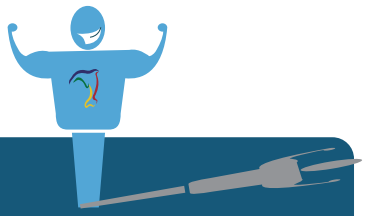
autentia

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...

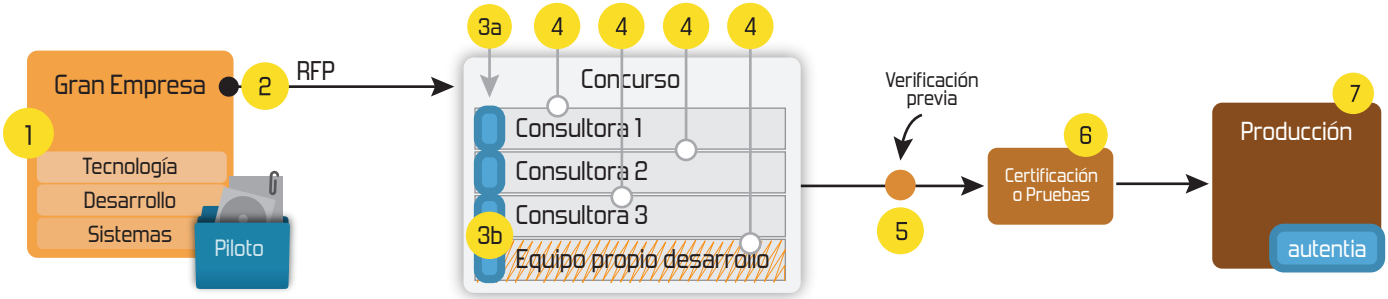
## 1. Desarrollo de componentes y proyectos a medida



## 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**

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 10 veces Descargar en PDF

# Titulo tutorial

## 0. Índice de contenidos.



## 1. Entorno

Este tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Intel Core 2 Duo CPU T8100 @ 2.10GHz x 2
- Sistema Operativo: Ubuntu 12.04 LTS x32
- Sublime Text 2

## 2. Introducción

A través de esta guía vamos a hacer una pequeña aplicación con Ember.js cubriendo algunos de los aspectos básicos del framework, de manera que podamos hacernos una idea de lo que Ember.js puede ofrecernos. Algunos de los aspectos básicos de ember están tratados en esta otra guia, es recomendable leerla antes para hacernos una idea de lo que trata de aportar Ember.js

Para realizar esta guía vamos a usar la versión de Ember 1.0.0-RC.2, y usaremos el Starter Kit que podemos descargar desde la página de Ember, el cual tiene la siguiente estructura:

```
starter-kit-1.0.0-rc.2
  css
    normalize.css
    style.css
  js
    libs
      ember-1.0.0-rc.2.js
      handlebars-1.0.0-rc.3.js
      jquery-1.9.1.js
    app.js
  index.html
  TODO.txt
```

El grueso de nuestra aplicación residirá en el archivo **app.js**, mientras que las vistas las almacenaremos en el archivo **index.html**, Handlebars ( el encargado de las vistas en nuestra aplicación ) también permite usar archivos externos para



**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**

» Descubriendo Responsive Web Design

» Primeros pasos para conocer Emberjs

» Webs que ofrecen plantillas CSS gratuitas

almacenar las vistas, pero de momento no vamos a hacer uso de esa funcionalidad.

El primer paso sería crear nuestra App, y asignarla al namespace que deseemos, en este caso será al namespace App.

Por defecto Ember nos crea una Ruta, un Controlador y una Vista, sin nosotros tener que hacer nada, en la siguiente imagen podemos hacernos una idea de la convención de nombres que sigue Ember.js, es importante saber que, aunque nosotros no hayamos hecho ninguno de esos componentes Ember los está haciendo por nosotros.

