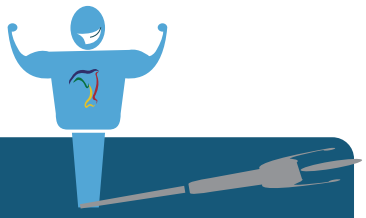
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

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

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

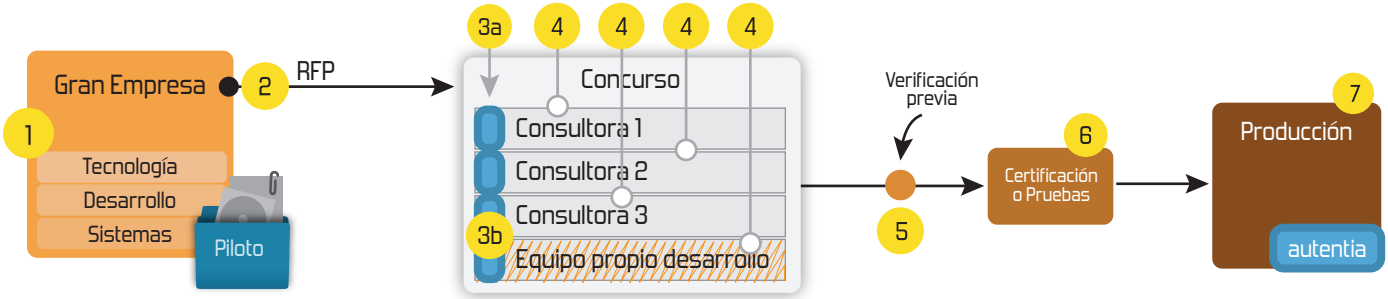
## 1. Desarrollo de componentes y proyectos a medida



## 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

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

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

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

Compartimos nuestro conocimiento en:
**www.adictosaltrabajo.com**

Para más información visítenos en:
**www.autentia.com**

DESARROLLADO POR:
Saúl García Díaz

Consultor tecnológico de desarrollo de proyectos informáticos.

Charla sobre LiquiBase

Charla sobre Alfresco Community Edition

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE



**Fecha de publicación del tutorial: 2009-02-26**



# Informes dinámicos con DynamicJasper

## 0. Índice de contenidos



## 1. Introducción

DynamicJasper es una buena librería de la que nos podemos servir para hacer nuestra vida un poco más fácil, principalmente cuando estamos trabajando en la generación de informes con JasperReports.

Es open-source y nos va a permitir crear informes de manera dinámica, definiendo en tiempo de ejecución gran parte de los elementos que pone a nuestra disposición JaperReports para crear informes (Definición de columnas, grupos, variables, tipos de letras, gráficos, subinformes etc .... ) .

Por experiencia, en la mayor parte de los proyectos en los que he tenido que trabajar con JasperReports, siempre he tendido a definir una plantilla para cada uno de los informes. Principalmete porque el diseño y la información de estos informes siempre era la misma (informes estáticos). Sin embargo, en ocasiones y debido a la complejidad de informe no era suficiente con una simple plantilla y era necesario mostrar un diseño y una información distinta en un único informe lo que hasta ahora, que no conocía esta librería, me había dado más de un quebradero de cabeza.

Ahora DynamicJasper nos da una solución facil para generar informes dinámicos, integrándose a la perfección con Maven y manteniendo la compatibilidad con los informes de Jasper ya que sólo se trata de una herramienta de ayuda a la creación de informes a nivel de programación.

Antes de continuar os dejo el código fuente con el ejemplo que veremos en el tutorial y la referencia a la página http://dynamicjasper.com/ a la página oficial de la librería donde podemos encontrar más información.

