KUBEET S. DE R.L. DE C.V.
AV. PONIENTE 7 No. 73 INT 2 COL. CENTRO
ORIZABA VER. C.P. 94300
01(272)72-5-94-28
+52 (272)122-11-11
www.kubeet.com
adolfo.centeno@kubeet.com

252233- FITOSMART: PLATAFORMA TECNOLÓGICA DE FITOMONITORIZACIÓN DE CULTIVO HIDROPÓNICO UTILIZANDO CÓMPUTO SENSIBLE AL CONTEXTO Y TÉCNICAS DE INTELIGENCIA ARTIFICIAL
(Tercera Etapa)

Programa de Estímulos a la Innovación

2018

CONACYT

AN_E2_03_Analisis y Diseño del módulo de aprendizaje CBR

Contiene la descripción de la arquitectura y diseño conceptual del módulo de razonamiento basado en casos en la plataforma web FitoSmart.

CONTENIDO

I. INTRODUCCIÓN

El presente documento describe el proceso de análisis y diseño del módulo de CBR para el proyecto Entorno Modular de Simulación en su segunda etapa; en él, se describen la estructura y arquitectura que tendrá el aplicativo que da funcionalidad al módulo de CBR. Dicho módulo tiene como finalidad llevar a cabo el proceso de un motor de CBR el cual de manera breve se puede describir en tres etapas:

1) La primera etapa es el almacenamiento de los Casos en la base de conocimientos de CBR. Dentro del contexto del proyecto, los Casos para el motor de CBR son las prácticas que pueden realizarse en el Fitotrón. Estas prácticas deben haber sido concluidas para formar parte de la base de conocimientos del motor de CBR y no deben ser alteradas en el futuro, para no generar errores de inconsistencia en la base de conocimientos.
2) La segunda etapa es el proceso de recuperación de casos; la recuperación de los casos se lleva a cabo a partir de una búsqueda sobre la base de conocimientos de CBR, usando como parámetro una práctica realizada en el Fitotrón y las variables que lo conforman; tomando en cuenta esa información, el motor de CBR realiza una búsqueda a través de los algoritmos de distancia euclidiana y el vecino más cercano que se encuentran para determinar el porcentaje de similitud que tiene el caso de búsqueda con los demás casos registrados en la base de conocimientos. Se utiliza la ponderación de las variables presentes en las prácticas (casos) para obtener la similitud entre casos.
3) En la tercera etapa, el módulo de CBR proporcionará una recomendación con base en la reputación de las prácticas realizadas en el Fitotrón; las reputaciones estarán dadas a partir de 2 criterios, primero las calificaciones proporcionadas por el usuario operador quién realiza la práctica y el usuario experto que evalúa la práctica realizada en el Fitotrón, las calificaciones son números enteros y se encuentran dentro de un rango de 1 a 10; el segundo criterio es el resultado del usuario experto que evaluará la práctica como un EXITO o FRACASO.

Una vez descrito el funcionamiento del módulo de CBR se procede a seleccionar las mejores herramientas tecnológicas para llevar a cabo el desarrollo del sistema.

Tomando en consideración que en la actualidad las plataformas de desarrollo de un motor de CBR están bien cimentadas en Java; productos maduros, estables y de comprobada reputación y rendimiento para esta tarea han dado resultados excepcionales para el desarrollo de software, cuya finalidad es la de implementar un motor de CBR. Es por ello que la plataforma Java ha sido elegida como la plataforma de desarrollo para el módulo CBR del Fitotrón.

Cabe mencionar que el desarrollo del software del Fitotrón está basado en la plataforma de .NET es por ello que el módulo de CBR debe proveer de las interfaces adecuadas para poder interactuar con .NET; es entonces que el desarrollo del módulo de CBR expondrá la funcionalidad de almacenamiento de los casos en la base de conocimientos y su posterior recuperación a través de servicios Web en formato SOAP.

Previo al análisis del módulo CBR, en la etapa inicial del proyecto se determinó que la administración de las variables obtenidas del proceso y las prácticas que podrán realizarse en el Fitotrón, así como los tipos de prácticas definidos en la primera fase del proyecto, los cuales serán administrados mediante catálogos.

En las siguientes secciones del documento se describen los detalles técnicos para la realización del módulo de CBR para el proyecto de FitoSmart.

II. DEFINICIONES, ACRONIMOS Y ABREVIATURAS

A. Paquete

Agrupación lógica de elementos estructurales, estos elementos pueden ser clases, interfaces u otros paquetes.

B. Modelo de codificación

El modelo de codificación está basado en lenguaje de programación Python los cuales describen que la asignación de nombres a una clase debe utilizar la notación "camel case", por ejemplo: MiClase en donde los nombres compuestos para una clase se conjuntan y siempre empieza con mayúsculas, para cada palabra incluida en el nombre de la clase. También se recomienda una anotación similar para los atributos y métodos de una clase sólo que siempre deben iniciar con minúscula, por ejemplo: lazyModel (atributo) y getLazyModel (método); se sigue también la especificación de Java para la documentación de código fuente a través de anotaciones de JavaDoc.

III. DEFINICIÓN DEL SISTEMA

El módulo de CBR se define como un aplicativo JEE basado en un enfoque SOA, con soporte para servicios Web SOAP con seguridad a través de la especificación WS Security y con la implementación de UsernameToken (autenticación por nombre de usuario y contraseña). Incluye también el soporte de jColibri 2 para la implementación del motor de CBR, y cuenta con un enfoque de desarrollo basado en Spring y Spring Security 3.0.7. El proyecto está administrado por Maven para manejar dependencias, plugins de construcción del aplicativo y perfiles para el despliegue del aplicativo en diversos entornos, ya sea de desarrollo, QA y finalmente de producción. Se incluye también el soporte de JPA con la implementación de Hibérnate para tener un desarrollo basado en el modelo de dominio y no depender de las capacidades específicas de un gestor de bases de datos en particular; no obstante se recomienda el uso de MySQL por su versatilidad, rapidez y gran eficiencia para manejar grandes volúmenes de información.

El proyecto de desarrollo se denomina *"Modulo CBR"* el cual es un proyecto Maven multi-módulos; lo cual significa que la funcionalidad completa del aplicativo estará divida físicamente en varios proyectos Java configurados por Maven para proveer la funcionalidad final de un aplicativo Web JEE.

Los módulos de los cuales se conforma el aplicativo *"Modulo CBR"* se describen a mayor detalle en el documento de arquitectura del módulo CBR.

A. Implementación de un motor CBR

La finalidad principal del aplicativo *"Modulo CBR"* es la de proveer un motor de CBR el cual sea capaz de registrar las prácticas (también denominadas casos en el contexto de CBR) realizadas por los operadores de la CABINA en la base de conocimientos de dicho motor, para posteriormente ser explotados a través de la similitud de las prácticas en base a las ponderaciones de las variables del proceso AHP. Y proveer una recomendación al usuario de cuales practicas se asemejan más a la práctica que funge como búsqueda y se agrega la reputación de la práctica para que seleccione la más adecuada para ser reutilizada dentro del aplicativo de .NET que interactuará con la CABINA.

Para llevar a cabo esta tarea se utiliza el soporte de jColibri 2; previo análisis para la implementación de un motor de CBR y su capacidad de realizar correctamente el cálculo de la similitud entre los casos de la base de conocimientos y esta puede explotarse a través un esquema de base de datos tradicional de SQL. JColibri 2 proporciona diversos mecanismos para llevar a cabo estas tareas; de los utilizados en el proyecto "*Modulo CBR*" se hace uso de los siguientes recursos Java:

LocalSimilarityFunction que es una interface la cual permite definir el mecanismo de comparación a ser utilizado para los casos.

CaseComponent que es una interface la cual permite definir que objetos serán utilizados como "casos" y cuál será su atributo identificador dentro de la base de conocimientos.

NNConfig que es una clase para definir el mecanismo de recuperación de los casos a través de la configuración de similitud utilizada, para el caso del aplicativo se utiliza el método de recuperación por promedio a través de la clase ***Average*** también del API de jColibri.

Los recursos descritos son los principales para la implementación de un motor de CBR con jColibri 2; el enfoque para crear la base de conocimientos se describirá más adelante.

B. Implementación de la base de conocimientos

La base de conocimientos en el contexto de CBR es un repositorio en donde se almacena la información más relevante para la definición de los casos; dicho repositorio tiene como función mantener un soporte histórico del registro de los casos y su recuperación a través de mecanismos de similitud como los provistos por jColibri 2.

Para el aplicativo del módulo CBR, la base de conocimientos se reflejará en una base de datos SQL y por el soporte de JPA con el que cuenta el aplicativo. Es posible utilizar diversos gestores de bases de datos como MySQL, PostgreSQL, MS SQL Server, Oracle, entre otros, con un mínimo de configuración para que sea portable el modelo de dominio de *"Modulo CBR"* para adecuarlo al gestor final donde quedará alojada la base de conocimientos.

Las entidades principales para la base de conocimientos se describen a continuación:

TipoVariableAHP. Es la entidad encargada de almacenar los tipos de variable definidos para el contexto del Fitotrón. A nivel de base de datos la entidad es mapeada con la tabla ***tbTipoVariable***.

TipoPractica. Es la entidad encargada de almacenar los tipos de práctica que pueden realizarse dentro del Fitotrón. A nivel de base de datos la entidad es mapeada con la tabla ***tbTipoPractica***.

VariableP. Es la entidad encargada de almacenar las variables presentes dentro de una práctica realizada en el Fitotrón. A nivel de base de datos la entidad es mapeada con la tabla ***tbVariable***.

PracticaCBR. Es la entidad encargada de almacenar las prácticas realizadas dentro de la CABINA, por lo tanto esta es la entidad que representa los casos en el sistema. Una práctica no debe ser aceptada dentro de la base de conocimientos a menos que ya se haya culminado y no se realicen modificaciones posteriores a la misma, pues esto afectaría el rendimiento y la finalidad del motor de CBR; sin importar cual fuese el resultado de la práctica, una vez concluida debe pasar a formar parte de la base de conocimientos del módulo de CBR. A nivel de base de datos la entidad es mapeada con la tabla ***tbPracticaCbr***.

Perfil. Es la entidad encargada de almacenar los perfiles de los usuarios que tendrán acceso al aplicativo Web y a los servicios Web, cada perfil determinará las acciones que puedan realizar los usuarios. A nivel de base de datos la entidad es mapeada con la tabla **tbPerfil**.

Usuario. Es la entidad encargada de almacenar la información de los usuarios que tendrán acceso al sistema. Esta entidad se conforma con base en los conceptos de seguridad descrito en Spring Security 3.0.7 para permitir a un usuario acceder a los recursos y páginas de un aplicativo Web Java al igual que permitir el acceso a los servicios Web publicados en el aplicativo. A nivel de base de datos la entidad es mapeada con la tabla ***tbUsuario***.

