
  
    @Before  
    private void init() {  
        total = -999;  
    }  
  
    @Given("^I have a calculator$")  
    public void initializeCalculator() throws Throwable {  
        calculator = new Calculator();  
    }  
  
    @When("^I add (-?\\d+) and (-?\\d+)$")  
    public void testAdd(int num1, int num2) throws Throwable {  
        total = calculator.add(num1, num2);  
    }  
  
    @Then("^the result should be (-?\\d+)$")  
    public void validateResult(int result) throws Throwable {  
        Assert.assertThat(total, Matchers.equalTo(result));  
    }  
}
```

ejem2

- Ejecución de los test

ejem2

The screenshot displays a Cucumber test runner interface. At the top, a summary bar shows 'Runs: 6/6', 'Errors: 0', and 'Failures: 0', accompanied by a green progress bar. Below this, a tree view shows the test structure. The selected test is 'es.codeurjc.test.cucumber.CalculatorTest [Runner: ...]'. It contains two scenarios: 'Feature: Calculator (0.008 s)' and 'Scenario: Add two numbers -2 & 3 (0.005 s)'. The first scenario has three steps: 'Given I have a calculator (0.002 s)', 'When I add -2 and 3 (0.003 s)', and 'Then the result should be 1 (0.000 s)'. The second scenario has three steps: 'Given I have a calculator (0.000 s)', 'When I add 10 and 15 (0.000 s)', and 'Then the result should be 25 (0.002 s)'. All steps are marked with green checkmarks, indicating successful execution. A 'Failure Trace' tab is visible on the right side of the interface.

Runs: 6/6 ✖ Errors: 0 ✖ Failures: 0

es.codeurjc.test.cucumber.CalculatorTest [Runner: ...] Failure Trace

- Feature: Calculator (0.008 s)
 - Scenario: Add two numbers -2 & 3 (0.005 s)
 - Given I have a calculator (0.002 s)
 - When I add -2 and 3 (0.003 s)
 - Then the result should be 1 (0.000 s)
 - Scenario: Add two numbers 10 & 15 (0.002 s)
 - Given I have a calculator (0.000 s)
 - When I add 10 and 15 (0.000 s)
 - Then the result should be 25 (0.002 s)

- Ejecución de los test

ejem2

Feature: Calculator

As a user

I want to use a calculator to add numbers

So that I don't need to add myself

Scenario: Add two numbers -2 & 3

Given I have a calculator

When I add -2 and 3

Then the result should be 1

es/codeurjc/test/cucumber/calc.feature:6

CalculatorRunSteps.initializeCalculator()

CalculatorRunSteps.testAdd(int,int)

CalculatorRunSteps.validateResult(int)

Scenario: Add two numbers 10 & 15

Given I have a calculator

When I add 10 and 15

Then the result should be 25

es/codeurjc/test/cucumber/calc.feature:11

CalculatorRunSteps.initializeCalculator()

CalculatorRunSteps.testAdd(int,int)

CalculatorRunSteps.validateResult(int)

2 Scenarios (2 passed)

6 Steps (6 passed)

0m0.068s

- Cuando tenemos varios escenarios similares, es mejor tener una plantilla (scenario outline)

Feature: Calculator

ejem3

As a user

I want to use a calculator to add numbers

So that I don't need to add myself

Scenario Outline: Add two numbers <num1> & <num2>

Given I have a calculator

When I add <num1> and <num2>

Then the result should be <total>

Examples:

num1	num2	total
-2	3	1
10	15	25
99	-99	0
-1	-10	-11

- **Steps con expresiones regulares**
 - Para que los steps sean flexibles y puedan ejecutar diferentes escenarios se tienen que implementar usando expresiones regulares
 - Las expresiones regulares
 - Definen el patrón que deben cumplir los steps
 - Capturan los valores para que se puedan usar en la implementación

<https://agileforall.com/just-enough-regular-expressions-for-cucumber/>

- **Steps con expresiones regulares**

- Inicio y fin del texto `^I'm logged in$`
- Cero o más caracteres `.*`
- Uno o más caracteres `.+`
- Cero o más dígitos `\d*` (en Java `\\d*`)
- Uno o más dígitos `\d+` (en Java `\\d+`)
- Cero o más caracteres entre comillas `"[^"]*" (Java \"[^\"]*\\")`
- Caracter opcional `a?`

- Grupos para capturar valores
 - Entre paréntesis se capturan los valores