## 2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware : Portátil Mac Book Pro 15" (2,6 Ghz Intel Core i7, 4 GB DDR3)
- Sistema Operativo:Mac OS X Snow Leopard 10.6.6
- Eclipse Galileo
- Apache Maven 2.2.1
- JaspertReports 3.5.2
- DynamicJasper 3.0.13

## 3. Características

Hay que decir que la mayoría de las características de la librería son proporcianadas por JasperReports pero haciendo uso del API de DynamicJasper obtenedremos buenos resultados con un mínimo esfuerzo. A continuación paso a destacar algunas de las principales características de DynamicJasper:

- **Informes con columnas dinámicas**

  Las propiedades de las columnas se pueden definir en tiempo de ejecución lo que nos proporciona el control total sobre el diseño del informe.

- **Creación y repetición de grupos**

  Esto es muy interesante ya que nos permite definir grupos en función a unos criterios (como con Jasper) pero además nos permite modificar propiedades de cada unos de los grupos tales como el encabezado o pie de página por ejemplo.

- **Informe de diseño automático**

  Con sólo definir un conjunto mínimo de opciones DynamicJasper se encarga de generar el informe de manera automática. Esta característica se convertirá en una de las más a útiles a la hora de trabajar con DynamicJasper.

- **Dynamic Crosstab**

  Con DynamicJasper el manejo de las populares tablas de referencias cruzadas (crosstabs) se hace de una manera mucho mas sencilla y conveniente.

- **SubInformes dinámicos**

  De la misma manera que con los informes con DynamicJasper podemos crear en tiempo de ejecución los subinformes. Esto supone que no tendremos que tener las plantillas que se definen normalmente para los subinformes en Jasper. Además tendremos la capacidad de que el diseño y la información de cada uno de los subinformes que pertenecen al informe principal pueda ser distinta.

- **Estilos**

  Cada columna puede tener sus propios estilos para el título y datos de detalle definiendo aspectos tales como color de borde, tamaño de la fuente, tipo y color, color de fondo ect....

- **Cálculo de variables**

  Con DynamicJasper es posible crear y modificar variables de informe con el resultado de una operación en un determinado campo (columna) y que podremos utilizar por ejemplo como condiciones a la hora de trabajar con grupos.

- **Uso de plantillas**

  Nos va a permitir hacer uso de plantillas jrxml básicas para definir los aspectos comunes y no perecederos en nuestros informes tales como el logotipo de la empresa, el fondo corporativo etc ...

## 4. Requerimientos y configuración

Según la documentación oficial para poder trabajar con esta librería es necesario tener una serie de dependencias. A continuación se indican las librerías mínimas para el correcto funcionamiento de DynamicJasper:

- **Librerías requeridas**

- JasperReports - Desde la versión 3.0.5 de DynamicJasper es necesario la versión 3.5.X de JasperReports.
- Jakarta Commons BeanUtils Component (versión 1.7 o posterior)
- Jakarta Commons Collections Component (versión 3.1 o posterior)
- Jakarta Commons Logging Component (version 1.0 o posterior)
- Eclipse jdt-core compiler

A pesar de ello para el ejemplo que os mostraré a continuación solo ha sido necesario las referencias a JasperReports y DynamicJasper. Por tanto en cuanto a la configuración se refiere no podía ser más sencillo ya que como he comentado anteriormente DynamicJasper se integra con Apache Maven y sólo tendremos que incluir en nuestro fichero de dependencias (pom.xml) las siguientes líneas:

```
view plain  print  ?
01. <dependency>
02.     <groupId>ar.com.fdvs</groupId>
03.     <artifactId>DynamicJasper</artifactId>
04.     <version>3.0.13</version>
05. </dependency>
06.
07. <dependency>
08.     <groupId>jasperreports</groupId>
09.     <artifactId>jasperreports</artifactId>
10.     <version>3.5.2</version>
11. </dependency>
```

## 5. Un par de ejemplos

**ReportBase.java** : Esta clase es la que contiene la lógica necesaria para generar el informe propieamente dicho y hace uso de JaperReports y DynamicReports. El resto de clases extiende de ésta e implementan la contrucción del informe a nivel de diseño y obtienen la fuente de datos.

