UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Pedidos Online - DUSA

Documentación Técnica

Versión 1.6

Proyecto Ingeniería Software 2013

Grupo 03

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. 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.

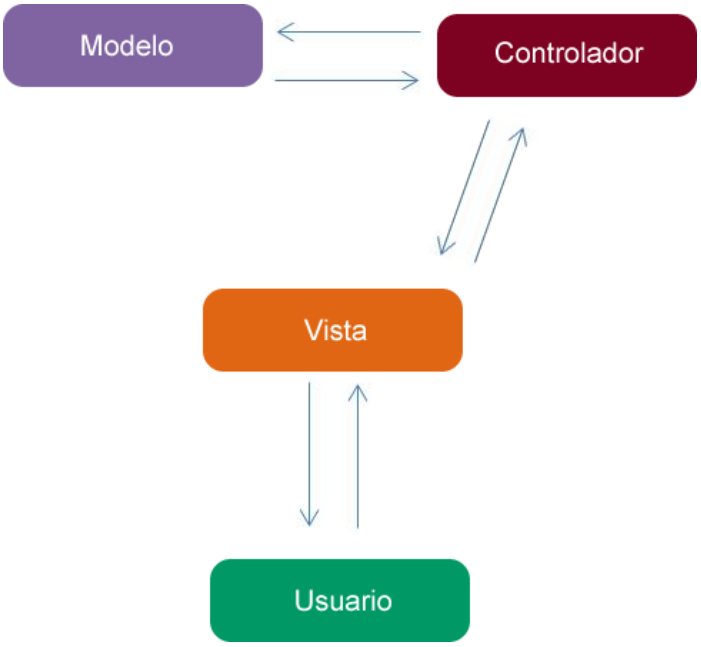
2. Front End

- 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

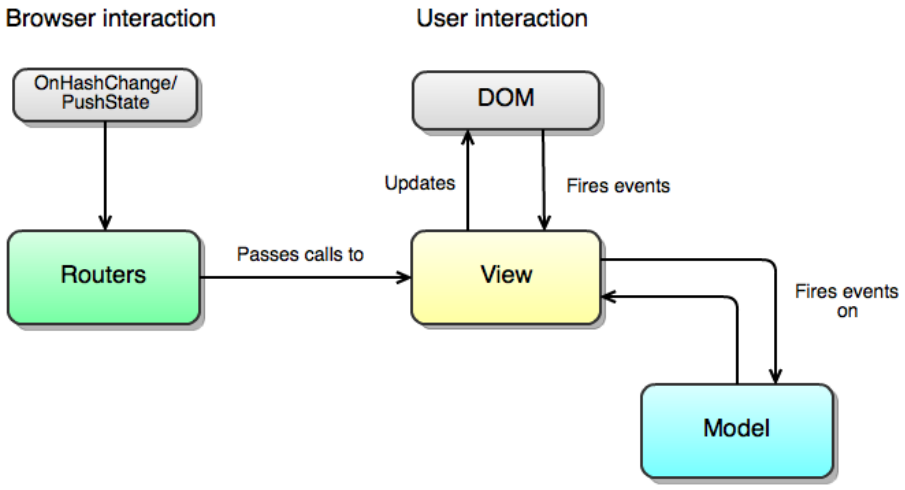
  El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.
  El MVC ideal funciona de la siguiente manera: el usuario se relaciona 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>
```

3. Motor de Búsqueda

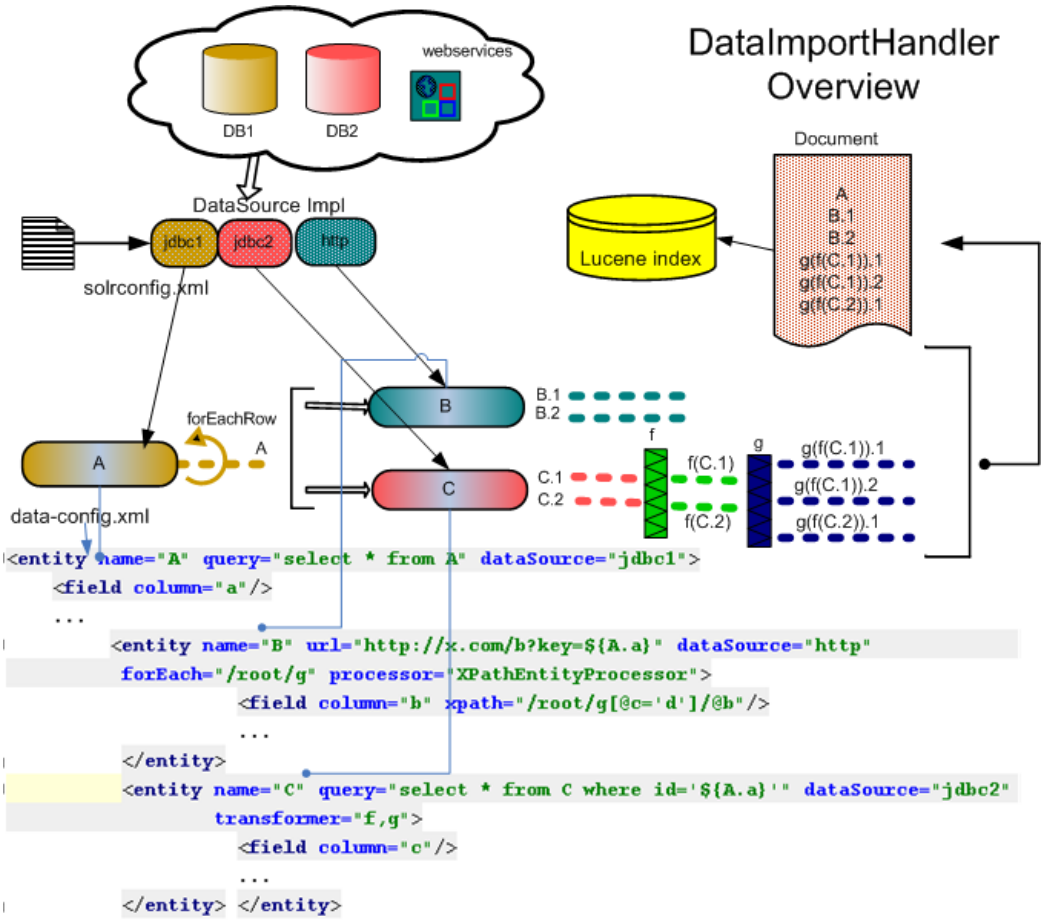
- **SOLR - Introducción**

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 una API en Java (SolrJ) con formatos XML/HTTP y JSON.

El motor de búsqueda solr se levanta sobre el servidor web Jetty, el cual viene configurado por defecto con la aplicación, este corre en el puerto 8983. Este puerto puede ser cambiado editando en el archivo solr.xml (ubicado en `/solr-4.4.0/solr_DUSA/dih/solr`) el atributo hostPort.

La incorporación de documentos al índice del motor puede ser realizada mediante archivos xml que contengan los distintos campos que se desean indexar, o mediante el Data Import Handler (DIH). El DIH permite realizar una consulta SQL sobre una base de datos para obtener así los documentos a indexar.

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

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 estar ubicado 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 la 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
```