## 3. Las partes de nuestra Aplicación

### 3.1. El Objeto Router

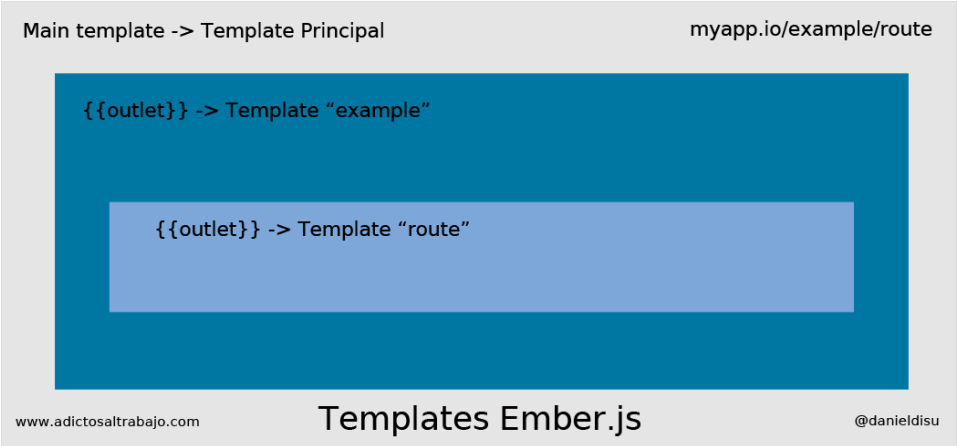
El objeto Router es el encargado de mantener el estado de nuestra aplicación a través de la URL, una vez mas, sino le decimos lo contrario Ember, usará el nombre de la ruta como ruta a capturar en la URL ( es decir, si no declaramos ninguna ruta a una ruta llamada lista, automáticamente se le asociará la URL **/lista**.

| URL | Route Name | Controller | Route | Template |
|---|---|---|---|---|
| / | index | IndexController | IndexRoute | index |
| /about | about | AboutController | AboutRoute | about |
| /favs | favorites | FavoritesController | FavoritesRoute | favorites |

De: Guia oficial de ember

### 3.2. La Vista

En cuanto a las **Vistas** o **Templates**, Ember usará automáticamente **dos** templates, uno sin nombre y otro con el nombre **index**, en la primera solo tendremos un título junto con el código {{outlet}} el cual Ember sustituirá por el contenido del template que pertenezca a la ruta actual.




### 3.2. El Modelo

En cuanto al **Modelo**, Ember nos provee de una clase llamada Ember.Data que nos permite sincronizar nuestros Objetos del modelo en la parte cliente con nuestro servidor, para esta aplicación nosotros vamos a hacer nuestros propios modelos sin sincronizarlos al servidor, ya que nos vamos a centrar en cómo interactúan las diferentes partes de la aplicación-

### 3.2. El Controlador

Por último los **Controladores** que en Ember pierden importancia y se usan meramente como "**Proxys**" para manipular los datos del modelo antes de mostrarlos por la **Vista**.

## 4. La Aplicación

### 4.1. El Índice

La aplicación que vamos a hacer nos va a permitir insertar un nombre, el cual tendremos que repetir para comprobar que se ha escrito correctamente, lo cual nos permitirá simular la funcionalidad de un registro y ver como Ember computa dinámicamente si los valores de los campos son iguales ( sin tener que entrar al DOM y ir actualizando manualmente ), y **actualizará automáticamente** el estado de la aplicación, en caso de ser correcto, o mostrará un mensaje de error en caso contrario.

Lo primero que vamos a hacer va a ser declarar el Objeto de modelo

```
1     App.User = Ember.Object.extend({
2     name : "",
3     nameRepeated : "",
4     isInvalid : function(){
5         var invalid = true;
6         var firstName = this.get("name");
7         var repeatName =  this.get("nameRepeated");
8         if(firstName != "" && repeatName != "" && (firstName == repeatName)){
9             invalid = false;
10        }
11        return invalid
12    }.property("name","nameRepeated")
13 })
```

» Resolver problema LockTimeoutException en Spring Web Flow

» WebSockets con Stomp y ActiveMQ: ¿chateamos?

**Linea 1:** Creamos el "tipo" de objeto User extendiendo el prototype Object de Ember que nos aportará mucha de la "magia" que hace posible que todo se actualice en tiempo real.

