Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación.

Experiencia Profesional.

Winbits

Emmanuel Maldonado Cuña

208367610

ClickOnero

M. en C. Luis Alejandro Flores Béjar

Trimestre 2015 Invierno

Yo, Luis Alejandro Flores Béjar, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma

Yo, Emmanuel Maldonado Cuña, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma

Tabla de contenido

I- Resumen

El presente documento describe el trabajo realizado en el desarrollo de un sistema de comercio electrónico, innovando la manera en la cual un usuario compra algún producto o servicio en línea, integrando y exponiendo servicios REST para que estos sean consumidos por la parte visual del sistema, la cual a su vez expone funciones que la tienda virtual o verticales puedan consumir.

Se necesita realizar un sistema en el cual será una página *web single-page,* con conectividad a los servicios web expuestos por aplicaciones REST; tomando la arquitectura de aplicaciones REST se llega a la conclusión que el sistema se puede dividir en distintos módulos en los cuales, se separa la lógica correspondiente con lleva el sistema, como por ejemplo una aplicación que se dedique exclusivamente a las órdenes de compra; teniendo cada una de ellas trabajando simultáneamente para hacer la integración de un sistema completo, esto nos lleva a un problema de comunicación entre cada una de las aplicaciones ya que como corren en diferentes procesos también debe de existir comunicación entre ellas, en lo cual se toma el concepto de procesos asíncronos para solventar este problema.

II- Introducción

ClickOnero es una empresa dedicada al comercio electrónico. Se especializa en venta, con descuento, de productos, viajes y cupones de servicios, que los proveedores mantienen en exceso, y que ven en la empresa, un canal de distribución alterno altamente eficiente. El proceso de venta está ligado a un esquema de lealtad que premia a los usuarios frecuentes, promoviendo de esta forma, las compras recurrentes.

En el departamento de sistemas se desarrolla un proyecto donde no solo sea una tienda involucrada en la venta de productos, sino formar un grupo de tiendas que compartan elementos comunes de operación, pero que para el usuario parezcan sitios individuales. El concepto básico de este proyecto es que varias tiendas, en este caso llamadas verticales, se comuniquen por un mismo dominio en el cual ya se encuentran los usuarios, registrados y con la ventaja de realizar el pago de todas las tiendas en un solo lugar.

Ejerciendo un puesto de desarrollador/analista *Grails/Groovy* para *backend* de servicios web y clientes web *api REST* [1], uso de *RabbitMQ* [2], para el paso de mensajes entre aplicaciones utilizando el protocolo *ampq* e implementación de módulos del administrador del proyecto. Generando pruebas unitarias con *JUnit*[3] y *Spock* [4], implementando la buena práctica de desarrollo TDD (*Test Driven*

Development) [5] y *Coffeescript*[6] para fronted, implementando conectividad con el backend mediante *Ajax*[7], uso del concepto de promesas de *JavaScript*[8]; integración del maquetado de la página web con el proyecto *Widget*, implementación del carrito de compra para múltiples verticales.

III- Antecedentes

Como se sabe internet es una gran herramienta que nos permite acceder con facilidad a la información que este alberga, dado esto, nos apoyamos de esta herramienta para realizar un sitio donde los usuarios puedan realizar compras en el mismo de manera confiable y en la comodidad de su hogar. Este sistema engloba la funcionalidad visual para desplegarla directamente en el navegador web, el cual tiene la funcionalidad de exponer los servicios, por ejemplo, agregar, quitar, modificar productos en el carrito de compra, manejo de los datos de usuario al igual que sus propios historiales de compra y de recompensas por parte de la plataforma; en la parte no visual del sistema se integran las funcionalidades que accedan a la base de datos y se realiza cada una de las reglas de negocio, como por ejemplo el inicio de sesión de un usuario o el guardado de una dirección de envió, para una orden de compra.

IV- Periodo

En el lapso de un año tres meses de trabajo, se realizaron varias pruebas para solventar el problema más importante del sistema, que era mostrar visualmente una aplicación que expusiera los servicios mencionados anteriormente y haciendo llamadas directamente a la parte no visual. Este desarrollo se logró concluir y exponerlo directamente para todo público.

V- Objetivo General

Desarrollo de una aplicación desplegable en un navegador web, mediante una sola línea de código la cual pueda exponer servicios de comunicación hacia el *backend* y exponer funciones que pueda integrar la vertical para la venta de los productos.

