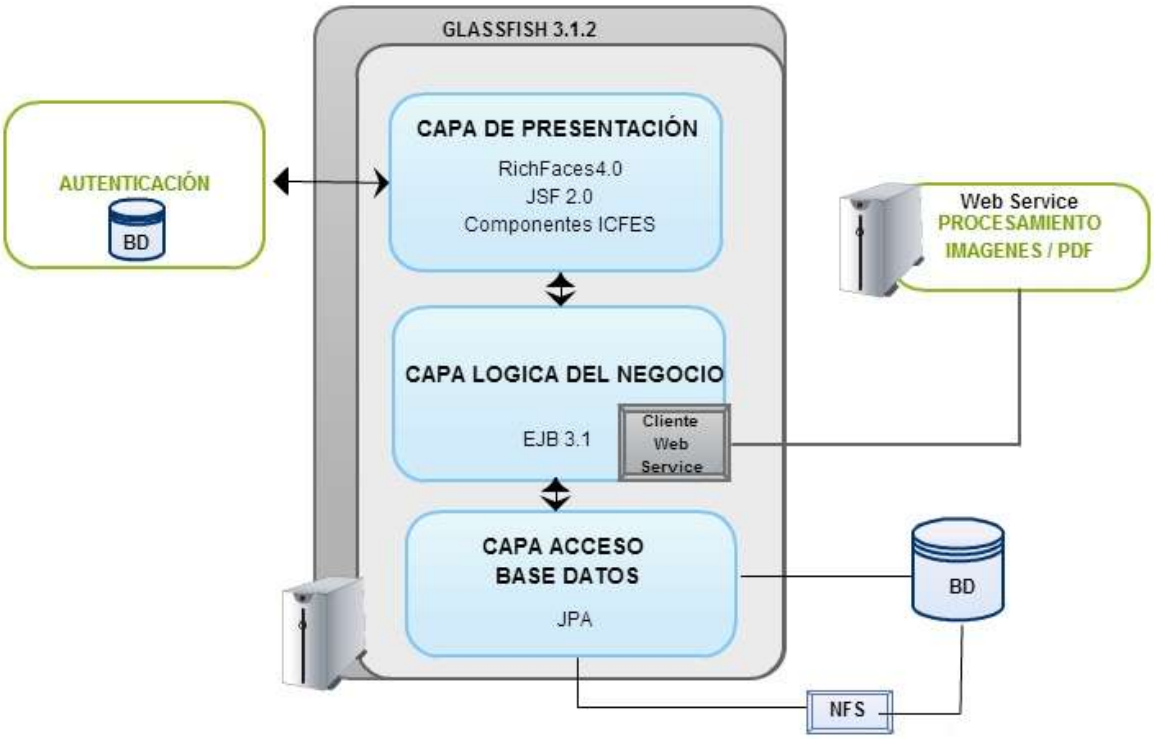
ICFES CALIFICACIÓN PREGUNTA ABIERTA WEB

Arquitectura

Versión <1.0 >

Contenido

1. ARQUITECTURA GENERAL



1.1 Autenticación de usuarios:

Se basa en el estándar de seguridad de java JAAS (Java Authentication and Authorization Service), utiliza un "securityrealms" tipo JDBC definido en el servidor de aplicaciones GlassFish 3.1.2, el cual se conecta a través de JDBC a una base de datos donde se encuentran los usuarios y grupos autorizados para ingresar a la aplicación, este compara los datos ingresados por el usuario con los existentes en la base de datos.

1.2 Capa de Presentación:

Contiene las interfaces de usuario. Las cuales muestran y capturan la información que el usuario ingresa en el sistema. El framework que se utiliza en esta capa es JSF 2.0 con la implementación de RichFaces 4.0 y componentes propios desarrollados por el ICFES. Esta capa se comunica con la capa de lógica del negocio.