Por el resto creo que no hay mucho que explicar ya que el código es bastante sencillo así que solo echadle un ojo.

view plain print ?

```
01. public abstract class ReportBase {
02. 
03.     protected static final Log log = LogFactory.getLog(ReportBase.class);
04.     protected JasperPrint jp;
05.     protected JasperReport jr;
06.     protected Map params = new HashMap();
07.     protected DynamicReport dr;
08. 
09. 
10.     public abstract DynamicReport buildReport() throws Exception;
11. 
12.     public abstract JRDataSource getDataSource();
13. 
14.     public void generateReport() throws Exception {
15. 
16.         dr = buildReport();
17. 
18.         /**
19.            * Ontenemos la fuente de datos en base a una colleccion de objetos
20.            */
21.         JRDataSource ds = getDataSource();
22. 
23.         /**
24.            * Creamos el objeto JasperReport que pasamos como parametro a
25.            * DynamicReport,junto con una nueva instancia de ClassicLayoutManager
26.            * y el JRDataSource
27.            */
28.         jr = DynamicJasperHelper.generateJasperReport(dr, getLayoutManager(), params);
29. 
30.         /**
31.            * Creamos el objeto que imprimiremos pasando como parametro
32.            * el JasperReport object, y el JRDataSource
33.            */
34.         log.debug("Filling the report");
35.         if (ds != null){
36.             jp = JasperFillManager.fillReport(jr, params, ds);
37.         }else{
38.             jp = JasperFillManager.fillReport(jr, params);
39.             log.debug("Filling done!");
40.             log.debug("Exporting the report (pdf, xls, etc)");
41.         }
42. 
43.         exportReport();
44. 
45.         log.debug("test finished");
46. 
47.     }
48. 
49. 
50. 
51.     protected LayoutManager getLayoutManager() {
52.         return new ClassicLayoutManager();
53.     }
54. 
55. 
56. 
57.     protected void exportReport() throws Exception {
58. 
59.             final String path=System.getProperty("user.dir")+ "/target/reports/" + this.getClass().getCanonicall
60. 
61.             log.debug("Exporing report to: " + path);
62.             JRPdfExporter exporter = new JRPdfExporter();
63.             File outputFile = new File(path);
64.             File parentFile = outputFile.getParentFile();
65.             if (parentFile != null)
66.                         parentFile.mkdirs();
67. 
68.             FileOutputStream fos = new FileOutputStream(outputFile);
69. 
70.             exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
71.             exporter.setParameter(JRExporterParameter.OUTPUT_STREAM, fos);
72. 
73.             exporter.exportReport();
74.             log.debug("Report exported: " + path);
75. 
76.     }
77. 
78. 
79.     protected void exportToJRXML() throws Exception {
80.         if (this.jr != null){
81.             DynamicJasperHelper.generateJRXML(this.jr, "UTF-
8",System.getProperty("user.dir")+ "/target/reports/" + this.getClass().getCanonicalName() + ".jrxml");
82. 
83.         } else {
84.             DynamicJasperHelper.generateJRXML(this.dr, this.getLayoutManager(), this.params, "UTF-
8",System.getProperty("user.dir")+ "/target
85.             /reports/" + this.getClass().getCanonicalName() + ".jrxml");
86.         }
87.     }
88. 
89. 
90. }
```

