UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

SISTEMA DE MONITOREO PARA UN CENTRO DE CONTACTO

REPORTE DE TRABAJO PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

JOSÉ CARLOS LEÓN PÉREZ

TUTORA:
DRA. ELISA VISO GUROVICH
2016

1. Datos del alumno
León
Pérez
José Carlos
53 38 61 26
Universidad Nacional Autónoma de México
Facultad de Ciencias
Ciencias de la Computación
100001347

2. Datos del tutor
Dra
Elisa
Viso
Gurovich

3. Datos del sinodal 1
Dr
José de Jesús
Galaviz
Casas

4. Datos del sinodal 2
Dra
María de Luz
Gasca
Soto

5. Datos del sinodal 3
Dra
Verónica Esther
Arriola
Ríos

6. Datos del sinodal 4
Dr
Canek
Peláez
Valdés

7. Datos del trabajo escrito
Sistema de monitoreo para un centro de contacto
43 p
2016

ÍNDICE GENERAL

1

INTRODUCCIÓN

Premium Restaurant Brands, nombre comercial del administrador de las marcas *Pizza Hut y KFC (Kentucky Fried Chicken)* en México, inició el proyecto de tener su propio sistema centralizado de pedidos por teléfono o Centro de Contacto. La empresa nombró al proyecto como *Contact Center*, nombre que se usará a lo largo de este trabajo para referirse al mismo. El sistema e infraestructura que administra las llamadas fue comprado a una empresa particular especializada en *call centers*, pero resolver las necesidades particulares de la empresa generó dos proyectos importantes, un *toma órdenes*, con el cual los agentes telefónicos pudieran realizar la venta de los productos de las marcas, y un sistema de monitoreo llamado *videowall*, que pudiese visualizar estadísticas e información relevante en tiempo real y en pantallas de 55 pulgadas, repartidas en el *Contact Center*, para que los supervisores de los agentes de ventas pudieran tener un mejor control de la operación, visualizando lo que sucede en el *Contact Center* y tener una respuesta inmediata a cualquier eventualidad.

Este trabajo será sobre el segundo proyecto planteado: el sistema *videowall*.

En particular, el sistema *Videowall del Contact Center de Pizza Hut* debería mostrar la siguiente a información:

- Una gráfica de pastel con el porcentaje de agentes que se encontraran: en llamada, listo y en espera de llamada, o bien en transición (pausa para recibir la llamada siguiente).
- Una tabla con los nombres de los agentes conectados y su estado.

- Una gráfica de barras con el número de llamadas en cola y, de acuerdo al tiempo que las llamadas se encuentran en espera, iría cambiando el color de la gráfica, para que sean notados y atendidos con mayor rapidez.
- Si las llamadas en cola de espera tardaran más de un minuto, la pantalla del sistema comenzará a parpadear en rojo, por lo que todas las pantallas del *Contact Center* lo harán.
- Automatización del sistema de apagado y encendido. El sistema debería apagar de manera automática las pantallas al finalizar el horario laboral y enceder las pantallas al inicio del horario de atención.

El nuevo *Contact Center*, en una etapa inicial, sólo contemplaría a no más de noventa agentes de venta en un solo turno, distribuidos en seis islas. Cada isla tiene como responsable un supervisor y una pantalla debería estar en funcionamiento por cada dos supervisores; sólo se contempló a la marca *Pizza Hut*. Por lo tanto, en esta etapa deberían estar tres pantallas alrededor del *Contact Center* y una dentro de la oficina del director responsable. El sistema debería ser capaz de escalar a treinta pantallas, para cincuenta y cuatro supervisores y puestos gerenciales, que están a cargo de docientos agentes de venta para la marca *Pizza Hut*, y quinientos agentes para la marca *KFC*, por turno, para poder atender todos los pedidos del país de ambas marcas.

Por ello el sistema debería diseñarse con una arquitectura robusta y bien definida, altamente escalable, para el manejo de información y estadísticas en tiempo real, con una interfaz de usuario visualmente amigable.

La empresa Asterisk, especialista en *call centers*, proporcionó el sistema base e infraestructura del *Contact Center*. Su sistema se ejecuta en un servidor Linux CentOS RedHat. El software administrativo de *call centers* para enrutamiento de llamadas, grabación de llamadas para el control de calidad, registros y diversas herramientas específicas, es un sistema web usando tecnología Java JSP (*Java Server Pages*) y como manejador de base de datos, usa MySQL.

La empresa para la que laboré, *Premium Restaurant Brands*, me eligió como responsable del desarrollo y me asignó las siguientes actividades:

- Toma de decisión de la arquitectura más adecuada para el nuevo sistema *videowall*.
- Diseño e implementación del sistema.
- Integración con el sistema base de la empresa especializada en *call centers*.
- Realización de pruebas y correciones del sistema.
- Dejar listo un esquema de control de versiones del sistema para futuros cambios o extensiones al mismo.
- Entrega de documentación del sistema, código y configuraciones realizadas al hardware para su correcto funcionamiento.
- Realizar la solicitud de hardware y equipo eléctronico necesario para llevar a cabo el proyecto.

En lo que sigue de este trabajo describiré el diseño y proceso de construcción de este sistema, así como las herramientas utilizadas.

2

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

La solución, el sistema *videowall*, fue desarrollado como un sistema web para mantener un sistema centralizado, disponible para múltiples usuarios (pantallas alrededor del *Contact Center*), desde una sola fuente, así como para realizar mantenimientos y aplicar mejoras al sistema de forma más fácil.

Como patrón de arquitectura se utilizó MVC (Modelo-Vista-Controlador) para mantener bien definida y separada la lógica, y los datos de la interfaz de usuario. Una de las tecnólogías que trabaja bajo esta arquitectura es el *framework* JSF (*Java Server faces*) para aplicaciones Java EE. Estas tecnologías se eligieron para tener similitud con el sistema base de la empresa Asterisk. Se utilizó Apache-Tomcat como servidor web HTTP, sobre un sitema operativo Linux Ubuntu Server 12.10.

Como *framework* para el desarrollo de interfaces se eligió *Primefaces*, por ser robusto, con buen desempeño, liviano, con una amplia documentación, tener una biblioteca muy amplia de componentes y ser muy fácil de usar.

Para lograr mostrar el sistema *videowall* en cada pantalla es necesario hacerlo mediante un navegador de Internet, por lo que se decidió usar la misma forma que se usa para mostrar videos publicitarios en las pantallas de los restaurantes. Usar una mini-computadora por cada pantalla, que ejecutaría un navegador de Internet donde mostraría el sistema *videowall*.

2.1 AUTOMATIZACIÓN DEL SISTEMA

Cada pantalla del *Contact Center*, debe tener una terminal (mini-computadora) con una distribución GNU/Linux ligera como sistetema operativo. La automatización del encendido y

apagado del sistema sólo tiene como propósito prender y apagar las pantallas, antes de iniciar la jornada laboral y después de terminar la jornada, respectivamente. Esto debido a que las pantallas, al colocarse en una visión disponible para todos, queda en un lugar poco accesible para apagarlas manualmente y de una por una.