```
When I'm logged as an admin  
When I'm logged as a user
```

```
@When("^I'm logged in as an? (.*)$")  
public void whenLogged(String role) throws Throwable {...}
```

- Con ?: no se capturan los valores

```
When I'm logged as an admin  
When I'm logged as a user
```

```
@When("^(?:I'm logged|I log) in as an? (.*)$")  
public void whenLogged(String role) throws Throwable {...}
```

- Implementa el código de pegamento para ejecutar los tests de los números complejos del ejercicio 1

- Introducción
- Gherkin
- Cucumber
- **Karate**



- Es una herramienta de testing que combina:
 - Automatización de test de API REST y UI
 - Mocks
 - Testing de rendimiento
- **Extiende la sintaxis** de Cucumber, pero evitandonos escribir el código “pegamento”.

<https://github.com/karatelabs/karate>



Testing de APIs REST

Scenario: create and retrieve a cat

Given url 'http://myhost.com/v1/cats'

And request { name: 'Billie' }

When method post

Then status 201

And match response == { id: '#notnull', name: 'Billie' }

Given path response.id

When method get

Then status 200

JSON is 'native'
to the syntax

Intuitive DSL
for HTTP

Payload
assertion in
one line

Second HTTP
call using
response data



- ¿Cómo lo añadimos a nuestro proyecto Spring?

```
<dependency>
  <groupId>com.intuit.karate</groupId>
  <artifactId>karate-apache</artifactId>
  <version>${karate.version}</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>com.intuit.karate</groupId>
  <artifactId>karate-junit5</artifactId>
  <version>${karate.version}</version>
  <scope>test</scope>
</dependency>
```

ejem4



• Ejemplo de Karate en la aplicación de Items

src/test/java/es/codeurjc/test/rest/items.feature

ejem4

Feature: items end-point

Background:

- * url 'http://localhost:8080'
- * configure logPrettyRequest = true
- * configure logPrettyResponse = true

Scenario: create and retrieve a item

Given path 'items/'

And request { description: 'Leche', checked: true }

When method post

Then status 201

And match response == { id: '#number', description: 'Leche', checked: true }

Definimos la URL base

Cuerpo de la petición
POST

Comprobamos la
respuesta

No sabemos el 'id' de la respuesta,
comprobará que 'id' es un número



- Ejemplo de Karate en la aplicación de Items

src/test/java/es/codeurjc/test/rest/ItemsControllerTest.java

ejem4

```
package es.codeurjc.test.rest;

import com.intuit.karate.junit5.Karate;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DefinedPort)
class ItemsControllerTest {

    @Karate.Test
    public Karate testSample() {
        return Karate.run("items").relativeTo(getClass());
    }

}
```

Nombre del fichero *.feature* (Debe estar en el mismo directorio)



- Comprobación de datos
 - Podemos validar el formato de de cada uno de los campos de la respuesta.

```
* def anuncio = { id: 3, contenido: "Vendo moto", pais: 'ES', activo: true }  
* match anuncio == { id: '#number', contenido: '#present', pais: '#regex [A-Z]{2}', activo: '#boolean' }
```

- **#number** → Comprobar que es un número
- **#present** → Comprobar que la clave está presente
- **#regex STR** → Comprueba que cumple la expresión regex STR
- **#boolean** → Comprueba que es un valor booleano

Mas validadores: <https://github.com/intuit/karate#fuzzy-matching>



- ¿Qué hacemos si queremos utilizar variables de Java en nuestro archivo .feature?
 - Es necesario incluir un **fichero Javascript** que haga de 'puente'
 - Este fichero debe estar situado en en la ruta **/src/test/java/** y nombrarse como **karate-config.js**



- Utilizar variables de Java en Karate

src/test/java/es/codeurjc/test/rest/ItemsControllerTest.java

ejem5

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)  
class ItemsControllerTest {  
  
    @LocalServerPort  
    int port;  
  
    @BeforeEach  
    public void setPort(){  
        System.setProperty("base.url", "http://localhost:"+port);  
    }  
  
    @Karate.Test  
    public Karate testSample() {  
        return Karate.run("items").relativeTo(getClass());  
    }  
}
```