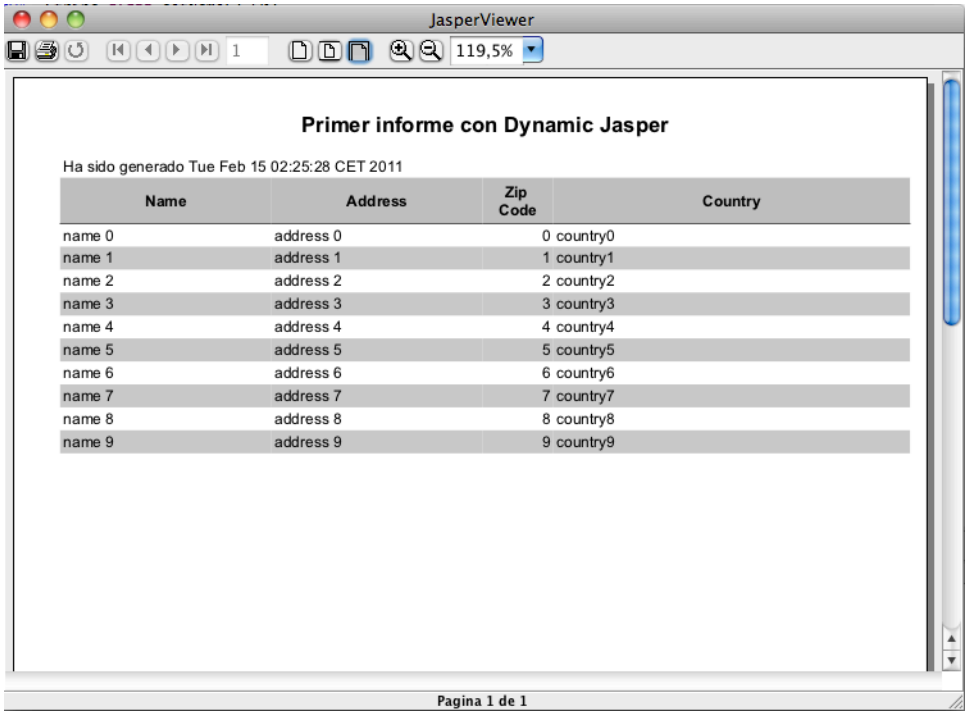
Ejemplo sencillo que muestra la configuración básica de un informe y sus columnas:

```
view plain  print  ?
01. public class SimpleReportOne extends ReportBase{
02.
03.
04.     public static void main(String[] args) throws Exception {
05.         SimpleReportOne simpleReportOne = new SimpleReportOne();
06.         simpleReportOne.generateReport();
07.         simpleReportOne.exportToJRXML();
08.         JasperViewer.viewReport(simpleReportOne.jp);     //finally display the report report
09.         JasperDesignViewer.viewReportDesign(simpleReportOne.jr);
10.     }
11.
12.     public DynamicReport buildReport() throws Exception {
13.
14.         FastReportBuilder drb = new FastReportBuilder();
15.
16.         drb.addColumn("Name", "name", String.class.getName(),30)
17.             .addColumn("Address", "address", String.class.getName(),30)
18.             .addColumn("Zip Code", "zipcode", Integer.class.getName(),10)
19.             .addColumn("Country", "country", String.class.getName(),50)
20.             .setTitle("Primer informe con Dynamic Jasper")
21.             .setSubtitle("Ha sido generado " + new Date())
22.             .setPrintBackgroundOnOddRows(true)
23.             .setUseFullPageWidth(true);
24.
25.         return drb.build();
26.
27.     }
28.
29.     public JRDataSource getDataSource() {
30.         Collection<SimpleReportOneBean> dataOneReport = getDataOneReportMock();
31.         dataOneReport = SortUtils.sortCollection(dataOneReport,dr.getColumns());
32.
33.         //Create a JRDataSource, the Collection used
34.         JRDataSource ds = new JRBeanCollectionDataSource(dataOneReport);
35.         return ds;
36.
37.     }
38.
39.     private Collection<SimpleReportOneBean> getDataOneReportMock() {
40.
41.         SimpleReportOneBean elementToInclude;
42.         Collection<SimpleReportOneBean > dataResult=new ArrayList<SimpleReportOneBean>();
43.
44.         for (int i = 0; i < 10; i++) {
45.             elementToInclude=new SimpleReportOneBean();
46.             elementToInclude.setName("name "+ i);
47.             elementToInclude.setAddress("address "+i);
48.             elementToInclude.setZipcode(i);
49.             elementToInclude.setCountry("country" + i);
50.             dataResult.add(elementToInclude);
51.         }
52.
53.
54.         return dataResult;
55.     }
56.
57. }
```

Como vemos la aplicación se encarga de generar un informe en función del diseño inicial que definimos en el la implementación del método buildReport(). En dicho método definimos cada una de las columnas indicando como parámetros:

- literal que se visualizará en la cabecera
- nombre de la propiedad del objeto que representan cada una de las lineas del informe
- tipo de objeto de la propiedad
- ancho de cada una de las columnas

Además estamos indicando un título, que se pinten las filas con colores de manera alternativa y que se ocupe todo el ancho posible de la página. Para poder ver el resultado pulsamos sobre las clase y ejecutar como --> Java Application . A continuación se mostrará el informe en una ventana como esta:



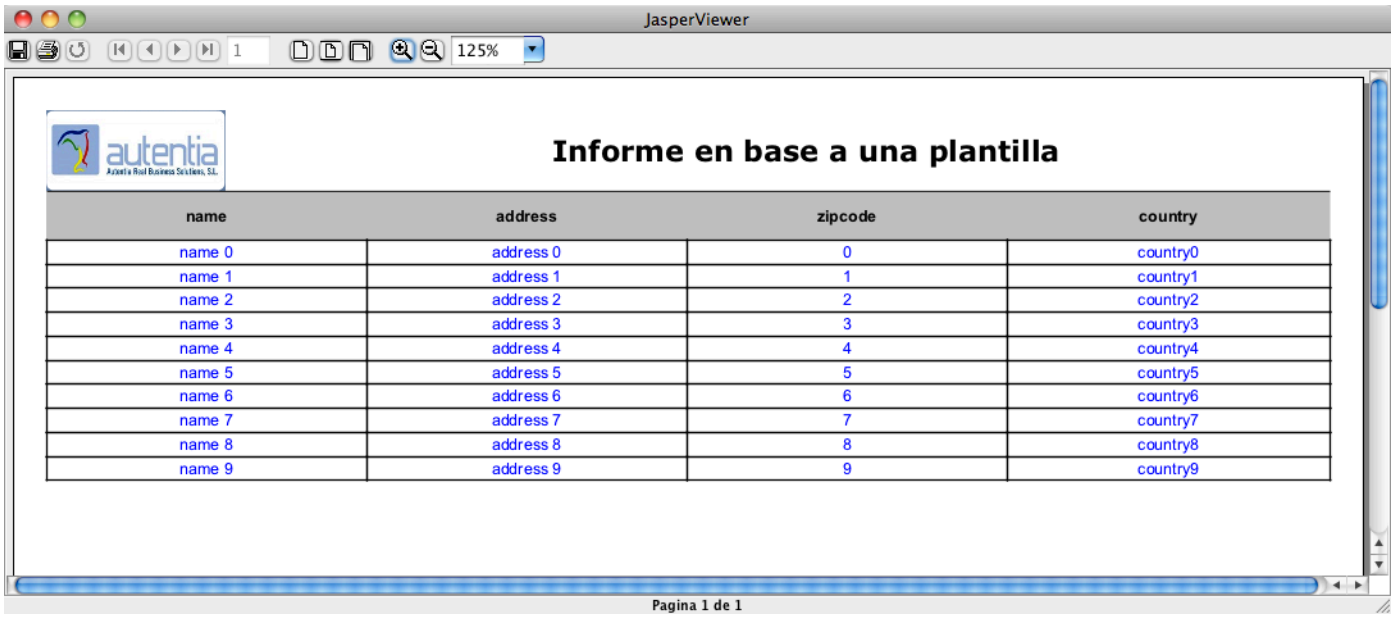

En el siguiente ejemplo vamos a ver cómo generar un informe en base a una plantilla predefinida y creada desde iReports. Esta plantilla sólo tiene campo para la imagen y otro para el título aunque podríamos definir todos los elementos comunes e inalterables del informe:

```
view plain  print  ?
01. public class SimpleReportTemplate extends ReportBase {
02.
03.     public static void main(String[] args) throws Exception {
04.         SimpleReportTemplate reportTemplate=new SimpleReportTemplate();
05.         reportTemplate.generateReport();
06.         reportTemplate.exportToJRXML();
07.         JasperViewer.viewReport(reportTemplate.jp); //finally display the report report
08.         JasperDesignViewer.viewReportDesign(reportTemplate.jr);
09.
10.     }
11.
12.     @Override
13.     public DynamicReport buildReport() throws Exception {
14.         // Definimos las columnas del informe
15.         final String[] columns = {"name","address","zipcode","country"};
16.         // Definimos el titulo y el logo como parametros del informe
17.         params =populateParametersMap("Informe en base a una plantilla");
18.         // Construimos el informe dinámico en funcionde las columnas y la lista de objetos que se p
19.         final DynamicReport dynamicReport = new ReflectiveReportBuilder(getDataOneReportMock(), columns).
20.         setTemplateFile(System.getProperty("user.dir")+ "/src/main/resources/dynamicJasperReportTemplate.jrxml"
21.
22.         // Definimos un estilo para el detalle de las properties
23.         Style styleColumns = new Style();
24.         styleColumns.setHorizontalAlign(HorizontalAlign.CENTER);
25.         styleColumns.setTextColor(Color.BLUE);
26.         styleColumns.setBorder(Border.PEN_1_POINT);
27.
28.         // Definimos algunas propiedades para las distintas columnas del informe
29.         for (int i = 0; i < columns.length; i++) {
30.             String name = columns[i];
31.             final PropertyColumn propertyColumn = (PropertyColumn)dynamicReport.getColumns().get(i);
32.             propertyColumn.setTitle(name);
33.             propertyColumn.setBlankWhenNull(true);
