

# ¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.  
 Ese apoyo que siempre quiso tener...

## 1. Desarrollo de componentes y proyectos a medida



## 2. Auditoría de código y recomendaciones de mejora

## 3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



## 4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,  
 HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)  
 Gestor de contenidos (Alfresco)  
 Aplicaciones híbridas

Tareas programadas (Quartz)  
 Gestor documental (Alfresco)  
 Inversión de control (Spring)

Control de autenticación y  
 acceso (Spring Security)  
 UDDI  
 Web Services  
 Rest Services  
 Social SSO  
 SSO (Cas)

JPA-Hibernate, MyBatis  
 Motor de búsqueda empresarial (Solr)  
 ETL (Talend)

Dirección de Proyectos Informáticos.  
 Metodologías ágiles  
 Patrones de diseño  
 TDD

BPM (jBPM o Bonita)  
 Generación de informes (JasperReport)  
 ESB (Open ESB)

AdictosAlTrabajo

Final de Terrakas  
¡¡Ven al estreno!!  
terrakas.com



autentia  
Soporte a desarrollo informático  
Hosting patrocinado por  
enREDados

Entra en Adictos a través de  



Entrar [Deseo registrarme](#)  
[Olvidé mi contraseña](#)



[Inicio](#) [Quiénes somos](#) [Formación](#) [Comparador de salarios](#) [Nuestros libros](#) [Más](#)

» Estás en: [Inicio](#) [Tutoriales](#) [Haciendo BDD con Cucumber](#)



Daniel Diaz Suarez

Daniel es un alumno becario en prácticas, procedente del I.E.S. Rey Fernando VI

[Ver todos los tutoriales del autor](#)

Fecha de publicación del tutorial: 2009-02-26

Tutorial visitado 1 veces [Descargar en PDF](#)

## Haciendo BDD con Cucumber

### 0. Índice de contenidos.

- 1. Entorno
- 2. Introducción
- 3. ¿En que consiste Gherkin?
- 4. Ejemplos Prácticos

### 1. Entorno

Este tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Intel Core 2 CPU T7200 @ 2.00GHz x 2
- Sistema Operativo: Ubuntu 12.04 LTS x32
- Sublime Text 2

### 2. Introducción

Cucumber es una herramienta, escrita originalmente en Ruby, que ejecuta descripciones funcionales en texto plano como test automáticos, aprovechando las ventajas del BDD como puede ser acercar la capa de negocio y tecnología de una empresa, permitiendo que no solo se desarrollen bien las funcionalidades, sino que se desarrollen las funcionalidades que mas valor van a aportar al usuario, a la vez que definimos casos de prueba y documentamos un proyecto.

La idea principal es que pueda ser entendido y usado no solo por desarrolladores sino por los clientes y gente de negocio.

El lenguaje que usa Cucumber para definir los requisitos se llama Gherkin, el cual nos permite traducir esas especificaciones en un lenguaje cercano al natural en especificaciones de test en el lenguaje que queramos, un ejemplo de Gherkin sería el siguiente:

Feature: Search courses

```
In order to ensure better utilization of courses
Potential students should be able to search for courses
```

Scenario: Search by topic

```
Given there are 240 courses which do not have the topic "biology"
And there are 2 courses A001, B205 that each have "biology" as one of the topics
When I search for "biology"
Then I should see the following courses:
```

Course code
A001
B205

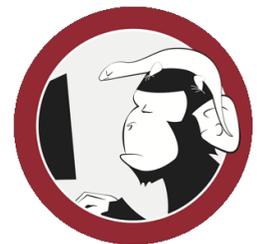
### 3. ¿En que consiste Gherkin?

Gherkin es el lenguaje que entiende Cucumber, es un DSL legible para gente no técnica, que permite definir el comportamiento del software sin detallar como está implementado, además de que nos permite documentar las funcionalidades a la vez que escribimos casos de prueba automáticos.

Otras ventajas que nos proporciona usar Gherkin y Cucumber son:

- Fáciles de leer
- Fáciles de entender
- Fáciles de Parsear
- Fáciles de discutir

### Catálogo de servicios Autentia



### Síguenos a través de:



### Últimas Noticias

» [Atención, APLAZADO Estreno último capítulo de Terrakas](#)

» [Vendedor: Soy inseguro, filtra o elige por mi: si quieres que te compre.](#)

» [Comentando el libro: El arte de pensar, de Rolf Dobelli](#)

» [Ya está a la venta mi segundo libro: Planifica tu éxito, de aprendiz a empresario](#)

» [Ya esta disponible en eBook mi primer libro: Informática Profesional](#)

[Histórico de noticias](#)

### Últimos Tutoriales

» [Manejo de test con TestLink](#)

» [Prototipado de pantallas con Pencil](#)

» [Como testear aplicaciones en Ember.js](#)

Gherkin es un lenguaje que usa el indentado para definir la estructura, de manera que los saltos de línea dividen las diferentes declaraciones, la mayoría de las líneas empiezan con palabras clave. El parser divide el texto en Features ( Características ), Scenarios y Steps, cuando pasas los casos de prueba, el parser busca un Step con ese nombre. Los Steps son los análogos de los Métodos en Java o las Funciones en Javascript.