Definimos una propiedad en el sistema





- Utilizar variables de Java en Karate

ejem5

src/test/java/karate-config.js

```
function fn() {  
  return { 'targetUrlBase': karate.properties['base.url'] };  
}
```

Recuperamos la propiedad (Karate puede leer las propiedades declaradas en Java)

Podemos pasarle al archivo *.feature* las variables declaradas en el objeto config



- Utilizar variables de Java en Karate

src/test/java/es/codeurjc/test/rest/items.feature

ejem5

Feature: items end-point

Background:

- * url baseUrl
- * configure logPrettyRequest = true
- * configure logPrettyResponse = true

Utilizamos la variable
definida en el archivo de
configuración

Scenario: create and retrieve a item

Given path 'items/'

And request { description: 'Leche', checked: true }

When method post

Then status 201

And match response == { id: '#number', description: 'Leche', checked: true }



- Ejercicio 2
 - Para la aplicación de anuncios, usar Karate para crear el siguiente escenario
 - Crear, recibir y borrar anuncio

TIP: Podemos obtener el id del objeto creado en otra petición para reutilizarlo:

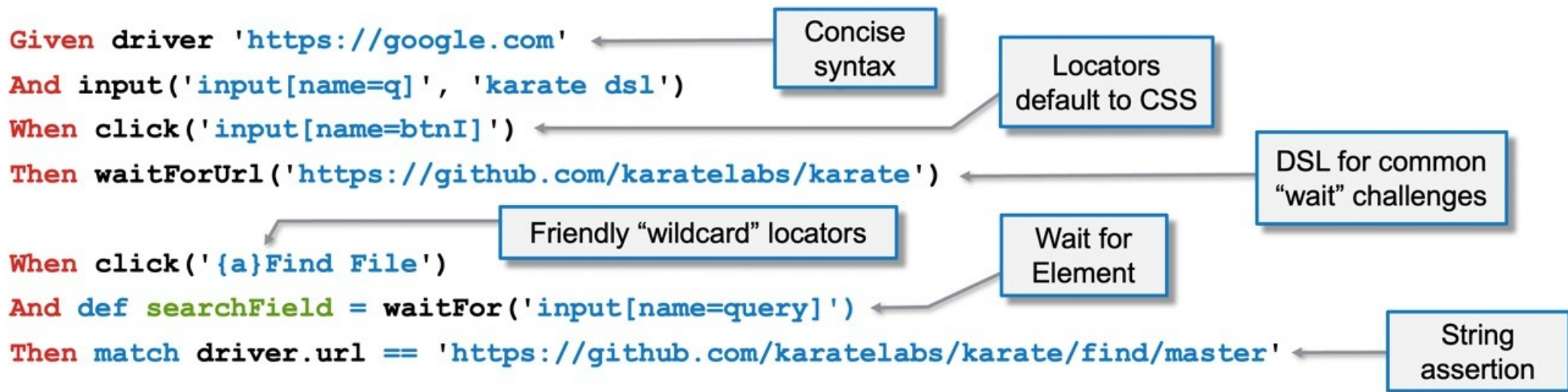
```
....
```

```
* def id = response.id
```

```
Given path '/', id
```

```
...
```


Karate UI – Testing de Interfaces



"The world needs an alternative to Selenium - so we built one"

<https://github.com/karatelabs/karate/tree/master/karate-core>

<https://hackernoon.com/the-world-needs-an-alternative-to-selenium-so-we-built-one-zrk3j3nyr>



Karate UI – Testing de Interfaces

- Permite hacer pruebas interactuando con los componentes de una aplicación gráfica a través de drivers:
 - Webs en navegadores (Chrome, Firefox, Safari y Edge)
 - Aplicaciones nativas de escritorio (WinAppDriver)
 - Aplicaciones móviles (Android/iOS)
- Características
 - Ejecución en paralelo
 - Grabación de test
 - Mocks de HTML
 - Sintaxis simplificada para la interacción (*locators*)
 - Buena integración con CI
 - Y más ..