La descripción de las entidades presentada no toma en cuenta los atributos de las clases Java y tampoco los campos con los cuales corresponden en las tablas de la base de datos, pues estos serán descritos en la sección del diccionario de datos.

C. Implementación de los servicios Web

La integración entre dos sistemas que no forman parte de la misma plataforma de desarrollo se puede hacer a través de diversas técnicas, y una de las más implementadas a nivel empresarial, por su rendimiento, capacidad de respuesta e interconexión son los servicios Web SOAP; los cuales permiten a través de un contrato de especificación y descripción del servicio Web denominado WSDL, permiten exponer las operaciones que pueden ser consumidas a través de mensajes XML y recibir las respuestas de dichas operaciones, también en formato XML, para posteriormente aplicar un mecanismo de conversión y obtener una respuesta acorde a la plataforma de desarrollo de la cual se consume el servicio.

Para el caso de *"Modulo CBR"* el aplicativo provee el soporte de servicios Web SOAP con Apache CXF que proporciona una pila muy extensa de tecnologías para proveer esta funcionalidad, incluyendo la implementación de WS Security.

La finalidad de los servicios Web que se exponen es la de almacenar las prácticas realizadas en el Fitotrón; y explotar posteriormente la base de conocimientos a través de una práctica previamente almacenada que servirá de método de búsqueda para realizar el procedimiento de comparación y obtención de las prácticas a través de jColibri 2 y añadiendo a los resultados la reputación de las prácticas como mecanismo de recomendación de las mismas.

El servicio Web implementado se denomina ***"módulo de razonador basado en casos"*** el cual proporciona las siguientes operaciones:

AlmacenarPracticaCBR. Cuya funcionalidad es la de registrar las prácticas en la base de conocimientos con las variables AHP asociadas a la misma, las calificaciones del operador y el experto, si el resultado es éxito o fracaso y un identificador único para la práctica proveniente del Fitotrón.

RecuperarPracticasCBR. Cuya funcionalidad es la de recuperar las prácticas más parecidas a aquella que se utilice como método de búsqueda, en caso de ser una práctica ya almacenada en la base de conocimientos bastará con proporcionar el identificador de la práctica del Fitotrón y el porcentaje de similitud que se debe cumplir entre los resultados obtenidos por los mecanismos de recuperación y cálculo de similitud de jColibri 2.

El servicio Web ***"módulo de razonador basado en casos"***, sus operaciones y sus resultados se describen con mayor claridad en el documento de integración de servicios Web del proyecto *"Modulo CBR"*.

IV. MODELO DE NEGOCIO

El modelo de negocio está basado en la premisa de que el módulo de CBR es una herramienta que debe acoplarse al sistema desarrollado para la plataforma web y las prácticas a realizarse dentro de la misma como se ha descrito anteriormente.

Por lo cual la finalidad que debe cumplir es la de crear y gestionar un motor de CBR y una base de conocimientos asociada a las prácticas realizadas dentro del Fitotrón.

A. Situación actual

La situación actual abordada por el proyecto "*Modulo CBR*" es el administrar a través de un proyecto multi-módulos de Maven; un aplicativo que sea escalable, robusto y escalable para garantizar un mejor desempeño en cuanto a su funcionalidad y finalidad.

La integración de los diferentes módulos que conforman el proyecto "*Modulo CBR*" se detalla en la descripción arquitectónica.

V. DESCRIPCIÓN ARQUITECTÓNICA

A continuación se establece una organización concreta de elementos que, en su conjunto, interactúan para constituir una propuesta de estructura inicial para el desarrollo de aplicativos Web, basados en reconocidos estándares tecnológicos que se enumeran a continuación.

1) Lenguaje de programación Python
2) Spring 3.0.7 como orquestador para los siguientes servicios:
    a. Inyección de dependencias
    b. Envío de correos
    c. Seguridad
    d. Conectividad a bases de datos
3) JPA 2 (Instrumentado por Hibérnate 4.2.10.Final como ORM)
4) Maven 3 como administrador de proyectos y control de dependencias
5) myCBR 3 SDK y jColibri 2 como librerías de CBR.

Como se mencionó en el inciso 4, la herramienta de compilación, empaquetado y despliegue que se utiliza es Maven, lo cual promueve la independencia de un IDE de desarrollo. Esto implica que el desarrollador puede, con toda libertad, utilizar el IDE de su preferencia, siempre y cuando respete la estructura de paquetes y clases propuesta por esta arquitectura.

El servidor de aplicaciones para el despliegue de los aplicativos desarrollados con esta arquitectura puede ser cualquier contenedor de Servlets 3.0. A continuación se presenta una lista con los nombres y versiones de los servidores de aplicaciones Java que a la fecha se ha identificado como aquellos capaces de soportar esta arquitectura:

1) Tomcat 7.0.x (aunque Tomcat no es un "full app-server", actualmente SI soporta la arquitectura)
2) Glassfish 3.1.2
3) WebLogic 12c
4) JBoss 6.0
5) WebSphere 8.0

VI. REQUERIMIENTOS

Los requerimientos mínimos necesarios que el ambiente de desarrollo debe cumplir, se enumeran a continuación:

1) Python 2.7 y 3
2) Maven y acceso a Internet
3) Contenedor de Servlets 3.0 o superior (Ejem. servidor de aplicaciones que soporte JSF 2.1)
4) Conectividad a un DBMS relacional con privilegios de lectura y escritura a la base de datos que la arquitectura ocupará.
5) Navegador
    a. IE 9.0 o superior
    b. FF 3.4 o superior
    c. Safari 5.1.4
    d. Google Crome en cualquiera de sus versiones
    e. Opera 5.0 o superior
6) Al menos 4GB de memoria RAM
7) Al menos un procesador Core2 DUO a 2.4 GHz

VII. ESTRUCTURA INICIAL

La propuesta arquitectónica establece una estructura de proyecto multi-módulos la cual se compone de varios directorios compatibles con Maven (proyectos Maven) que, inicialmente, deberán respetar la siguiente organización:

```
Proyecto Maven
    ├pom.xml
    ├──src
        ├──main
        │   ├──java
        │   └──resources
        │   └──webapp
        └──test
            ├──java
            └──resources
```

Cada uno de estos elementos será discutido en las secciones siguientes y se hará énfasis en aquellas partes que serán susceptibles de sufrir cambios o adiciones al momento de extender la arquitectura para satisfacer los requerimientos específicos del aplicativo en desarrollo.

A. El modelo de objetos del proyecto

Maven establece, como uno de sus principales archivos de operación, el archivo pom.xml, que contiene la mayor parte de información requerida para compilar, ensamblar, empaquetar y distribuir un proyecto. El archivo pom.xml define, en su estructura, varias secciones. Las cuatro secciones más relevantes para propósitos de explicación de esta propuesta son:

1) Cabecera
2) Repositorios
3) Dependencias
4) Plugins
5) Construcción

B. Cabecera del archivo pom.xml

La cabecera define las propiedades generales del aplicativo, como las que se muestran en la Tabla I.

Tabla I Propiedades generales

Propiedad	Descripción
- Nombre	Nombre corto del aplicativo
- Id del grupo	Por ejemplo, mx.com.rodascomputacion
- Id del artefacto	Por ejemplo, cabinaCBR
- Versión	Por ejemplo, 0.1 -SNAPSHOT
- Empaquetado	En esta arquitectura siempre será 'war'
- Descripción	Descripción general del aplicativo

El ejemplo anterior indica que se creará un proyecto llamado "*Modulo CBR*" con una jerarquía de paquetes de la siguiente forma: mx.com.kubeetcomputacion, que será empaquetado como un archivo "war" y cuya versión

será 0.1-SNAPSHOT, que es una convención de nombrado que indica que el aplicativo se encuentra en etapa de desarrollo. La convención para denotar una versión liberada sería 1.x.y-RELEASE. Para obtener información más detallada acerca de las convenciones de nombrado que deberá llevar cada liberación.

C. Repositorios

Los repositorios indican las fuentes (locales o remotas) de donde se recuperan las bibliotecas que son requeridas y solicitadas en la sección de dependencias. Cada repositorio establece tres secciones: id, nombre y URL. Las tres secciones son obligatorias, pero la de mayor relevancia es la de 'URL', ya que esta sección es utilizada para encontrar la ubicación del servidor Maven de bibliotecas del cual serán obtenidos los recursos y sus dependientes. Este documento no tiene como finalidad explicar el funcionamiento y configuración general de Maven. Únicamente señala las secciones integrantes de la arquitectura que pueden ser extendidas y/o reconfiguradas para que se ajusten a las necesidades específicas de un aplicativo en particular. Será común la necesidad de definir nuevos repositorios adicionales a los que se incluyen en el pom.xml original. A continuación en la Tabla II se presenta una lista de repositorios de uso frecuente:

Tabla II repositorios de uso frecuente

Incluido	URL
SI	http://repository.primefaces.org
SI	http://download.java.net/maven/2/
NO	http://mvnrepository.com/
NO	http://nexus.sonatype.org/

D. Dependencias

La sección de dependencias lista una serie de bibliotecas que deberán ser obtenidas al momento de ensamblar el aplicativo, así como algunas otras requeridas para la compilación del mismo. Esta sección generalmente incrementará el número de dependencias en función de las necesidades del nuevo aplicativo basado en esta arquitectura, sin embargo, las dependencias ya incluidas son necesarias para el mínimo funcionamiento de ésta arquitectura. Actualmente, las dependencias incluidas se presentan en la Tabla III.

Tabla III Lista de dependencias incluidas

Dependencia	Dependencia	Dependencia
jsf-impl	**Testng**	**mysql-connector-java**
Primefaces	Jstl	Visualization-datasource
servlet-api	jsp-api	Hibernate-annotations
jsf-api	guava	
c3p0		

E. Construcción

En la sección de construcción se configura al proyecto para realizar la construcción del paquete, además se declaran los plugins que proveen al proyecto de herramientas de análisis y verificación.