VI- Objetivos Específicos

Para poder desplegar esta aplicación directamente en un navegador web se desglosa en los siguientes objetivos:

- ✓ Utilizar el patrón de arquitectura de software MVC.
- ✓ Integrar el maquetado o HTML.
- ✓ Soportar una sesión de usuario en diferentes dominios.
- ✓ Realizar comunicación con el *backend*
- ✓ Ejecutar por una sola línea de código desde una página HTML
- ✓ Implementar cada uno de los módulos del maquetado.
- ✓ Realizar pruebas unitarias automatizadas para comprobar el funcionamiento correcto de cada módulo.

VII- Justificación

Dado que existen diferentes sitios de venta por internet, el concepto que estos sitios tienen es la integración de un solo sitio con diferentes tipos de productos, en el cual solo existe una forma de pago o incluso un método para poder realizar nuestras compras. Lo el proyecto integra es la posibilidad de que diferentes tiendas no importa el dominio y el sitio puedan unirse, por ejemplo, si estoy en la tienda Xventas.com y he agregado algo al carrito de compra, no importa si me voy a la tienda cosas.com; porque mantengo agregado en el carrito de compra el mismo producto en la misma orden de compra y así poder comprar lo que requiero englobado como si este fuera un mismo sitio.

VIII- Fundamento Teórico

Dada la complejidad completa del proyecto, se tienen en cuenta principalmente los siguientes temas los cuales son en esencia el núcleo de la plataforma:

1- Aplicaciones REST y Servicios REST

REST por sus siglas en ingles *Representational Slate Transfer*, es un tipo de arquitectura de desarrollo WEB totalmente apoyado en el estándar HTTP. Este tipo de arquitectura nos permite crear servicios y aplicaciones web que pueden ser usadas por cualquier dispositivo o cliente que entienda el protocolo, que a su vez es más simple y son otras alternativas a los servicios SOAP y XML-RPC.

Para el uso correcto de este tipo de arquitectura se deben de tener en cuenta tres reglas:

- Uso correcto de las URIs.

    a. Las URLs nos permiten acceder a cada una de las páginas, secciones o documentos web, en el caso de REST esto es llamado recursos, por lo tanto es la información a la cual queremos tratar con ella.

b. Deben de ser únicas, no se debe tener más de una URI para identificar un mismo recurso.
c. Deben de ser independiente de formato.
d. Deben de mantener una jerarquita lógica.
e. Los filtrados de la información no se deben de hacer en las URI.

- Uso correcto de HTTP.

a. Conocer los métodos HTTP.
b. Códigos de estado.
c. Aceptación de tipos de contenido.

- Implementar Hypermedia.

a. Conectar mediante vínculos las aplicaciones clientes con las APIs.
b. Despreocupar al usuario como acceder a los recursos.

2- HTTP (*Hypertext Transfer Protocol*)

El protocolo HTTP es el método más común de intercambio de información en internet; en este protocolo existen algunos métodos con los cuales debemos operar para la comunicación de las APIs.

1. GET: Consultar o leer recursos.
2. POST: Crear recursos.
3. PUT: Editar recursos.
4. DELETE: Eliminar recursos.
5. PATCH: Editar partes muy concretas de un recurso.

A la hora de generar un servicio web, se declara en el a qué tipo de método va a responder y mediante que URL.

3- JSON (*JavaScript Object Notation*)

Cada uno de los servicios *REST*, expuesto por las aplicaciones pueden recibir algunas cabeceras y su respuesta de los mismos es mediante *JSON,* se optó esta notación por ser muy ligera y rápida, a comparación del XML.

En el código 1.1, se muestra un ejemplo de una respuesta dada por un servicio REST, el cual contiene dos partes principalmente. Una donde se regresa una etiqueta *"meta"* en la cual se nos indica el estatus con el cual nos ha respondido el servicio, en esta etiqueta pueden existir igual mensajes, en caso de

error para indicar el porque del mismo error; en la segunda etiqueta *response* viene en otras etiquetas o en forma de arreglo, la información que nos responde, en este caso la información es una transacción de una cantidad de 10 y un concepto de “ejemplo de respuesta”.

4- Procesos Asíncronos

En el proceso de desarrollo existen algunos procesos que por su naturaleza son tardados o por el mismo flujo del proceso no es tan necesario esperar alguna respuesta, es por eso que se utilizó un broker llamado RabbitMQ el cual nos ayuda a solucionar este problema.

