

¿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  

E-mail

Contraseña

Entrar

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



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

» Estás en: [Inicio](#) [Tutoriales](#) [Uso de Requirejs para modularizar una App creada con Emberjs](#)



Rafael Macías Rodríguez

Rafael 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: 2013-04-24

Tutorial visitado 260 veces [Descargar en PDF](#)

Uso de Requirejs para modularizar una App creada con Emberjs

0. Índice de contenidos.

- 1. Entorno
- 2. Introducción
- 3. ¿Qué vamos a utilizar?
- 4. Primer paso, Modularizar nuestra aplicación
- 5. Iniciar nuestra app
- 6. [Cambiar de vista](#)
- 7. Cambio de estado mediante controladores
- 8. Conclusiones

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

En este tutorial voy a enseñaros como utilizar Requirejs con una aplicación anteriormente creada con Emberjs, voy a utilizar como ejemplo la aplicación que utilizó Daniel en el anterior [tutorial](#).

3. ¿Qué vamos a utilizar?

Vamos a modularizar nuestra app con todas las ventajas que nos proporciona Requirejs. Dado que en esta aplicación utilizamos Ember la convención de nombres será especial, la iremos nombrando más adelante.

Para saber más de requirejs ver [tutorial de require](#).

Os pongo un código de ejemplo en el que trabajé con Daniel: [Enlace a github](#)

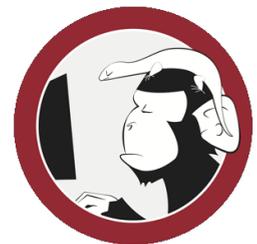
4. Primer paso, Modularizar nuestra aplicación

Si le echásteis un ojo a la anterior aplicación, podréis ver que toda la app estaba en el mismo fichero, y era lo que conocemos como 'espagueti' o 'chorizo'.

Y como es obvio no nos conviene tener un código de este tipo. Lo que vamos a hacer es separar nuestra app en cuanto a vistas, modelos, controladores y templates en diferentes módulos.

Lo que haremos sera crear un fichero para cada cosa, y separarlo en carpetas, que tendremos que definir después en un fichero 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 aprendiz a empresario](#)

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

[Histórico de noticias](#)

Últimos Tutoriales

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

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



Como podéis ver tenemos nuestro fichero config al lado de nuestra app 'main.js' lo que tomaremos como directorio raíz. Nota: Este no es el modelo de la App, sino el fichero principal desde el que crearemos la App.

Vamos a definir un módulo de configuración para utilizarlo después a la hora de crear nuestra app.

```

1 | define({
2 |   app_name: "App", //nombre de la aplicacion
3 |   shim : {
4 |     'ember' : {
5 |       deps: ['handlebars', 'jquery'],
6 |       exports: 'Ember'
7 |     }
8 |   },
9 |   paths : {
10 |     'App': 'app/main',
11 |     'models': 'app/models',
12 |     'views': 'app/views',
13 |     'controllers': 'app/controllers',
14 |     'routes': 'app/routes',
15 |     'templates': 'app/templates',
16 |     /*libs*/
17 |     'jquery': 'libs/jquery/1.9.1/jquery',
18 |     'handlebars': 'libs/handlebars/1.0.rc.3/handlebars',
19 |     'ember': 'libs/ember/1.0.0-rc.2/ember',
20 |     /*requirejs-plugins*/
21 |     'text': 'libs/requirejs-plugins/text',
22 |     'hbs': 'libs/requirejs-plugins/hbs',
23 |     'domReady': 'libs/requirejs-plugins/domReady',
24 |     'store' : 'app/store'
25 |   },
26 |   /*hbs plugin options*/
27 |   hbs: {
28 |     disableI18n: true,
29 |     templateExtension: "html"
30 |   }
31 | });
32 |

```

Como podéis ver aquí definimos (en path) todas las rutas que vamos a utilizar más adelante, a la hora de definir las dependencias. En este mismo código, línea 4, podemos ver como definimos una variable ember a la cual le asignamos unas dependencias. Deps.

5. Iniciar nuestra app

Una vez hecho esto, vamos a iniciar nuestra app. Vamos a definir unas dependencias con require al fichero de configuración:

```

1 | require(["config"], function (config) {
2 |   requirejs.config (config);
3 | });

```

Dentro de esta función de callback, crearemos nuestra app. Por lo que vamos a definir nuestras dependencias antes de hacerlo:

```

1 | require(["App", "ember", "store"], function (App, Ember, store) {
2 |
3 | });

```

Como podéis ver, por parámetros vamos a recibir estas dependencias, por lo que podremos utilizar todos los atributos, funciones... de estas.

» [Introducción a Require.JS](#)

» [Conexión con mysql desde iSeries](#)

» [Descubriendo Responsive Web Design](#)

Últimos Tutoriales del Autor

» [Primeros pasos para conocer Emberjs](#)

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.](#)

Vamos a crear ahora nuestra app:

```
1 | root[app_name] = App = Ember.Application.create(App, {LOG_TRANSITIONS: true});
```