Sin embargo, conviene mencionar que, la sección de 'Construcción' en el ambiente de desarrollo de Eclipse IDE (al menos en Helios 3.6.1) es irrelevante. Esta sección puede estar comentada, ya que Eclipse instrumenta su propia estrategia de construcción y despliegue.

a. Plugins

Dentro de la sección de plugins, contenida en la sección de construcción, la arquitectura define 7 plugins, a saber:

1) El plugin "maven-compiler-plugin" determina la versión de java (1.7) y el file encoding que será utilizado por Maven.
2) El plugin "maven-war-plugin" determina la forma en la que el aplicativo será ensamblado (modalidad 'war') y posiblemente desplegado por Maven.
3) El plugin "maven-site" permite generar reportes a partir del proyecto, a su vez es plugin utiliza los siguientes plugins para los reportes para JavaDoc:
4) El plugin "cobertura-maven" permite verificar el porcentaje de código que las pruebas unitarias están revisando.
5) El plugin "maven-javadoc" genera la documentación JavaDoc en base al código fuente
6) El plugin "maven-checkstyle" realiza una verificación del estilo en base a los estándares sugeridos por Oracle.
7) Finalmente, el plugin "findbugs-maven" realiza una revisión al código fuente en busca de violaciones a criterios establecidos que pudieran conllevar a fallas de seguridad y/o de desempeño

Generalmente no habrá cambios en esta sección. En caso de requerir información general acerca de la configuración de un plugin o sección específica del POM (Project Objec Model) o en general, de Maven, por favor refiérase a la guía de uso de Maven.

F. El directorio de fuentes

El segundo elemento dentro de la jerarquía principal de la estructura de proyecto es el directorio de fuentes 'src'. Acorde a la definición de Maven para el directorio de fuentes, éste deberá estar organizado en subdirectorios con la siguiente estructura:

```
src
├──main
│   ├──python
│   └──resources
│
└──test
    ├──java
    └──resources
```

Las dos principales ramas son 'main' y 'test' que tienen una estructura similar. En ambas, será posible definir paquetes, clases y archivos de configuración potencialmente idénticos. Sin embargo, el propósito concreto de la rama 'main' es contener las piezas que definen, efectivamente, al aplicativo. Mientras que la rama 'test' contendrá código dedicado a probar el que se encuentra en 'main'.

Cada una de las pruebas definidas en la rama 'test' será ejecutada de manera automática por Maven al momento de ser invocado el comando 'mvn package'. En caso de que la prueba de unidad falle, Maven no

producirá el empaquetado y terminará el ciclo de vida con un mensaje indicando cual prueba de unidad ha fallado.

Con base en lo anterior, se procederá a describir la subestructura de la rama 'main', que podrá ser tomada como base para la estructura de la rama 'test'.

Es importante mencionar que todos y cada uno de los métodos que son (y serán) implementados en esta propuesta, deberán ser compatibles con el formato de codificación: "CAMEL-CASE", en donde la primera palabra del método debe estar enser minúsculas y cada palabra siguiente deberá comenzar con su primera letra mayúscula. Por ejemplo: salvaDatosUsuario es un nombre CORRECTO, en cambio, SalvaDatosusuario es INCORRECTO.

1) Rama 'java'

La rama 'java' contiene la estructura tradicional de paquetes y clases contenidas en dichos paquetes. Con base en la información definida en el archivo pom.xml sugerido en esta propuesta (ver definición inicial) la rama 'java' expondrá la siguiente estructura:

```
Java
  └──mx
      └──com
          └──kubeetcomputacion
              └──cabinaCBR
                  ├──service
                  ├──web
                  ├──model
                  ├──persistence
                  └──support
```

De forma resumida, la rama principal 'mx.com.kubeetcomputacion.cabinaCBR' contendrá cinco módulos, a saber:

1) support
2) model
3) persistence
4) service
5) web

2) Modulo 'Model'

Módulo dedicado a contener clases de dominio de problema, que generalmente serán POJOS, también conocidos como DTO's o VO's, es decir, "Data Transfer Objects" o bien, "Value Objects". Se requiere el cumplimiento de las siguientes políticas:

Interfaces a implementar obligatoriamente:

- Serializable

Atributos obligatorios:

- SerialVersionUID (privada, estática, final, tipo long)

Métodos a sobrescribir:

- equals
- hashCode
- toString

Constructores

- Constructor por default
- Constructor mínimo con el ID único del objeto
- Constructor con todos los atributos de la clase

En la Fig. 1 se muestran las dependencias entre y hacia los módulos que 'model' tendrá:

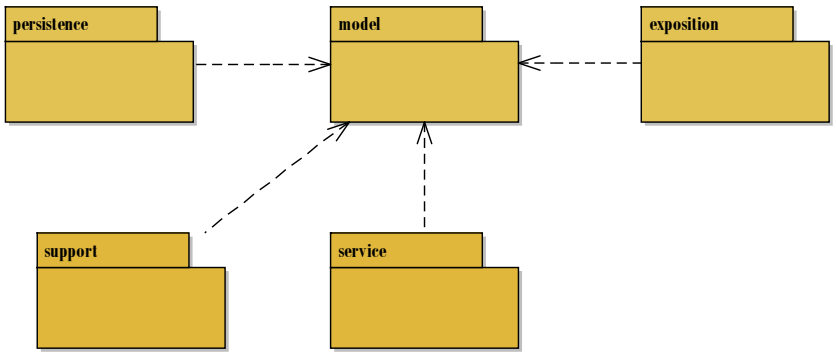


Fig. 1 Dependencias entre módulos 'model'

3) Modulo 'persistence'

Módulo dedicado a contener piezas para el control de objetos persistentes. En general, en este paquete las piezas serán interfaces, con apego a la nomenclatura: *DAO.java y deberán obligatoriamente extender a la interfaz GenericDAO que es genérica y que obligatoriamente requiere de la definición de una clase base que usará para determinar el tipo de objetos que recibirán y retornarán ciertos métodos a ser implementador por JPA 2 (Hibérnate Annotations).

Nota: SI es posible que extiendan a otra interfaz, siempre y cuando ésta última sea una interfaz descendiente de la interfaz GenericDAO.

Las firmas de los métodos que cada interfaz definirá representan las operaciones de persistencia que, en tiempo de ejecución, serán invocados por elementos del módulo 'service' en clases *ServiceImpl.java.

Los métodos que serán expuestos por todas las piezas persistentes (sin necesidad de incorporarlos explícitamente, ya que son heredados de la interfaz GenericDAO) son:

- T findById(ID id);
- List<T> findAll();
- List<T>findByExample(T exampleInstance, String... excludeProperty);
- void persist(T entity);
- void merge(T detachedEntity);
- void refresh(T oldEntity);

- void remove(T entity);
- void flush();

Sin embargo, es factible agregar, a cada interfaz, más métodos en función de las necesidades de negocio, p.e.: User getUserByCorreo (String correo);

Las interfaces *DAO.java son utilizadas por el Hibérnate a través de anotaciones provistas por JPA 2.

La Fig. 1 muestra las dependencias entre y hacia los paquetes que el módulo 'persistence' tendrá:

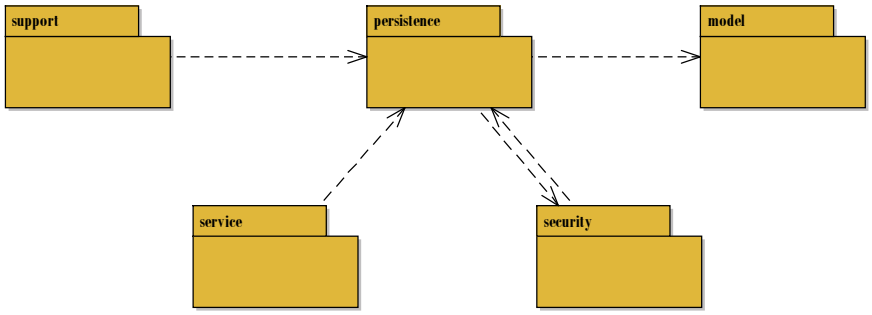


Fig. 2 Dependencias entre paquetes módulo 'persistence'

4) Módulo 'web'

Módulo dedicado a contener las clases que serán consumidas desde la interfaz de usuario del aplicativo. Los clientes típicos de estas piezas son los elementos xhtml y en general, los elementos de 'web' serán 'JSF ManagedBeans'.

La función principal de un ManagedBean es la interacción entre los servicios y la vista. Un ManagedBean recupera los datos capturados en la vista (o JSF 2) y puede efectuar validaciones y cálculos que invoca desde un servicio para retroalimentar a la vista y en su caso, desplegar mensajes de validación o bien mandar ejecutar la funcionalidad de guardado o de actualización de información.

Una de las principales funciones de un ManagedBean es garantizar que la integridad de la información es correcta para poder, con toda seguridad, realizar el proceso de ésta por el *backend* del aplicativo.

Cada ManagedBean deberá ser nombrado acorde a la convención: *MB.java y llevará forzosamente al menos dos anotaciones:

- @Controller y
- @Scope

(Esta última anotación puede ser de tipo: Application, Session, Request o View)

@Controller no lleva atributos extras (como 'name' u otro). De esta forma, los ManagedBeans serán accedidos, desde el jsf, con su mismo nombre, pero sin capitalizar la primera letra. Por ejemplo: PaisMB será accedido como: paisMB desde el *frontend*.

Los ManagedBeans deberán implementar la interfaz 'Serializable', incluir el atributo serialVersionUID, ofrecer un atributo final y estático para 'logging' llamado LOG y en general, serán consumidores del paquete de servicios. No está permitido que un ManagedBean haga referencia directa a métodos de persistencia. Tales métodos serán siempre provistos a través de la capa de servicios. En este punto conviene mencionar que los elementos de tipo ManagedBeans soportan el uso de anotaciones como @Autowired y @Value.

NOTA: De forma general, se deberá observar la política siguiente: Un ManagedBean no podrá poseer un constructor distinto al constructor por defecto.

Si se desea simular la implementación de un constructor no trivial, se deberá implementar un método anotado de la siguiente manera: @PostConstruct. Sin embargo, aun así éste método no puede recibir ningún parámetro formal. Se recomienda que el nombre del método sea "init ()".

El módulo de web poseerá un subpaquete llamado servlets que tiene el propósito de contener las clases de negocio que descienden de HttpServlet y que serán declaradas en el archivo web.xml.

La Fig. 3 muestra las dependencias entre y hacia los paquetes que el módulo 'web' tendrá:

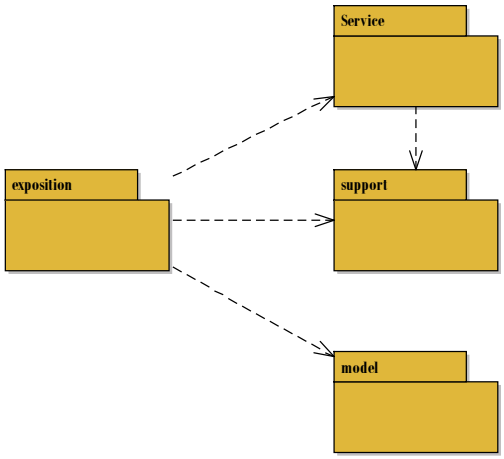


Fig. 3 dependencias entre paquetes módulo 'web'

5) Módulo 'service'

Módulo dedicado a contener clases puramente de negocio. Admite la inclusión de interfaces a ser implementadas por tales clases de negocio. La convención de nombrado será acorde al estándar siguiente:

- Clase de negocio: *ServiceImpl.java
- Interface: *Service.java

Por ejemplo, RealizaPagoNominaService.java y RealizaPagoNominaServiceImpl.java representan la interfaz y la clase que la implementa respectivamente.