Este broker es un sistema de mensajes encolados, el cual nos permite enviar múltiples mensajes y que ciertos consumidores los lean uno a uno para poder continuar con su flujo.

En el diagrama 1.1 nos muestra el flujo de una compra exitosa, en dicho flujo existe el proceso que publica un mensaje dentro del broker en este se almacena y encola hasta que un consumidor o listener lo toma para procesarlo, de esta manera el flujo de la compra exitosa sigue sin ninguna interrupción y se agiliza el proceso.

5- Pruebas unitarias.

Para realizar las pruebas unitarias sobre el código del widget, nos referimos a probar la funcionalidad de cada una de las vistas, es decir, si se le da click en algun boton que debería de hacer, como se muestra en el código 1.2.

En el código 1.2, se muestra el ejemplo de una prueba unitaria, en ella se ve la descripción de la prueba de que es lo que se está probando, por cada una de las pruebas se tiene una pequeña configuración donde se carga la vista y su modelo método “*beforeEach*”, pero por cada una de las pruebas se tienen que regresar a su estado original método “*afterEach*”, en este caso se prueba “*Header*”, cada una de las pruebas se reconocen por el prefijo “*it*” y la descripcion de la misma, en este caso cada expectativa se acierta con el método “*expect”,* en este caso se acierta que la vista tenga la clase “*widgetWinbitsHeader*”.

6- Handlebars

Una de las tecnologías que se ocuparon, para el renderizado de la vista fue Handlebars, esta se comunica directamente con el modelo que nos proporciona *ChaplinJs,* se integra directamente a un código *HTML,* con anotaciones de variables entre llaves “*{{variable}}*”, este tipo de archivo se usa

como una plantilla donde por cada variable y lista son pintados en el compilado. Como se muestra en el ejemplo código 1.3.

Para poder meter más lógica dentro de la misma pantalla existen algunas notaciones dentro del handlebar donde se nos permite hacer un *if, else, unless,* aun así esto puede ser configurable generando anotaciones personalizadas donde se declara la función y parámetros a recibir ya sea para comprar esta información o para tratarla de tal manera que el resultado sea desplegado en la pantalla, como en el código 1.3 se muestra la notación *{{eachActiveVertical}}* en donde por medio del modelo se toma la información y se hace una validación para pintar el siguiente código HTML.

7- TDD (*Test driven development*)

Una metodología de desarrollo, que consiste en tres pasos principalmente.

- Generar una prueba de la funcionalidad de un módulo.
- Hacer que esta prueba pase dicha funcionalidad.
- Reorganizar el código para dejarlo para dejarlo lo mas limpio posible.

Estos pasos se van repitiendo cada vez que se añade complejidad o un módulo extra, se repiten los pasos, corriendo al final todas las pruebas y verificar que no hubo algo que modificará la lógica que se tenía en un principio o que llegue a afectar alguna regla de negocio.

8- GIT

Un sistema de versionamiento distribuido, este sistema se maneja mediante ramas o *branches* los cuales se separan principalmente en dos, *master* y *develop*. En *master* se encuentra el código que ya se ha probado y esta listo para ser productivo; en la rama de *develop* es donde se encuentran los recientes cambios o nuevos módulos en la lógica del sistema, en esta rama es donde se realizan las pruebas antes de poder pasarlo a *master*.

IX- Descripción del trabajo.

Dentro del proyecto, existen diferentes sub-proyectos, en donde se integra desde la parte de los servicios o *backend* hasta la parte visual o *frontend*; mismos que están desarrollados con diferentes tecnologías y conectados mediante una arquitectura basada en aplicaciones *REST* mediante el paso de mensajes con el formato *JSON*[9], cabe mencionar que todos los proyectos tienen un sistema de

versiones basado en la tecnología *GIT*[10] que es un sistema de versionamiento distribuido, que nos ayuda a tener de manera local el ambiente y poder seguir desarrollándolo de una manera más ágil.

En los diferentes sub-proyectos en los que se participó fueron principalmente:

1- Widget

El problema a afrontar en este proyecto, es que se necesitaba desplegar un *Widget* con la declaración de una sola línea de código en una página html, de manera rápida y estable. Para esto se decidió utilizar *ChaplinJs*[11] que nos facilita la manera de organizar el código mediante una estructura Modelo-Vista-Controlador (MVC) *y BrunchJs*[12] que nos ayuda a concentrar todo el código generado en un solo archivo. El patrón de diseño MVC, permite separar el código de tal manera que pueda ser escalable, modificable y organizado. Por ejemplo para una vista debe de tener un controlador el cual le indica cuando esta se debe de renderizar y un modelo para que sepa cuáles son los datos con el cual esta se renderiza, ver diagrama 1.2.

