Sistema de gestión de farmacias

Documentación Técnica

Versión 1.2

Historia de revisiones

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 08/11/2014 | 1.0 | Se documentaron las características técnicas más relevantes del sistema. | Victoria Díaz, Ignacio Decia |
| 09/11/2014 | 1.1 | Se agregaron datos de la arquitectura | Ignacio Decia |
| 09/11/2014 | 1.2 | Se agregaron datos de configuración de Solr | Victoria Díaz |

**Contenido**

1. Introducción 3

2. Arquitectura 3

3. Tecnologías utilizadas 3

4. Configuraciones Técnicas 4

5. Librerías utilizadas 5

6. Problemas encontrados 6

1. Introducción
   1. Propósito

El objetivo de este documento es brindar una descripción general de los aspectos técnicos de la aplicación y de las funcionalidades configurables en el sistema.

1. Arquitectura
   1. Descripción

La arquitectura del sistema está dividida en varias capas, estas son: capa de presentación, capa lógica, capa de acceso y capa de persistencia.

Este modelo sigue una estructura jerárquica donde cada capa podrá comunicarse únicamente con su capa inmediata inferior.

Cada capa tiene una responsabilidad y la comunicación entre ellas será mediante una interfaz bien definida:

* La **capa de presentación** se encarga de presentar los datos y procesar las entradas de los usuarios.
* La **capa lógica** se encarga principalmente de la lógica de negocio, en este caso todo lo que comprende la operativa de una farmacia.
* La **capa de acceso** permite la comunicación con la base de datos y los servicios externos de D.U.S.A.
* La **capa de persistencia** consiste en la base de datos física.

Además se cuenta con un conjunto de entidades que son objetos que se utilizan para la comunicación entre las distintas capas. De esta forma se tiene una forma uniforme para la comunicación entre ellas. Estos objetos representan entidades de la realidad, por ejemplo, artículos, ventas, pedidos, etc.

1. Tecnologías utilizadas

Se utilizó la plataforma Java SE Development 7u67 para el desarrollo de la aplicación y el servidor Apache Tomcat 7.0.55 para la ejecución del mismo.

La base de datos utilizada es PostgreSQL y en nuestro caso, el desarrollo del software se realiza utilizando el mismo como servidor (localhost) para la base de datos y para el Tomcat. Igualmente este es un parámetro configurable y se podría modificar en un posterior desarrollo y utilización del producto.

Además, se utilizó el motor de búsqueda Apache Solr 4.10.1.

1. Configuraciones Técnicas

A continuación se describen las configuraciones necesarias para la correcta interacción entre los distintos componentes del sistema.

* 1. Configurar conexión con la base de datos

La conexión con la base de datos se hace a través del servidor, para lograr la misma se necesita configurar la url, usuario y contraseña de la base de datos así como también una librería.

Estos parámetros se deben incluir al path de sistema de la siguiente manera:

CATALINA\_OPTS =

-Dpostgres.url="jdbc:postgresql://localhost:5432/##NOMBRE BD##"

-Dpostgres.user="##USER BD##"

-Dpostgres.pass="##PASS BD##"

-Dpostgres.driver="org.postgresql.Driver"

Como se puede observar, el último parámetro referencia a la librería de postgres utilizada. Esta misma se encuentra en la carpeta WebContent/WEB-INF/lib.

* 1. Configuración de Apache Solr

Para configurar Solr son necesarios una serie de librerías en la carpeta de tomcat (detalladas más adelante), otras librerías que se deben encontrar en el WebContent/WEB-INF/lib para poder utilizar Sorj, la API de Solr para Java.

Por otra parte, les será entregada una carpeta con los archivos de Solr, la cual tiene que ser ubicada en algún lugar del servidor.

Al igual que con la base de datos, es necesario agregarle un parámetro de tomcat en el path del sistema de la siguiente manera:

CATALINA\_OPTS =

-Dsolr.solr.home="##RUTA A LA CARPETA SOLR EXTRAIDA##"

* 1. **Configurar archivos de propiedades**

Se tienen dos archivos de propiedades con configuraciones para las alertas del sistema. Para especificar los momentos en los que se disparan las alertas se utilizan expresiones de Cron. Una expresión de Cron es un String que representa un momento dado en el tiempo. Puede ser, determinado día a determinada hora, todos los 25 de cada mes, etc. Las expresiones de Cron son una forma estándar y flexible que nos permiten indicar los momentos exactos en los que se debe ejecutar una alerta.

* alertasVenta.properties que tiene los siguientes valores:
  + diaDelMes: expresion de Cron que indica el día del mes en que se dispara la alerta.
  + mesesAtras: cantidad de meses que se va a tener en cuenta en la operación *articulosConPocasVentasEnLosUltimosMeses(mesesAtras)*.
* alertasStock.properties que tiene los siguientes valores:
* fechaUltimaActualizacion: corresponde a la fecha de la última actualización realizada. El formato de la fecha es el mismo que el de la clase Date.
* horas: expresión Cron que indica las horas del día en la que se dispara la alerta.

1. Librerías utilizadas

A continuación se describen las librerías utilizadas para el proyecto.

Las siguientes librerías se encuentran en WebContent/WEB-INF/lib:

* SolrJ
  + commons-io-2.3.jar
  + httpclient-4.3.1.jar
  + httpcore-4.3.jar
  + httpmime-4.3.1.jar
  + noggit-0.5.jar
  + wstx-asl-3.2.7.jar
  + zookeeper-3.4.6.jar
* Mínimos cuadrados
  + commons-math3-3.3-javadoc.jar
  + commons-math3-3.3.jar
* Alertas
  + quartz-2.2.1.jar
  + quartz-jobs-2.2.1.jar
* Prime faces
  + primefaces-5.0.jar
* Postgresql
  + postgresql-9.2.1003.jdbc4.jar
* Mail
  + javax.mail.jar

Por otro lado, se deben incluir en la carpeta /lib de Tomcat las librerías:

* Solr
  + jcl-over-slf4j-1.7.6.jar
  + jul-to-slf4j-1.7.6.jar
  + log4j-1.2.17.jar
  + slf4j-api-1.7.6.jar
  + slf4j-log4j12-1.7.6.jar

1. Problemas encontrados

Se reportó un problema de performance en el caso de uso Generar Pedido Automático. El problema se debía a que en cada operación de la persistencia se abría una nueva conexión con la base de datos, ejecutaba la consulta y luego cerraba la conexión. Dado que abrir una conexión con la base de datos es una operación costosa, se decidió utilizar un pool de conexiones abiertas, cada vez que una operación requiere comunicarse con la base de datos pide por una conexión disponible en el pool y cuando termina la libera. De esta forma se evita abrir y cerrar conexiones de manera innecesaria.