|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Autor | Versión  Aprobada | Referencia al cambio |
| *06/03/2020* | *Ennovasoft* | *1.0* | *Se aprueba documento.* |

**Paquete de Datos Técnicos**

# Contenido

[1 Contenido 2](#_Toc45610034)

[2 Datos del proyecto 3](#_Toc45610035)

[3 Permisos necesarios para integrantes del proyecto 4](#_Toc45610036)

[4 Arquitectura del producto 4](#_Toc45610037)

[5 Arquitectura de Acceso a Datos 5](#_Toc45610038)

[5.1 Datos de la base de datos General 5](#_Toc45610039)

[5.2 Efusion 5](#_Toc45610040)

[6 ANEXOS 6](#_Toc45610041)

[6.1 Cuenta para consolas 6](#_Toc45610042)

[6.2 Utilidades 6](#_Toc45610043)

[6.2.1 Diseño de botones y formularios 6](#_Toc45610044)

[6.2.2 Imágenes Default 6](#_Toc45610045)

[6.2.3 Manejo de cantidades 6](#_Toc45610046)

[6.3 Notificaciones 6](#_Toc45610047)

[6.4 Envió de información al server central y/o subida de archivos al servidor local 7](#_Toc45610048)

[6.4.1 Envío de imágenes al server central y subida de imágenes al servidor local 7](#_Toc45610049)

[6.4.2 Subida de archivos (puede ser pdf, Excel o de cualquier tipo) al servidor local 8](#_Toc45610050)

[6.5 Correos 9](#_Toc45610051)

[6.5.1 Imágenes 9](#_Toc45610052)

[6.5.2 Generación de plantilla HTML 9](#_Toc45610053)

[6.5.3 Envío Email Java 9](#_Toc45610054)

[6.6 Validaciones 9](#_Toc45610055)

[6.6.1 Validaciones Java 9](#_Toc45610056)

[6.6.2 Validaciones SQL 10](#_Toc45610057)

[6.7 Componentes Externos 10](#_Toc45610058)

[6.8 Encriptado 12](#_Toc45610059)

[6.8.1 Encriptación 12](#_Toc45610060)

[6.8.2 Desencriptación 15](#_Toc45610061)

[6.9 Rutas 15](#_Toc45610062)

[6.9.1 Configuración 15](#_Toc45610063)

[6.9.2 Archivo Rutas JS 16](#_Toc45610064)

[6.9.3 Llamar el routing 16](#_Toc45610065)

[6.10 Configuración de idiomas en el aplicativo 16](#_Toc45610066)

[6.10.1 Idioma en JSP 16](#_Toc45610067)

[6.10.2 Idiomas en las validaciones de Bootstrap validator 18](#_Toc45610068)

# Datos del proyecto

|  |  |
| --- | --- |
| ExtraCoupon WebAdmin | |
| Cliente: | Ing. Mario Alejandro Guillen Ramírez |
| Responsable del proyecto: | Sarai Moreno Moreno |
| Fecha Inicio |  |
| Fecha fin |  |
| Elaborado por: | Sarai Moreno Moreno |

# Permisos necesarios para integrantes del proyecto

* Herramienta Efusion proyecto ExtraCouponV3
* Redmine ExtraCupon\_v3
* Repositorio para el proyecto ExtraCoupon\_V3 (centralizador de memorias, server móvil y proyecto WebAdmin)