Este aspecto, aunque aparentemente trivial, no es directo, pues se tuvo que implementar a nivel del sistema operativo y con cableado específico, dada su poca accesibilidad física.

Para lograr esto se usó el administrador de procesos de segundo plano del sistema operativo *cron*. Se agregó al calendarizador las tareas de ejecutar un programa *Bash* con dos parámetros, uno que se ejecutara por la mañana, diez minutos antes de iniciar la jornada laboral para encender la pantalla, y el otro diez minutos después de terminar la jornada laboral, para realizar el apagado de la pantalla.

2.1.1 *Comunicación*

Para lograr la comunicación entre una terminal y su pantalla, se necesitó conectar un cable entre ambos dispositivos. Un cable fue para la transmisión de video VGA (*Video Graphics Array*), mismo que se utilizará para la transferencia de comandos de la terminal a la pantalla.

La Figura 1 muestra cómo se realizó la conexión entre una pantalla y la terminal.

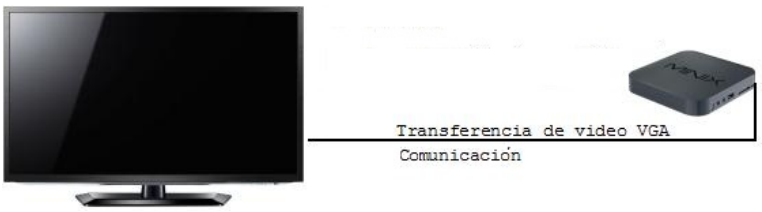


Figura 1: Conexión Pantalla con Mini-PC

El cable VGA se utilizó con la configuración universal.

2.1.2 *Script de apagado y encendido*

El script para enviar los comandos de apagado y encendido a la pantalla desde la terminal se hizo en lenguaje *Bash*; el código del script *turn.sh* se muestra a continuación.

```
#!/bin/bash
export DISPLAY=:0.0

if [ $1 = "off" ]; then
  echo -en "Turning monitor off..."
  xset dpms force off
elif [ $1 = "on" ]; then
  echo -en "Turning monitor on..."
  xset dpms force on
else
  echo usage: $(basename $0) "on|off"
```

Al ejecutarse el script *turn.sh* puede recibir dos valores para el parámetro; *off* para ejecutar el comando de apagado, *on* para ejecutar el comando de encendido. En una terminal puede ejecutarse el *script* de la siguiente manera:

Para encender la pantalla.

```
$ ./turn.sh on
```

Para apagar la pantalla.

```
$ ./turn.sh off
```

De esta manera el *script* se agregó al calendarizador para ser ejecutado en el horario correcto.

2.1.3 *Calendarizador*

Para realizar la ejecución automática del script *turn.sh* en el horario solicitado se agregó el script al administrador de tareas de segundo plano *cron*. Todas las tareas del calendarizador están en un archivo con un formato particular para que cada

tarea sea ejecutada en el tiempo especificado. Este archivo tiene como nombre *crontab*. Para agregar una tarea al *crontab* es suficiente con editar el archivo y actualizar los cambios. El formato para agregar una tarea es el siguiente:

```
mi ho dm me ds usuario tarea
```

Donde los valores `mi, ho, dm, me, ds`, definen de manera precisa el calendario de ejecución de la **tarea** por un **usuario** en particular. El valor `mi` se refiere al minuto con valores de 0 a 59; `ho` es la hora del día con posibles valores de 0 a 23; `dm` significa el día del mes; `ds` corresponde al día de la semana y debe ser un número de 0 a 7, donde 0 y 7 corresponden a domingo; o se pueden usar las primeras tres letras del día en inglés (*mon, tue, wed, thu, fri, sat, sun*). Si se desea tomar todo el rango de valores posibles de un campo se usa un asterisco (*). El valor `usuario` indica el usuario que ejecutará la tarea; y por último, `tarea` es la tarea a ejecutar.
En el caso particular de este proyecto se agrego el script *turn.sh* al archivo */etc/crontab* de la siguiente manera.

```
50 9 * * * root /home/videwall/scripts/turn.sh on
10 23 * * * root /home/videwall/scripts/turn.sh off
```

Lo que realiza esta configuración del *crontab* para la primera tarea es ejecutar *turn.sh* con parámetro *on* para encender la pantalla, por el usuario *root* en el minuto 50; a las 9 horas del día (9:50 am); todos los días del año. La segunda tarea ejecutará el script *turn.sh* con parámetro *off* para apagar la pantalla en el minuto 10, a las 23 horas del día (11:10 pm), todos los días del año.

2.2 ACCESO AUTOMÁTICO AL SISTEMA *videowall*

Cuando la terminal (mini-computadora) es reiniciada o encendida nuevamente de manera manual, por cualquier eventualidad, ya sea un apagón eléctrico, falla del dispositivo o simplemente se forzó el reinicio, la terminal, al encender, es capaz de abrir nuevamente la aplicación de forma automática. Es decir,

la terminal, al terminar de cargar todas las aplicaciones que necesita el sistema operativo para funcionar, el mismo ejecutará un comando adicional para abrir el navegador *Chrome* en pantalla completa, redireccionándose a la URL del sistema *videowall*, pasando la autenticación de acceso al sistema de manera directa y automática por reconocimiento de la dirección IP de acceso. Terminará esta inicialización mostrando la página principal del sistema *videowall*.

2.2.1 *Lanzamiento de la aplicación al iniciar el sistema operativo*

La configuración para lanzar el navegador *Chrome*, una vez iniciado el sistema operativo, se realizó modificando el archivo de configuración de carga de aplicaciones al iniciar una sesión local (/etc/rc.local), pues al ser una aplicación que necesita un ambiente gráfico debe ser ejecutada por el propio usuario con su sesión abierta. Adicionalmente se configuró que inicie la sesión del sistema operativo sin solicitar contraseña al encender o reiniciar la terminal. Se necesitó hacer un script que abra el navegador y que éste sea ejecutado por el cargador de aplicaciones. El archivo se modificó agregando la ejecución del *script openChrome.sh*, de la siguiente manera.

```
#!/bin/sh
# rc.local
#
sh /home/videowall/openChrome.sh
exit 0
```

El *script openChrome.sh* se ejecutará en cuanto la sesión inicie. Además de abrir el navegador de forma automática se necesita que abra en una `url` específica, la del sistema *videowall*, y que el navegador se ejecute en pantalla completa. Estas configuraciones están en *openChrome.sh*, como se puede ver a continuación:

```
#!/bin/sh
# openchrome.sh
chrome --kiosk
   http://10.203.12.28:8080/videowall/jsf/login.xhtml
```

El argumento `--kiosk` es para que el navegador inicie en pantalla completa.