Gherkin está localizado en muchos idiomas entre ellos el español:

```
# language: es
Característica: adición
  Para evitar hacer errores tontos
  Como un ser humano
  Quiero saber la suma de los números

Esquema del escenario: Sumar dos números
  Dado que he introducido en la calculadora
  Y que he introducido en la calculadora
  Cuando oprimo el
  Entonces el resultado debe ser en la pantalla
```

```
Ejemplos:
| entrada_1 | entrada_2 | botón | resultado |
| 20        | 30        | add   | 50        |
| 2         | 5         | add   | 7         |
| 0         | 40        | add   | 40        |
```

Generaría

```
# encoding: utf-8
begin require 'rspec/expectations'; rescue LoadError; require 'spec/expectations'; end
require 'cucumber/formatter/unicode'
$: .unshift(File.dirname(__FILE__) + '/../lib')
require 'calculador'

Before do
  @calc = Calculador.new
end

Dado /que he introducido (\d+) en la calculadora/ do |n|
  @calc.push n.to_i
end

Cuando /oprimo el (\w+)/ do |op|
  @result = @calc.send op
end

Entonces /el resultado debe ser (.*) en la pantalla/ do |result|
  @result.should == result.to_f
end
```

Cada feature ( característica ) se define en un archivo, una característica normalmente consiste en una serie de escenarios, el texto entre Característica y "Esquema del escenario" permite definir libremente ya que este texto no será usado para ninguna funcionalidad de Cucumber, es meramente descriptivo.

Este código nos generará un esqueleto ( dependiendo del lenguaje en el que vayamos a producir los test y el adaptador que estemos usando ), el cual pasará los test con los distintos casos que le hayamos definido. Luego será nuestra tarea la de definir los casos de prueba que comprobarán las diferentes características.

Se pueden diferenciar claramente tres grupos que cumplen funciones distintas:

- **Given (Dado)** : El propósito de los "Dado" es el de poner al sistema en el estado deseado para pasar los test deseados, antes de que el usuario ( o un sistema externo ), interactúe con el sistema.
- **When ( Cuando )** : En este caso el propósito de los "Cuando" es el describir la acción que realiza el usuario, la cual vamos a probar.
- **Then ( Entonces )** : El propósito de los "Entonces" es observar los resultados, estas observaciones han de ser realizadas desde el punto de vista de negocio y el valor para el usuario. Estas observaciones también deben de estar en algún sitio visible para el usuario, no en una base de datos o capas inferiores del sistema.
- **And, But ( Y, Pero )**: Estas "keywords" están destinadas a aumentar la legibilidad y pueden ser usadas en vez de definir repetidas veces cualquiera de las anteriores.

Al ejecutar Cucumber, buscará un Step que coincida con cada definición, siendo irrelevante la "keyword" usada para otra cosa que no sea el orden, esto cambia ligeramente dependiendo del lenguaje que estemos probando.

## 4. Ejemplos Prácticos

Podemos encontrar ejemplos en distintos lenguajes como:

**Java**, mostrando el archivo desde el que se parte y el archivo que se generaría, incluyendo algún test de ejemplo ya implementado, [Feature & Step](#).

Y **Ruby** [Feature & Steps](#).

## A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)

» [Internacionalizar una aplicación creada con Ember](#)

» [Control de la calidad, aseguramiento de la calidad y calidad total en el desarrollo de software](#)

## Últimos Tutoriales del Autor

» [Como testear aplicaciones en Ember.js](#)

» [Introducción a Require.JS](#)

» [Nuestra Primera App con Ember.js](#)

## Últimas ofertas de empleo

2011-09-08

[Comercial - Ventas - MADRID.](#)

2011-09-03

[Comercial - Ventas - VALENCIA.](#)

2011-08-19

[Comercial - Compras - ALICANTE.](#)

2011-07-12

[Otras Sin catalogar - MADRID.](#)

2011-07-06

[Otras Sin catalogar - LUGO.](#)

## Por favor, vota +1 o compártelo si te pareció interesante

Share |

0

¡Anímate y coméntanos lo que pienses sobre este **TUTORIAL**:



» **Regístrate** y accede a esta y otras ventajas «



Esta obra está licenciada bajo licencia [Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

PUSH THIS

Page Pushers

Community

Help?

----  
no clicks

0 people brought clicks to this page

+ + + + + + + +

powered by [karmacacy](#)

Copyright 2003-2013 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) | [Contacto](#)