Este módulo estará altamente acoplado con los módulos 'model', 'persistence' y 'web'. Adicionalmente, las piezas contenidas en este paquete deberán cumplir con la siguiente reglamentación:

- No tienen dependencias a elementos de 'web' (aunque las clases del módulo de 'web' si dependen de las clases de 'service')
- Incorpora la anotación: @Service que generalmente irá sin parámetros

- Puede hacer uso de la anotación: @PostConstruct, dado que en una clase de servicios no está permitido definir constructores distintos al constructor trivial. Nuevamente, al igual que en las clases de 'web' se hace notar que tales métodos no deben incorporar parámetros formales y se recomienda que el nombre del método anotado de esta forma sea llamado 'init ()'.
- Puede hacer uso de la anotación: @Value para obtener valores asociados a llaves definidas en el archivo de propiedades 'application.properties'.
- Usarán Postfijos Service y ServiceImpl para interfaces y clases respectivamente
- Usan piezas de 'model' y de 'persistence' vía dependencias de SPRING
- Puede hacer uso de la anotación: @Autowired para hacer uso de los objetos de 'persistence' y del mismo 'service'
- Son usadas por piezas de 'web' vía dependencias de SPRING
- Las excepciones a las reglas anteriores deberán estar plenamente justificadas

Uno de los objetivos principales de las anteriores políticas es garantizar, en la medida de lo posible, que los servicios contenidos en el paquete 'service' pueden, en un futuro, ser expuestos, por ejemplo, como 'webServices'. De lo anterior, se infiere que debe tener un bajo o nulo nivel de dependencias a piezas asociadas al *frontend*.

La Fig. 4 muestra las dependencias entre y hacia los módulos que 'service' tendrá:

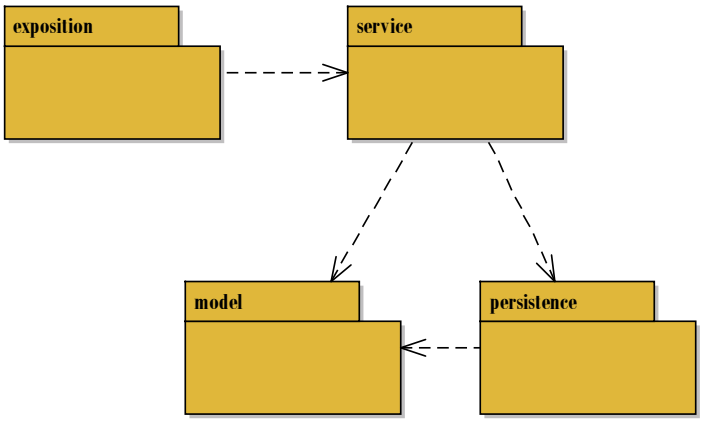


Fig. 4 Dependencia de módulos 'service'

6) Módulo 'support'

El módulo de 'support' incluye cuatro clases utilitarias de propósito general que sirven de apoyo a diversos procesos vitales dentro del aplicativo. Tales piezas se muestran en la Tabla IV.

Tabla IV Clases

Clase	Descripción
LanguageMB	Clase auxiliar para su uso desde el front-end. Permite obtener y definir un idioma (o código de localización) determinado para su uso en la presentación de mensajes obtenidos desde un archivo de recursos adaptado para soportar localización.
PrimeFacesRequestContext	Permite guardar un parámetro que le indica a PrimeFaces si la petición se realizó correctamente.

En general, es altamente recomendable no modificar ninguno de los elementos constituyentes del módulo de support.

La Fig. 5 muestra las dependencias entre y hacia los módulos que 'support' tendrá:

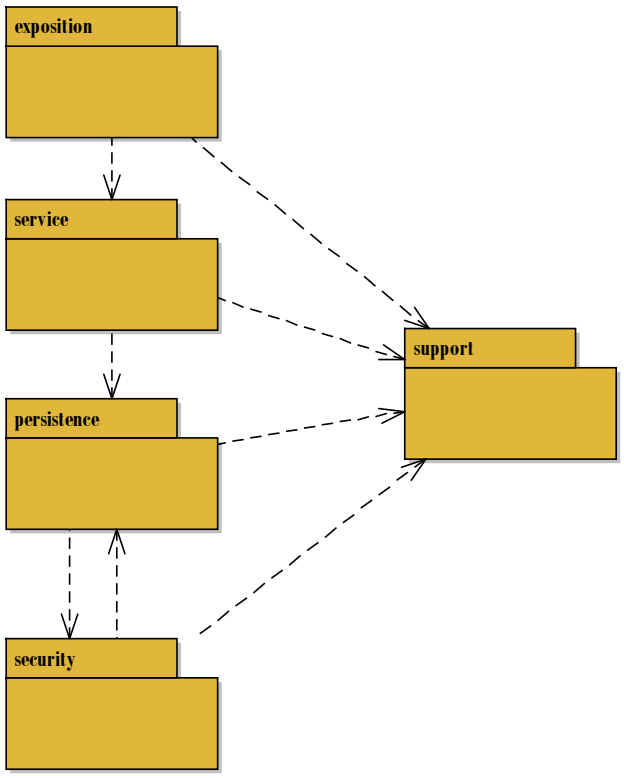


Fig. 5 Dependencias entre módulos 'support'

a. Rama 'Resources'

La rama de 'recursos' contiene una serie de cinco directorios y dos archivos expuestos en el siguiente esquema:

```
src
└──main
└──resources
├──i18n
├──jasper
├──mail
├──spring
├──META-INF
├──log4j.properties
└──application.properties
```

Cada uno de los elementos anteriores será expuesto con detalle en las secciones siguientes.

1) Archivo 'log4j.properties'

Define los appenders para el "log" del aplicativo. El único valor que se recomienda cambiar es el del archivo de log, que en este caso es la ruta y/o nombre del archivo: "frontend.log" y el nivel de debug. Actualmente existen cuatro niveles disponibles: info, debug, warning y error.

2) Archivo de propiedades 'application.properties'

El archivo 'application.properties' contiene la mayoría de valores que, en cierto momento, podrán ser ajustados a los rangos que el aplicativo demande. La Tabla V contiene una lista de las llaves, descripciones de las mismas y valores de ejemplo que podría llevar cada llave.

Tabla V Lista de llaves

Llave	Descripción	Ejemplo
db.driver	Nombre del driver de la base de datos	com.mysql.jdbc.Driver
db.url	Cadena JDBC de conexión a la base de datos	jdbc:mysql://localhost/prueba
db.username	Nombre de usuario de conexión a la base de datos. Éste puede ir encriptado, si se desea, al igual que el valor de cualquier otra llave.	ENC(4loy4c1OKXg9IqwLP0QzIk)
db.password	Clave de conexión a la base de datos. Puede ir encriptado, si se desea.	ENC(sL8nFtHP5rlk77AX2EuKfaz)
mail.active	Establece si el correo será efectivamente enviado o no	FALSE
mail.url	Establece la ruta base que el motor de correo usará ´para establecer la parte inicial de las trayectorias tipo liga que le ser´pan enviadas a un receptor de correo.	/200.38.189.150/arq.front/
mail.forgotPassword.url	Establece la ruta relativa de la forma para restaurar una clave	content/common/restorePassword.jsf
mail.preregistro.url	Establece la ruta relativa de la forma para completar un registro	content/common/CompletaRegistro.jsf
mail.stopMail.url	URL que se usa para solicitar que ya no se mande correo promocional a un usuario dado.	content/common/StopMail.jsf
mail.resources	Directorio de recursos (relativo a 'resources') que contiene las plantillas y archivos de soporte al envío de correos electrónicos.	mail/
mail.template	Plantilla que se usa para enviar correos de recuperación de claves de acceso.	mail.forgot.skeleton
mail.repeatInterval	Número de milisegundo que debe transcurrir entre un envío masivo de correos y el siguiente.	300000
mail.maxQueueSize	Máximo número de correos que pueden ser encolados antes de ser enviados	25
mail.sid.length	Longitud de las cadenas encriptados de recuperación y confirmación de registro del aplicativo	50
mail.host	Host SMTP	webmail.infotec.com.mx
mail.port	Numero de puerto SMTP	25

mail.username	Nombre de usuario de envío de correo	hudson@infotec.com.mx
mail.password	Clave del usuario de envío de correo	ENC(E6+XZLq3koKDivTs4Cp4n7hPerliIM=)
cron.db.ping.interval	#ping to db each 30 minutes	1800000
login.blokedWindowTime	Tiempo que el usuario permanecerá bloqueado (en milisegundos)	300123
login.maxInvalidTries	Número máximo de intentos permitidos antes de bloquear al usuario	5
restore.password.ticketLiveTime	Horas que el "ticket para reestablecer la clave" será válido (OJO: en horas)	24
idle.timeout	Idle Timeout para inactividad en milisegundos: 1000x60x29 = 29 minutos (un minuto menos que el tiempo de la sesión del web.xml)	1740000
passwordValidator.messageResolver	Archivo de mensajes para traducir al castellano los errores de validación de fortaleza de claves	/vt.properties
usuario.minLength	Longitud mínima del usuario	5
defaultTargetUrl	Ruta relativa a donde se re direccionará al usuario en caso de que el proceso de registro al sistema sea correcto	/content/restricted/MenuPrincipal.jsf
defaultFailureUrl	Ruta relativa a donde se re direccionará al usuario en caso de que el proceso de registro al sistema sea incorrecto	/content/common/login.jsf
customSalt	Cadena utilizada para cifrar los passwords de los usuarios	Cadena cifrada
numDaysBitacoraFlush	Número de días que se mantendrán un mensaje en la bitácora de seguridad	7
max.inputText.length	Tamaño máximo definido en las entradas, se usa por medio del bean	500
gmaps.api.key	Llave para las apis de google maps, requerida si se usa gmps4jsf	RZeziNr2i1m...Px4c9ZOmNL0VzM6Y

Para las llaves que tienen asociados valores encriptados, conviene mencionar que tales valores fueron generados mediante el uso de la herramienta JASYPT que permite generar encriptación inclusive ‘distintas’ para la misma cadena de texto a encriptar, esto gracias a que se emplean algoritmos de encriptación de dos vías.

3) Directorio ‘jasper’

Directorio sugerido para albergar los archivos jasper creados para generar reportes por medio de JasperReport.