Ahora vamos a definir nuestra App. Será un js que ejecutaremos al utilizar la dependencia del App, visto en el punto anterior.

```
1 | Define([
2 | //definimos las dependencias
3 | "app/router",
4 | "views/ApplicationView",
5 | "controllers/ApplicationController",
6 | "views/IndexView",
7 | "controllers/IndexController",
8 | "routes/IndexRoute",
9 | /*Aquí podemos incluir el resto de ruters, controladores, vistas... */
10 | ], function(){
11 |
12 | // Aquí inicializamos estas dependencias
13 | //Podríamos incluirlas como parámetros.
14 |
15 | /* Router */
16 | var Router = require("app/router");
17 |
18 | /* Global */
19 | var ApplicationView = require("views/ApplicationView");
20 | var ApplicationController = require("controllers/ApplicationController");
21 |
22 | /* Controladores globales */
23 | var UserController = require("controllers/UserController");
24 | var ColorController = require("controllers/ColorController");
25 | var ColorsController = require("controllers/ColorsController");
26 |
27 | /* Index */
28 | var IndexView = require("views/IndexView");
29 | var IndexController = require("controllers/IndexController");
30 | var IndexRoute = require("routes/IndexRoute");
31 |
32 | /*Aquí podemos incluir el resto de ruters, controladores, vistas... */
33 |
34 | /*Aplicamos el patron de módulos a nuestra app.*/
35 | /*Aquí le damos uso a esas dependencias
36 | var App = {
37 |   Router: Router,
38 |   ApplicationView: ApplicationView,
39 |   ApplicationController: ApplicationController,
40 |   UserController: UserController,
41 |   ColorController: ColorController,
42 |   ColorsController: ColorsController,
43 |   IndexView : IndexView,
44 |   IndexController : IndexController,
45 |   IndexRoute : IndexRoute,
46 |   /*Aquí podemos incluir el resto de ruters, controladores, vistas... */
47 | };
48 | return App; //Retornamos la App, ya que es un módulo para utilizarla fuera.
49 | });
```

Como veis definimos una serie de dependencias en el primer parámetro de define, luego las inicializamos, aunque podríamos recibirlas por parámetro de la función de callback, y saltarnos este paso.

Pero para tener ambos ejemplos y no tener demasiados parámetros... mejor mostraros la otra forma, y ya sabeis lo dicho: si hay más de 3 parámetros en una función, divide y vencerás.

6. Cambiar de vista

Para poder navegar entre views, o vistas, lo que necesitamos es, definir en un módulo router.js nuestras rutas. Serán similares a las de la aplicación anterior, dado que para navegar entre templates es exactamente igual, pero esta vez definido como módulo.

```
1 | define(["ember"], function(Ember){
2 |   var Router = Ember.Router.extend({});
3 |   Router.map(function() {
4 |     this.resource("menu", {path : "/menu" }, function(){
5 |       this.route("selectColor");
6 |       this.route("watchColor");
7 |     })
8 |   })
9 |   return Router;
10 | });
```

Creamos un módulo para el controlador

```
1 | define(["ember"], function(Ember){
2 |   var ApplicationController = Ember.Controller.extend();
3 |
4 |   return ApplicationController;
5 | });
```

Ahora vamos a crear un módulo para nuestro template:

```
1 | //<div class="container"> Comentado para que podais verlo.
2 |   {{t main.application.title tagName="h1"}}
3 |   {{outlet}}
4 | //</div>
```

Finalmente crearemos un módulo para nuestra view:

```
1 | define([
2 |   "ember",
3 |   "text!templates/applicationTemplate.html"
4 | //definimos como dependencia los templates
```

```

5     ], function(Ember, applicationTemplate) {
6
7     var ApplicationView = Ember.View.extend({
8       //Aquí compilaremos con handlebars nuestro template
9       defaultTemplate: Ember.Handlebars.compile(applicationTemplate)
10    });
11  });
12  return ApplicationView;
13  });

```

Nota: para poder enlazar bien nuestras rutas, controlar que habéis realizado estos pasos, incluso definir en vuestra app cada view, cada controlador, todo...

7. Cambio de estado mediante controladores

Para mantener un estado en nuestra aplicación, basta con tener un Store donde almacenamos nuestras variables, o acceder al controlador que mantiene estos estados.

Para acceder a los atributos, o funciones de un controlador desde otro basta con definir una dependencia como atributo llamado "needs:" ejemplo:

```

1  define(['ember'], function(Ember) {
2  MenuSelectColorController = Ember.Controller.extend({
3    needs: ['Color', 'Colors'],
4    init: function() {
5      this._super();
6    },
7    changeSelectedColor : function(newColor) {
8      var colorController = this.get("controllers.Color");
9      colorController.changeSelectedColor(newColor.name);
10   }
11  });
12  return MenuSelectColorController;
13  });

```

De esta manera conseguiremos acceder a las funciones y atributos de ColorController y ColorsController.

Como podéis ver la sintaxis de ember es similar, pero debemos definir las dependencias.

Una duda que me surgió es como enlazar un `{{outlet}}` a una vista determinada, y bastaba con tener bien enrutada nuestra app en el Router.

Si tenemos un `{{outlet}}` en menu, cargará a continuación menu/index, por lo tanto si queremos acceder a otras rutas desde nuestra app, por ejemplo a menu/selectColor, lo tendremos que definir en nuestro router, y después acceder a esta ruta mediante `linkTo {{#linkTo "menu.selectColor"}}`

Otra manera es renderizar mediante el helper `{{render}}` de handlebars. Ej: `{{render "user" this}}`

Cargará el template: userTemplate El segundo parámetro es el contexto a usar. This dentro de este bloque es user.

8. Conclusiones

De este modo tendremos nuestra app separada en distintos módulos, por lo que tendremos todas las ventajas de este sistema, ya sea para reutilizar nuestro código, para tener un código más limpio y ordenado, facilitar una futura refactorización y facilitar también la comprensión.

Más adelante haremos un tutorial de como internacionalizar una aplicación con Ember-i18n y separada en módulos con Requirejs y así profundizar un poco más.

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)

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

Share |

0

Animáte 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](#)

IMPULSA

Impulsores

Comunidad

¿Ayuda?

4
clicks

1 personas han traído clicks a esta página



powered by [karmacrazy](#)

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