# Arquitectura del producto

|  |  |
| --- | --- |
| IDE (Entorno de Desarrollo Integrado) | Eclipse |
| Lenguaje | JAVA |
| Modelo de desarrollo | MVP |
| Plataformas | * Java 1.8 * redHat sistema operativo |
| Tipo de Aplicación | Web |
| Servidor de aplicaciones | Apache Tomcat 7 |
| Librerías | JQuery, Bootstrap, angular JS, Librería Efusion. Bootstrap validator, itext, itextPDF, thymeleaf, bcrypt |
| Modelo de acceso a datos | Conexión mediante Ejecutor (Pool de Conexiones) |
| matriz de trazabilidad |  |
| Diseño interfaz | http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Proyectos%20Internos/ExtraCupon\_V3/WebAdmin/An%C3%A1lisis%20y%20Dise%C3%B1o/Dise%C3%B1o |
| Administración del proyecto | http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Proyectos%20Internos/ExtraCupon\_V3/WebAdmin/An%C3%A1lisis%20y%20Dise%C3%B1o/Planeacion\_ExtraCoupon\_WebAdmin.mpp |
| Seguimiento a incidencias | http://192.168.0.65:70/redmine/projects/web-administrador |
| Tiempo de respuesta máximo | 5 milisegundos máximo en tiempo de consultas en el server móvil |
| Casos de uso | http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Proyectos%20Internos/ExtraCupon\_V3/WebAdmin/An%C3%A1lisis%20y%20Dise%C3%B1o |
| Modelo base de datos | http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Proyectos%20Internos/ExtraCupon\_V3/WebAdmin/An%C3%A1lisis%20y%20Dise%C3%B1o/Dise%C3%B1o/Base%20de%20datos/DBS\_ExtraCoupon.mwb |
| Metodología de desarrollo | http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Biblioteca%20de%20Activos/Procesos/Comercial/Manuales/Metodolog%C3%ADa%20Ennovasoft.docx |
| Pruebas Unitarias | http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Proyectos%20Internos/ExtraCupon\_V3/WebAdmin/Construcci%C3%B3n/PSU\_Extracoupon\_WebAdmin.xlsm |
| Extras | ***Centralizador de Memorias***  *http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Proyectos%20Internos/ExtraCupon\_V3/Componentes/Construccion/Codigo%20Fuente/CentralizadorMemoria/Trunk*  ***Server Móvil (Imágenes)***  *http://192.168.0.140:8080/svn/RepositorioEnnovasoft/Proyectos%20Internos/ExtraCupon\_V3/Componentes/Construccion/Codigo%20Fuente/ServerMovil/Trunk* |

# Arquitectura de Acceso a Datos

## Datos de la base de datos General

|  |  |
| --- | --- |
| Gestor | *MYSQL 5.7* |
| IP servidor | *192.168.0.89:3306/BDD\_ExtraCouponV3* |
| Usuario | BDD\_ExtraCouponV3 |
| Contraseña | ##.%C.upN!#L3 |

## Efusion

|  |  |
| --- | --- |
| JDNI | jndi.ExtraCouponV3 |
| proyecto en efusion | Pool-db\_ ExtraCouponV3 |
| Nombre del proyecto | ExtraCouponV3 |
| ID del proyecto | 1226 |

# ANEXOS

## Cuenta para consolas

|  |  |
| --- | --- |
| Usuario | Pass |
| [extracupon.ennovasoft@gmail.com](mailto:extracupon.ennovasoft@gmail.com) | ExtraCupon%260918 |

La cuenta en la tabla de arriba será la que se ocupará para la creación de proyectos en consolas como Firebase, Facebook o Google y para el envío de correos.

## Utilidades

### Diseño de botones y formularios

Para el diseño de botones y formulario se dejarán en un CSS en la carpeta utilerias/base.css

### Imágenes Default

La imagen por default será la ubicada en resources/imagenes/ImagenDefault

### Manejo de cantidades

Todas las cantidades deberán formatearse en miles (1,250.00) con decimales en caso de cantidades de dinero, las cantidades deben de truncarse en caso de contener mas de dos decimales (ejemplo 1,250.5663 quedaría 1,250.56)

## Notificaciones

Las notificaciones a los dispositivos móviles se harán a través del uso del bean tipo singleton llamado “Notificaciones”. Instanciar la clase del Bean

@Autowired

@Qualifier("Notificaciones")

**private** Notificaciones notify;

* Llenar objeto con los datos Input10ParamsVO (com.efusion.input)

La tabla de valores se ira llenando conforme se necesite.

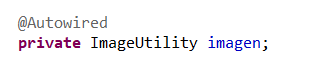
|  |  |
| --- | --- |
| Parámetro | Valor |
| Param1 | Este valor debe contener el id de la notificación, el cual puede ser  “1” – notificación Afiliado |
| Param2 | Este valor depende directamente del tipo de notificación  Tipo 1 – el valor deberá ser el Nombre del Afiliado |

* Ejecutar método de notificación

notify.SendNotificaciones(objNotificacion);

## Envió de información al server central y/o subida de archivos al servidor local

### Envío de imágenes al server central y subida de imágenes al servidor local

Hay una clase en el paquete utilerias.imagenesWebp que se utiliza para hacer esta implementación, primero que nada, crearemos una variable de la clase **ImageUtility** de la siguiente manera:

Posteriormente se manda a llamar el método **loadServerImage**(String img, String rootpath, String pathCompl, String dirServer, String nombre, String idUser) donde:

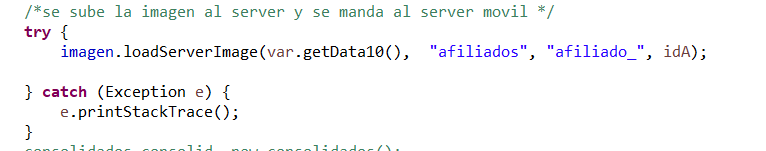
* **imgBase64:** Es la imagen en base64
* **dirServer:** carpeta donde será almacenada la imagen, esta debe corresponder al módulo (ejemplo Afiliados)
* **Nombre:** Un identificador para que las imágenes
* **idImagen:** Un identificador para que las imágenes no se repitan o simplemente el id del registro insertado

Hay que meter la invocación de método en un try catch par cachar cualquier error, el Ip, puerto y timer del server central se configuran en el archivo spring-servlet.xml como se muestra en la siguiente imagen, al igual se definió una ruta para los resources de las imágenes, en caso de cambio se debe actualizar el valor resourcesImagenes:



El método retorna un true o false si se recibió correctamente la imagen en el server central.

Ejemplo de implementación:



### Subida de archivos (puede ser pdf, Excel o de cualquier tipo) al servidor local

En la misma clase de **ImageUtility**, se manda a llamar el método subirPdf(String base64pdf, String path) donde:

* **Base64pdf:** Es el archivo que queremos subir en base64
* **Path:** Ruta donde se quiere subir el archivo, junto con el nombre que se le quiere poner al documento y su respectiva extensión

Este método retorna un true o false si se subió el archivo en el servidor local.

Ejemplo de implementación:



## Correos

### Imágenes

Para las imágenes de correo se hará uso de las imágenes que se encuentran ya cargadas en las siguientes rutas:

* **Header:** <http://www.ennovasoft.com/ExtraCupon/Imagenes/correoi-s.png>
* **Footer:** <http://www.ennovasoft.com/ExtraCupon/Imagenes/correoi-i.png>

### Generación de plantilla HTML

Se necesita generar la respectiva plantilla del correo que se enviara, se recomienda hacer pruebas con diferentes servidores de correo Gmail, Hotmail, outlook.

Las plantillas generadas se almacenarán en el directorio /extraCupon/WebContent/WEB-INF/templatesEmail, Nombrado según lo que se enviara, en esta plantilla se permite dejar estilos directos de CSS.

Dentro de la plantilla debemos agregar la etiqueta <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">, para poder hacer uso de las etiquetas de thymleaf, a continuación, se listan las más usadas:

* **th:text**: esta etiqueta se usa para colocar texto, ejemplo de uso <span style="color:#3a67ac" th:text="${data1}"></span>, donde **${data1}** es un valor que se manda desde java.
* **th:href**: etiqueta para hacer referencia a un link <a th:href="${data2}">Go</a>

### Envío Email Java

* Se debe crear un objeto de la clase **VariablesEmail** llenando los datos que necesitamos, esta clase se puede modificar en caso que necesitemos enviar mas datos de los proporcionados, para lo cual se solicita seguir con la estructura de los data.
* Con la etiqueta **@Qualifier** mandamos a instanciar la clase **EnvioDeEmail**
* Utilizamos el método **getInfoEmail** pasándole como parámetro nuestro objeto lleno de la clase **VariablesEmail** (nos regresara un context que enviaremos al método siguiente)

Por último, ejecutamos el método enviarHtml, pasandole los parámetros necesarios

## Validaciones

### Validaciones Java

Todos los campos de los formularios que se manden al controlador se tendrán que replicar sus respectivas validaciones de Bootstrap y cachar la excepción, en caso que los datos se cachen en una vista se tendrá que mostrar el modal con el mensaje de “**Verificar datos**” y al dar clic en aceptar se cerrará dicho modal **(Ver anexo con id 04)**. En caso de cacharse la excepción en un modal, se tendrá que visualizar el error en un mensaje de Bootstrap con el contenido de **“Verificar datos”** y al dar clic en cerrar se ocultará el mensaje **(Verificar anexo con id 10)**. Para realizar dichas validaciones en java, se tiene una clase en el paquete **singleton** llamada **InputValidaciones** (Aquí se pueden encontrar validaciones básicas de textos, párrafos, números, correo y selects, en caso no encontrar un método de acuerdo a la necesidad de la validación del campo existe un método llamado **validRegex**), para crear una variable de esta clase se hace de la siguiente manera:



Posteriormente se manda a llamar el método que cumpla con la validación del campo correspondiente a validar como se muestra en la siguiente imagen:



Todo esto se debe cachar en un if, ya que todos los campos tienen que ser válidos para proceder enviar la información al servicio.

