Capítulo IV. Implementación.

4.1 Instalación.

La implementación del sistema comienza con la instalación de las herramientas y tecnologías a utilizar. Empezamos instalando el contenedor web que nos ayudara a desplegar la aplicación en un navegador web, para nuestro caso utilizaremos Apache Tomcat versión 5.5 y elegimos el nombre de usuario y contraseña, y el puerto 8080.

A continuación instalamos el IDE de desarrollo Net Beans versión 6.0 y elegiremos como navegador predeterminado Mozilla Firefox versión 3.0.8 y como contenedor web el apache Tomcat versión 5.5. Utilizaremos Firefox para el desarrollo, sin embargo la aplicación será realizada para ser soportada por cualquier navegador como Safari e Internet Explorer.

Una vez instalado el IDE de desarrollo y el contenedor web necesitamos instalar nuestra base de datos, para nuestro caso utilizamos Oracle versión10g y el puerto 1521, elegimos un nombre de usuario y agregamos al paquete de librerías del Net Beans el archivo ojdbc14.jar para la conexión de la base de datos con las clases java.

Para el uso del framework Hibernate necesitamos agregar a la librería los siguientes archivos:

antrl.jar	cglib.jar	hibernate3.jar
asm.jar	commons-collections.jar	hsqldb.jar
asm-attre.jar	commons-logging.jar	jta.jar
c3p0.jar	dom4j.jar	

Para utilizar el Framework Scriptaculous para la utilización de Ajax se copiaron los archivos js en el siguiente directrio de la aplicación:

Javascript\lib\prototype.js

Javascript\scriptaculous\...

builder.js

controls.js

dragdrops.js

effects.js

scriptaculous.js

slider.js

sound.js

unitest.js

Una vea agregado a la librería, basta con agregarlos en las paginas donde serán utilizados mediante el siguiente código.

```
<script type="text/javascript" src="javascript/lib/prototype.js">
</script>
<script type="text/javascript"
src="javascript/scriptaculous/scriptaculous.js"> </script>
```

Para utilizar el Framework Dojo para la interaccion con Ajax de lado del cliente se agrega la carpeta descomprimida al directorio de la librería javascript y se agrega el siguiente código para utilizar cualquier método:

```
<script type="text/javascript" src="dojo/dojo.js"
      djConfig="parseOnLoad: true">
</script>
<script type="text/javascript">
  dojo.require("dijit.form.DateTextBox");
  dojo.require("dojo.parser");
</script>
```

4.2 Descriptor de la aplicación.

Toda aplicación web cuenta con un descriptor, para poder ser desplegado, en este documento XML se agregan los servlets y el mapeo hacia ellos, en este caso todas las peticiones son mapeadas a la clase FrontController que será descrita más adelante. A continuación se muestra parte de la descripción de la aplicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app       version="2.4"       xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
```

```
<servlet>
    <description>FrontController</description>
    <servlet-name>FrontController</servlet-name>
    <servlet-class>Controlador.FrontController</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>FrontController</servlet-name>
    <url-pattern>/Login</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>FrontController</servlet-name>
    <url-pattern>/Registro</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>FrontController</servlet-name>
    <url-pattern>/almacenarCita</url-pattern>
</servlet-mapping>
 …
 …
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

4.3 Manejo de sesión.

EL manejo de la sesión se crea desde el momento que el usuario ingresa al sistema en la clase Login.java, en la sesión agregamos el nombre de usuario, el nombre y correo, y se agrega un atributo del objeto usuario para el manejo de las opciones. En las clases restantes podemos obtener los datos para la consulta en la base de datos.

```
…
HttpSession session = request.getSession(true);
session.setAttribute("id", id);
session.setAttribute("nombre", nombre);
session.setAttribute("correo", correo);
session.setAttribute("usuario", usuario);
…
```

4.4 Controlador.

Utilizaremos el patrón FrontController para las múltiples peticiones, al igual que el patrón Model View Controller, cuenta con cada uno de estos elementos, en este caso la vista son los JSP, el controlador un servidor que extiende de la clase HttpServlet y el modelo, las clases para el manejo de los objetos de negocio, para almacenar citas, usuarios y páginas con lo que conlleva las mismas.