4) Directorio i18n

Contiene los archivos de propiedades para internacionalización de mensajes. Un archivo de propiedades para internacionalización de mensajes puede tener cualquier nombre y es establecido en el archivo de configuración faces-config.xml. Sin embargo, la nomenclatura que debe tener para que ‘faces’ pueda llevar a cabo el proceso de internacionalización es la siguiente:

- nombre.properties nombre base
- nombre_xx_YY.properties nombre acorde al idioma y país

Los postfijos 'xx' y 'YY' indican idioma y país respectivamente y con apego al estándar internacional de idiomas y países i18n.

Por ejemplo, en la Tabla VI se muestran idiomas – países:

Tabla VI Idiomas y países

xx	YY	Idioma	País
en	US	Inglés	Estados Unidos
en	UK	Inglés	Reino Unido
es	MX	Español	México
fr	CA	Francés	Canadá
zh	CN	Chino	China

5) Directorio mail

Contiene recursos auxiliares para el envío de correos electrónicos. Tales recursos pueden ser plantillas, imágenes y documentos binarios (como pdf's, archivos Word, Excel, entre otros)

No existe una restricción al tipo y cantidad de archivos que es posible almacenar en el directorio, sin embargo, se recomienda mesura. La forma en la que un recurso interno puede acceder a los que se encuentran contenidos en el directorio mail, es a través de su ruta absoluta, que puede ser calculada gracias a valores específicos en el archivo application.properties. En secciones posteriores, se discutirá más acerca del archivo global de propiedades 'application.properties'.

6) Directorio 'spring'

El directorio 'spring' contiene cinco archivos de configuración XML de propósito específico, a saber:

- applicationContext.xml
- applicationContext-cronTasks.xml
- applicationContext-mail.xml
- applicationContext-security.xml

El contexto de Spring (que en este caso incluye los cuatro archivos anteriores, y un archivo adicional dentro del directorio src/main/profiles/ dentro de la estructura de los perfiles de 'development', 'production', 'qa' con el prefijo applicatioContext-persistence*.xml, el cual contiene la configuración de conectividad a base de datos según el perfil y el motor de bases de datos) será cargado por el aplicativo gracias a el listener "mx.com.rodascomputacion.cabinaCBR.support.ProxyContextLoaderListener" declarado en el archivo web.xml.

En las siguientes secciones cada uno de estos archivos será discutido. Sin embargo, conviene mencionar en este punto que es posible agregar uno o más archivos de configuración, acorde a las necesidades del aplicativo que se base en esta arquitectura. La única recomendación sugerida es no eliminar o fusionar alguno de los anteriores archivos. Y respetar la nomenclatura: applicationContext-xyz.xml.

Tal y como se ha venido anunciando, el objetivo de este documento no es proporcionar una referencia del marco de trabajo Spring. La principal intención es constituir una guía para la configuración y distribución de elementos que podrán ser reconfigurados, extendidos y analizados para obtener un mejor entendimiento y/o rendimiento de los aplicativos que adopten esta arquitectura.

7) ApplicationContext.xml

El archivo de configuración applicationContext.xml es el principal archivo de configuración de Spring. En una gran cantidad de casos, éste es el único archivo de configuración, sin embargo, esto ha probado ser una práctica débil, ya que la extensión de tal archivo, puede ser muy grande.

En la propuesta actual, se sugieren las siguientes responsabilidades para este archivo:

1) Definir el bean "propertyConfigurer" que determina cuál será el archivo de propiedades que contendrá las llaves (y valores) que serán requeridos en cualquier archivo de configuración de Spring.(en este caso es el archivo "application.properties")
2) Definir el bean "log4jInitializer" que establece el archivo de propiedades de configuración de log4j.
3) Definir el bean "passwordEncoder" que podrá ser utilizado a lo largo del aplicativo para aplicar encripciones de tipo MD5.

8) ApplicationContext-cronTasks.xml

Este archivo contiene la configuración requerida para las tareas a ejecutar. En este caso se utiliza Spring con Quartz, al momento de bajar la arquitecura se tiene configurado una tarea para el envío regular de correos electrónicos.

El bean "sendMail" especifica que clase y que método serán ejecutados regularmente por el cron de Spring. En este caso se hace referencia al bean 'notificationService' que es el que configura el servicio de correo y se verá en la siguiente sección.

El bean "sendMailTrigger" define la tarea a ejecutar y el tiempo con el que la terea se ejecutará repetidamente. Éste último valor es tomado del archivo application.properties.

Algunos ejemplos de la configuración del cronTrigger son expuestos a continuación:

"0 0 12 * * ?"	Fire at 12pm (noon) every day
"0 15 10 ? * *"	Fire at 10:15am every day
"0 15 10 * * ?"	Fire at 10:15am every day
"0 15 10 * * ? *"	Fire at 10:15am every day
"0 15 10 * * ? 2005"	Fire at 10:15am every day during the year 2005
"0 * 14 * * ?"	Fire every minute starting at 2pm and ending at 2:59pm, every day
"0 0/5 14 * * ?"	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
"0 0-5 14 * * ?"	Fire every minute starting at 2pm and ending at 2:05pm, every day
"0 10,44 14 ? 3 WED"	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
"0 15 10 ? * MON-FRI"	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday
"0 15 10 15 * ?"	Fire at 10:15am on the 15th day of every month
"0 15 10 L * ?"	Fire at 10:15am on the last day of every month
"0 15 10 ? * 6L"	Fire at 10:15am on the last Friday of every month
"0 15 10 ? * 6L"	Fire at 10:15am on the last Friday of every month
"0 15 10 ? * 6#3"	Fire at 10:15am on the third Friday of every month

9) ApplicationContex-mail.xml

Este archivo contiene la configuración requerida para utilizar el api de spring para envío de correo electrónico y adicionalmente, se definen los "beans" para provocar el envío masivo de aquellos mensajes que actualmente se encuentran en espera de ser enviados.

Los "beans" definidos en este archivo de configuración son los siguientes:

- mailSender
- notificacionService

- detalleTarea
- cronTrigger
- schedulerFactoryBean
- sendMailMB

El bean "mailSender" establece las propiedades requeridas por "org.springframework.mail. javamail.JavaMailSenderImpl" que son: "host", "port", "username", "password", "defaultEncoding" y "javaMailProperties". Todos los anteriores, recuperados del archivo de propiedades "application.properties", excepto "javaMailProperties", que establece propiedades especiales, como "mail.debug", "mail.smtp.auth" y "mail.smtp.starttls.enable" que se encuentran alambradas a este rubro, pero que pueden, si se desea, ser establecías también en el archivo de propiedades.

El bean "notificacionService" (que realmente es un proxy) utiliza al bean "mailSender" para poder inicializar las propiedades de la clase "MailEngine" (ver documentación del API de la arquitectura) y al mismo tiempo sirve como un bean que provee un método (llamado "notificar") a ser invocado regularmente por un cron de spring. El método "notificar", a su vez, invoca el envío masivo de correos de MailEngine, llamado sendAllMessages.

10) ApplicationContext-persistence.xml

El archivo "applicationContext-persistence.xml" está destinado a establecer y contener las definiciones de beans de persistencia, incluida la configuración de JPA y las definiciones de datasources.

Actualmente, el archivo "applicationContext-persistence.xml", presenta tres importantes secciones:

- Bean dataSource
- Bean jpaVendorAdapter
- Bean entityManagerFactory
- Bean transactionManager

El bean "dataSource", en esta propuesta, hace uso de un pool de conexiones "enterprise": C3P0 (utilizado en los casos en los que el servidor de aplicaciones no provee un pool de conexiones propio). Adicionalmente, incorpora la recuperación de valores del archivo de propiedades y por otro lado, incorpora valores alambrados, que se considera, cambiarán muy esporádicamente. Sin embargo, tales valores pueden, si se desea, ser generalizados en el archivo de propiedades, de la manera usual.

El bean "jpaVendorAdapter" determina la implementación de JPA en este caso con Hibérnate, la definición del gestor de base de datos, el mostrar el código SQL generado por Hibérnate, y el dialecto de conexión a la base de datos.

El bean "entityManagerFactory" es el encargado de fungir como fábrica de entidades (POJOs anotados con JPA) en donde reutilice el 'dataSource' para conectarse a la base de datos, el adaptador de JPA 'jpaVendorAdapter' y la definición de la unidad de persistencia de JPA.

Finalmente, el bean "transactionManager" es el encargado de administrar el proceso de transacciones a través del bean 'entityManagerFactory' y la conexión a base datos con el bean 'dataSource'.

11) ApplicationContext-securirty.xml

El archivo de configuración 'applicationContext-security.xml' es uno de los más relevantes de esta arquitectura. Establece, vía el contexto global de Spring, las características, tanto de autenticación como de autorización a los recursos del sistema. Particularmente, los recursos asociados al *frontend* del aplicativo.

En este punto conviene mencionar que el modelo de seguridad de Spring (SS3) es realmente una cadena de filtros que interceptan una solicitud y la procesan secuencialmente acorde al orden establecido para la cadena de filtros. Es posible sustituir un filtro específico por uno personalizado, estableciendo de esta manera un nuevo comportamiento para la intercepción dada.

La cadena de filtros existente se muestra en la Tabla VII:

Tabla VII Cadena de filtros

Filtro	Clase
CHANNEL_FILTER	ChannelProcessingFilter
CONCURRENT_SESSION_FILTER	ConcurrentSessionFilter
SECURITY_CONTEXT_FILTER	SecurityContextPersistenceFilter
LOGOUT_FILTER	LogoutFilter
X509_FILTER	X509AuthenticationFilter
PRE_AUTH_FILTER	AstractPreAuthenticatedProcessingFilter
CAS_FILTER	CasAuthenticationFilter
FORM_LOGIN_FILTER	UsernamePasswordAuthenticationFilter
BASIC_AUTH_FILTER	BasicAuthenticationFilter
SERVLET_API_SUPPORT_FILTER	SecurityContextHolderAwareFilter
REMEMBER_ME_FILTER	RememberMeAuthenticationFilter
ANONYMOUS_FILTER	AnonymousAuthenticationFilter
SESSION_MANAGEMENT_FILTER	SessionManagementFilter
EXCEPTION_TRANSLATION_FILTER	ExceptionTranslationFilter
FILTER_SECURITY_INTERCEPTOR	FilterSecurityInterceptor
SWITCH_USER_FILTER	SwitchUserFilter

Retomando la estructura original del archivo propuesto, las secciones representativas de tal archivo de configuración son las siguientes:

- Sección http
- bean 'usuarioService'
- Sección authentication-manager
- bean 'accessDecisionManager'
- bean 'decisorDeRoles'
- bean 'decisorDeAutenticacion'

12) Sección http

En el archivo propuesto, el tag ‘http’ define tres propiedades: auto-config, access-decision-manager-ref y access-denied-page.