2.2.2 *Acceso automático*

Para que un usuario pueda ingresar al sistema *videowall* necesita autenticarse por usuario y contraseña, pero eso debe ser para los usuarios que traten de ingresar desde una computadora normal; en cambio, al ingresar desde una terminal de las pantallas del *Contact Center*, esta autenticación no se deberá realizar. Estas terminales tienen una autenticación especial, que es con base en su dirección IP, pues cada una de las terminales cuentan con una dirección IP fija.

La autenticación por dirección IP inicia cuando el servidor del sistema recibe una petición de un cliente que solicita su acceso. El servidor verifica la ip de donde proviene la petición y valida si la ip es parte del catálogo de ip válidas en base de datos. El servidor regresará un *true* si la ip es valida o un *false* si no lo es, y la asignará a una variable, donde una función de *Javascript* (`autenticaIp(args)`) recuperará el resultado de la autenticación por dirección IP, que usará para el manejo de la interfaz de usuario. Para recuperar el resultado de manera automática utiliza un evento `html onload()`, como se muestra en el siguiente código *html*.

```
<!DOCTYPE html>
<html>
  <body onload="autenticaIp(args)">
    ...
    <!--Codigo-->
    ...
  <body>
</html>
```

La función `autenticaIp()` recupera el resultado de un *JSF managed bean*, que en el servidor realiza dos tareas importantes: la obtención de la dirección IP local de la terminal o computadora, y una consulta a la base de datos para comparar si la dirección IP local está en la lista de IP autorizadas para ingreso al siste-

ma, sin autorización por usuario y contraseña.
A continuación el código java que obtiene la IP local y la verifica.

```
import org.primefaces.context.RequestContext;
import java.net.InetAddress;
...
    try {
        String ipLocal =
            InetAddress.getLocalHost().getHostAddress();
        boolean ipAutorizada = isIPAutorizada(ipLocal);
        context.addCallbackParam("ipAutorizada",
            ipAutorizada);
    } catch (UnknownHostException e) {
            log.(e.getClass()+" "+e.getCause());
    }
...
```

Este código java o *JSF managed bean* mostrado anteriormente se ejecuta en el servidor, la cual usa la clase importada `java.net.InetAddress`. Después de obtener la dirección IP local, llama al método `isIPAutorizada()`, que realiza una consulta a la base de datos para validar que la variable `ipLocal` sea una IP autorizada para acceder directamente al sistema *videowall* y dar una sesión activa.
Finalmente, la función *JavaScript* `autenticaIp()`, que recibe el resultado de la validación del servidor, si `ipAutorizada` es verdadera redirecciona al sistema *videowall*, como se muestra a continuación:

```
<script type="text/javascript">
    function autenticaIp(args) {
        if(args.ipAutorizada) {
            location.href = videowall;
        }
    }
</script>
```

Este código javascript se encuentra en el mismo archivo xhtml `login`, pantalla principal de acceso al sistema.

2.3 DISEÑO DE LA SOLUCIÓN

El sistema *videowall* es sólo una página de HTML donde muestra toda la información relevante para el equipo responsable de la operación diaria del *Contact Center*, realizando consultas a dos bases de datos diferentes: la primera base de datos es para la autenticación de acceso al sistema, consulta que debe realizarse a la base de datos de recursos humanos del corporativo para conocer el estado del empleado y su perfil, así como su contraseña maestra de acceso a todos los sistemas del corporativo y, de esta manera, poder validar su acceso; la segunda base de datos es propia del *Contact Center* para extraer los datos necesarios para el despliegue de información del sistema *videowall*, los estados de los agentes, la cola de llamadas en espera y su tiempo de espera.

La Figura 2 muestra una vista de alto nivel de la arquitectura propuesta.

