Pedidos Online - DUSA
Documentación Técnica
Versión 1.6

Historia de revisiones

Fecha	Versión	Descripción	Autor
07/08/2013	1.0	Descripción SolR	Juan Magrini, Ignacio Gil
08/11/2013	1.1	Persistencia	Dahiana Morales
08/11/2013	1.2	Revisión del documento	Patricia Rolandi
09/11/2013	1.3	Agregado front-end	Sergio Bonilla
10/11/2013	1.4	Agregado Mail	Rodrigo Lago
12/11/2013	1.5	Revisión del documento	Patricia Rolandi
12/11/2013	1.6	Revisión Responsable SQA	Verónica Gamarra

Índice

1. Introducción

1.1. Propósito

El propósito de este documento consiste en describir las especificaciones técnicas relevantes para la correcta administración del Producto "Pedidos Online - DUSA".

1.2. Referencias

En la siguiente tabla se listan nombre y descripción de los documentos que son referenciados en este documento:

Nombre	Descripción

1.3. Visión General

En el documento se presentan cada una de las tecnologías utilizadas en el sistema. Para cada tecnología se especifica la capa del sistema donde es utilizada y parámetros de configuración que deben ser considerados para mantenimiento y actualización de las distintas funcionalidades.

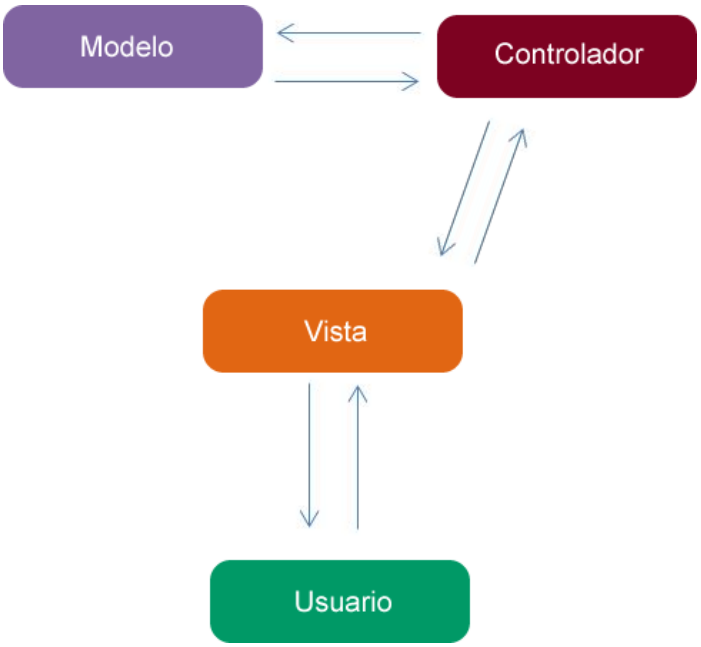
2. Front End

- Backbone

  Backbone.js, es un framework que provee de estructuras para el desarrollo de aplicaciones usando Javascript tomando como base el patrón MVC (modelo-vista-controlador).
  Backbone es dependiente (y está basada) en una librería llamada Underscore.js que provee de funciones para el manejo más cómodo de JavaScript.