### Validaciones SQL

Se deberá de validar caracteres que permitan la inyección SQL, a su vez algunos caracteres no permitidos por la herramienta, si se pueden enviar acentos y ñ , en el proyecto ya se cuenta con una clase en el paquete singleton llamada **validacionSqlInjection**, la manera de instanciar esta clase en su servicio se muestra en la siguiente imagen:

Se manda a llamar el método reeplaceCaracter como se visualiza en la siguiente imagen:



No olvidar que siempre hay usar el método trim() para evitar los espacios al inicio y final de la cadena a enviar al repositorio.

## Componentes Externos

Para ocupar los componentes tanto centralizador como server móvil, utilizaremos el método **SendInfoComponente** dentro de la clase AppConfig, el cual nos retornara un String como respuesta, para esto primeramente realizaremos la instancia a la clase AppConfig



Después llenaremos los valores necesarios para el uso del método anteriormente mencionado, siendo estos los siguientes:

* **Componente:** deberá contener el valor 1 cuando se necesite ocupar el server móvil y un 2 para el centralizador de memorias.
* **Caso:** deberá ser algún id definido en la tabla de abajo.
* **ObjetoEnviar:** deberá ser el elemento que se solicite en datos de envío de la tabla de abajo , puede ser un objeto de clase o un String.

A continuación, se presenta un ejemplo del método a usar



#### Casos a usar

|  |  |  |  |
| --- | --- | --- | --- |
| ID Caso | Componente | Descripción | Datos envío |
| SRL2004 | Memorias | Caso para actualizar la memoria de la configuración general | null |
| SRL2002 | Memorias | Caso para actualizar la memoria de él orden de consumo | null |
| SRL2005 | Memorias | Caso para realizar actualización de id de consolidado por afiliado |  |
| SRL2007 | Memorias | Caso para actualizar las promociones | True |
| SRL2006 | Memorias | Caso para realizar la actualización de la configuración de operación con INNE | **Input 20**  Param1: país  Param2: montoRetiro\_con  Param3: monto retiro sin  Param4: monto transferir con  Param5: monto transferir sin  Param6: montocupon con  Param7: monto cupon sin  Param8:monto solicitud con  Param9: monto solicitud sin  Param10: check canje con  Param11: check canje con  Param12: check retiro con  Param13: check retiro con  Param14: check transferir con  Param15: check transferir con  Param16: check solicitud con  Param17: check solicitud con  Nota: para los valores de los check seria 1 si esta check y 0 sino |
| EXCISV008 | server móvil | Caso para realizar envió de la imagen.  Nota: este método ya esta implementado en la clase de envió de imagen no es necesario ocuparlo en cada módulo. | InputByte input = **new** InputByte();   * **IdUser**: identificador de la imagen * **Imagen**: String Base64(imagen) compresa con Gzip * **Nombre**: nombreimagen + ".webp"; * **setPath**: directorio donde quedara en el server; |

