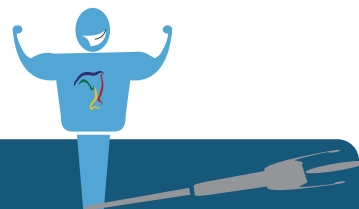


¿Qué ofrece Autentia Real Business Solutions S.L?

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

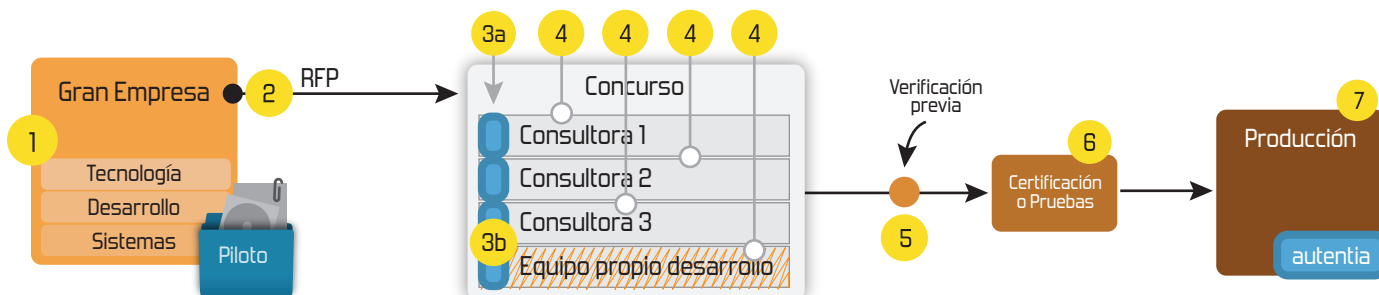
1. Desarrollo de componentes y proyectos a medida



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



Entra en Adictos a través de  

E-mail

Contraseña

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



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

» Estás en: [Inicio](#) [Tutoriales](#) [Como testear aplicaciones en Ember.js](#)



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: 2013-04-29

Tutorial visitado 13 veces [Descargar en PDF](#)

Como testear una aplicacion de Ember.js con Jasmine

0. Índice de contenidos.

- 1. Entorno
- 2. Introducción
- 3. Mavenizando nuestra Aplicación
- 4. Haciendo los test con Jasmine
- 5. Conclusiones

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

Los cambios que han sucedido en los últimos años han propiciado el crecimiento del uso de Javascript, así como el tamaño de los programas escritos con Javascript, lo cual nos ha obligado a replantearnos la manera en la que programamos con Javascript.

Por ello, y aprovechando los conocimientos adquiridos en el lado servidor, se está intentando traspasar las buenas practicas ya logradas al lado del cliente.

Practicas como pueden ser la implementación de Test Unitarios y de Integración, el uso de herramientas de automatización a la hora de desplegar las aplicaciones y demás herramientas para asegurar la calidad del código.

Es importante comprender que Javascript es un lenguaje dinámico, de tipado débil, lo cual no lo hace ni mejor ni peor, y hay que tratarlo como tal, para lo bueno y para lo malo.

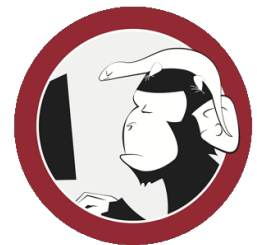
Este tutorial hará uso de diferentes herramientas y frameworks para conseguir esas metas, las herramientas que vamos a usar son las siguientes:

- Ember.js: Es un framework MVC que nos aporta abstracciones a la hora de conseguir seguir el patrón MVC, además de otras herramientas que facilitaran el desarrollo de aplicaciones SPA (Single Page Application) . Si quieres saber mas sobre Ember puedes echar un vistazo a los siguientes tutoriales: [Primeros Pasos con Ember](#) y [Primera Aplicacion Ember](#)
- Require.js: Require.js es un framework de Javascript que nos permite modularizar nuestro código, de manera lógica y física, aparte de ofrecernos la posibilidad de cargar módulos "on-demand", lo que nos vendrá muy bien a medida que nuestra aplicación vaya creciendo. Si quieres saber mas sobre Require.js, o sobre como hemos aplicado Require a nuestro proyecto, puedes mirar los siguientes tutoriales: [Introducción a Require](#) y [Modularizar tu aplicacion de Ember con Require](#)
- Jasmine: Jasmine nos ofrece una serie de herramientas con las que podremos realizar nuestros test, además de proporcionarnos la posibilidad de agruparlos y nos ofrecerá distintos plugins con los que acoplar nuestros test a nuestro sistema de despliegue automático favorito. Puedes ver un tutorial sobre Jasmine en el siguiente enlace: [Primeros pasos con Jasmine](#)
- Maven: Maven es una herramienta que nos permite gestionar y construir proyectos en Java, este tutorial se centrará en como instalar y configurar el plugin de Jasmine para que pase los test de Javascript a la vez que los de Java a la hora de desplegar nuestra aplicación.

3. Mavenizando nuestro proyecto

Lo primero que vamos a hacer es integrar nuestro Proyecto dentro de un proyecto de Maven, nuestra aplicación residirá en "src/main/webapp/scripts/app" siendo main.js nuestro punto de entrada, tendremos dos archivos de configuración,

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 aprendizaje a empresario

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

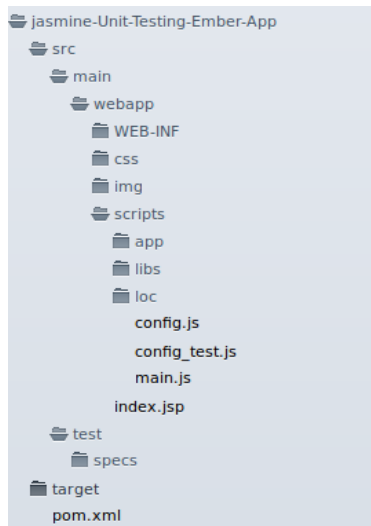
[Histórico de noticias](#)

Últimos Tutoriales

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

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

uno para el entorno de producción y otro para el de pruebas.



Por ultimo, los test residirán en el archivo test/specs, en el pom.xml tenemos que añadir el plugin de Jasmine y la configuración específica del plugin.

```

1  <plugins>
2  <plugin>
3  <groupId>com.github.searls</groupId>
4  <artifactId>jasmine-maven-plugin</artifactId>
5  <version>1.3.1.1</version>
6  <executions>
7  <execution>
8  <goals>
9  <goal>test</goal>
10 </goals>
11 </execution>
12 </executions>
13 <configuration>
14 <!-- Le indicamos al plugin de Jasmine que queremos realizar los test con Require.js
15 <specrunnerTemplate>REQUIRE_JS</specrunnerTemplate>
16 <!-- Esta directiva indica donde se encuentran los archivos que conforman nuestra aplicación
17 <jssrcdir>${project.basedir}/src/main/webapp/scripts</jssrcdir>
18 <!-- La ruta por defecto donde buscará los test es src/test/javascript, si queremos
19 <jstestsourcedir>${project.basedir}/src/test/specs</jstestsourcedir>
20 <!-- Esta directiva indica donde está el archivo de configuración que buscará require.js
21 en el archivo config deberemos poner las rutas a partir de nuestro directorio java
22 en este caso hemos creado un archivo config para el entorno de testing, de manera que
23 <!--<customRunnerConfiguration>${project.basedir}/src/main/webapp/scripts/config_test.js
24 <!-- Esta directiva indica al plugin de Jasmine donde se encuentra la librería require.js
25 <preloadSources>
26 <source>${project.basedir}/src/main/webapp/scripts/libs/requirejs/2.1.2/require.js
27 <source>${project.basedir}/src/main/webapp/scripts/libs/jquery/1.9.1/jquery.js
28 <source>${project.basedir}/src/main/webapp/scripts/libs/handlebars/1.0.rc.3/handlebars.js
29 <source>${project.basedir}/src/main/webapp/scripts/libs/ember/1.0.0-rc.2/ember.js
30 <source>${project.basedir}/src/main/webapp/scripts/config_test.js
31 </preloadSources>
32 </configuration>
33 </plugin>
34 </plugins>

```

Como nuestro proyecto hace uso de Require.js para cargar nuestros módulos, hay una dificultad añadida a la hora de usar Jasmine, así que hacemos uso de la opción que nos brinda el plugin de Maven.