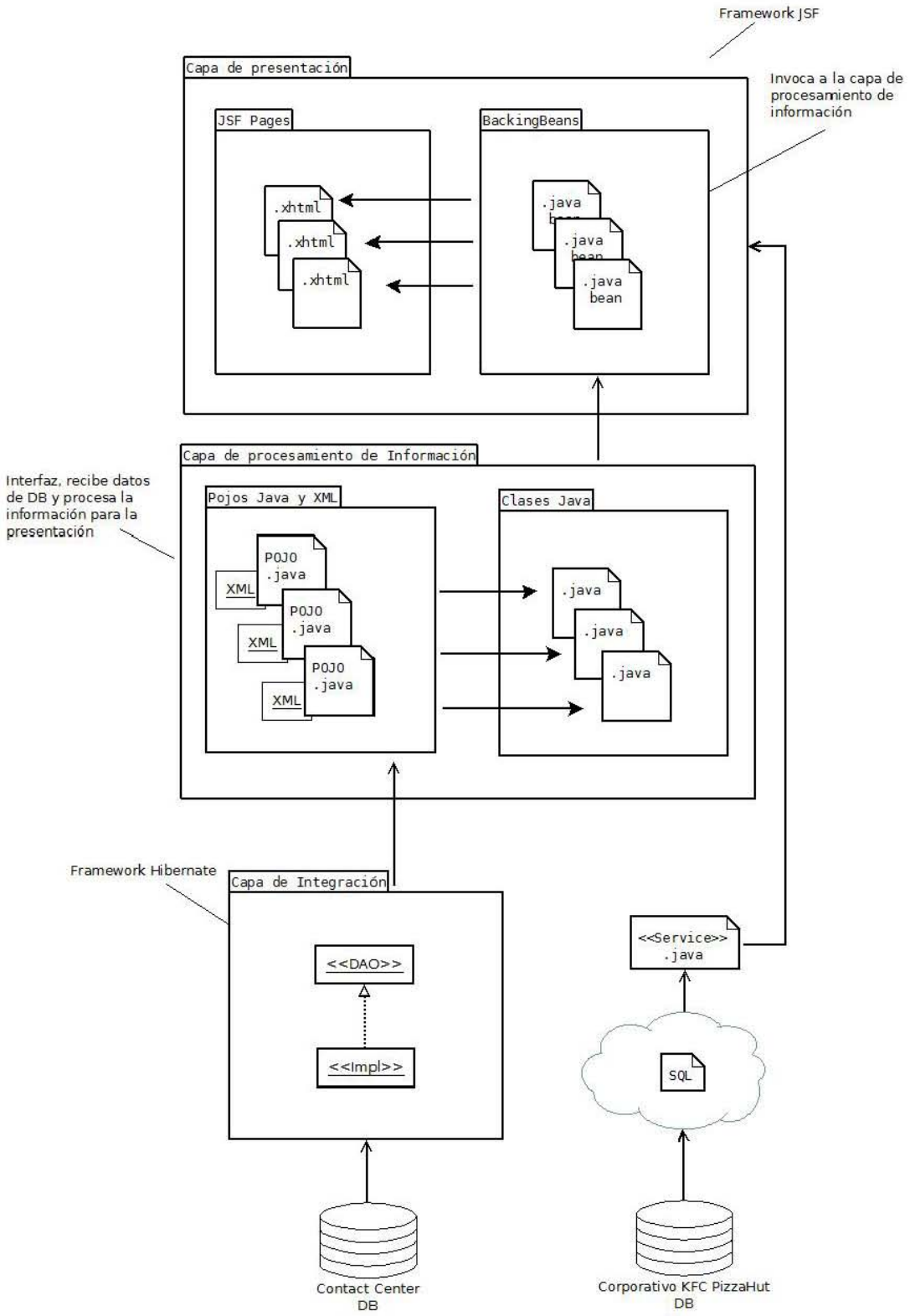


Figura 2: Arquitectura del sistema *videowall*

2.4 CAPA DE PERSISTENCIA DE DATOS

Una opción para manejar la persistencia de datos, si utilizamos JavaEE JSF, es utilizar una herramienta de mapeo objeto relacional (ORM por sus siglas en inglés). Esto ayuda también al *Framework Primefaces*, pues facilita el manejo de la información que reciben de las consultas realizadas a la base de datos, ya que los *frameworks Primefaces y JSF* consumen objetos *Plain Old Java Object* (POJO por sus siglas en ingles) que contienen la información que muestran en las pantallas XHTML; de ese modo manipularán con mayor naturalidad los datos. En particular para el desarrollo de está aplicación se eligió el framework ORM Hibernate.

Este *Framework* tiene como objetivo mapear cada tabla de una base de datos, preservando la relación que exista entre cada una de ellas. Así el programador podrá pensar sólo en manipulación de objetos, haciendo más fácil la manipulación de datos en un lenguaje orientado a objetos. Por ejemplo, uno de los componentes que tiene el sistema *videowall* es mostrar los datos de los agentes y su estado, datos que son consultados de la tabla *agent* (que se muestra a continuación) usando una consulta SQL. La tabla fue reducida para fines ilustrativos.

```
mysql> SELECT * FROM agent;
+----+--------------+----------+-------+
| ID | NAME         | LASTNAME | STATE |
+----+--------------+----------+-------+
|  1 | Juan Carlos  | Cano     | 2     |
|  2 | Ana Maria    | Lopez    | 3     |
|  3 | Antonio Mora | Montes   | 1     |
|  4 | Esmeralda    | Lozano   | 2     |
+----+--------------+----------+-------+
4 rows in set (0.00 sec)
```

Para poder realizar el mapeo de la tabla anterior a un objeto relacional, Hibernate genera dos archivos principales: El primero es un documento XML que guarda todas las características de la tabla, como el nombre de la tabla y de sus columnas, el

tipo de dato que almacenan, a cuál base de datos pertenece la tabla, y la relación con otras tablas o catálogos. El segundo es un POJO que es el objeto modelado con base en el mapeo del documento XML a una clase Java. A continuación se muestra el XML de la tabla *agent*:

```
<?xml version="1.0"?>
<hibernate-mapping>
 <class name="pojos.Agent" table="agent"
    catalog="ContactCenterDB">
  <id name="id" type="java.lang.Long">
    <column name="ID" />
    <generator class="identity" />
  </id>
  <property name="name" type="string">
    <column name="NAME" length="180" />
  </property>
  <property name="lastname" type="string">
    <column name="LASTNAME" length="180" />
  </property>
  <many-to-one name="state" class="pojos.State"
     fetch="select">
    <column name="STATE" />
  </many-to-one>
  <set name="ventas" table="ventas" inverse="true"
     lazy="true" fetch="select">
    <key>
     <column name="id" not-null="true" />
    </key>
    <one-to-many class="pojos.Ventas" />
  </set>
 </class>
</hibernate-mapping>
```

Este XML nos dice que el nombre de la tabla es *agent* pero será mapeada a Java por la clase `Agent` del paquete `pojos`; la tabla pertenece a la base de datos `ContactCenterDB`; tiene como identificador el campo `ID`; tiene un campo `NAME` que podrá ser mapeado como `name` y alamacena datos de tipo cadena; lo mismo con el mapeo `lastname`; el campo `STATE` será mapeado

como `state` y hace referencia a otra tabla de la base de datos (un catálogo) con una relación de muchos a uno. También esta tabla es referida por otra tabla por el mapeo de su llave primaria `id` con nombre `ventas` y con una relación de uno a muchos, por lo que esta relación puede traer consigo una colección de objetos. El POJO que corresponde a la tabla *agent* y al mapeo *xml* es el siguiente:

```
public class Agent implements java.io.Serializable {

     private Long id;
     private String name;
     private String lastname;
     private State state;
     private Set ventas = new HashSet(0);

    public Agent(){
    }
    public Agent(Long id, String name, String
    lastname,State state, Set ventas) {
       this.catEdociv = catEdociv;
       this.catNaci = catNaci;
       this.catSex = catSex;
       this.catTipcli = catTipcli;
       this.ven = ven;
       this.clinom = clinom;

    }
    /* Getters y setters */
}
```

En este POJO, se puede observar que preservó las características de la tabla, así como las relaciones que tiene con otras tablas.
Para poder realizar una consulta a la tabla *agent*, Hibernate tiene su propio lenguaje SQL, llamado HQL *(Hibernate Query Language)*. Para ejecutar una consulta es suficiente con escribir un método DAO (*Data Access Object*) que abra una conexión a la base de datos y ejecute la instrución, para de este modo regresar

un objeto o una lista de objetos, dependiendo el tipo de consulta que se realice. Una consulta HQL de nuestra tabla *agent* que regrese un registro con un mapeo de la clase *Agent* se muestra a continuación:

```
public Agent getAgent(int id){
    return (Agent)sesion.createQuery("from Agent where
        id="+id).uniqueResult();
}
```

2.5 DESARROLLO DE PANTALLAS

Como se mencionó anteriormente el sistema *videowall* tiene sólo una pantalla HTML principal, que contiene todos los componentes que muestran la información, y una pantalla adicional para el *login* de acceso al sistema. La pantalla principal (como también se ha mencionado) muestra tres componentes:

- Una gráfica de pastel para mostrar la porción de agentes que se encuentren en los estados **"En llamada", "En espera" o "En transición"** y sea más visible esta información para el supervisor.
- Una tabla con el nombre completo y su estado de los agentes que deben ser notados por algún ordenamiento discriminatorio.
- Una gráfica de barra que muestra el número de llamadas en cola organizadas por el tiempo que lleven en espera.