- Ejemplo de Karate UI – Probando Wikipedia

src/test/java/es/codeurjc/test/web/wikipedia.feature

ejem6

Feature: Finding Rick Astley in Wikipedia

Background:

* **configure** driver = { type: 'chrome', showDriverLog: true }

Scenario: Search for Rick Astley in Wikipedia and find the RickRoll reference

Given driver 'https://wikipedia.org'

And input('input[name=search]', 'Rick Astley')

When submit().click("button[type=submit]")

Then **match** html('#mw-content-text') **contains** 'RickRoll'



• Ejemplo de Karate UI – Probando Wikipedia

src/test/java/es/codeurjc/test/web/wikipedia.feature

Feature: Finding Rick Astley in Wikipedia

Background:

```
* configure driver = { type: 'chrome', showDriverLog: true }
```

Scenario: Search for Rick Astley in Wikipedia and find

```
Given driver 'https://wikipedia.org'
```

```
And input('input[name=search]', 'Rick Astley')
```

```
When submit().click("button[type=submit]")
```

```
Then match html('#mw-content-text') contains 'RickRoll'
```

ejem6

Configuramos el Driver
del navegador

Navegamos a la página

Interactuamos con
la web (rellenar un
formulario y darle a
enviar)

Comprobamos la
respuesta



• Ejemplo de Karate UI – Probando Wikipedia

src/test/java/es/codeurjc/test/web/wikipedia.feature

ejem6

Feature: Finding Rick Astley in Wikipedia

Background:

```
* configure driver = { type: 'chrome', showDriverLog: true }  
* configure driverTarget = { docker: 'ptrthomas/karate-chrome',  
                             showDriverLog: true }
```

Scenario: Search for Rick Astley in Wikipedia and find the RickRoll reference

```
Given driver 'https://  
  And input('input[  
When submit().click("b  
Then match html('#mw-c
```

Podemos configurar el driver para que use un navegador *Dockerizado*. La imagen Docker `ptrthomas/karate-chrome` graba automáticamente la sesión del navegador para que podamos visualizarla en caso de fallo en el test



- Ejemplo de Karate UI – Probando la app de Mensajes (Spring)

ejem7

```
<body>
  <h1>Message forum</h1>
  <form id="form" action="/" method="post">
    Title: <input id='title-input' type='text' name='title' />
    Body: <input id='body-input' type='text' name='body' />
    <input id='submit' type='submit' value='New' />
  </form>
  <div id="messages">
    {{#messages}}
    <p>Message: <b>Title:</b> <span id='title'>{{title}}</span>
    <b>Description:</b> <span id='body'>{{body}}</span><br>
    {{/messages}}
  </div>
</body>
```



- Ejemplo de Karate UI – Probando la app de Mensajes (Spring)

src/test/java/es/codeurjc/test/web/message.feature

ejem7

Feature: Create new message

Background:

```
* configure driver = { type: 'chrome', showDriverLog: true }
```

Scenario: Create new message and check it

```
Given driver targetUrlBase
  And input('#title-input', 'Hola Mundo')
  And input('#body-input', 'Estoy usando Karate')
When submit().click("input[type=submit]")
Then match html('#title') contains 'Hola Mundo'
```

src/test/java/karate-config.js

```
function fn() {
  return { 'targetUrlBase': karate.properties['base.url'] };
}
```



- Ejemplo de Karate UI – Probando la app de Mensajes (Spring)
- Si queremos utilizar un navegador Dockerizado, no podremos utilizar localhost (la red del contenedor y de la aplicación no es la misma)
- Hay soluciones avanzadas para dockerizar también la aplicación
- A continuación mostramos una solución “sencilla”

```
@BeforeEach
public void setPort() throws SocketException{
    try (final DatagramSocket datagramSocket = new DatagramSocket()) {
        datagramSocket.connect(InetAddress.getByName("8.8.8.8"), 12345);
        String baseUrl = "http://" + datagramSocket.getLocalAddress().getHostAddress() + ":" + port;
        System.setProperty("base.url", baseUrl);
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }
}
```

ejem7

Este fragmento de código obtiene la IP de nuestra máquina (192.168.1.XXX) para crear la URL de la aplicación