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 (Analyzers, Tokenizers, TokenFilters, TokenizerFactories).

solrconfig.xml

Solrconfig.xml es la configuración general del motor, 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>
```

Otra parte importante es el requestHandler utilizado por defecto, que es quien en nuestro caso resuelve las consultas de búsquedas realizadas en el sitio. Debajo se ve la configuración actual, donde se tiene entre otros parámetros, el porcentaje de coincidencia con la búsqueda (mm), el peso que debe tener cada campo para luego ordenar los resultados (qf y mlt.qf), la cantidad default de resultados de búsqueda por página (rows, aunque este valor es sobrescrito luego por el valor utilizado en el archivo properties), el tipo de búsqueda que se hace (defType), entre otros parámetros.

```
<requestHandler name="/select" class="solr.SearchHandler" default="true">
  <!-- default values for query parameters -->
   <lst name="defaults">
     <str name="echoParams">explicit</str>
     <str name="defType">edismax</str>
     <str name="qf">
        descripcion^10 lab^5
     </str>
     <str name="df">text</str>
     <str name="mm">100%</str>
     <str name="q.alt">*:*</str>
     <str name="rows">10</str>
     <str name="fl">*,score</str>
     <str name="wt">json</str>
     <str name="mlt.qf">
        descripcion^10 lab^5</str>
   </lst>
</requestHandler>
```

En este link debajo (además de en las referencias citadas al final del documento) podrán encontrar más información sobre este archivo y sus diferentes configuraciones:
http://wiki.apache.org/solr/SolrConfigXml

db-data-config.xml

DataImportHandler gestionará las peticiones de la url /dataimport, usando la configuración definida en este archivo. En él, indicamos el driver de conexión a la base y la configuración de conexión (url, usuario, contraseña, etc.). Además, también se incluye la consulta a la base para obtener los datos a indexar y un mapeo entre las columnas de la tabla obtenida y los campos dentro del esquema definido anteriormente.

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`

Dar de baja Solr

Basta con ingresar ‘ctrl c’ o ejecutar el comando kill con el PID asignado a solr para dar de baja a la aplicación.

4. Persistencia

El modelo de Datos del sistema se encuentra en el documento "Documentación Modelo de Datos.pdf"

- Para gestionar la persistencia se utilizó **JPA - Java Persistence API**:
  Es un framework del lenguaje de programación Java que maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard (Java SE) y Enterprise (Java EE). El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional) y permitir usar objetos regulares (conocidos como POJOs).
  En particular se utilizó el proveedor **EclipseLink.**

- EL driver utilizado para conectar la aplicación con la base de datos es **oracle.jdbc.OracleDriver**.

- Se utilizó **JPQL - Java Persistence Query Language:** un lenguaje de consulta orientado a objetos independiente de la plataforma definido como parte de la especificación Java Persistence API (JPA).
  JPQL es usado para hacer consultas sobre entidades almacenadas en una base de datos relacional. Está inspirado en gran medida por SQL, y sus consultas se asemejan a las consultas SQL en la sintaxis, pero opera con objetos entidad de JPA en lugar de hacerlo directamente con las tablas de la base de datos.

  Dicho lenguaje se utiliza en el Módulo AccessDB, para obtener, modificar y eliminar entidades.

- 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:@servidor7.dusa.com.uy:1521: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>
```

  - La frecuencia con que se persistirán los datos alojados en memoria en la Base de datos por el thread comentado en el documento Modelo de Datos.pdf es configurable desde el archivo **config.properties** que se encuentra en el home del servidor cabo: **/home/grupo3/ config.properties**.
    La propiedad se llama **timeBackup** y se expresa en milisegundos, actualmente dicho tiempo es media hora, pero puede ser modificada en cualquier momento sin detener el servidor.

5. Comunicación

Para comunicar los distintos subsistemas se utilizaron servicios web basados en diseño REST.

El estilo se centra más en interactuar con recursos con estado, que con mensajes y operaciones. Es más escalable y particularmente útil en dispositivos con escasos recursos como PDAs o teléfonos móviles, esto pensando en usuarios con recursos de hardware limitados o futuro desarrollo de una aplicación movil.

Servicios - REST

REST (Representational State Transfer) es un estilo de arquitectura para desarrollar servicios. Los servicios web que siguen este estilo deben cumplir con las siguientes premisas

- Cliente/Servidor: Como servicios web son cliente servidor y definen un interface de comunicación entre ambos separando completamente las responsabilidades entre ambas partes.
- Sin estado: Son servicios web que no mantienen estado asociado al cliente. Cada petición que se realiza a ellos es completamente independiente de la siguiente. Todas las llamadas al mismo servicio serán idénticas.
- Servicios Uniformes :Todos los servicios REST compartirán una forma de invocación y métodos uniforme utilizando los métodos GET, POST, PUT, DELETE.

Servicios para Comunicación con Aplicación Web

En la clase BrowserController.java dentro del paquete "WebController" se implementan los servicios que consume la Aplicación Web, estos servicios se publican en la url relativa: "Web/ws/api/NombreServicio".

Servicios para Comunicación con la Lógica

En la clase ControladorServicios.java dentro del paquete "services" se implementan los servicios que comunican la Aplicación Web con la Lógica. Estos servicios se publican en la url "http://localhost:9999/ws/api/NombreServicio".

Anotaciones para servicios REST:

```
//Tipo de operación
@POST, @GET, @PUT, @ DELETE
//Path relativa del servicio
@Path("/ServicioEjemplo")
//Indica que el servicio consume Json
@Consumes(MediaType.APPLICATION_JSON)
//Indica que el resultado de la ejecución del servicio se devuelve en formato Json
@Produces(MediaType.APPLICATION_JSON)
```

Para consumir un servicio REST debemos realizar lo siguiente:

```
String REST_URI_PATH = "http://localhost:9999/ws";
```

```
String SERVICIOEJEMPLO_PATH = "api/ServicioEjemplo"
ClientConfig config = new DefaultClientConfig();
Client client = Client.create(config);
WebResource service = client.resource(REST_URI_PATH);

//Consumimos el servicio indicando que la respuesta sea un objeto JSON, la
//respuesta del servicio se carga en RespuestaServicio

ClientResponse response = service.
                          path(SERVICIOEJEMPLO_PATH).
                          type(MediaType.APPLICATION_JSON).
                          post(ClientResponse.class,RespuestaServicio);
```