1.3 Capa Lógica del negocio:

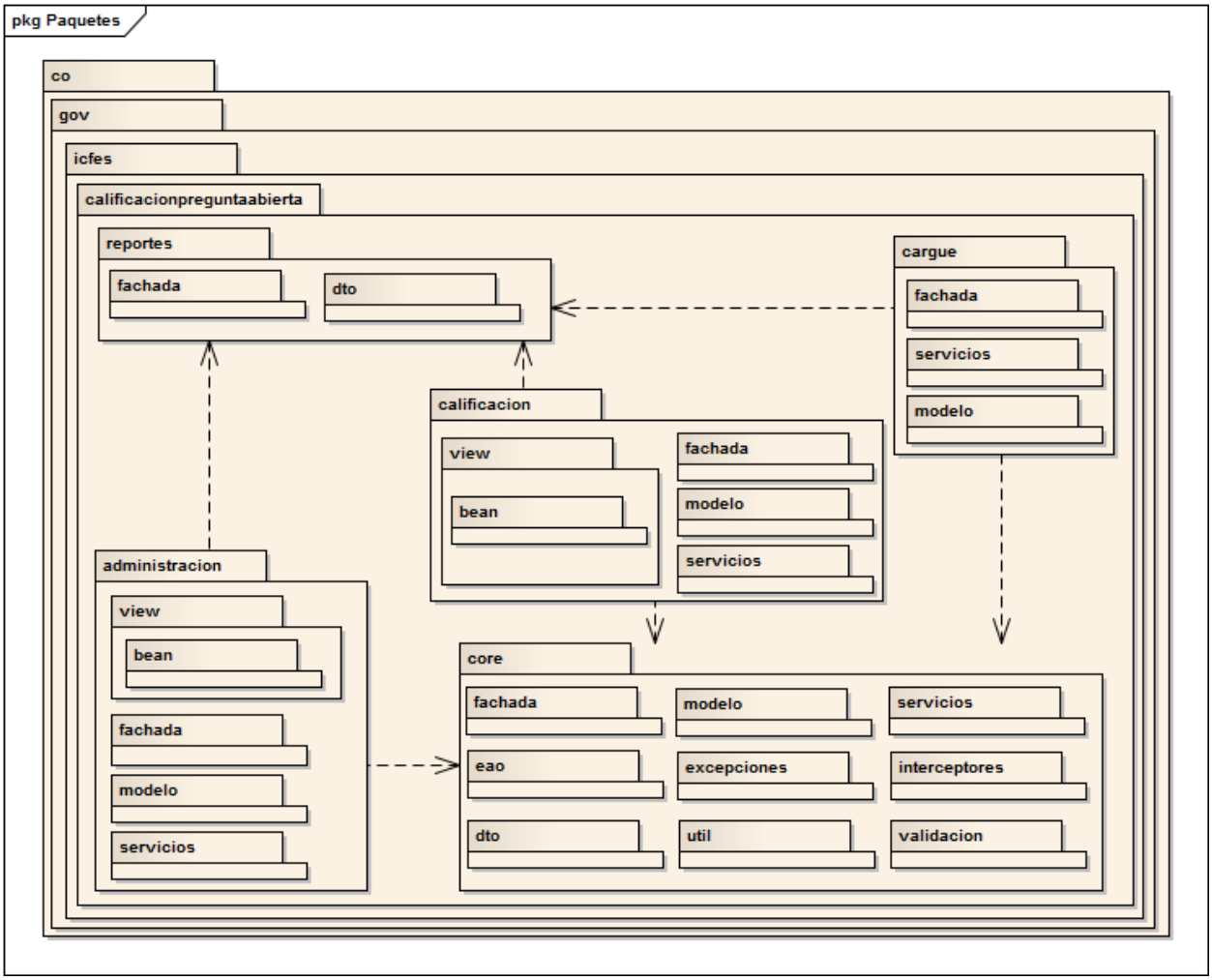
Esta capa se comunica con la capa de presentación para recibir las solicitudes, presentar los resultados y con la capa de acceso a datos para solicitar al gestor de la base de datos almacenar o recuperar información. En esta capa se utiliza el API EJB 3.0.