La interface Action denota un comportamiento para todas las acciones del modelo, todas las clases del controlador la implementan, consta de dos métodos, el principal es perform que recibe la petición y realiza la acción con el código dentro del método. El segundo método solo regresa el nombre de la clase.

```
public interface Action
{
```

```
      public void perform(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException;
      public String getName();
  }
```

Todas las peticiones hechas al servidor llegan al controlador (FrontController.java) este recibe los llamados y por cada llamado busca y obtiene el URL de la petición, ya sea almacenar citas o usuarios, redirige la petición según sea el caso al modelo correspondiente. El modelo genera una respuesta y muestra el resultado en la vista. La clase FrontController extiende de HttpServlet, el método más importante del servidor es procesRequest, recibe la petición, busca la acción mediante el método findAction, una vez encontrada la ruta ejecuta el método perform del Action.

```
public Action findAction(String servletPath){
      int index = servletPath.lastIndexOf('/');
      servletPath=servletPath.substring(index+1, servletPath.length());
      return (Action)actions.get(servletPath);
}
protected void processRequest(HttpServletRequest request,
     HttpServletResponse response)
            throws ServletException, IOException {

      response.setContentType("text/html");
      response.setHeader("Cache-Control", "no-cache");
      String path = request.getRequestURL().toString();
      Action action = findAction(path);
      if(action != null)
            action.perform(request, response);
      else{
```

```
            HttpServletResponse httpResponse =
            (HttpServletResponse)response;
            httpResponse.sendError(HttpServletResponse.SC_NOT_FOUND);
      }
         return;
   }
```

Al iniciar el servidor se ejecuta el método init que está sobrecargado para agregar a la hashtable en este caso llamado actions todas las clases del controlador para almacenar, obtener, modificar y borrar objetos de la aplicación, así como la petición para el acceso y registro, como las ligas para el repositorio y página.

```
public void init() throws ServletException
{
      actions = new HashMap();
      actions.put("Login", new Login());
      actions.put("Registro", new Registro());
      actions.put("almacenarCita", new AlmacenarCita());
      …
      …
      actions.put("EliminarArchivo", new BorrarArchivo());
}
```

El controlador a su vez también contiene las acciones Almacenar, Borrar, Modificar y Obtener cualquier objeto. El código para las acciones que se ejecutan de forma dinámica sigue un patrón para que la aplicación sea flexible. En el siguiente código se muestra la acción de agregar una nota dinámicamente.

```
  public class AlmacenarNota implements Action {
      private String name;
      public AlmacenarNotaAjax() {
```

```
            this.name = "AlmacenarNota";
        }
        public String getName() {
            return name;
        }
        public void perform(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            HttpSession session = request.getSession();
            ModeloPagina mPagina = new ModeloPagina();
            String nom_us = (String) session.getAttribute("id");
            String id = request.getParameter("titulo");
            String id_pagina = request.getParameter("id_pagina");
            String nota = request.getParameter("nota");
            Nota Nota = new Nota(id, id_pagina, nom_us, nota);
            try {
                mPagina.insertar(Nota);
            } catch (Exception e) {
            }
        }
    }
```