Cada uno de estos componentes será actualizado automáticamente cada seis segundos. Cada página XHTML, para poder usar los *frameworks JSF y Primefaces*, necesitan una configuración inicial en cada una de las páginas XHTML para que cada *framework* se identifique dentro del código HTML y tenga definido correctamente cada una de sus acciones, de tal manera que puedan ejecutarse correctamente sin conflicto entre sí, así como información de qué es lo que le corresponde resolver a

cada *framework*. A continuación la configuración inicial de todo XTHML que desee utilizar *JSF y Primefaces*:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui">
```

El primer atributo `xmls` es para definir que se utilizará el estandar XHTML; el segundo atributo `xmls:h` define que cualquier nombre de etiqueta que contenga a `:h` será manipulado por el *framework JSF*; el tercer atributo define que las etiquetas `:f` ejecutarán funciones especiales del *framework JSF*; por último, el atributo `xmls:p` define que todas las etiquetas que contengan `:p` serán componentes propios del *framework primefaces*.

2.5.1 *Actualización de los componentes del sistema videowall*

La pantalla HTML del sistema *videowall* tiene un componente `poll` que proporciona el framework *Primefaces* y realiza una actualización de todos los elementos HTML de la pantalla que necesita refrescar para mantener los datos actualizados. El componente se muestra a continuación:

```
<h:form>
    <p:poll interval="6" update="estadoAgentesPie" />
    <p:chart id="estadoAgentesPie" type="pie"
        model="#{chartView.livePieModel}"
        style="width:400px;height:300px"/>
</h:form>
```

Este código nos dice que *poll* realizará una actualización de una gráfica de pastel `estadoAgentesPie` cada seis segundos, pues `inverval` define el intervalo de tiempo de actualización. Le proporciona datos a la gráfica de pastel `estadoAgentesPie` para pintarlos en la pantalla, realizando una consulta a la base de datos para obtener el estado de los agentes actualmente conectados. Para realizar esta consulta usa un `ManagedBean` con la clase `CharView`.

La clase se muestra a continuación:

```
import javax.faces.bean.ManagedBean;
import org.primefaces.model.chart.PieChartModel;

@ManagedBean
public class ChartView implements Serializable {
    private PieChartModel livePieModel;

    public PieChartModel getLivePieModel() {
        HashMap<String,Integer> agentes =
            Dao.getEstadoAgentes();
        String[] keys = agentes.getKeys();
        for(String key: keys){
            livePieModel.getData().put(key,
                agentes.get(key));
         }
        return livePieModel;
    }
}
```

En general, un *JSF managed bean* de Java será quien maneje las peticiones del cliente al servidor, reciba sus respuestas y entregue los datos a los componentes *primefaces*, para que cada uno manipule los datos de manera particular y muestre la información en pantalla correctamente, actualizada y de una manera más amigable al usuario. Cada componente realiza una consulta a la base de datos de un tabla de transición particular, donde encontrará el estado de la actividad del *Contact Center* en ese momento. Estas consultas se realizan a la par con la actualización de los componentes, cada seis segundos. Lo que se pretende con este modo de implementación es mantener los datos en pantalla lo más actualizados posibles, sin saturar las conexiones y consultas a la base de datos, para evitar bloqueos.

2.5.2 *Estado de agentes en una gráfica de pastel*

Este componente de *primefaces* muestra de manera general el número de agentes conectados agrupados por su estado, de tal forma que de un vistazo rápido un supervisor del *Contact*

Center debe poder ver con certeza las porciones de agentes que se encuentran en una llamada realizando una venta; los agentes que se encuentran en espera, es decir disponibles para recibir una nueva llamada; y los agentes que se encuentrar en estado de transición, que se refiere al tiempo de recuperación del agente entre la finalización de una llamada y una nueva.

Para agregar este componente a una página XHTML se agrega la etiqueta `<p:chart />` con el atributo `type='pie'`, que define una gráfica de tipo pastel, y éste a su vez debe recibir un modelo con los datos que utilizará para poder pintar la gráfica. El siguiente código muestra cómo se agregó el componente a la página XHTML principal del sistema *videowall*.

```
<p:chart type="pie" model="#{chartPieView.pieModel}"
    id="piemodel" style="width:600px;height:400px" />
```

La etiqueta `model='chartPieView.pieModel'` recibe el modelo de datos de la clase `PieCharModel` que fue resultado de una consulta a la base de datos y estos son procesados para hacer una estructura de datos correcta para este modelo *Chart*. Los datos son alcanzados por el *ManageBean* `ChartPieView` que recibe la petición (*request*) del XHTML. A continuación el *ManageBean* `ChartPieView`.

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import org.primefaces.model.chart.PieChartModel;

@ManagedBean
@RequestScoped
public class ChartPieView implements Serializable{
    private PieChartModel pieModel;

    public PieChartModel getLivePieModel() {
        HashMap<String,Integer> agentes =
            Dao.getEstadoAgentes();

        String[] keys = agentes.getKeys();
```

(Continua en la página siguiente.)

```
        for(String key: keys){
            livePieModel.getData().put(key,agentes.get(key));
        }
        return pieModel;
    }
}
```

Este *ManageBean* realiza la consulta a la base de datos con el método `getEstadoAgentes()` de la clase `Dao`; también realiza un procesamiento para organizar la información de tal manera que sea la correcta para el modelo `PieCharModel`. Lo que regresa el método `getEstadoAgentes()` es un *HashMap* donde las llaves serán el tipo de estados encontrados en la base de datos `('En llamada','En Espera','En transición')` y el valor de cada llave será el conteo de agentes que se encuentren en cada uno de esos estado. Por ejemplo, si los siguientes datos fuesen los contenidos en el `HashMap agentes` del siguiente código:

```
livePieModel.getData().put("En llamada", 45);
livePieModel.getData().put("En espera", 15);
livePieModel.getData().put("En transicion", 5);
```

se daría como resultado una gráfica de pastel como la de la siguiente figura.

En este caso la gráfica nos dice que hay 45 agentes en una llamada realizando una venta, 15 agentes se encuentran dis-

ponibles para recibir una nueva llamada, mientras que cinco agentes acaban de finalizar una llamada y se están alistando para estar disponibles nuevamente.

2.5.3 *Tabla de agentes*

Este componente de *primefaces* muestra de manera específica la información de cada agente, su nombre completo y el estado en el que se encuentra en una tabla dinámica. Los datos mostrados se ordenan por el tipo de estado del agente, donde el que tiene más relevancia es el de 'En transición', ya que un agente no debe perder mucho tiempo en finalizar una llamada y estar disponible para una nueva. La siguiente de importancia es el estado 'En llamada' y al final los que se encuentran en 'En espera', pues al estar disponibles ya no hay nada que cuidar del agente.

Para agregar este componente a una página XHTML se agrega la etiqueta `<p:dataTable />` y para pintar los valores en una tabla de *primefaces* sólo debe recibir una lista de objetos en el atributo `value='agenteView.agentesList'`, que proviene de un *ManageBean*. El atributo `var='agente'` será la variable para recorrer la lista de objetos y pedir los atributos de cada objeto, como se haría en un `for each`, donde cada objeto que se recorre es un renglón a pintar de la tabla y cada uno de sus atributos será pintado en una columna de la tabla, como se muestra en el siguiente código.

```
<p:dataTable var="agente"
    value="#{agenteView.agentesList}" rows="15"
            paginator="true">
    <p:column headerText="Agente" width="250">
        <h:outputText value="#{agente.nombre}" />
    </p:column>
    <p:column headerText="Estado" width="120">
        <h:outputText value="#{agente.estado}" />
    </p:column>
    <p:column headerText="Agente" width="250">
```

(Continua en la página siguiente.)

```
        <h:outputText value="#{agente.nombre2}" />
    </p:column>
    <p:column headerText="Estado" width="120">
        <h:outputText value="#{agente.estado2}" />
    </p:column>
</p:dataTable>
```

El atributo `rows='15'` limita a la tabla a 15 renglones, mostrando a 30 agentes de un total de 70 agentes conectados simultáneamente. Para que no se pierdan los demás agentes conectados se activó el atributo `paginator='true'` que reparte a todos los agentes conectados en varias páginas de 15 renglones cada una, manteniendo el orden de prioridad a mostrar.