34.             propertyColumn.setWidth(25);
35.             propertyColumn.setStyle(styleColumns);
36.         }
37.
38.
39.         return dynamicReport;
40.     }
41.
42.
43.     private Map<String, String> populateParametersMap(String title) {
44.
45.         final Map<String, String> parameters = new HashMap<String, String>();
46.         final String pathLogo=System.getProperty("user.dir")+ "/src/main/resources/autentialogo.png";
47.
48.             parameters.put("REPORT_LOGO", new File(pathLogo).getPath());
49.             parameters.put("REPORT_TITLE", title);
50.         return parameters;
51.     }
52.
53.
54. }
```

Como podemos observar la única diferencia con el ejemplo anterior es la implementación del builReport(), donde definimos un diseño distinto en algunos detalles como por ejemplo:

- Línea 19 : Construimos el informe en base a las columnas que sólo están definidas y la fuente de datos. Lo importante aquí es el setTemplateFile donde indicamos la plantilla básica que hemos diseñado en anterioridad.
- Línea 23-26: Se define un estilo que se aplicará al detalle de cada una de las columnas. (Alineación, color del texto y border de cada campo).
- Línea 29-36: Es donde se definen unas propiedades mínimas para cada propiedad incluyendo los estilos.

Si ejecutamos el programa veremos el siguiente informe:




| name | address | zipcode | country |
|---|---|---|---|
| name 0 | address 0 | 0 | country0 |
| name 1 | address 1 | 1 | country1 |
| name 2 | address 2 | 2 | country2 |
| name 3 | address 3 | 3 | country3 |
| name 4 | address 4 | 4 | country4 |
| name 5 | address 5 | 5 | country5 |
| name 6 | address 6 | 6 | country6 |
| name 7 | address 7 | 7 | country7 |
| name 8 | address 8 | 8 | country8 |
| name 9 | address 9 | 9 | country9 |

## 6. Conclusiones

Con este simple ejemplo hemos podido comprobar lo sencillo y rápido que puede resultar generar un informe con DynamicJasper. Y lo mejor de todo es que esto es sólo un adelanto ya que la librería nos ofrece un gran número de posiblidades que intentaré ir desgranando en próximos tutoriales.

Como ya he comentado en alguna ocasión, desde Autentia os animamos a que probéis todas aquellas herramientas que puedan hacer vuestra vida como desarrolladores un poco más facil. Intentad no caer en la tentación de programar todo ya que en muchísimas ocasiones hay muy buenas librerías que hacen justo lo que necesitamos y además con una buena calidad como DynamicJasper.

Espero que os sirva de utilidad.

Un saludo.

Saúl García Díaz

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

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

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

## COMENTARIOS