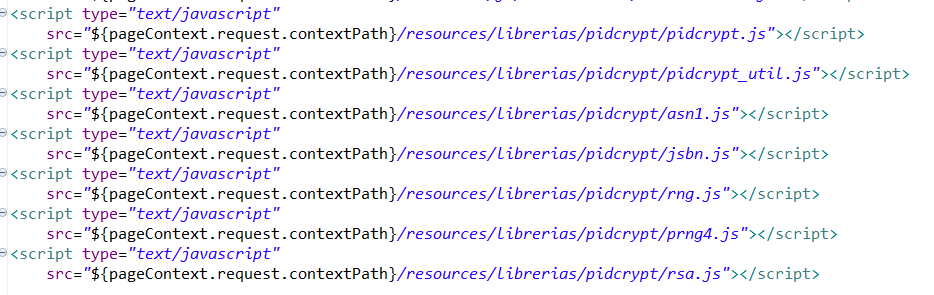
#### Errores Librería

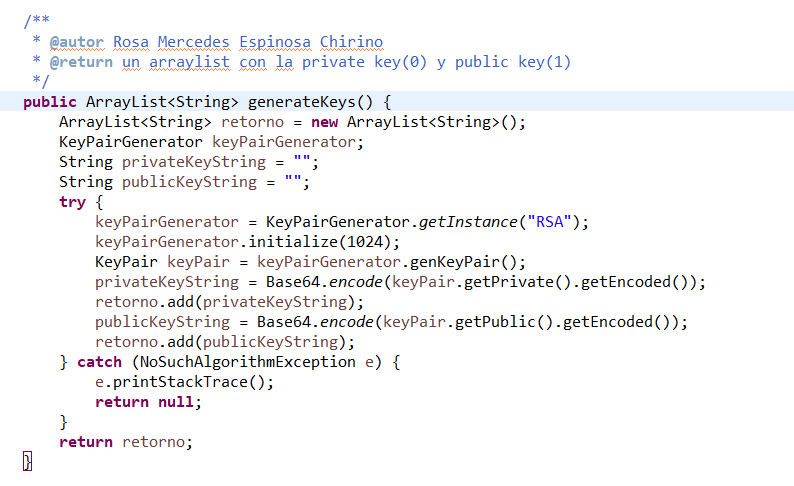
|  |  |
| --- | --- |
| Código Error | Descripción |
| ECM001 | No se conectó al servidor móvil |
| ECM002 | No se recibió respuesta del server móvil posible reinicio de conexión |
| ECM003 | No se conectó al server móvil timeout socketcliente |
| ESM001 | No se encontró modulo Servicio |
| ESM002 | No se envió el correo |
| ESM003 | Error de comunicación con Efusion |

## Encriptado

### Encriptación

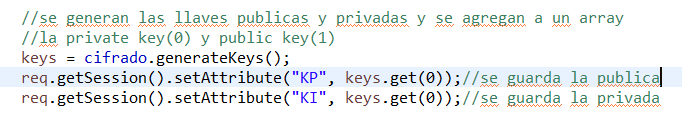
**Nota: Se encriptarán todos los parámetros que se envíen mediante Ajax sin excepción alguna.**

* Primero que nada, tienen que importar las librerías en el contenedor principal o en caso de que se ocupe una pantalla que no ocupe angular se tendrá que volver a declarar estas librerías en el mismo orden que se visualiza en la siguiente imagen. (Estas librerías ya se encuentran en el proyecto)
* Ahora se procede generar 2 llaves (**privada y pública**), esto se generará en el controlador o servicio del login (quien realice el inicio de sesión tendrá que generar dichas llaves y guardarlas en una variable de sesión para que todos los módulos puedan acceder a ella, estas llaves serán necesaria para la encriptación (la privada nos servirá para desencriptar y la pública para encriptar). Todos los integrantes deben tener conocimiento de cómo se llamaran estas variables ya que únicamente se crearán un par de llaves. La clase que tendrán que ocupar será **com.efusion.ExtraCouponCash.Singleton.CifradoTarjeta**, el método a utilizar será **generateKeys()**, el cual nos proporcionará dicha llaves.



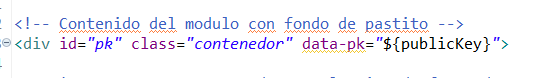
Para realizar dicha obtención de llave se tiene que realizar de la siguiente manera:



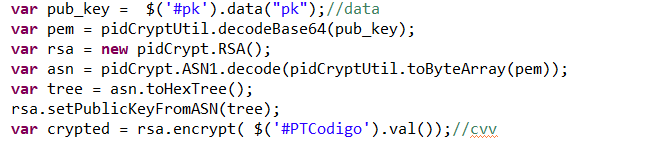


Tratar de ponerle un nombre no tan obvio a las variables que guardarán ambas llaves.

* Se tiene que guardar la llave pública en alguna parte de la vista, esta no importa si el usuario logra verla al inspeccionar el código, pero será necesario tenerla en alguna parte de del jsp, como esta guardado en el servidor, no es necesario mandarlo mediante la vista, si no únicamente asignarlo.