El *ManagedBean* que realiza la consulta a la base de datos y entrega la lista de objetos es el siguiente:

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ApplicationScoped;

@ManagedBean
@ApplicationScoped

public class AgenteView implements Serializable{

    private List<Agente> agentesList;

    public List<Agente> getAgentesList() {
        agentesList = Dao.getAgentesDoblePorRenglon();
        return agentesList;
    }
```

Este *ManagedBean* `AgenteView` obtiene una lista de objetos de una consulta a la base de datos y es procesada para agrupar dos agentes por objeto para poder pintar dos agentes por renglón. El resultado se muestra en la Figura 3.

Agentes conectados			
Agente	Estado	Agente	Estado
Cesar Rosas Lopez	Transicion	Marco Garcia Ramirez	En llamada
Maria Ramirez Rodriguez	Transicion	Roberto Gonzalez Martinez	En llamada
Carlos Gomez Shanchez	Transicion	Cesar Hernandez Castillo	En llamada
Antonio Shancez Meza	Transicion	Maria Hernandez Castillo	En espera
Roberto Gomez Shanchez	Transicion	Margarita Hernandez Castillo	En espera
Marta Gomez Shanchez	Transicion	Manuel Hernandez Castillo	En espera
Marco Hernandez Castillo	Transicion	Carlos Villa Flores	En espera
Carlos Gonzalez Martinez	Transicion	Margarita Shancez Meza	En espera
Margarita Gomez Shanchez	Transicion	Rodrigo Ramirez Rodriguez	En espera
Roberto Garcia Ramirez	Transicion	Rodrigo Rosas Lopez	En espera
Manuel Ramirez Rodriguez	En llamada	Antonio Shancez Meza	En espera
Carlos Rosas Lopez	En llamada	Maria Gonzalez Martinez	En espera
Rodrigo Garcia Ramirez	En llamada	Rodrigo Jimenez Mu	En espera
Margarita Shancez Meza	En llamada	Maria Leon Munive	En espera
Manuel Gomez Shanchez	En llamada	Marco Jimenez Mu	En espera
1			

Figura 3: Tabla de agentes y su estado

2.5.4 *Llamadas en espera*

Este componente de *Primefaces* es muy importante para el sistema *videowall*, pues nos muestra información crítica para toma de decisiones de un supervisor del *Contact Center*, ya que muestra el número de llamadas que se encuentran en cola de espera cuando todos los agentes se encuentren atendiendo una llamada. En situaciones como ésta ningún agente podrá tener tiempo de transición y debe atender una nueva llamada inmediatamente, pues una llamada perdida es una venta perdida. Además de conocer el número de llamadas, también se debe conocer el tiempo de espera; para ello las llamadas deben ser agrupadas por rangos de tiempos para conocer la gravedad de cada situación en cada tiempo de espera. Son cuatro rangos de tiempo de espera: el primero es de 0 a 15 segundos, el segundo es de 15 a 30 segundos, el tercero de 30 a 60 segundos y el cuarto cuando la llamada lleva más de un minuto en espera.

Para agregar este componente a un XHTML se usa la etiqueta `<p:chart />` con el atributo `type='bar'` para definir una gráfica de tipo barra; éste a su vez debe recibir un modelo con los datos que utilizará para poder pintar la gráfica. El siguien-

te código muestra cómo se agregó el componente a la página principal XHTML del sistema *videowall*.

```
<p:chart type="bar"
    model="#{chartViewBar.horizontalBarModel}"
    style="height:200px;width: 800px;font-size: 24px" />
```

La etiqueta `model='chartViewBar.horizontalBarModel'` recibe el modelo de datos de la clase `HorizontalBarChartModel`, que fue resultado de una consulta a la base de datos procesados para hacer una estructura de datos correcta para este modelo *Chart*. Los datos se alcanzan desde el *ManageBean* `ChartViewBar`, que recibe la petición (*request*) del *xhtml*. A continuación se muestra el *ManagedBean* `ChartViewBar`.

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import org.primefaces.model.chart.ChartSeries;
import org.primefaces.model.chart.HorizontalBarChartModel;

@ManagedBean
@RequestScoped

public class ChartViewBar implements Serializable{

    private HorizontalBarChartModel horizontalBarModel;

    private HorizontalBarChartModel
        getHorizontalBarModell() {

        horizontalBarModel = new HorizontalBarChartModel();
        int[] esperaRango = Dao.getLlamadasEsperaRangos();
        int llamadasEspera = Dao.getLlamadasEspera();
        ChartSeries espera0_15 = new ChartSeries();
        espera0_15.setLabel("0s-15s");
        espera0_15.set("En espera", esperaRango[0]);
        ChartSeries espera15_30 = new ChartSeries();
        espera15_30.setLabel("15s-30s");
```

(Continua en la página siguiente.)

```
        espera15_30.set("En espera", esperaRango[1]);
        ChartSeries espera30_60 = new ChartSeries();
        espera30_60.setLabel("30s-60s");
        espera30_60.set("En espera", esperaRango[2]);
        ChartSeries espera1min = new ChartSeries();
        espera1min.setLabel("+1min");
        espera1min.set("En espera", esperaRango[3]);
        horizontalBarModel.addSeries(espera0_15);
        horizontalBarModel.addSeries(espera15_30);
        horizontalBarModel.addSeries(espera30_60);
        horizontalBarModel.addSeries(espera1min);
        horizontalBarModel.setTitle("Llamadas en Cola:
            "+llamadasEspera);
        horizontalBarModel.setLegendPosition("e");
        horizontalBarModel.setStacked(true);

    }
}
```

Este *ManagedBean* realiza dos consultas a la base de datos con el método `getLlamadasEsperaRangos()` para saber la cantidad de llamadas por rangos de tiempo y el método `getLlamadasEspera()` que regresa el número total de llamadas en espera; por ejemplo, si el arreglo de datos de la variable `esperaRango` fuesen los que se declaran a continuación:

```
int[] esperaRango ={12,6,10,3};
```

El arreglo `esperaRango` daría como resultado la siguiente gráfica de barra como se observa en la Figura 4.