Los datos se manipulan en formato JSON, pueden importarse en cualquier sitio Web, se representa mejor la estructura de los datos y se requiere menos codificación y procesamiento. Además, la velocidad de procesamiento es mayor que con XML.
Para el manejo de datos JSON hemos utilizado la librería: org.codehaus.jettinson.json

6. Seguridad

- Encriptación de Datos

Para la encriptación de las contraseñas desde el browser hasta el servidor se utilizó el algoritmo MD5 de la misma forma que es utilizado en sistemas UNIX y GNU/Linux para calcular el hash de las claves de los usuarios.

- Autenticar Usuario.

Los usuarios inician sesión indicando al sistema su usuario web y password mediante el servicio web de iniciar sesión (/iniciarSesion). Cuando un usuario inicia sesión en el sistema, en el sistema se genera un token único que tendrá una validez predefinida. El token es devuelto al servidor web y recordado mediante el uso de cookies.

Si un usuario intenta ejecutar un método desde otro navegador, o luego de expirado el período de validez del token generado entonces no será autorizado. En el caso que un usuario inicie sesión desde un nuevo navegador, entonces el token será regenerado y el período de validez es renovado, al regenerarse el token para un mismo id de usuario, la sesión abierta en el primer navegador expira ya que los tokens no coincidirán. De la misma forma, al finalizar sesión, el id de usuario y su token asociado son dados de baja en la memoria del servidor de aplicación: MemoryController.java y se eliminan las cookies que recuerdan los mismos datos en la aplicación web para complementar lo anterior.

Para generar los token, se utiliza la librería java.util.UUID y se invoca el método UUID.randomUUID. De esta manera generamos identificadores de sesión única, difícil de forzar y predecir.

Se trata de una librería utilizada con estos propósitos, es una versión disponible a partir de Java 5 como alternativa a los métodos anteriores que utilizaban SecureRandom y MessageDigest.

El token generado al iniciar sesión para un usuario y el timestamp son guardados en un datatype asociado al usuario: TokenInfo.java.

Para la autenticación de usuarios se dispone de un web service publicado en la clase BrowserController.java (/autenticar). El mismo recibe como parámetro el id de usuario y el token de la sesión, y devuelve una respuesta de éxito o error dependiendo si la autenticación en el servidor de aplicación es exitosa o falla.

Previo a la invocación de un método en el servidor web, se invoca el servicio autenticar para solo admitir usuarios autenticados en el sistema. En caso de una autenticación exitosa continua la ejecución con normalidad, si la autenticación falla, entonces el servidor web devuelve un error de autenticación http. (403 – Forbbiden).

Creemos que una posible mejora para implementar en futuras iteraciones sería evitar la invocación del servicio web para autenticar y que el id de usuario y token sean enviados como parámetros en todas las invocaciones de los métodos. De esta manera, en la lógica se ejecuta el método de autenticación previo a la ejecución de cualquier otro método y se evita consumir un servicio web exclusivamente para autenticar.

- Cookies.

Cuando un usuario inicia sesión, su nombre de usuario, token de sesión e id de farmacia son recordados mediante el uso de cookies. El sitio requiere el uso de cookies, por lo tanto los navegadores deberán tener el uso de cookies habilitado.