Se usa la plataforma Linux para este desarrollo, corriendo una integración continua con *BrunchJs*, el cual compila al momento de existir un cambio en el código y así realizar el desarrollo más ágil. La parte visual del sistema es desarrollado en un 70% de *CoffeeScript* y el 30% de *Handlebars*[13]; realizando pruebas unitarias con *MochaJs*[14]*, ChaiJs*[15] y *SinonJs*[16].

En el *Widget* se albergarán las diferentes verticales comunicándose mediante un dominio tercero como se muestra en el diagrama 1.3, el cual sirve como mediador para el alojamiento de información, principalmente el carrito de compra y el *token* de la sesión si esta existe. Esta comunicación se hace gracias a facilidad que nos da para mandar la información el *plugin* de *EasyXDM,* utilizando una metodología de *RPC*[17] (Por sus siglas en inglés, llamada de un método remoto), ya que con una petición de algún método configurado podemos almacenar o extraer la información que se encuentra guardada.

La responsabilidad en este proyecto, fue desde un principio analizar, investigar y experimentar con el *framework ChaplinJs*, entendiendo su estructura, función y como genera en el compilado archivos de salida; implementación del dominio tercero para el uso del dominio cruzado y compartición de datos entre verticales, implementación de los módulos *login* o inicio de sesion con sus diferentes flujos, completar registro, recordatorios de completar registro*,* recuperar contraseña, flujo de usuario ya registrado pero no confirmado, historial de compras, historial de bits, listado de direcciones de envío,

nueva dirección de envió, editar alguna dirección de envió, integración de campañas para la venta de artículos en las verticales, módulo de *mailing,* modales de aviso, integración de activación del teléfono celular para poder seguir contestando encuestas, historial de compras e historial de bits .

2- MIS (*Message Interface Service*)

El problema a atacar en esta aplicación era tener una aplicación que gestiona el envío de correos, se atacó el problema con la tecnología Groovy/Grails[18] con integración de *RabbitMQ.*Utilizando *listeners* o escuchas de Rabbit*MQ* se envían los mensajes con los datos requeridos, es decir si se va a enviar un correo a un usuario para que confirme la cuenta que acaba de registrar, entonces la aplicación pone en la cola el mensaje con el nombre, correo y la url del registro; del lado de la aplicación "mis" se capta el mensaje y se procesa, validando cada uno de los campos enviados, si todos ellos son correctos este consume un servicio *REST* a una instancia externa para el envío de estos mensajes.

La responsabilidad en este proyecto fue generar cada una de las colas en un *exchange* de *RabbitMQ*, en el cual se alojaban los mensajes, realizar los consumidores de estas colas para el tratado de cada uno de los diferentes mensajes, hacer la validación por cada uno de los campos y consumiendo el servicio *REST* para el envío del correo; realizando pruebas unitarias y de integración mediante *Spock y JUnit*.

3- Orders

En esta aplicación se ataca directamente el problema de los pagos, generación de órdenes de compra y carrito por usuario, guardado de tarjetas de crédito mediante una librería llamada *CyberSource*, apartado de producto en caso de generación de una orden, envío del mensaje anunciando que se ha realizado una compra al usuario, guardado de campañas por cada carrito de compra y orden generada por usuario para su estudio por el área de Inteligencia del negocio. Desarrollado con *Groovy/Grails* por su fácil implementación para el desarrollo de una *api REST* , para el consumo de estos mediante el *Widget*.

La parte dentro de este proyecto en la cual participé, fue la exposición de dos servicios web y la integración de las campañas por carrito de compra y por orden de compra, teniendo en cuenta cuando viene un solo producto o cuando viene una lista de ellos para su persistencia en la base de datos. Realizando la buena práctica de *Test Driven Development,* su implementación de este tipo de servicios bajo el patrón de diseño de Modelo-Vista-Controlador que nos proporciona *Groovy/Grails,* se realice

de una manera más ágil y con una tendencia de errores en cuanto a lógica menor a la que normalmente se suele cometer.