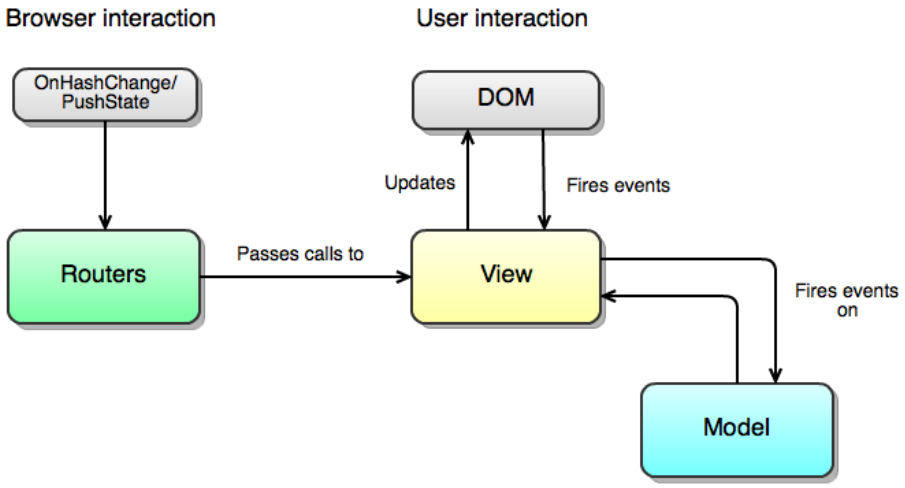
- Hacia el MVC

  El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.
  El MVC ideal funciona de la siguiente manera: el usuario se relaciona directamente con la vista. La vista reporta eventos al controlador. El controlador interactúa con el modelo y reporta a la vista los cambios del modelo. Y la vista se renderiza de forma que se le muestra al usuario los cambios hechos.



Como hemos dicho, este es el modelo ideal, pero en la realidad no se implementa así.

- Cómo implementa backbone a MVC?
  Backbone implementa MVC de una manera un poco diferente al modelo real. Se utilizan vistas, modelos y routers.
  El usuario va a interactuar con la vista, la vista va a reportar los eventos que generó el usuario al modelo y finalmente el modelo reporta sus cambios a la vista mediante Event Listeners. Y mediante el router se renderiza la vista correspondiente.



- Principales estructuras de Backbone
  Modelo: es el que maneja los datos interactivos y el comportamiento de los mismos dentro del dominio de la aplicación, así como una gran parte de la

lógica involucrada: conversiones, validaciones, etc. Responde a peticiones de información acerca del estado de los datos e instrucciones para cambiar el estado de esos datos.

Ejemplo de un modelo

```
Person = Backbone.Model.extend({
        defaults: {
                name: 'Fetus',
                 age: 0,
                child: ""
        },
        initialize: function(){
                 alert("Welcome to this world");
        }
});
 var person = new Person({ name: "Thomas", age: 67, child: 'Ryan'});
```

Vista: es la que está encargada de reflejar la información del modelo en una forma que sea más presentable para la interacción. Es la encargada también de escuchar a los eventos y reaccionar frente a ellos. Típicamente renderiza los datos del modelo en elementos de interfaces de usuario.

```
 SearchView = Backbone.View.extend({
        el: "#search_container",
        initialize: function(){
                this.render();
        },
        render: function(){
                var template = _.template( $("#search_template").html(), {}
);
                this.$el.html( template );
        },
         events: {
                 "click input[type=button]": "doSearch"
        },
        doSearch: function( event ){
                 // Button clicked, you can access the element that was
clicked with                                event.currentTarget
                alert( "Search for " + $("#search_input").val() );
        }
});
```

Router: se utiliza para manejar las URL's de la aplicación usando hash tags (#). Este es un agregado del MVC tradicional.

```
var AppRouter = Backbone.Router.extend({
        routes: {
                "*actions": "defaultRoute"
        }
});

 // Initiate the router
var app_router = new AppRouter;
app_router.on('route:defaultRoute', function(actions) {
         alert(actions);
})
```