1.4 Capa de Persistencia:

Permite obtener información y realizar las operaciones deseadas en la base de datos solicitadas por la capa de lógica del negocio. Utiliza el API de persistencia JPA. Algunas operaciones se hacen directamente desde los entity y las demás se hace a través de procedimientos almacenado o por consultas named-query ó named-native-query.

2. VISTA LOGICA

La vista lógica está segmentada en las tres capas antes mencionadas (Modelo, Negocio, Presentación).



2.1 ESQUEMA GENERAL

2.1.1 Capa de datos (Modelo)

Basados en el uso de JPA2 se esquematizó el diseño de esta capa de la siguiente manera. Las entidades que hacen parte del modelo se dividieron de acuerdo al componente funcional al cual pertenece .Existen archivos ORM, para la creación de los JPQL

El uso de *@NamedQuery* se prohíbe en favor de usar archivos orm.xml.

Para el manejo de la auditoria se creó una superclase EntidadAuditable con la finalidad de facilitar y automatizar la asignación del usuario y fecha de creación y/o modificación.

El alias de los NamedQuery nunca se deben "quemar" en el código, se crea una constante en la entidad de la cual se genera la consulta y se define un alias para cada consulta NamedQuery realizada. La convención para la constante es:

```
public static final String QUERY_CONSULTAR_POR_<nombre_parámetros> =
        "Entidad.QUERY_CONSULTAR_POR_<nombre_parámetros>";
```

En el archivo ?-orm.xml correspondiente, se crean los <named-query />

En las entidades además de los atributos junto con sus métodos accesores, se van a implementar métodos de simples de negocio que realice lógica y tomas de decisión sobre sus atributos y atributos anidados.

2.1.2 Acceso a Datos

Se creó una serie de clases para el tema de las operaciones con la base de datos además de desacoplar la aplicación de JPA (dependencias con clases y librerias propias del API JPA o EclipseLink).

implementó una clase EAOGenerico la cual implementa los métodos necesarios para el tema de persistencia, esta clase se implementó como un EJB de sesión sin estado para poder hacer uso de todos los servicios provistos por el servidor de aplicaciones y ser inyectado en los servicios que requieran la funcionalidad de persistencia. ParametroQuery<T **extends** Serializable> permite usar una estructura flexible para enviar parámetros a los métodos de EAOGenerico (generalmente para el envío de parámetros a las consultas).

2.1.3 Servicios de negocio

Los servicios están divididos en paquetes de acuerdo al componente funcional al cual pertenece.

En los servicios estan implementados con las funcionalidades y reglas del negocio, se implementarán como EJB de sesión sin estado, se agregará a nivel de clase la siguiente anotación @TransactionAttribute mediante la cual se definirá el estado transaccional correcta para cada caso, las fachadas que invocan los servicios deben garantizar que se inicie la transacción donde se requiera.

Todo servicio debe incluir los siguientes atributos:

```
private CatalogoCache cache = CatalogoCache.getInstance();
@EJB
private EAOGenerico eaoGenerico;
```

Cuando no se realizan modificaciones sobre la base de datos, se debe agregar a nivel de método la siguiente anotación @TransactionAttribute(TransactionAttributeType.*SUPPORTS*) con el fin de no sobrecargar la aplicación iniciando una transacción para una consulta, ya que no es necesario.

2.1.4 Fachadas

Las fachadas están divididas en los siguientes paquetes de acuerdo al componente funcional al cual pertenece.

Las fachadas exponen servicios a alto nivel, generalmente son las funcionalidades descritas en un caso de uso, las fachadas actúan como wrapper y orquestador entre los servicios para implementar la funcionalidad. En las fachadas no debe implementarse directamente la lógica de negocio, para ese propósito están los servicios.