La propiedad ‘auto-config’ determina si el modelo de seguridad será interno y especifica que no estarán definidas implementaciones personalizadas de ningún filtro de la cadena de seguridad de Spring.

La propiedad ‘decision-manager-ref’ apunta a un bean definido dentro del propio archivo de configuración que determina cómo será la estrategia de decisión de acceso. En este caso, será definida por la clase de spring ‘AffirmativeBased’.

La propiedad ‘access-denied-page’ determina cuál será la página (Ejem. URL) a mostrar en caso de que una solicitud no esté autorizada a acceder a un cierto recurso.

Las partes constituyentes del tag ‘http’ son los que se mencionan en la Tabla VIII:

Tabla VIII Partes que constituyen el tag "http"

Sección	Propiedades	Descripción
intercept-url	pattern access	Ruta de directorios a interceptar. Roles que podrán acceder a esa ruta.
form-login	login-processing-url login-page default-target-url authentication-failure-url	URL que procesará la petición de log Página para firmarse al sistema. Si login ok, ir a este URL. Si login falla, ir a este URL
logout	logout-url logout-success-url	Página de logout Si logout OK, ir a este URL
session-management	invalid-session-url	Si sesión inválida, ir a este URL
remember-me	Key	Identificador genérico. No importa.

13) bean 'UsuarioService'

El bean 'usuarioService' es una referencia al POJO que contiene los atributos mapeados de la tabla de Usuarios en la base de datos del aplicativo. Cada uno de los campos de tal entidad estará definido como un atributo de este POJO.

14) Sección authentication-manager

El administrador de autenticación establece el mecanismo con el cual, Spring determinará si un intento de firma al sistema es válido o no. Esta sección puede referenciar a un archivo plano, una configuración XML, una base de datos o una clase de modelo. En este caso, se propone una clase de modelo que fue definida en la sección anterior: el bean 'usuarioService'.

Aquí también es posible definir el modo de encripción para la columna de 'password'. Actualmente está sugerida 'plainText', pero se recomienda usar MD5 o SHA.

15) bean 'accessDecisionManager'

Bean utilizado como una propiedad del tag 'http'. Como se mencionó anteriormente, este vean apunta a la clase: 'org.springframework.security.access.vote.AffirmativeBased'. Contiene una definición llamada 'decisionVoters' que es una lista de beans (decisorDeRoles y decisorDeAutenticacion) que serán discutidos a continuación:

16) bean 'decisorDeRoles'

En esta propuesta, el decisor de roles define el prefijo con el que serán nombrados los distintos roles que soportará el aplicativo. Para este caso en concreto es: 'PERFIL_', que puede ser cambiado en cualquier momento, siempre teniendo cuidado de que el prefijo coincida con los valores establecidos en la base de datos en la tabla Perfiles.

17) bean 'decisorDeAutenticacion'

Finalmente, el decisor de autenticación está determinado por una clase de spring denominada: org.springframework.security.access.vote.AuthenticatedVoter. Esta clase instrumenta una serie de mecanismos 'estándar' de autenticación, pero es posible usar una implementación propia, aunque no se recomienda, para esta arquitectura, dado que desde un inicio se propuso que la propiedad 'auto.config' del tag http estuviera apuntada a 'true'.

G. Pruebas unitarias y pruebas de integración

Las pruebas unitarias deberán ser confeccionadas bajo el directorio de "src/test/java" y podrán (o no) hacer uso de los archivos de configuración ubicados tanto en "src/test/resources" como en "src/main/resources".

La misma infraestructura servirá para confeccionar pruebas de integración.

Cada una de las clases de prueba deberá llevar el sufijo "Test". Por ejemplo, la clase que incluye pruebas de unidad para la clase "Calculadora.java" será llamada "CalculadoraTest.java".

De igual manera, los archivos de configuración 'de prueba' estarán ubicados en la trayectoria "src/test/resources" y llevarán el sufijo "-test.*". Por ejemplo: "applicationContext-mail-test.xml".

Todas las clases de prueba extenderán a la clase abstracta "AbstractTransactionalTestNGSpringContextTests" contenida en el paquete org.springframework.test.context.testng, para lo cual se requiere de la inclusión de la dependencia "spring-test" en nuestro pom.xml.

Conviene mencionar que la anotación "ContextConfiguration" es obligatoria, ya que determina el conjunto de archivos de configuración a utilizar durante la prueba. La anotación "ContextConfiguration" requiere de un argumento, que es realmente un arreglo de cadenas, por ejemplo:

@ContextConfiguration({"/bar/applicationContext.xml", "/foo/app.xml"})

Que también acepta el uso de comodines:

@ContextConfiguration({"/spring/applicationContext*.xml"})

Dado lo anterior, es factible (y recomendable) hacer uso de las anotaciones @Autowired para poder hacer uso de las distintas inyecciones de Spring.

TestNG es una extensión de JUnit, un sucesor que ofrece mayores capacidades y facilidades para la realización de pruebas unitarias. Integra, diversas anotaciones, entre las cuales, identificamos algunos muy importantes, como:

@Test
@AfterTest
@BeforeTest
@AfterSuite
@BeforeSuite
@AfterMethod
@BeforeMethod

A continuación, se dará una breve explicación acerca de cada uno de ellos:

1) @Test
   Es quizá la más importante, dentro de la selección hecha. Debe ser asociada al método que representa una prueba y puede estar presente en varios métodos de la misma clase. La anotación "@Test" no requiere de argumentos.

2) @AfterTest
   Indica que el método se ejecutara después de todas las clases con la etiqueta <test>

3) @BeforeTest
   Indica que el método se ejecutará antes de todas las clases con la etiqueta <test>

4) @AfterSuite
   Indica que el método se ejecutara después de todas las pruebas de la suit

5) @BeforeSuite
   Indica que el método se ejecutara antes de todas las pruebas de la suit

6) @AfterMethod
   Indica que el método se ejecutará después de todos los métodos de pruebas

7) @BeforeMethod
   Indica que el método se ejecutará antes de todos los métodos de pruebas

Atributos comunes para las anotaciones anteriores se muestran en la **Tabla *IX* Atributos comunes.**

Tabla IX Atributos comunes

Agregar nombre	Agregar nombre
alwaysRun	Si se establece como verdadero, el método siempre se ejecutará incluso si depende de métodos que no se hallan fallado en la prueba
dependsOnGroups	La lista de grupos de los que éste método depende
dependsOnMethods	La lista de métodos de los que éste método depende
Enabled	Indica sí los métodos de esta clase o método están habilitados
Groups	Lista grupos a los que esta clase o método pertenece
inheritGroups	S el valor del atributo es true, el método pertenece a los grupos declarados por la anotación Test a nivel clase

* El atributo inheritGroups, no está disponible para la anotación @Test.

Atributos exclusivos para la anotación @Test se enlistan en la Tabla X.

Tabla X Atributos exclusivos para anotación @Test

Atributo	Descripción
dataProvider	El nombre del data provider para esta prueba.
dataProviderClass	El nombre de la clase que va a proveer los datos. Sí no se especifica se buscará en la clase actual.
Description	Descripción para éste método.
expectedExceptions	La lista de excepciones que éste método podría lanzar; sí la excepción que el método lance no está en esta lista, la prueba se considera como un fracaso.
invocationCount	Cantidad de veces que este método debe ser invocado
invocationTimeOut	Tiempo máximo en milisegundos que ésta prueba puede durar, durante el acumulado de todos los invocationCount. Este se ignora si invocationCount no es especificado.
priority	Prioridad para éste método, la prioridades menores de ejecutan primero.
successPercentage	El porcentaje de éxito que se espera para éste método.
singleThreaded	Si el valor es true, todos los métodos de esta clase se ejecutarán en el mismo hilo.

timeOut	Tiempo máximo en milisegundos que ésta prueba puede durar.
threadPoolSize	El tamaño del pool de threads para este método, la idea es que se use este pool para ejecutar el método las veces que indica invocationCount. Si invocationCount no se especifica se ignora.

H. El directorio base para WEB

En la subrama 'webapp', de la estructura de proyecto, encontramos el directorio base para la organización de los elementos WEB.

```
/src/main/webapp
    ├──index.xhtml
    ├──content
    │     ├──common
    │     ├──restricted
    │     └──error
    ├──resources
    └──WEB-INF
          ├──faces-config.xml
          └──web.wml
```

a. Archivo index.xhtml

El archive index.xhtml representa la página de entrada al aplicativo.

El directorio 'content' sirve como un núcleo de agrupación para los directorios que realmente tienen funciones aplicativas. Este directorio no requiere de explicación, a diferencia de su contenido, que se expone a continuación.

b. Directorio 'common'

El directorio 'common' contiene todas las páginas que podrán ser accedidas sin requerir que el usuario esté firmado al sistema. Este directorio puede, opcionalmente, contener subdirectorios, que también serán accedidos sin necesidad de firma. La arquitectura está diseñada para tal efecto. Sin embargo, es posible redefinir, mover o eliminar cualquier directorio para que éste requiera, o no, autentificación al sistema. Este tipo de configuraciones puede ser llevado a cabo vía el archivo de configuración applicationContext-security.xml discutido en secciones previas de este documento.

El directorio 'common' incluye aquellos elementos que generalmente no requieren de autenticación al sistema.

c. Directorio 'restricted'

El directorio 'restricted' contiene todos aquellos recursos que sólo podrán ser accedidos si el usuario está firmado al sistema y su rol tiene los suficientes privilegios para acceder a ese recurso.

Al igual que 'common' este directorio puede contener subdirectorios, ser modificado, reubicado o eliminado configurando adecuadamente el archivo applicationContext-security.xml que fue discutido en secciones previas de este documento.

d. Directorio 'error'

El directorio 'error' contiene las páginas que serán mostradas en caso de que ocurra un error en el flujo del sistema. Tales páginas fueron especificadas en el archivo web.xml. Es posible (y recomendable) extender este conjunto de páginas, realizando las declaraciones adecuadas en el archivo web.xml. Por favor, refiérase a la sección de descripción de tal archivo en secciones posteriores de este documento.

e. Directorio 'templates'

El directorio 'templates' contiene la definición de secciones de páginas, (como header o footer) y la definición global de la estructura visual que rige a todas las páginas del aplicativo. Su objetivo principal es la reutilización de código JSF al definir una sola vez una sección y poder reutilizar tal sección en varias páginas.

El directorio 'templates' es, al igual que 'common' y 'restricted' altamente configurable, vía el archivo de configuración de SS3 (applicationContext-security.xml) sin embargo, se recomienda no modificarlo en etapas avanzadas del desarrollo del proyecto.

f. Directorio 'resources´

El directorio 'resources' contiene los elementos que jsf utilizará para dotar al aplicativo de un 'look & feel' atractivo y compatible con los estándares gráficos institucionales organizados en dos subdirectorios: 'css' y 'images'.