- Ejemplo de ciclo completo backbone

  A continuación se muestra un pequeño ejemplo usando backbone.js que calcula el área de un rectángulo dado su alto y ancho y que lo muestra en pantalla. Por un lado tendremos el modelo, en nuestro caso el rectángulo con las propiedades alto y ancho que nos servirán para calcular el área, por el otro la vista que se actualizará según los datos del modelo introducidos en dos campos de texto, captura los eventos en los campos de texto, modifica el modelo y el modelo notifica a la vista para que se actualice.

```
var Rectangulo = Backbone.Model.extend({
        defaults: {
                alto: 4,
                ancho: 3,
        },
        area: function() {
                return this.get('alto') * this.get('ancho');
        },
        toTemplateJSON: function() {
                var json = this.toJSON();
                json.area = this.area();
                return json;
        }
});
var Vista = Backbone.View.extend({
        el: '#area',
                events: {
                "change input[name='alto']": 'onChangeAlto',
                "change input[name='ancho']": 'onChangeAncho'
        },
        initialize: function() {
                _.bindAll(this, 'render', 'onChangeAlto', 'onChangeAncho');
                this.model = new Rectangulo();
                this.model.on('change', this.render);
                this.render();
        },
        onChangeAlto: function() {
                var v = parseInt($("input[name='alto']", this.el).val());
                this.model.set('alto', v);
```

```
        },
        onChangeAncho: function() {
                var v = parseInt($("input[name='ancho']", this.el).val());
                this.model.set('ancho', v);
        },
        render: function() {
                $("input[name='alto']", this.el).val(this.model.get('alto'));
                $("input[name='ancho']", this.el).val(this.model.get('ancho'));
                var texto = Mustache.render('El área de un rectángulo de alto
{{alto}} y ancho
                        {{ancho}}          es          <b>{{area}}</b>',
this.model.toTemplateJSON()
                $('#resultado', this.el).html(texto);
        },
});

var              v              =              new              Vista();

<div id="area">
        <label for="alto">Alto</label>
        <input id="alto" name="alto" value="">

        <label for="ancho">Ancho</label>
        <input id="ancho" name="ancho" value="">

        <div id="resultado" style="margin-top: 2em"></div>
</div>
```