7. Configuraciones

7.1 TimeOut de Sesión de Usuario Web

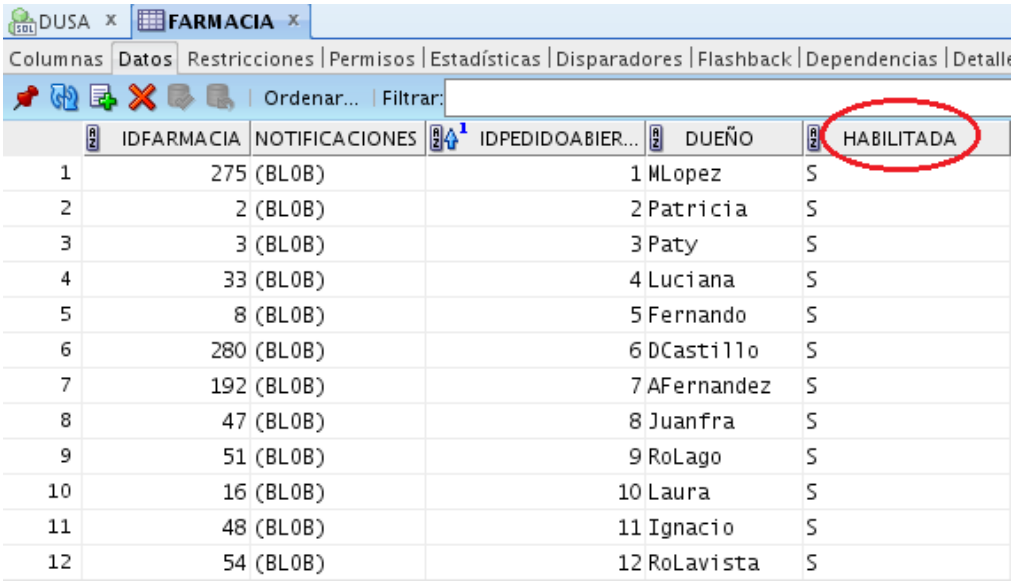
El tiempo de timeout de la sesión de usuario es configurable desde el archivo **config.properties** que se encuentra en el home del servidor cabo: **/home/grupo3/ config.properties**.
La propiedad a modificar es **ttlSesionUsuario**, actualmente está seteada con 1.800.000 milisegundos (equivalente a 30 minutos).
Una vez modificada la propiedad se debe realizar el Deploy de la aplicación nuevamente, para que los cambios sean reflejados en la aplicación.

7.2 Habilitar o Deshabilitar Farmacia

Mediante la modificación del atributo HABILITADA en la tabla FARMACIA se puede controlar el acceso al sitio web.

Para restringir el acceso de determinadas Farmacias se debe setear dicho atributo en "N".
Para Habilitar una Farmacia se debe setear en "S".

Una vez modificado el atributo solo los usuarios asociados a una farmacia habilitada podrán ingresar a la aplicación.



DUSA | FARMACIA

Columnas | Datos | Restricciones | Permisos | Estadísticas | Disparadores | Flashback | Dependencias | Detalle

Ordenar... | Filtrar:

	IDFARMACIA	NOTIFICACIONES	IDPEDIDOABIER...	DUEÑO	HABILITADA
1	275	(BLOB)	1	MLopez	S
2	2	(BLOB)	2	Patricia	S
3	3	(BLOB)	3	Paty	S
4	33	(BLOB)	4	Luciana	S
5	8	(BLOB)	5	Fernando	S
6	280	(BLOB)	6	DCastillo	S
7	192	(BLOB)	7	AFernandez	S
8	47	(BLOB)	8	Juanfra	S
9	51	(BLOB)	9	RoLago	S
10	16	(BLOB)	10	Laura	S
11	48	(BLOB)	11	Ignacio	S
12	54	(BLOB)	12	RoLavista	S

7.3 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

- 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

- Librerías utilizadas en /Logica:

Librerías utilizadas para implementación de Servicios REST:

- genson-0.98.jar
- gson-2.2.4.jar

Librería utilizada para el envío de mails:

- javax.mail.jar

Librerías de Solr:

- solr-analysis-extras-4.4.0.jar
- solr-cell-4.4.0.jar
- solr-clustering-4.4.0.jar
- solr-core-4.4.0.jar
- solr-dataimporthandler-4.4.0.jar
- solr-dataimporthandler-extras-4.4.0.jar
- solr-langid-4.4.0.jar
- solr-solrj-4.4.0.jar
- solr-test-framework-4.4.0.jar

- solr-uima-4.4.0.jar
- solr-velocity-4.4.0.jar
- commons-io-2.1.jar
- httpclient-4.2.3.jar
- httpcore-4.2.2.jar
- httpmime-4.2.3.jar
- jcl-over-slf4j-1.6.6.jar
- jul-to-slf4j-1.6.6.jar
- log4j-1.2.16.jar
- noggit-0.5.jar
- slf4j-api-1.6.6.jar
- slf4j-log4j12-1.6.6.jar
- wstx-asl-3.2.7.jar
- zookeeper-3.4.5.jar

JDBC Driver 11g Release 1:

- 11gR1-11.1.0.6.0-ojdbc6.jar

10. Referencias:

[SOLR] http://www.brujuleo.es/esquemas-en-solr/
[SOLR] http://wiki.apache.org/solr/
[SOLR]http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=apacheSolrIntro
[SOLR] Libro: Apache Solr 4 Cook Book, Capítulo 2 , How to import data using Data ImportHandler and delta query
[JPA] http://docs.oracle.com/javaee/5/tutorial/doc/bnbpz.html
[JPQL] http://docs.oracle.com/javaee/6/tutorial/doc/bnbtg.html