**Linea 2-3:** Declaramos dos atributos simples que tendrá el Objeto

**Linea 4-11:** Una de las mayores ventajas de Ember, los **atributos computados**, nos permite la posibilidad de crear un atributo que se recalcule automáticamente cada vez que cambie alguno de los atributos por los que está compuesto ( todos o alguno de ellos, en la línea 11 se le indica los atributos que quieres observar )

En el template **Index** vamos a tener el siguiente código:

```
1  script type="text/x-handlebars" data-template-name="index">
2      {{#if App.mainUser.isInvalid }}
3          <div class="error">
4              <p>Los nombres no coinciden</p>
5          </div>
6      {{/if}}
7      <div class="inputGroup">
8          <label> Introduce tu nombre: </label> {{view Ember.TextField valueBinding="App.ma
9      </div>
10     <div class="inputGroup">
11         <label> Repite tu nombre: </label> {{view Ember.TextField valueBinding="App.mainU
12     </div>
13 {{#linkTo "menu"}}<input type="button" {{bindattr="" disabled="App.mainUser.isInvalid" }}
```

**Linea 1:** Podemos ver como estamos llamando al template index, en este caso cogerá automáticamente la ruta "/".

**Linea 2:** Usamos una función condicional de Handlebars, en caso de que exista esa variable Y sea true, mostraremos un DIV con la clase error, si quisiéramos internacionalizar o poner varios tipos de errores solo tendríamos que mostrar por pantalla un String que contuviese el error y Ember se encargaría de mostrarlo.

**Linea 8:** Usamos uno de los métodos que nos proporciona el objeto **View** de Ember, el cual nos creará un Input Text cuyo valor estará asociado directamente a un atributo de un objeto del modelo que indiquemos.

**Linea 13:** En esta línea podemos observar dos cosas, el método {{#linkTo }} el cual creará un enlace a la ruta que le pongamos sin necesidad de saber nosotros donde nos encontramos. Tambien tenemos el método {{bindAttr disabled}} el cual nos deshabilitará el botón en caso de que la variable que le pasemos sea true, hay que tener en cuenta que esto se actualizará automáticamente cada vez que la variable cambie de valor.

Por ultimo solo necesitamos crear la ruta "Menu que hemos indicado en la línea 13, para ello le tenemos que pasar al objeto Router un JSON con todas las rutas que tenemos en nuestra aplicación, de momento solo tenemos dos rutas.

```
1  App.Router.map(function() {
2      this.route("index, {path: "/ } ) // Está ruta no hace falta declararla ya que ember lo
3  // hace por nosotros
4      this.resource("menu",{path : "/menu" },function(){
5
6      })
7  });
```

Cómo se puede observar,hemos declarado la ruta menú, como un recurso, el cual a su vez puede tener otras rutas.

### 4.2. El Menú

El Template:

```
1  script type="text/x-handlebars" data-template-name="menu/index">
2  <div class="welcomeBox">
3      <h3>Bienvenido {{App.mainUser.name}}</h3>
4  </div>
5  <div class="listBox">
6      {{#linkTo "menu.selectColor"}}<input type="button" value="Ver Lista Colores">{{/linkTo
7      {{#linkTo "menu.watchColor"}}<input type="button" value="Ver Color Seleccionado">{{/li
8  </div>
```

```
1  this.resource("menu",{path : "/menu" },function(){
2      this.route("selectColor");
3      this.route("watchColor");
4  })
```

Generamos las tres rutas, en este caso estará la primera ruta "/menu", y las dos subrutas, /menu/selectColor y /menu/watchColor

### 4.3. El Selector de Color:

Lo siguiente será la parte necesaria de elegir un color y que se actualice en toda la aplicacion donde necesitemos usar el color utilizado ( pensar que el color puede ser cualquier opción que pueda llegar a tener nuestra aplicacion )

```
1  App.Color = Ember.Object.extend({
2      name: ""
3  });
4
5  App.selectedColorName = "red";
6
7  App.MenuSelectColorController = Ember.ArrayController.extend({
8      changeSelectedColor : function(selectedColor){
9          /*Cambiando de  App.selectedColorName  a selectedColor.get("name")*/
10         App.set("selectedColorName", selectedColor.get("name"));
11     },
12 })
13
14 App.MenuSelectColorRoute = Ember.Route.extend({
15     model : function(){
16         App.colors=[];
17         var color1 = App.Color.create({name: 'red'})
18         var color2 = App.Color.create({name: 'yellow'})
19         var color3 =App.Color.create({name: 'blue'})
```

```
            App.colors.push(color1)
            App.colors.push(color2)
            App.colors.push(color3)
        return App.colors;
    }
});
```

**Linea 1-3:** Declaramos un objeto de tipo color que va a tener un atributo nombre.

**Linea 5:** Por defecto el color seleccionado en la aplicacion será el rojo, como se puede ver, podemos agregar cualquier atributo o funcion al Namespace App, de manera que tendremos acceso a el desde cualquier parte de la apicación, incluidos los templates.