Adicionalmente, contiene un subdirectorio llamado 'js' que será el contenedor de archivos '*.js' (javaScript) para todo el aplicativo.

Cabe mencionar que los archivos '*.css' (o archivos de estilos) deben preservar un estándar de notación establecido por jQuery.

Por otro lado, una alternativa a la modificación manual de los archivos de estilo es el uso de la herramienta themeRoller de jQuery, que permite efectuar modificaciones 'al vuelo' y de forma visual, pudiendo en todo momento obtener o bajar el nuevo estilo completamente personalizado.

g. Directorio 'WEB-INF

El directorio 'WEB-INF' contiene una gran variedad de elementos en función de la plataforma de despliegue y del servidor de aplicaciones que se usará como base de desarrollo / pruebas / producción.

Para el ambiente de desarrollo en el que se implementó esta propuesta arquitectónica (Apache Tomcat 7) los archivos relevantes a discutir son dos:

- web.xml
- faces-config.xml

1) Archivo faces-config.xml

Gracias al uso de JSF 2.1 el archivo de configuración faces-config.xml expone una estructura simple y reducida. A continuación se muestra el archivo faces-config.xml propuesto para esta arquitectura:

```
01  <?xml version="1.0" encoding="UTF-8"?>
02
03  <faces-config
04      xmlns="http://java.sun.com/xml/ns/javaee"
05      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
06      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
07  http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
08      version="2.0">
09
10      <application>
11        <message-bundle>
12                  validationMessages
13          </message-bundle>
14          <resource-bundle>
15              <base-name>i18n.textMessages</base-name>
16              <var>translatedMessage</var>
17          </resource-bundle>
18          <el-resolver>
19                  org.springframework.web.jsf.el.SpringBeanFacesELResolver
20          </el-resolver>
21      </application>
22          <lifecycle>
23          <phase-listener>
24                  mx.com.rodascomputacion.cabinaCBR.support.LoginErrorPhaseListener
25          </phase-listener>
26      </lifecycle>
27
28  </faces-config>
```

Cinco son las líneas relevantes en este archivo de configuración: 12, 15, 16, 19 y 24 que han sido marcadas en negrita y rojo. Se procede a su explicación:

12: Nombre del archivo de mensajes (que se encuentra en src/main/resources y que fue explicado en secciones anteriores) conteniendo los textos que serán empleados por JSF para presentar mensajes generales de advertencia o errores de validación. Para internacionalización es factible utilizar la técnica descrita para los archivos ubicados en el directorio i18n expuesta con anterioridad.

15: Trayectoria (relativa a src/main/resources) y nombre del (o de los) archivo(s) de recursos que soportará(n) internacionalización.

16: Nombre de la variable que se referenciará dentro de un JSF para acceder a capacidades de internacionalización.

19: Soporte de para instrumentar la inyección de ManagedBeans desde el contexto de Spring

24: Interceptor de faces de JSF para el procesamiento de intentos de login fallidos y la presentación de mensajes de error en el frontend.

2) Archivo 'web.xml'

Para discutir a profundidad el archivo 'web.xml', se procederá a exponer la totalidad del mismo y comentar las secciones más relevantes.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">
    <display-name>PruebaPrototipo</display-name>

    <context-param>
        <param-name>primefaces.PUBLIC_CAPTCHA_KEY</param-name>
        <param-value>6LeYZcwSAAAAAPjeT6Ve9Nkbu2MV9Scumxl_S6w9</param-value>
    </context-param>
    <context-param>
        <param-name>primefaces.PRIVATE_CAPTCHA_KEY</param-name>
        <param-value>6LeYZcwSAAAAADCXWHYHuX_h6TgiIO6l-PtgHujr</param-value>
    </context-param>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/classes/spring/applicationContext*.xml</param-value>
    </context-param>

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>Captcha</servlet-name>
        <servlet-
class>mx.com.infotec.samples.pruebaprototipo.exposition.servlet.CaptchaServlet</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Captcha</servlet-name>
        <url-pattern>/Captcha.jpg</url-pattern>
    </servlet-mapping>