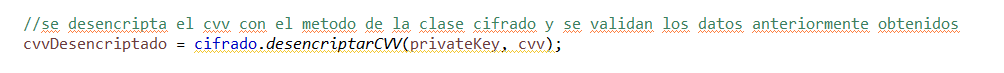
* Por último, al momento de enviar los datos al controlador mediante Ajax, se tendrá que realizar lo siguiente:



Primero se obtiene la clave publica que se tiene en el jsp, posterior se procede a encriptar, en donde dice $('#PTCodigo').val() será el valor de los inputs y la variable que se mandará al controlador sería la variable crypted. Con esto sería todo lo de la encriptación. **De preferencia generar una función en js para reutilizar el método de encriptación.**

### Desencriptación

En el controlador nuevamente se ocupará la clase CifradoTarjeta la cual ya se encuentra en el repositorio, pero ahora se mandará a llamar el método **desecriptarParametros,** mandando como parámetro la clave privada y el parámetro encriptado, esto retornará un String, con el parámetro desencriptado, para que posteriormente proceda a validar su estructura (longitud, regex) etc. Y con esto sería todo.



## Rutas

### Configuración

En el JSP que fungirá como plantilla se coloca las siguientes etiquetas

* <html ng-app="app">, donde app sería el nombre de nuestra aplicación
* ng-include="'header.html'", es opcional si se desea incluir alguna vista, en este caso este redirecciona al controlador quien redirecciona a la vista.
* ng-view se debe colocar en el apartado que fungirá como enrutador, donde se cargaran las vistas
* se deben cargar estos JS
  + <script type=*"text/javascript"* src=*"*${pageContext.request.contextPath}*/resources/librerias/angular/angular.min.js"*></script>
  + <script type=*"text/javascript"* src=*"*${pageContext.request.contextPath}*/resources/librerias/angular/angular-route.min.js"*></script>

### Archivo Rutas JS

Para las rutas se utiliza el archivo ruta.js, donde se definirán las redirecciones a los controladores, cuando sea un cambio de página tipo GET, modificaremos el apartado $routeProvider.

* Para agregar un nuevo caso se coloca lo siguiente

.when('/testuno',{

title : 'Test',

cache: **false**,

controller: 'antiCache',

templateUrl: **function**(params) {

**return** creadorUrl(params, "testuno.html");

}

})

Donde /testuno/' es la ruta que aparecerá en la URL, 'Test' es el título de la pestaña y "testuno.html" es el valor del requestMapping del controlador.

El apartado de la ruta se puede complementar si mandamos parámetros desde URL, colocando /:param1?/:param2? Después de testuno, al colocar **?** estamos diciendo a Angular que el parámetro es opciona. Es importante señalar que el nombre que le coloquemos a los parámetros serán los mismos que obtendremos con el método req.getParameter("param1");

Para regresar las vistas desde el controlador se debe de colocar la ruta completa en el name de la vista, contenedor/modulo.jsp

### Llamar el routing

Se debe colocar la ruta absoluta seguida de #!+RutaURL, como ejemplo <http://192.168.0.89:7070/Efusion/app/index.html#!/bussines/?configid=modulos-list-1219>

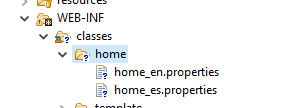
## Configuración de idiomas en el aplicativo

### Idioma en JSP

#### Estructura de archivos:

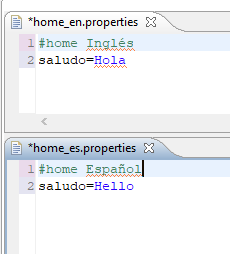
Creamos una carpeta con el nombre de la vista (para un mejor orden) está en la carpeta “classes” que se encontrara en la carpeta “WEB-INF”, dentro de ella creamos los archivos de los idiomas a agregar con la extensión properties.

En la siguiente imagen se visualizan los idiomas inglés y español, la estructura al nombrar estos archivos es importante que el primer prefijo, nombre o palabra que se desee asignar sea el mismo en todos los archivos, en este caso esta como “home”, seguido por la abreviación delo idioma, en el caso del inglés “en” y español “es”.