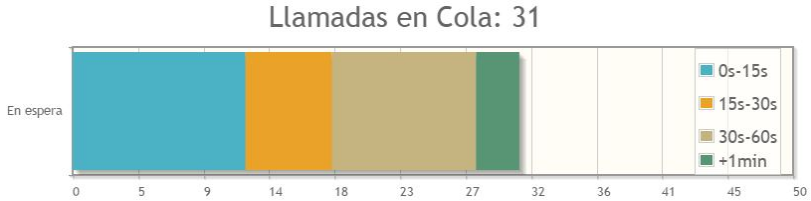


Figura 4: Llamadas en espera total y el número de llamadas por rangos de tiempo

En este caso la gráfica nos dice que que hay 31 llamadas en cola de espera, de las cuales 12 están en el rango de espera de 0 a 15 segundos, 6 están en el rango de 15 a 30 segundos, 10 están en el rango de 30 a 60 segundos y 3 llamadas llevan más de un minuto.

2.5.5 *Parpadeo de pantallas*

Como se mencionó al inicio de este trabajo, uno de los objetivos del sistema *videowall* es hacer notar los eventos críticos de la operación del *Contact Center*, y uno de estos eventos críticos es cuando una llamada o más han alcanzado un tiempo de espera mayor a un minuto, lo que nos dice que existe un alto riesgo de perder ventas, algo que no pueden permitir los supervisores y tienen que tomar acciones inmediatas.

Para implementar esta acción se escribió un *script* en *JavaScript* que verificará el número de llamadas que estuvieran con una espera mayor a un minuto. Se usará el mismo componente *poll* de *primefaces* que actualiza a todos los componentes del sistema. Como se muestra en el siguiente código XHTML.

```
...
<h:body id="body" styleClass="backgroundWhite">
       <h:form>
       ...
           <p:outputLabel id ="max_calls_queued"
              value="#{chartViewBar.maxCallsQueued}" />
           <p:poll interval="6"
              update="pie,bar,table,max_calls_queued"
              oncomplete="setBackgroundColor()"/>
       ...
       </h:form>
</h:body>
```

El componente `<p:outputLabel>` guardará el número de llamadas en cola con más de un minuto en espera y se mantendrá actualizado, como los demás componentes del sistema, por el componente `<p:poll>`. Su etiqueta de ejecución `oncomplete='setBackgroundColor()'` es la que ejecutará nuestro *script* des-

pués de haber realizado la actualización. Se puede observar que se ocupó el mismo *ManagedBean (chartViewBar)* que utiliza el componente `<p:chart type='bar'>` (llamadas en espera), sólo se agregó el siguiente código para obtener el número de llamadas en espera con más de un minuto (`maxCallsQueued`).

```
... private int maxCallsQueued;
    public int getMaxCallsQueued() {
      maxCallsQueued =
          Dao.getCuentaLlamadasEsperaCriticas();
      return maxCallsQueued;
    }...
```

Recibe el resultado de este valor `<p:outputLabel>` en su atributo `value='chartViewBar.maxCallsQueued'`, que utiliza el *script* `setBackgroundColor()` para hacer el cambio de fondo de color. El código del *script* se muestra a continuación:

```
<script type="text/javascript">
  function setBackgroundColor(){
    var max_calls_queued =
        document.getElementById("form:max_calls_queued")
            .innerHTML;
    if(max_calls_queued>0){
      document.getElementById("body").className=
        "backgroundRed";
    }else{
      document.getElementById("body").className=
        "backgroundWhite";
    }
  }
</script>
```

Este script se ejecutará siempre, después de que los componentes sean actualizados, y validará si el número de llamadas en espera con más de un minuto es mayor a cero; si es mayor cambiará el valor de la etiqueta `styleClass` del componente `<h:body>` por el valor `'backgroundRed'`, que es un estilo de *CSS*; en caso contrario cambiará el valor por `'backgroundWhite'` que también es un estilo. A continuación el código de estilos de *CSS* utilizados.

```
<style>
.backgroundRed {
    background-color: #ff0000;
}
.backgroundWhite {
    background-color: #ffffff;
}
</style>
```

En los lapsos de tiempo de mayor concurrencia de llamadas que pueden cambiar el fondo del sistema en rojo para advertir que hay llamadas en cola con más de un minuto, provoca que los supervisores hagan todo lo que esté a su alcance por atender esas llamadas críticas lo más pronto posible, ya sea con acciones de advertir a los agentes que no realicen ventas sugeridas o incluso que ellos mismos comiencen a contestar llamadas. Por esta razón en cuestion de segundos puede que ya no haya llamadas con más de un minuto de espera cambiando el fondo a blanco, pero la concurrencia de llamadas puede hacer que retorne de nuevo a color rojo, haciendo un efecto de parpadeo en las pantallas del *Contact Center*. Si las llamadas con más de un minuto de espera se mantienen en la cola, ó en la pantalla se mantendrá en rojo.

2.5.6 *Login*

El componente *login* es el que da acceso al sistema *videowall* por usuario local. Se había mencionado anteriormente que esta página XHTML tiene una acción *onload()*, que al cargar la página intentará realizar una autenticación por dirección IP; si ésta es satisfactoria ingresará al sistema, en caso contrario mostrará una ventana de diálogo (*login*) para iniciar sesión por usuario y contraseña, lo que nos dice que el ingreso al sistema será desde una computadora personal y no desde una de las pantallas del *Contact Center*.

Para agregar este componente al XHTML se agrega la etiqueta `<p:dialog/>` y el atributo `widgetVar='dlg'`, que sólo es

el nombre con el que va a ser identificado. El siguiente código muestra cómo se agregó el componente al XHTML:

```
<p:dialog header="Login" widgetVar="dlg"
    resizable="false">
   <h:panelGrid columns="2" cellpadding="5">
     <h:outputLabel for="username" value="Username:" />
     <p:inputText id="username"
         value="#{userLoginView.username}"
     required="true" label="username" />
     <h:outputLabel for="password" value="Password:" />
     <p:password id="password"
         value="#{userLoginView.password}"
         required="true" label="password" />
     <f:facet name="footer">
       <p:commandButton value="Login"
          actionListener="#{userLoginView.login(event)}"
                    oncomplete="handleLoginRequest(xhr,
                        status, args)" />
     </f:facet>
   </h:panelGrid>
</p:dialog>
```

El *ManageBean* que va a manipular las cadenas recibidas por el usuario es `userLoginView`. Este componente usa un actionListener que será manipulado por un método *login(event),* el cual realiza la validación del usuario y contraseña. A continuación el código del *ManagedBean* `userLoginView`.

```
import java.awt.event.ActionEvent;
import javax.faces.bean.ManagedBean;
import javax.faces.context.FacesContext;
import org.primefaces.context.RequestContext;

@ManagedBean
@RequestScoped
public class UserLoginView {

   private String username;
```

(Continua en la página siguiente.)

```
    private String password;

    /*Getters and setters*/

    public void login(ActionEvent event) {
        RequestContext context =
            RequestContext.getCurrentInstance();
        boolean loggedIn =
            Dao.isloggedIn(username,password);
        context.addCallbackParam("loggedIn", loggedIn);
    }
}
```

Al finalizar el método `login(event)`, dado el atributo `oncomplete='handleLoginRequest(xhr, status, args)'`, se ejecuta un *script JavaScript* que redirige al sistema *videowall* cuando la autorización es exitosa; de no ser así realiza un evento de sacudir el componente `dialog`. A continuación el código *JavaScript*.

```
function handleLoginRequest(xhr, status, args) {
       if(args.loggedIn) {
           PF('dlg').hide();

           $('/faces/videowall.xhtml').fadeOut();

       }
       else{
           PF('dlg').jq.effect("shake", {times:5}, 100);
       }
}
```

La variable `loggedIn` es la que se evalúa, pues es la que regresa en el *call back* del método `login(event)`. La Figura 5 muestra el componente ya en ejecución.