    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>


    <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
```

```
63          <filter-name>springSecurityFilterChain</filter-name>
64          <url-pattern>/*</url-pattern>
65          <dispatcher>FORWARD</dispatcher>
66          <dispatcher>REQUEST</dispatcher>
67      </filter-mapping>
68
69
70      <filter>
71          <filter-name>PrimeFaces FileUpload Filter</filter-name>
72          <filter-class>org.primefaces.webapp.filter.FileUploadFilter</filter-class>
73          <init-param>
74              <param-name>thresholdSize</param-name>
75              <param-value>51200</param-value>
76          </init-param>
77      </filter>
78      <filter-mapping>
79          <filter-name>PrimeFaces FileUpload Filter</filter-name>
80          <servlet-name>Faces Servlet</servlet-name>
81      </filter-mapping>
82
83      <listener>
84          <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
85  class>
86      </listener>
87
88      <welcome-file-list>
89          <welcome-file>index.jsf</welcome-file>
90      </welcome-file-list>
91
92      <error-page>
93          <exception-type>javax.faces.application.ViewExpiredException</exception-type>
94          <location>/content/support/err/ErrorViewExpired.jsf</location>
95      </error-page>
96      <error-page>
97          <error-code>404</error-code>
98          <location>/content/support/err/PageNotFound.jsf</location>
99      </error-page>
100     <error-page>
            <error-code>500</error-code>
101         <location>/content/support/err/ServerError.jsf</location>
102     </error-page>
103
104
105     <session-config>
106         <session-timeout>30</session-timeout>
107     </session-config>
108 </web-app>
109
```

A continuación se mencionan las partes más relevantes del archivo web.xml:

1) Declaración del “Faces Servlet” (líneas 21-25 y 51-55)
2) Declaracion del Filtro de “EncodingFilter” de Spring (58-73)
3) Declaración del Filtro de “Spring Security 3” (líneas 74-83)
4) Declaración del Listener del contexto general de “Spring” (líneas 99-101)
5) Declaración de páginas de error (líneas 107-118)
6) Declaración del tiempo máximo de inactividad en una sesión (claro líneas 121-123)

3) Declaración del “Faces Servlet”

Biblioteca que permite utilizar el framework de Java Server Faces. Para esta arquitectura se utilizará la versión 2.1. La implementación específica de JSF es PrimeFaces V 2.2, que actualmente es la única implementación AJAX que soporta la especificación 2.1 de JSF.

PrimeFaces no requiere de ninguna configuración o declaración especial dentro del archivo 'web.xml' (ni de ningún otro archivo de configuración) y es por ello que en él no aparece ninguna mención específica a esta implementación. Sin embargo, para el uso específico de alguna configuración especial, como 'fileUpload' o 'captcha' será requerida configuración extra, que no está cubierta por este documento.

El marco de trabajo JSF permite definir (con independencia de la extensión real de los archivos para frontend –generalmente .xhtml-) extensiones "virtuales". En este caso, se propone la extensión ".jsf" (lineas 23-26 en web.xml)

4) Declaración del Filtro de "EncondingFilter" de Spring

El filtro "EncodingFilter" es usado para indicarle a spring la codificación de caracteres que deberá usar en sus peticiones.

Este filtro debe ser el primer filtro en declárase, incluso antes que el filtro de seguridad de spring.

5) Declaración del Filtro de "Spring Security 3"

La declaración del filtro de seguridad de spring se realiza en las líneas 28-43 de 'web.xml'. Este filtro realmente representa una cadena de filtros que inicia con la petición de un recurso y concluye con una respuesta de JSF. Los recursos que pasarán por la cadena de filtros JSF pueden ser limitados o abiertos. En este caso, se filtrarán todas las peticiones de todo tipo de archivo. (Líneas 40, 41 y 42 de web.xml)

6) Declaración del Listener del contexto general de "Spring"

La operación de Spring está sujeta a las directivas que se establecen en uno o varios archivos XML de configuración (explicados con anterioridad) ubicados en lugares específicos y cargados por una clase maestra. Las líneas de la 28 a la 43 establecen la ubicación de tales archivos y la clase (o listener) **org.springframework.web.ContextLoaderListener** tiene la responsabilidad de efectuar la carga inicial de la información de tales archivos para inicializar Spring.

7) Declaración de páginas de error

La sección de "páginas de error" establece, por el momento, tres páginas que presentan mensajes específicos a los siguientes errores:

- Vista expirada
- Error en el server
- Página no encontrada

Cada una de ellas será invocada automáticamente en el momento en el que el error específico ocurra. Obviamente es posible extender el uso y cantidad de manejadores de errores acorde a las necesidades particulares del aplicativo.

8) Declaración del tiempo máximo de inactividad en una sesión

Determina el tiempo en el que se da término de una sesión por inactividad en ésta. El tiempo está determinado en minutos y comúnmente oscila entre 15 y 45 minutos.

VIII. VISTA LÓGICA

A continuación se muestran los diagramas de clases y de paquetes asociados a la arquitectura propuesta:

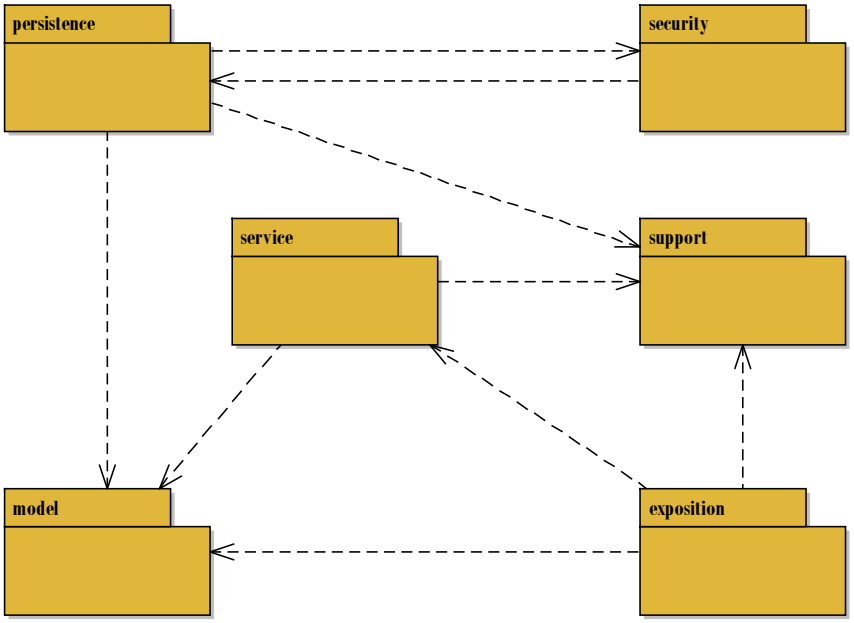


Fig. 6 Diagrama de paquetes

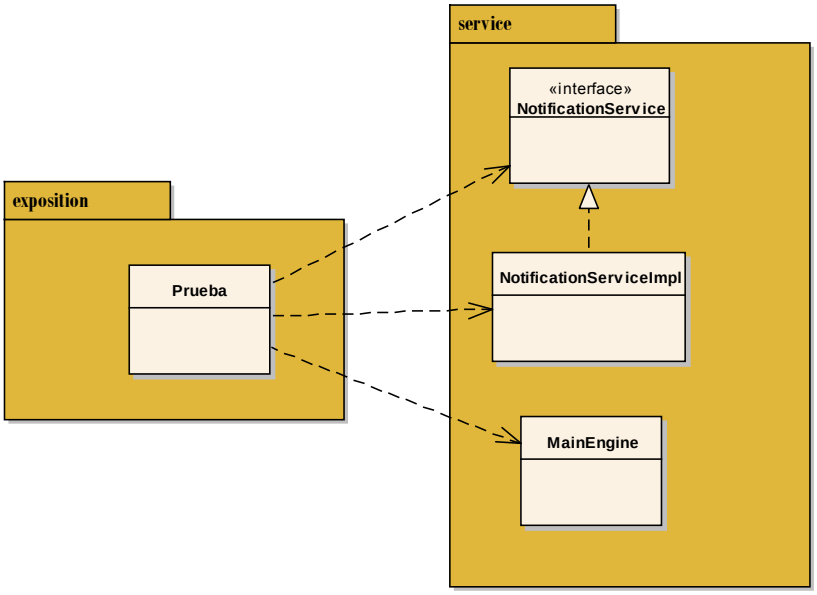


Fig. 7 Diagrama de clases 1 ª pag.

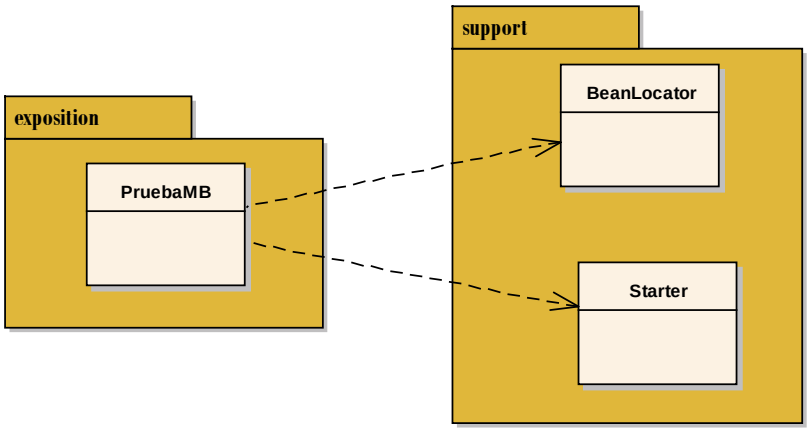


Fig. 8 Diagrama de clases 2 ª pag

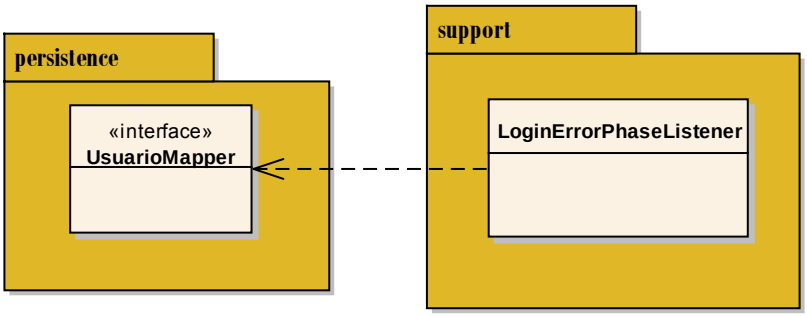
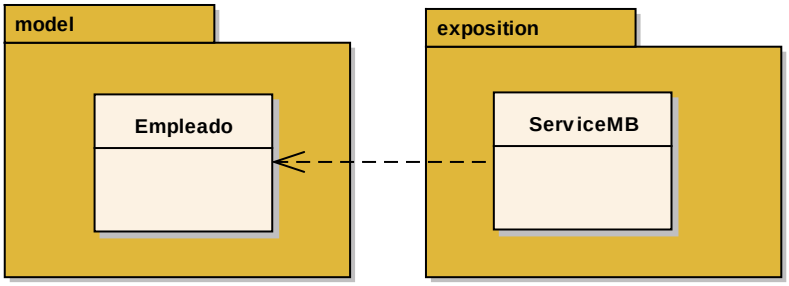


Fig. 9 Diagrama de clases 3ª pag



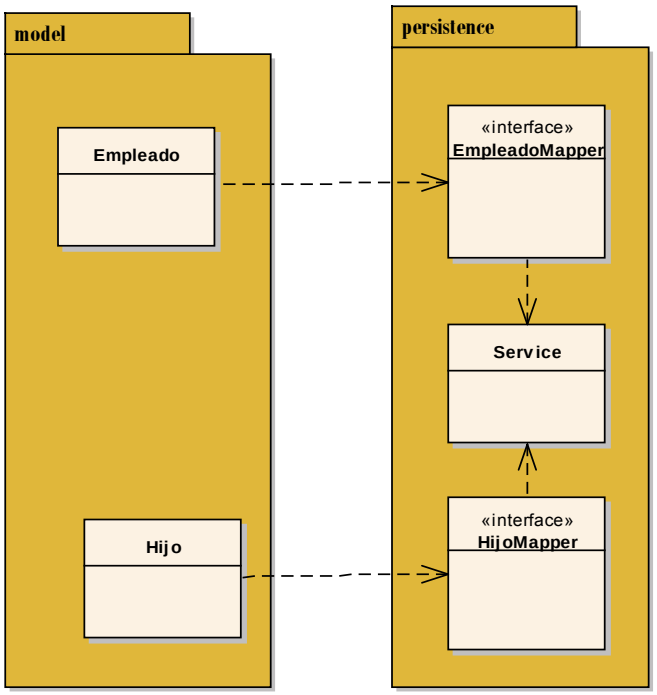


Fig. 10 Diagrama de clases 4ª pag

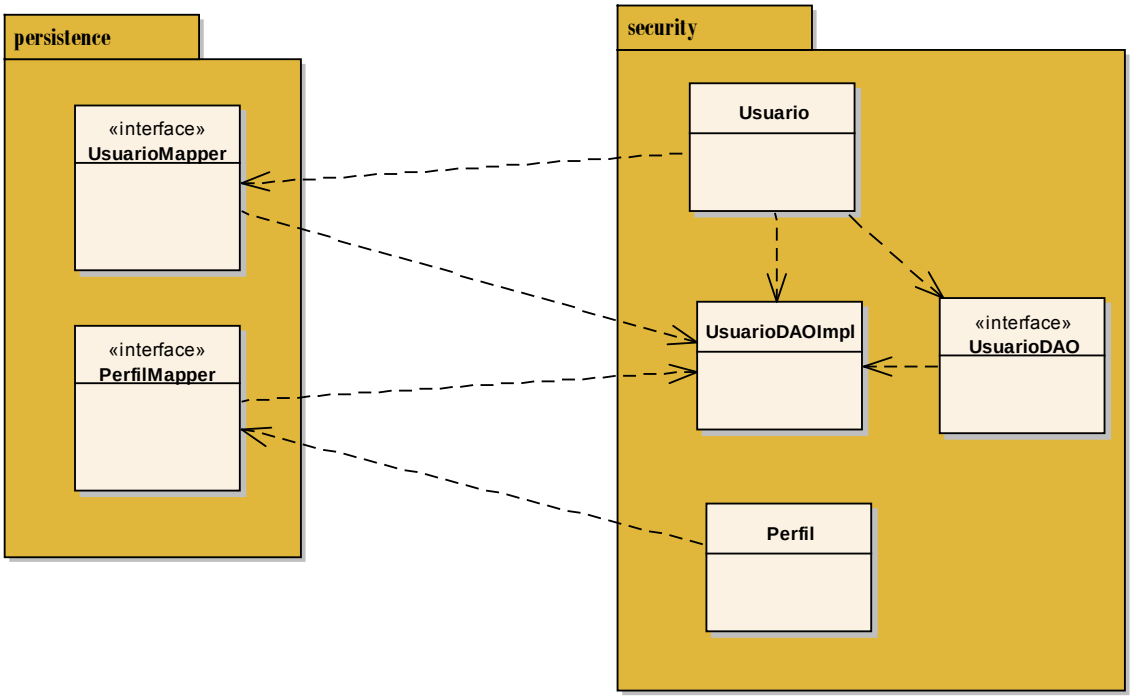


Fig. 11 Diagrama de clases 5ª pag.

IX. PLATAFORMAS Y LENGUAJES

Se requiere	Versión	Debido a:	Comentarios
Python	2.7 and 3	Soporte mínimo necesario para ejecutar el aplicativo y librerías correspondientes	
Apache Tomcat	7.0.X	Soporte mínimo necesario para desplegar el aplicativo Web	Puede utilizarse un servidor de aplicaciones como GlassFish, WebLogic o JBoss
MySQL	5.5	Soporte mínimo necesario para la base de datos	Es posible utilizar otros motores de bases de datos como MS SQL Server, PostgreSQL, entre otros; pero se necesitarían hacer ajustes al modelo de dominio de JPA

X. CONCLUSIONES

En la etapa de análisis y diseño del módulo de CBR del Entorno Modular de Simulación en su segunda etapa; se delimitaron los requerimientos y funcionalidad de dicho módulo para comprender cuál es su interacción con los servicios del Entorno Modular de Simulación y la plataforma de desarrollo .NET, siendo capaz de ser utilizado como un componente externo que es configurable para capitalizar de forma eficiente los casos en la base de conocimientos de CBR; a través de mecanismos transaccionales que garantizan la integridad de la información; y la integración de los algoritmos de búsqueda y cálculo de similitud del módulo CBR son implementados correctamente en base al estudio previo del proceso para el proceso de recuperación de los casos incluyendo aquellos casos que son recomendados en base a la reputación de las prácticas. Por lo que el módulo de CBR del Entorno Modular de Simulación cumple con las características necesarias para la implementación de un motor de CBR dinámico en base a los requerimientos del proyecto de FitoSmart.

La necesidad de la gran variedad teórica planteada en este documento es para dar a comprender al usuario final el modo de funcionamiento del proyecto con respecto a la sección del razonador basado en casos. Con lo cual se analizan los parámetros con respecto los modulo físicos como son el módulo de adquisición de variables e imágenes.