Por ultimo pre-cargamos nuestra librerías para evitarnos problemas a la hora pasar los tests.

Una vez tengamos todo listo, hacemos un "mvn install" para que se nos instalen las dependencias necesarias para el plugin.

Para pasar los test de Javascript tenemos dos opciones, pasarlos junto a los test de Java, usando la directiva "mvn test" o bien usar la propia directiva que nos ofrece el plugin de Jasmine "mvn jasmine:bdd".

A la hora de hacer TDD/BDD, la segunda opción se presenta muy útil, pues nos inicia un servidor, en el cual podemos pasar todos nuestros test y ver los resultados en tiempo real, actualizándose cuando hacemos cambios en cualquier test o archivo de la aplicación.

Probamos con el comando mvn jasmine:bdd que todo se ha instalado correctamente, en la consola nos debería mostrar la dirección en la que se ha creado nuestro servidor el cual nos permitirá ver los test, en un principio deberíamos ver algo así:

» [Uso de Requirejs para modularizar una App creada con Emberjs](#)

» [Instalación de Redmine \(Bitnami\) e integración con Subversion.](#)

» [Introducción a Require.JS](#)

Últimos Tutoriales del Autor

» [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

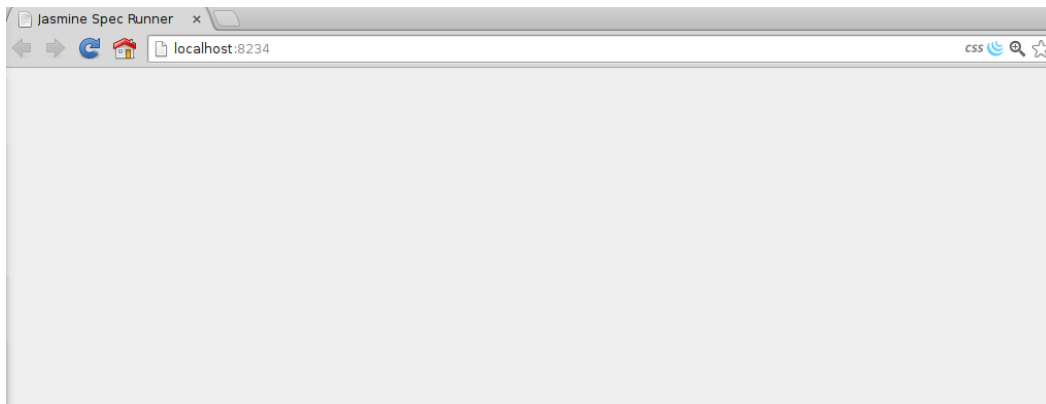
[Comercial - Compras - ALICANTE.](#)

2011-07-12

[Otras Sin catalogar - MADRID.](#)

2011-07-06

[Otras Sin catalogar - LUGO.](#)



¡Es hora de hacer nuestro primer test!

4. Haciendo Test con Jasmine para nuestra aplicación con Ember y Require

Es importante tener en cuenta que al estar usando Require y Ember, la manera en la que tenemos que hacer los test se tiene que adaptar a las tecnologías usadas. Para testear con Require, es recomendable convertir los test en módulos de Require, en los cuales puedes declarar las dependencias y así mantener un nivel de encapsulación óptimo para el entorno de pruebas que nos asegurará mejores resultados.

El primer test que vamos a hacer va a comprobar que se están cargando todas las librerías.