4- Admin

El problema a atacar en esta aplicación, es la administración y operación del sistema en general, desde el desbloqueo de un usuario, hasta cancelar o reembolsar una orden de compra. Desarrollado en *Groovy/Grails* con integración de una librería interna de clientes *REST.*

Mi participación en este proyecto, es integrar el cambio de talla/color de una orden generada, integrando el maquetado que nos estrega el área de diseño, con su respectivo patrón de diseño que se ha utilizado en todo el proyecto MVC, integrando funcionalidad en el maquetado mediante *JavaScript* y realizando las peticiones a los servicios mediante *ajax*.

5- Api-Clients

Este desarrollo se enfocó en realizar múltiples *plugins* en un solo proyecto y publicarlos en un repositorio donde las aplicaciones lo obtendrán, actualizado, estos serán clientes *REST* de las mismas aplicaciones, por ejemplo si por parte de un usuario se requiere saber cuales son sus tarjetas de crédito, mediante un cliente de este proyecto se ejecuta y nos trae la información requerida. Es este proyecto se utiliza *Groovy/Gradlew*[20] este pequeño framework nos ayuda con la generación de los plugins y la publicación en el repositorio y el plugin *Wslite* que nos ayuda a el desarrollo de clientes *REST* y para la realización de pruebas se utiliza *JUnit*.

Mi participación en este proyecto fue realizar el cliente para las aplicaciones de "*MIS*" y "*Orders*", de esta manera se integraban a cada uno de las aplicaciones mandandolas llamar por inyección de dependencias y poder consumir los servicios.

6- Winbits-jobs

Esta aplicación esta enfocada en su totalidad, a tener procesos que se ejecutan de manera repetitiva cada cierto tiempo, un ejemplo de esto es la cancelación de ordenes de compra con un tiempo mayor a 72 horas, que están en el estatus de pendiente. En este proyecto se utiliza *Groovy/Grails* y se utiliza la configuración de *CRON* que nos ofrece el sistema operativo.

El desarrollo que se lleve a cabo en esta aplicación, fue realizar una conexión a un servidor con *SQL SERVER* para realizar una consulta a un *Store Procedure,* el cual nos entrega información de los productos activos y con *stock* activo; esta información se organiza y se escribe en un archivo xml, para que sea consumido por otra aplicación externa.

X- Conclusiones

La creación de un sistema no monolítico es una buena solución para creación de un sistema de comercio electrónico, ya que esto separa la lógica del negocio en cada una de las instancias, esto también separa los recursos del servidor y evita que pueda colapsar en un momento de mucha concurrencia.

La desventaja de este tipo de solución es que es necesario realizar tanto servicios y clientes web para hacer alguna comunicación entre estas aplicaciones o existen algunos métodos que por la necesidad del negocio no es necesario que se espere una respuesta inmediata, es decir, el enviar la petición y que otra aplicación la tome; esto lo solucionamos mediante el sistema de envió de mensajes RabbiMQ, que como vimos mediante la publicación de un mensaje, las colas del mismo sistema y los escuchas, se solventa este problema y hace que el sistema pueda ser mucho mas independiente.

XI- Referencias

[1] DocuSign, “REST API Guide”, version 2, California, 2013

[2]http://www.rabbitmq.com/documentation.html

[3]http://junit.sourceforge.net/javadoc/

[4]http://spock-framework.readthedocs.org/en/latest/

[5]Robert C. Martin, “The Clean Coder”, Página 77, Edition 1, Libro de divulgación científica, Editorial: Prentice Hall, Nueva York, 2012.

[6] http://coffeescript.org/

[7] José Antonio Muñoz Jiménez, “Programación con AJAX”, Trabajo terminal , México 2010

[8] http://www.html5rocks.com/en/tutorials/es6/promises/

[9] Douglas Crockford ,”JSON The x in Ajax”,Yahoo! Inc.