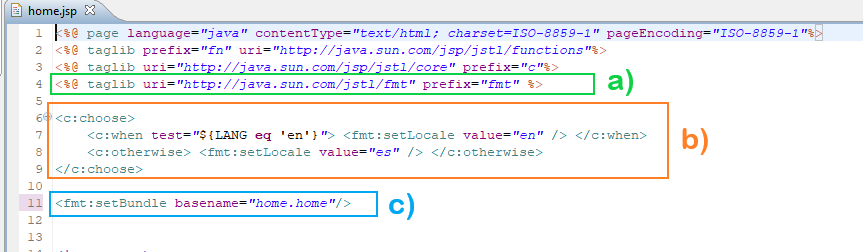
#### ESTRUCTURA DEL CONTENIDO DE LOS ARCHIVOS DE CONFIGURACION DE IDIOMAS

Se utiliza una key(identificador del texto) para reconocer el texto en la vista, en este caso utilizamos saludo, en todos los archivos de los idiomas utilizamos el mismo key, para mandar a llamar el texto en la vista. Los hash “#” se utilizan para hacer un comentario como en cualquier archivo con la extensión properties.



#### CARGAR IDIOMAS A LA VISTA (ARCHIVO JSP)

1. Declaramos la librería (fmt), junto con las demás al inicio del archivo.
2. Asignamos el idioma en el que está el usuario, (la variable es declara en una variable de sesión con un valor por default, en este caso es LANG), se compara si es en, se asigna como idioma el inglés, y si es algún otro valor se deja por default español.
3. Se carga el archivo del idioma con la siguiente sentencia, donde **basename** está compuesto del nombre que le asignamos a la carpeta correspondiente a la vista, en este caso es “home”, y el prefijo a los archivos de lenguaje que creamos, en este caso seria” home”, estos concatenados con un punto.



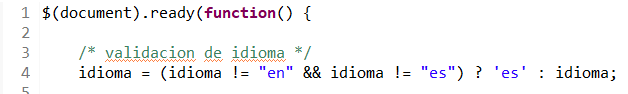
#### UTILIZAR TEXTOS EN LA VISTA

se manda a llamar utilizando la siguiente línea, donde el key es la que se declara en el archivo de configuración del idioma.

### Idiomas en las validaciones de Bootstrap validator

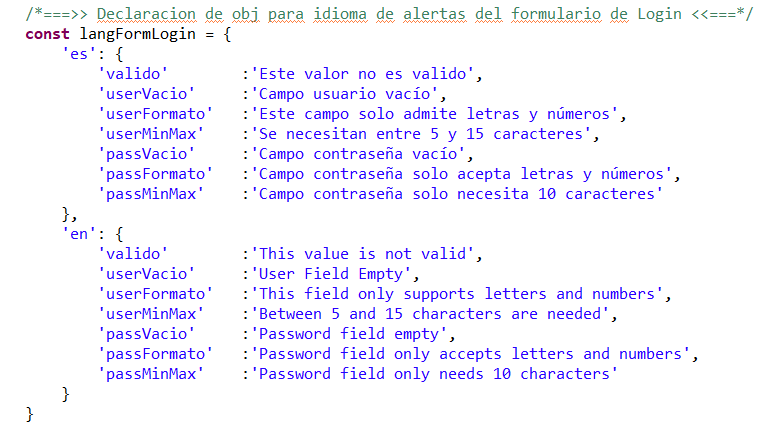
Para las validaciones con multi idioma se creó una variable en el jsp, la cual usaremos en las validaciones.

Entonces en el archivo Validaciones.js tendremos lo siguiente.

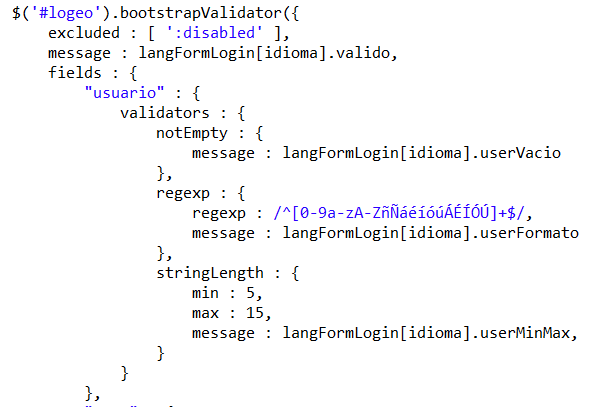


Donde la variable idioma, se evalúa, si está vacía se le asigna un valor por default

(es -> español), de lo contrario conserva su valor.

Seguido de esto, se crea una variable **const** que tendrá los mensajes de las validaciones, para los idiomas correspondientes.

A continuación, en el mensaje de la validación se coloca la variable **const** y la variable de idioma. Ej. “langFormLogin[idioma].valido”



Así automáticamente al realizar una validación se mostrar el mensaje en el idioma determinado.