Figura 5: Diálogo de inicio de sesión del sistema *videowall*

Este componente se sitúa en el centro de la pantalla.

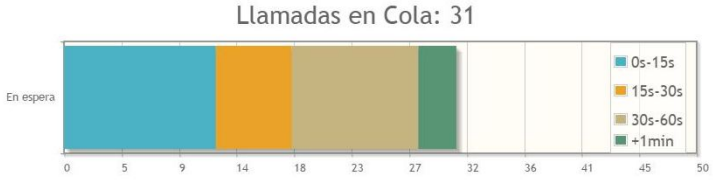
3

RESULTADOS

El sistema se entregó con todas las descripciones de este trabajo totalmente en funcionamiento. Las pruebas de funcionamiento se hicieron directamente en producción, sin pasar por un ambiente de pruebas, ya que la dirección de sistemas no cuenta con una área específica de control de calidad. Sólo antes de montar el sistema en ambiente productivo, se realizaron pruebas de funcionalidad en ambiente de desarrollo. Una vista del sistema *videowall* se muestra en la siguiente figura.

Agentes conectados

Agente	Estado	Agente	Estado
Cesar Rosas Lopez	Transicion	Marco Garcia Ramirez	En llamada
Maria Ramirez Rodriguez	Transicion	Roberto Gonzalez Martinez	En llamada
Carlos Gomez Shanchez	Transicion	Cesar Hernandez Castillo	En llamada
Antonio Shancez Meza	Transicion	Maria Hernandez Castillo	En espera
Roberto Gomez Shanchez	Transicion	Margarita Hernandez Castillo	En espera
Marta Gomez Shanchez	Transicion	Manuel Hernandez Castillo	En espera
Marco Hernandez Castillo	Transicion	Carlos Villa Flores	En espera
Carlos Gonzalez Martinez	Transicion	Margarita Shancez Meza	En espera
Margarita Gomez Shanchez	Transicion	Rodrigo Ramirez Rodriguez	En espera
Roberto Garcia Ramirez	Transicion	Rodrigo Rosas Lopez	En espera
Manuel Ramirez Rodriguez	En llamada	Antonio Shancez Meza	En espera
Carlos Rosas Lopez	En llamada	Maria Gonzalez Martinez	En espera
Rodrigo Garcia Ramirez	En llamada	Rodrigo Jimenez Mu	En espera
Margarita Shancez Meza	En llamada	Maria Leon Munive	En espera
Manuel Gomez Shanchez	En llamada	Marco Jimenez Mu	En espera



En las pantallas del *Contact Center* el sistema usa los logotipos de *Pizza Hut*, que no aparecen en estas figuras por no haber obtenido permiso para ello.

Uno de los beneficios muy claros desde la liberación del proyecto fue una rápida reacción en situaciones de saturación de llamadas en espera, para cambiar la estrategia de venta de los agentes y cuanto antes finalizaran su venta, por ejemplo sin ofrecer productos adicionales para que demoraran poco en cada llamada atendida.

4

CONCLUSIONES

Al inicio no se contempló la autenticación de usuario o por dirección IP, ya que sólo se pensó en el uso exclusivo del sistema en las pantallas del *Contact Center*.

Del componente que muestra las llamadas en cola, en un principio sólo se deseaba saber el número de llamadas en cola y que mostrara una gráfica que representara ese número. Después de mostrar por primera vez el sistema *videowall* y que ya estuviese en producción, se solicitó que se pudiera visualizar el número de llamadas en cola por rangos de tiempo, para saber con mayor certeza cuánto debían presionar a los agentes para que concluyeran con su venta y recibieran una nueva llamada; de ese mismo modo solicitaron que emitiera un sonido al momento que existieran llamadas en cola con más de un minuto en espera. El ruido que provocaban todas las pantallas en conjunto fue poco funcional, por lo que se cambió a un parpadeo en rojo de las pantallas como una alarma visual muy notoria para supervisores y agentes.

La automatización del apagado y encendido de las pantallas no se contempló tampoco en un principio para el *Contact Center*, pero al surgir el mismo problema de no poder apagar las pantallas fácilmente, lo que ocurre en los restaurantes, se implementó la misma solución, su automatización.

Este tipo de cambios, u otros posteriores, afectan directamente la arquitectura del software, por lo que no es conveniente hacerlos apresuradamente.Afortunadamente las complicaciones surgidas por los cambios, me dieron la oportunidad de resolver con tiempo suficiente.

Este proyecto ha sido una de mi mayores experiencias cumpliendo con todas las expectativas técnicas y profesionales que la empresa *Premium Restaurant Brands* corporativo de *KFC y Pizza Hut* me brindó para realizar este proyecto, otorgándome por primera vez en el campo laboral el rango de programador *senior*, que para la empresa significa estar a cargo de un nuevo desarrollo y liderear cada etapa del mismo.

Una posible extensión al sistema es que las pantallas muestren promociones del día (*banner*) o mensajes importantes de la operación, escritas y editadas por un administrador.

Este sistema sólo se realizó para la marca *Pizza Hut* pero se diseñó para poder agregar la marca *KFC* sin problema alguno. Ya quedará a elección del gerente de sistemas si sólo se genera una copia del sistema para *KFC* y tener completamente separados la visualización de ambas marcas o incluir a ambas en el mismo *videowall*. Si desean usar el mismo sistema para ambas marcas se tendrá que hacer cambios en la tabla de agentes. Por ejemplo el tiempo de llamada del agente y ordenarlos por esa prioridad, además de agregar el número de isla donde se encuentra físicamente.

BIBLIOGRAFÍA

[1] Gavin King Christian Bauer. *Java Persistence with Hibernate*. Manning Publications, 2006.

[2] Mert Cal Oleg Varaksin. *Primefaces Cookbook*. Packt Publishing, 2012.

[3] ORACLE. *Documentación lenguaje java 7.0*.

[4] Abraham Silberschatz. *Fundamentos de Base de Datos (5ª Ed.)*. S.A. Mcgraw-Hill / Interamericana de España, 2006.
[1, 4, 2, 3]