Para ello crearemos un fichero Javascript en nuestra carpeta de test, en el ejemplo la carpeta se encuentra en "src/main/test/specs"

```

1  define(function() {
2    describe("Comprobacion de que las librerias se hayan cargado correctamente", function()
3      it("Ember se ha cargado", function() {
4        expect(Ember).not.toBeUndefined();
5      });
6      it("jQuery se ha cargado", function() {
7        expect($).not.toBeUndefined();
8      });
9      it("Handlebars se ha cargado", function() {
10       expect(Handlebars).not.toBeUndefined();
11     });
12     it("Require se ha cargado", function() {
13       expect(require).not.toBeUndefined();
14     });
15   });
16 });
17

```

Gracias a la sintaxis de Jasmine, los test son muy legibles, y auto-explicativos, lo cual facilita mucho la tarea de compartir los test con otras personas.

Si recargamos el navegador, la pagina que nos había preparado Maven ahora debería mostrar algo así:



A la hora de hacer test unitarios de nuestras aplicaciones de Ember, hay que tener en cuenta como funciona Ember, ya que al iniciar la aplicación, Ember crea una serie de "Listeners" de eventos que tienen un numero alto de probabilidades de chocar con otros frameworks y herramientas como pueden ser Maven o Jasmine.

Por ello Ember provee de un metodo llamado deferReadiness(), que nos permitirá preparar la aplicación para ser testeada, pero sin interferir con el resto de frameworks, en el siguiente ejemplo, podemos ver como podríamos testear los métodos de un controlador de Ember.

El codigo del controlador a testear lo podemos encontrar [aquí](#) , y el codigo del test sería el siguiente:

```

1  define(['controllers/MenuSelectColorController'], function(menuSelectColorController) {
2
3    describe("Conjunto de test que prueban el MenuSelectController", function(){
4
5      var stateManager = jasmine.createSpyObj('stateManager', ['set', 'saveState']);
6
7      var dummyObject = {};
8
9      window.App = Ember.Application.create({
10        ColorsController : dummyObject,
11        ColorController : dummyObject
12        menuSelectColorController : menuSelectColorController

```

```

13    })
14
15    App.stateManager = stateManager;
16
17    /*
18     By default, calling `Ember.Application.create()` will automatically initialize
19     your application by calling the `Ember.Application.initialize()` method. If
20     you need to delay initialization, you can call your app's `deferReadiness()`
21     method. When you are ready for your app to be initialized, call its
22     `advanceReadiness()` method.
23     */
24    App.deferReadiness();
25
26    var controlador = menuSelectColorController.create({
27      container : App.__container__
28    })
29
30    var color = {
31      name : "red"
32    }
33
34    it("Cuando el controlador recibe el evento lo propaga correctamente al State Manager", function() {
35      controlador.changeSelectedColor(color)
36
37      expect(stateManager.set).toHaveBeenCalled();
38      expect(stateManager.saveState).toHaveBeenCalled();
39    });
40  });
41 })

```

Línea 1 : Lo primero que hacemos es definir las dependencias que tendría el test, en este caso la única dependencia fuera de las librerías que estamos usando es la del propio controlador que vamos a testear.

Línea 5 : Creamos un espía que nos dirá si el Controlador que estamos usando usa los métodos del otro objeto.

Línea 7 : Creamos un objeto vacío que le pasaremos a Ember como si fuesen otros dos objetos. Debido a que el Controlador no usa en ningún momento ninguno de esos dos objetos, Ember no comprueba que esos objetos tengan métodos ni nada, solo comprueba que las variables estén definidas. (Estos dos objetos son necesarios para la vista, ya que son los encargados de mostrar la lista de colores, pero el controlador no los usa para nada)

Línea 9 : Iniciamos la aplicación en el namespace global, y le pasamos los objetos que hemos creado y el objeto que deseamos testear.

Línea 26 : Iniciamos manualmente el controlador.

Líneas 34-39 : Realizamos el test que comprobará el correcto funcionamiento de nuestro Controlador.

5. Conclusiones

Como se puede observar, hacer test para nuestras aplicaciones de Ember conlleva escribir un poco más de código de la cuenta, lo bueno es que Ember lleva en su interior una serie de Test de Integración que nos ayudarán a completar nuestra Suite de Test, de manera que nos podamos centrar más en los test unitarios de las funciones que añadamos nosotros.

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**:

» [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](#)