- Bootstrap

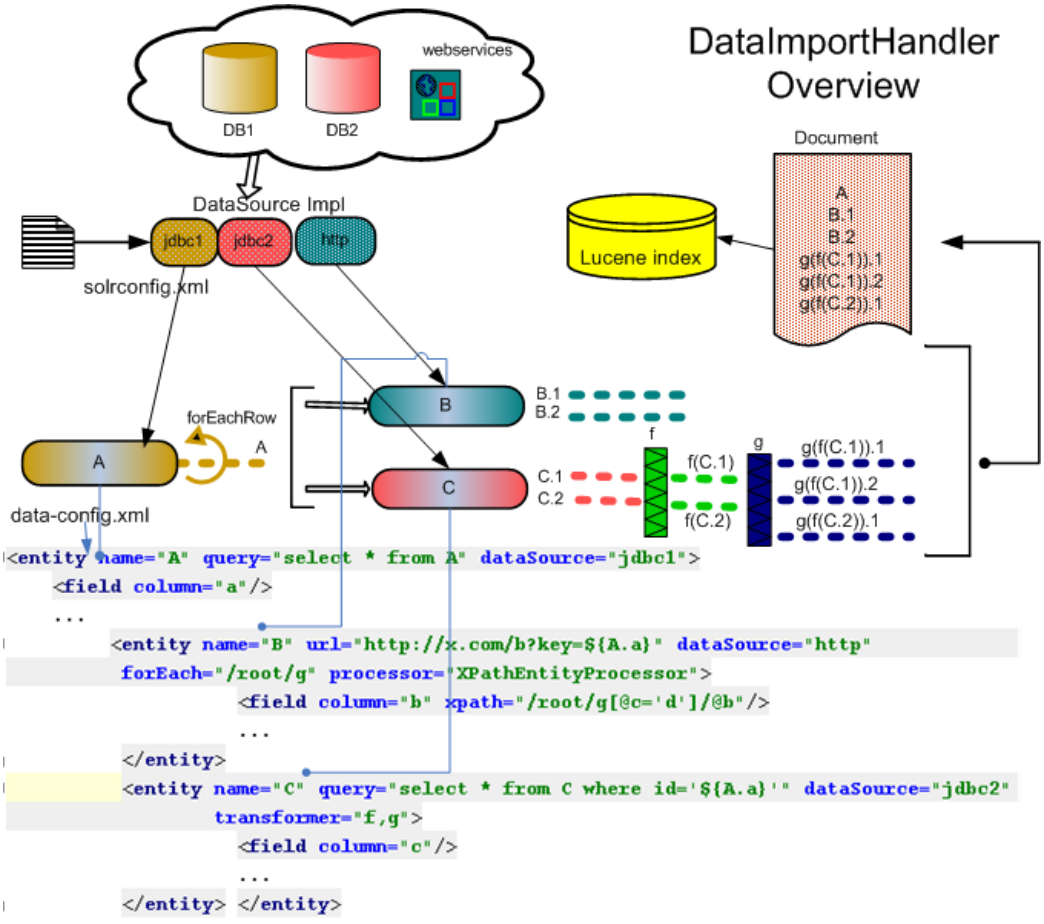
3. Motor de Búsqueda

- **SOLR - Introducción**

Apache solr es un popular motor de búsqueda de código abierto que proporciona potentes funcionalidades de búsqueda y navegación por facetas, es decir , explorar la información desde diversas perspectivas. Solr está basado en la biblioteca Java del proyecto Lucene, con APIs en XML/HTTP y JSON, resaltado de resultados búsqueda por facetas entre otros servicios.
El motor de búsqueda solr se levanta mediante el servidor Jetty, el cual viene configurado por defecto con la aplicación, este corre en el puerto 8983.

En nuestro caso utilizamos el data Import Handler (dih) de solr como home al correr la aplicación, este es el manejador de datos de solr.
Esto fue utilizado para indexar los datos a buscar a partir de la base de datos de DUSA.

La siguiente figura describe el comportamiento de Solr usando el data import handler:



Los archivos más importantes que se deben configurar son:

- schema.xml
- solrconfig.xml
- db-data-config.xml

Estos se encuentran en el directorio:
solr-4.4.0/example/example-DIH/solr/db/conf/

En nuestro caso estarán ubicados en el siguiente directorio:
solr-4.4.0/solr_DUSA/dih/solr/db/conf/

Por otra parte se debe incluir el driver de la base de datos utilizada para realizar la indexación de los datos en solr.
El driver debe ser colocado en el siguiente directorio:
solr-4.4.0/solr_DUSA/dih/solr/db/lib/

Observación: en cada carpeta se encuentra un README.txt el cual posee información sobre la configuración de dichos archivos.

schema.xml

El esquema define la estructura de los documentos a gestionar: los campos que componen a un documento, el formato o tipo de cada uno, y como van a ser analizados los datos en el momento del indexado.
Lo primero que debemos definir en nuestra estrategia de indexado es qué datos vamos a almacenar en el índice y con qué objeto.
El resultado de la búsqueda se presenta como un listado que muestra una serie de datos del registro en cuestión. Independientemente de si un campo participa o no en el indexado, podemos almacenarlo en el índice y recuperarlo junto con los resultados de nuestras consultas.
Podríamos dividir las aplicación de los campos en los siguientes grupos:

- Campos para la búsqueda de texto

Son los campos principales de búsqueda. Un mismo dato puede analizarse de diferentes formas generando distintos campos de búsqueda en el esquema.

- Facetas y filtros

Campos que nos permiten afinar la búsqueda, ya sea sesgando el resultado con filtros o rangos, o distribuyendo los resultados según taxonomías.

Un campo puede pertenecer a uno o más de estos grupos. Es por tanto importante que sepamos qué campos queremos incorporar y cual va a ser su papel en el buscador.

Los campos que queremos indexar se configuran de la siguiente manera:

```
<field name="nro_articulo" type="long" indexed="true" stored="true"
required="true"     multiValued="false" />
<field name="lab" type="text" indexed="true" stored="true"/>
<field name="lin" type="text" indexed="true" stored="true"/>
```