[10] http://git-scm.com/documentation

[11]http://docs.chaplinjs.org/

[12]http://brunch.io/

[13]http://handlebarsjs.com/

[14]http://visionmedia.github.io/mocha/

[15]http://chaijs.com/guide/

[16]”Sinon Documentation”http://sinonjs.org/docs/

[17] “RPC Llamada a procedimientos remotos”, http://www.tamps.cinvestav.mx/~vjsosa/clases/sd/RPC_notas.pdf

[18] “Grails Tutorial”, http://grails.org/doc/latest/guide/single.pdf

[19] “Gradle”http://www.gradle.org/docs/current/userguide/userguide.html

XII- Anexos

Código 1.1

Ejemplo de respuesta JSON de un servicio REST

```
{
 "meta": {
      "status": 200
   },
  "response": {
      "transactions":
       [
         {
            "amount": 10,
            "concept": "Ejemplo respuesta"
         }
       ],
      "balance": 10
   }
}
```

Código 1.2

Ejemplo de prueba unitaria

```
describe 'HeaderViewSpec', ->

  beforeEach ->
    @model = new Header
    @view = new HeaderView model: @model

  afterEach ->
    @view.dispose()
    @model.dispose()

  it 'should be rendered', ->
    expect(@view.$ '.widgetWinbitsHeader').to.exist
```

Código 1.3

Ejemplo Handlebars

```
<div class="knowMoreMax-item">
      <div class="knowMoreMax-icons">
          {{#eachActiveVertical}}
            <a href="{{baseUrl}}" class="iconVertical-w vertical{{id}}" title="{{name}}"></a>
          {{/eachActiveVertical}}
      </div>
       <h2>VARIOS SITIOS</h2>
 </div>
```

Diagrama 1.1

Proceso de envío de mensajes a RabbitMQ proceso de compra exitosa

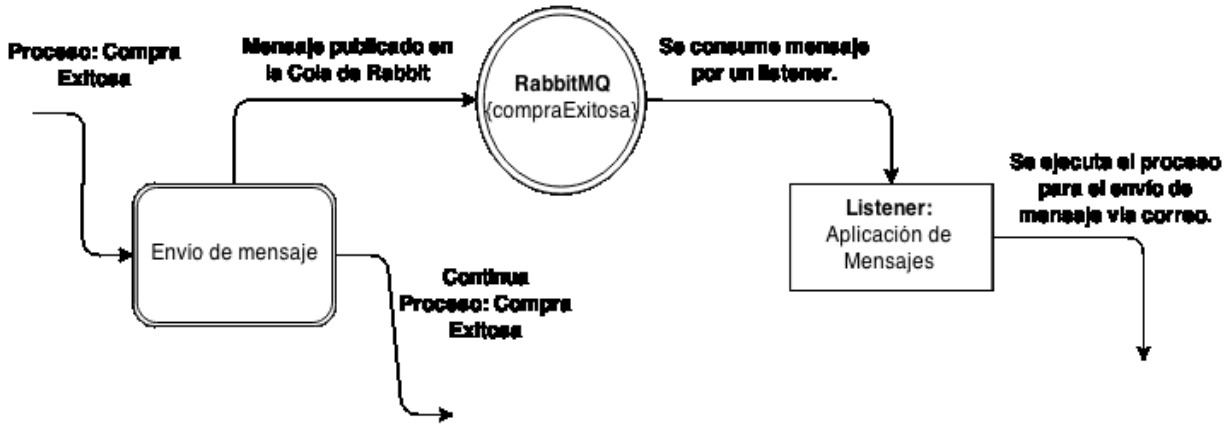

Diagrama 1.2

Integración CoffeeScript/ChaplinJs/BrunchJS

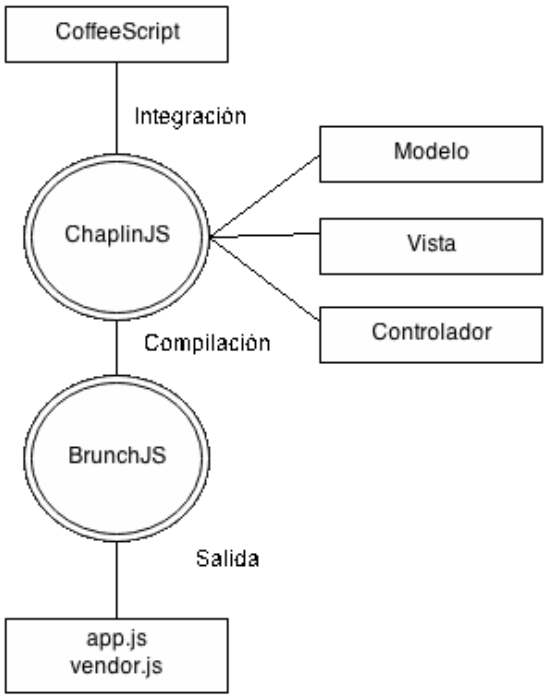

Diagrama 1.3

Arquitectura general del sistema.