Las fachadas se implementarán como EJB de sesión sin estado, se agregará a nivel de clase la siguiente anotación @TransactionAttribute(TransactionAttributeType. *REQUIRES_NEW*) ,lo cual permite garantizar que se inicie una nueva transacción cada vez que se invoque un método de una fachada.

Cuando no se realizan modificaciones sobre la base de datos, se debe agregar a nivel de método la siguiente anotación @TransactionAttribute(TransactionAttributeType.*SUPPORTS*) con el fin de no sobrecargar la aplicación iniciando una transacción para una consulta, ya que no es necesario.

2.1.5 Presentación

Para la parte de presentación se divide la implementación en baking beans y plantillas xhtml.

A cada backing bean se inyecta su respectiva fachada, no se debe usar directamente los servicios.

```
@EJB
private Fachada nombreFachada;
```

Se deben anotar como @RequestScoped y por medio de ajax simular un scope de conversación.

A excepcion del bean CalificarTarea que pertenece al modulo de seleccion, su anotación debe ser @SessionScoped

Se creó una serie de componentes xhtml.

selectOneRadio.xhtml calendar.xhtml inputTextArea.xhtml dataTable.xhtml

panelForm.xhtml inputSecret.xhtml inputText.xhtml selectOneMenu.xhtml

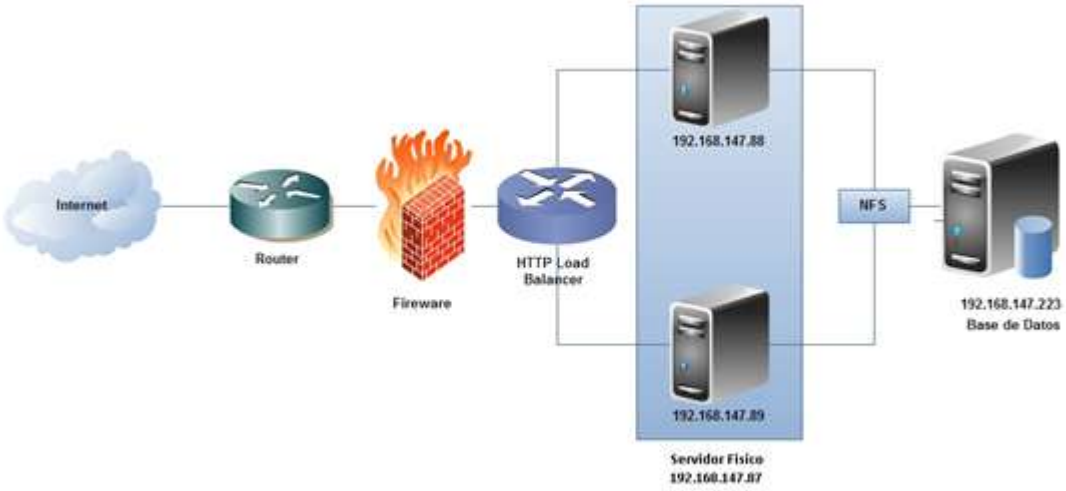
Para usar estos componentes se debe importar el xmlns de la siguiente manera, xmlns:icfes="http://java.sun.com/jsf/composite/icfes"

Y los componentes se invocan de la siguiente manera:

```
<icfes:inputText id="inputext" label="inputText" value="#{user.name}"required="true" validatorId="prueba" />
<icfes:selectOneMenu id="selectonemenu" label="selectOneMenu" required="true"
    selectItems="#{catalogoCache.itemsValidacion}" />
<icfes:panelForm for="commandbutton" label=" commandbutton">
    <h:commandButton id="commandbutton" value="Say Hello" />
</icfes:panelForm>
```

3. DIAGRAMA DE DESPLIEGUE

Las configuraciones físicas sobre las cuales el sistema interactúa, son las indicadas en la grafica.



4. VISTA DE IMPLEMENTACIÓN

La siguiente imagen es una muestra de la estructura general del modelo de implementación del software en niveles y subsistemas , incluyendo los componentes arquitectónicos significativos.