**Linea 7-11:** Le añadimos un metodo al controlador de la ruta Menu/SelectColor, como podemos ver le estamos pasando por parametro algo, mas adelante veremos como el template es el que se encargará de pasarle el color en el que se haga click

**Linea 14 -24 :** Aqui estamos declarando el Modelo que va a usar la ruta/vista/controlador MenuSelectColor, simplemente le estamos devolviendo un array de colores, que tanto la vista como mas adelante el controlador, se encargaran de pedir. En una aplicación real, aqui tendriamos que tener un intermediario que se encargase de hacer una peticion REST a un servidor.

Por ultimo hacemos los templates que se encargaran de mostrar los colores y llamar a los eventos correspondientes cuando estos sucedan

```
script type="text/x-handlebars" data-template-name="menu/selectColor">
    <h2> Seleccione un color </h2>
    {{#each App.colors}}
        <div {{bindattr="" class=":colorBox name" }}="" {{action="" 'changeselectedcolor'
    {{/each}}

    <h2> Color Seleccionado: </h2>
    <div class="seletedColorBox">
        <div {{bindattr="" class=":selectedColor App.selectedColorName" }}=""></div>
    </div>

    {{#linkTo 'menu'}}<button class="backMenu">Seleccionar color</button>{{/linkTo}}
/script>

script type="text/x-handlebars" data-template-name="menu/watchColor">
    <h2> Color Seleccionado: </h2>
    <div class="seletedColorBox">
        <div {{bindattr="" class=":selectedColor App.selectedColorName" }}=""></div>
    </div>
    {{#linkTo 'menu'}}<button class="backMenu">Volver al Menu</button>{{/linkTo}}
/script>
```

**Linea 3-5:** En este fragmento de codigo tenemos que destacar la funcion {{#each}} que nos proporciona handlelbars, que nos permite mostrar un Array y por cada objeto del array hacer algo, recordar como antes hemos creado un Array de colores y lo hemos declarado como el modelo para esta vista/ruta. En este caso estamos creando un rectangulo por cada color, con una accion interna, la cual llamará al metodo changeSelectedColor del controlador de la ruta, y le pasaremos por parametro "this", que será el modelo ( color ) que estamos pintando.

**Linea 9:** Aquí usamos otro helper de Handlebars, el cual nos va a permitir enlazar un dato que se está mostrando en la vista con un dato del modelo/aplicación, esto nos permite que si este dato cambia en el modelo, automaticamente cambiará en la vista y viceversa, lo cual nos ahorra mucho esfuerzo a la hora de tener que buscar cambios y actualizar la vista en caso de que estos existan, en este caso concreto el cambio solo se propagará del modelo a la vista, ya que viceversa no se podrá cambiar.

**Linea 9 y 18:** Podemos comprobar como hemos escrito dos veces ":selectedColor", esto simplemente es para indicarle a Handlebars que queremos añadir la siguiente palabra literalmente, no queremos que busque ninguna variable en el modelo con ese nombre, esto nos sirve para darle nombres fijos a las clases para luego aplicarle estilos, pero a la vez teniendo otra clase variable.

## 5. Conclusiones

Como hemos podido comprobar Ember es un framework MVC potente, que nos permitirá hacer Aplicaciones Single Page con relativa facilidad una vez le hemos cogido el truco, a pesar de la dificultad inicial, y la falta ( o mas bien dispersion ) de documentación actualizada, puede hacer dificil los primeros pasos con Ember, pero las muchas ventajas que nos ofrece hacen que merezca la pena.

En cuanto al futuro, el equipo de Ember, está dando los ultimos retoques a la libreria Ember Data, la cual nos va a permitir ( a modo de ORM en el cliente ), actualizar automaticamente los datos del modelo en el cliente con el servidor, sin necesidad de programar nosotros nuestra propia solucion, lo que puede hacer del desarrollo Frontend una verdadera delicia.

Podemos encontrar todo el codigo de la aplicación en GitHub

### A continuación puedes evaluarlo:

Regístrate para evaluarlo

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

Share | 0

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

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