En el ejemplo se ven indexados los campos nro_articulo, lab, y lin.
lab y lin se indexan como tipos "text" , este tipo se define en la siguiente configuración:

```
<fieldType name="text" class="solr.TextField" positionIncrementGap="100">

<analyzer type="index">
        <charFilter class="solr.MappingCharFilterFactory" mapping="mapping-
FoldToASCII.txt"/>
      <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="0" catenateNumbers="0"
catenateAll="0"                    preserveOriginal="1"/>
      <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt" />
      <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
      <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.NGramFilterFactory" minGramSize="1"
maxGramSize="40"/>
    </analyzer>
    <analyzer type="query">
        <charFilter class="solr.MappingCharFilterFactory" mapping="mapping-
FoldToASCII.txt"/>
      <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="0" catenateNumbers="0"
catenateAll="0"
                preserveOriginal="1"/>
      <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt" />
      <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
      <filter class="solr.LowerCaseFilterFactory"/>
    </analyzer>
  </fieldType>
```

Dentro de la configuración del field Type podemos encontrar los siguientes campos:

- En esta etiqueta se configuran los parámetros para indexar los documentos de búsqueda deseados en solr:
  <analyzer type="index">

  </analyzer>

- En esta etiqueta se configuran los parámetros para buscar los documentos indexados:
  <analyzer type="query">

  </analyzer>

- Dentro de cada etiqueta analyzer se pueden definir filtros de búsqueda e indexado, por ejemplo:

- El siguiente filtro no hace diferencias entre mayúsculas y minúsculas
  <filter class="solr.LowerCaseFilterFactory"/>

- El siguiente filtro hace un match de las palabras con diferentes delimitadores, por ejemplo la etiqueta catenate Words: catenateWords="0" devuelve como resultado wi , fi
  "wi-fi" => "wi" , "fi"
  mientras que
  catenateWords="1" devuelve como resultado wifi
  "wi-fi" => "wifi"

<filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0" preserveOriginal="1"/>

- El siguiente filtro busca los sinónimos configurados en el archivo synonyms.txt y los reemplaza para la palabra buscada:

<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false"/>

En la referencia de la página de apache solr se puede encontrar más información sobre los filtros de búsqueda(AnalyzersTokenizersTokenFilters#TokenizerFactories).

solrconfig.xml
Solrconfig.xml és la configuración general del servidor, donde caben parámetros de gestión de la caché, memoria, buffers, etc. Así como también se referencia el uso el Data Importa Handler antes mencionado.

A continuación se muestra parte de la configuración del solrconfig.xml donde se indica el uso del Data Import Handler y el archivo donde está configurado la consulta a la base para indexar los elementos en solr.

```
<requestHandler name="/dataimport"
class="org.apache.solr.handler.dataimport.DataImportHandler">
   <lst name="defaults">
        <str name="config">db-data-config.xml</str>
   </lst>
 </requestHandler>
```

db-data-config.xml

DataImportHandler gestionará las peticiones de la url /dataimport, usando la configuración definida en solr/config/db-data-config.xml.
En este archivo indicamos la consulta a la base , la configuración de la base (url,usuario, contraseña, etc), y el driver de la misma.

Por ejemplo: la siguiente etiqueta indica que el atributo CLAVE1 se indexa en un campo de solr llamado clave1 configurado en el schema.xml

```
 <field column="CLAVE1" name="clave1" />
```

```
<dataConfig>
   <dataSource name="dusa" driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1234:develop" user="GRUPO3"
password="gr3841"/>
  <document name="item">
   <entity name="stock" dataSource="dusa" query="select NRO_ARTICULO,
NOMLAB, LIN from stock natural join laboratorios where HABILITADO = 'S' AND
PRECIO_VENTA > 0">
     <field column="NRO_ARTICULO" name="nro_articulo" />
     <field column="NOMLAB" name="lab" />
     <field column="LIN" name="lin" />
       </entity>
 </document>
</dataConfig>
```

Una forma de indexar los documentos una vez configurado el anterior archivo es:
http://localhost:8983/solr/solr/dataimport?command=full-import
este comando se ejecuta en la logica.jar del sistema construido, una vez que arranca la aplicación.
También se cuenta con un script para re-indexar los datos en solr.

Ejecutar Solr

Para ejecutar solr desde consola debemos hacer lo siguiente:

1. posicionarse en el directorio solr_DUSA
2. ejecutar lo siguiente:
   java -Dsolr.solr.home="./dih/solr/" -jar start.jar

Con la siguiente ejecución solr queda corriendo en el servidor Jetty.

Otra forma es ejecutarlo mediante el script:

1. 1solr.sh el cual contiene:

java -DSTOP.PORT=1891 -DSTOP.KEY=DUSA -Dsolr.solr.home="./dih/solr/" -jar start.jar

El puerto 1891 es un puerto donde solr escucha para darlo de baja.

Dar de baja Solr

Para el primer caso basta con ingresar 'ctrl c'.

Para el segundo caso se puede dar de baja mediante el script:
pararSolr.sh

Referencias:

En los siguientes sitios se encuentra información clara y concisa de la configuración y comportamiento de Solr.
http://www.brujuleo.es/esquemas-en-solr/
http://wiki.apache.org/solr/
http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=apacheSolrIntro

4. Persistencia

El modelo de Datos del sistema se encuentra en el documento Modelo_De_Datos.doc

- JPA - Persistencia de Datos en una Base de Datos relacional
    - comentarios de qué es, y en qué módulos del sistema se utiliza,

  Se utilizó la API JPA la cual ofrece la funcionalidad de realizar un mapeo objeto-relacional.

  Dicha API facilita el mapeo de objetos java a objetos de una Base de Datos Relacional
  para utilizarla se deben agregar anotaciones en las clases para indicar por ejemplo multiplicidades, claves foráneas, etc.

- JDBC - Para conexión a la Base Datos
  Se utilizó el driver utilizado fue oracle.jdb.OracleDriver.

- JPQL - Lenguaje de consulta sobre Base de Datos relacionales persistida con JPA - Dicho lenguaje se utiliza en el Modulo AccessDB, para obtener, modificar y eliminar entidades.

  Un ejemplo de su uso:

```
public Usuario QObtenerUsuario(String idUsuario) throws DusaException {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("Persistence");
        EntityManager em = emf.createEntityManager();
        Usuario u= null;
        try {
            em.getTransaction().begin();
            u = em.find(Usuario.class, idUsuario);
            em.getTransaction().commit();
            if(u == null)
                throw new DusaException(1, "El Usuario no se encuentra en la
base datos del Sistema");

        } catch (Exception e) {
            e.printStackTrace();
            em.getTransaction().rollback();
            throw new DusaException(11, "Ha ocurrido un error en la Base de
Datos del Sistema, disculpe las molestias ocasionadas");
        } finally {
            em.close();
            emf.close();
        }
        return u;
```

```
}
```

- Configuración
  Para configurar el acceso a la base de datos se debe modificar el archivo persistence.xml el cual se encuentra en Logica / src / META-INF / persistence.xml
  El contenido del mismo es el siguiente:

```
 <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:oracle:thin:@localhost:1234:develop"/>
      <property name="javax.persistence.jdbc.password" value="gr3841"/>
      <property name="javax.persistence.jdbc.driver"
value="oracle.jdbc.OracleDriver"/>
      <property name="javax.persistence.jdbc.user" value="GRUPO3"/>
      <property name="eclipselink.ddl-generation" value="create-tables"/>
    </properties>
  </persistence-unit>
</persistence>
```

- Comentar
  Se ha implementado un thread en la clase threadDB, el cual se encarga de actualizar en la Base de Datos la información de los pedidos abiertos que existen en el sistema cada un cierto período de tiempo configurable en el archivo que se encuentra en
  / Logica / src / com / dusa / logica / configuraciones / config.properties.
  la propiedad se llama timeBackup y se expresa en milisegundos, por defecto dicho tiempo es media hora. pero puede ser modificada en cualquier momento sin detener el servidor.

Referencias:
[JPA] http://docs.oracle.com/javaee/5/tutorial/doc/bnbpz.html
[JPQL] http://docs.oracle.com/javaee/6/tutorial/doc/bnbtg.html

5. Comunicación

Comentar las distintas formas de interacción del sistema mediante base de datos y servicios, descripción, hacer referencia al documento de Arquitectura y diseño para esto.

- Servicios - REST
  - en qué capa y módulos se utilizan, mostrar un ejemplo de invocación,
  - comentar como modificaria un servicio si quisiera agregar o quitar parámetros que retorna (poner un ejemplo)

6. Seguridad

- Encriptación de Datos
    - Para la encriptación de las contraseñas desde el browser hasta el servidor se utilizó MD5.

7. Configuraciones

- Especificar la configuración del timeout de la sesión.

- Especificar la configuración de habilitar y deshabilitar una farmacia para que utilice la web.

7.1 Configuración de envío de Mail

Para realizar el envío de correo electrónico mediante la aplicación se utilizó una librería de Java llamada "JavaMail".

Configuración del archivo Config.properties

- Nombre del servidor de correo
  smtpHost=mail.dusa.com.uy

- Puerto del Servidor
  smtpPort=587

- Dirección de envío
  smtpUserFrom=web.gr3@dusa.com.uy

- Contraseña de la Dirección de envío
  smtpPass=1j5Ke9Vn

- Dirección que recibe el correo electrónico, esta dirección será utilizada para el caso en que un Usuario-Web envía un mensaje a D.U.S.A. a través de la Aplicación Web.
  smtpUserTo=web.gr3@dusa.com.uy

Configuración Archivo Mail.java

```
Properties props = new Properties();

// nombre del servidor de correo
props.setProperty("mail.smtp.host",Config.getProperty("smtpHost"));

// puerto del servidor para envío de correos
props.setProperty("mail.smtp.port",Config.getProperty("smtpPort"));
```

```
     // usuario que envía el correo
    props.setProperty("mail.smtp.user", Config.getProperty("smtpUserFrom"));


    // si requiere autenticación
    props.setProperty("mail.smtp.auth", "true");

   //conexión segura
  props.put("mail.smtp.starttls.enable", "true");
  props.put("mail.smtp.ssl.trust",Config.getProperty("smtpHost"));

 //obtengo sesión
 Session session = Session.getDefaultInstance(props);

//nuevo mensaje
MimeMessage message = new MimeMessage(session);

// usuario que envía el correo
message.setFrom(new InternetAddress(Config.getProperty("smtpUserFrom")));

//usuario que recibe el correo: si el envío es por Restablecimiento de contraseña o
por Envío de //Comprobante se especifica el mail del Usuario Web, si es por Envío
de Mensaje a través de //Contacto se especifica la dirección que recibirá los
mensajes
message.addRecipient(Message.RecipientType.TO, new
InternetAddress(mailUsuarioTo));

//asunto de mensaje
message.setSubject("Asunto");

// cuerpo de mensaje
message.setText(texto);


 //si el envío es por Restablecimiento de contraseña o por Envío de Comprobante se
envía el mensaje como Html, si es por Envío de Mensaje a través de Contacto se
envía texto plano.
if(!tipo.equals(tipoMail.Contacto)){
        message.setText(texto,"ISO-8859-1","html");
}

//obtengo objeto transport para enviar mail
Transport t = session.getTransport("smtp");



//me conecto al servidor de correo con un Usuario y contraseña
t.connect(Config.getProperty("smtpHost"),Config.getProperty("smtpUserFrom") ,
Config.getProperty("smtpPass"));
```

```
//envío Mensaje
t.sendMessage(message, message.getAllRecipients());

//cierro la conexión
t.close();
```

8. Infraestructura

Plataformas soportadas (versiones de navegadores y SO, requerimientos mínimos del sistema):

- El sistema soporta los siguientes navegadores desde la versión mencionada en adelante: Internet Explorer 8.0, Mozilla Firefox 12.0, Chrome 30.0.

9. Librerías

- comentar las librerías que sean importantes y no se hayan comentado en las secciones anteriores