El controlador también es el encargado del manejo de las links, este recibe la petición y te manda la página seleccionada. El patrón FrontController nos ayuda a proteger los archivos jsp sin mostrar la extensión de las paginas. En la siguiente fracción de código se muestra como el controlador abre el repositorio.

```
    public void perform(HttpServletRequest request, HttpServletResponse
response)
                throws ServletException, IOException {
```

```
            HttpSession session = request.getSession();


            String nom_us = (String)session.getAttribute("id");


            try {
                if (nom_us != null) {
                    ModeloPagina mPagina = new ModeloPagina();
                    ArrayList paginas = mPagina.obtenerPaginas(nom_us);
                    request.setAttribute("paginas", paginas);
                    RequestDispatcher rd =
request.getRequestDispatcher("Repositorio/index.jsp");
                    rd.forward(request, response);
                    return;
                }
                else{
                    RequestDispatcher rd =
request.getRequestDispatcher("error/start.jsp");
                    request.setAttribute("status", "La pagina no existe");
                    rd.forward(request, response);
                    return;
                }
            }
            catch(SQLException e){
                e.printStackTrace();
            }
        }
```

4.5 Modelo.

Existen tres modelos, para cita, página y usuario, cada uno de ellos contiene los métodos para insertar, borrar, modificar u obtener, dependiendo sea el caso, cualquiera de los objetos de negocio manejados. A diferencia del modelo cita y modelo usuario que manejas los objetos cita y usuario respectivamente, el modelo pagina maneja a su vez archivos, notas, listas, objetos listas y fotos.

```
  public class ModeloCita {
      Conexion conn = null;
      Statement stmt = null;
      ResultSet res;
      public ModeloCita(){
          conn = new Conexion();
      }
      public int insertar(Cita cita)  throws SQLException {
            …
            …
      }
      public Cita obtener(String nom_us, String fecha_inicio, String
hora_inicio)throws SQLException {
            …
            …
      }
      public boolean borrar(String nom_us, String fecha_inicio, String
hora_inicio)throws SQLException {
            …
            …
      }
  }
```

Parte del modelo se basa en los objetos de negocio (Cita, Usuario, etc.). El siguiente código muestra parte del bean Usuario.

```
public class Usuario {
    private Long id;
    private String nom_us;
    private String nombre;
    private String apellido_pa;
    private String apellido_ma;
    private String correo;
    private String contrasena;
    public Usuario(){
    }
    public Usuario(String nom_us, String nombre, String apellido_pa,
String apellido_ma, String correo, String contrasena){
        this.nom_us = nom_us;
        this.nombre = nombre;
        this.apellido_pa = apellido_pa;
        this.apellido_ma = apellido_ma;
        this.correo = correo;
        this.contrasena = contrasena;
    }
    public String getNom_us() {
        return nom_us;
    }
    public void setNom_us(String nom_us) {
        this.nom_us = nom_us;
    }
     …
```

```
    …
    public Long getId() {
        return id;
    }
     public void setId(Long id) {
        this.id = id;
    }
}
```

La otra parte del modelo es la conexión a la base de datos es manejada por la clase Conexión.java, la clase para almacenar las citas, usuarios y páginas con lo que conlleva, o el acceso al administrador. La conexión cuenta con dos métodos, el primero para abrir una sesión con la base de datos y realizar las operaciones necesarias y el segundo para cerrarla y guardar los cambios realizados actualizaciones, inserciones y borrado.

```
  public class Conexion {

      public Conexion(){}

      Connection conn = null;
      Statement stmt = null;
      static final String CONN_URL =
                      "jdbc:oracle:thin:@localhost:1521:XE";
      static final String USER = "system";
      static final String PASSWD = "system";

      public Statement abrir(){
            …
            …
      }
```

```
    public void cerrar() {
          …
          …
    }
}
```

Los objetos de negocio son manejados por el Framework hibérnate, tanto para almacenarlos y recuperarlos, con la clase HibernateUtil.java del paquete persistence se obtiene una sesión y se cierra.

```
public class HibernateUtil {
    private static SessionFactory sessionFactory;
    static{
        try{
            sessionFactory = new
         Configuration().configure().buildSessionFactory();
        }
        catch(Throwable ex){
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory(){
        return sessionFactory;
    }

    public static void shutdown(){
        getSessionFactory().close();
    }
}
```

4.6 Motor de inteligencia.

El motor es activado desde el momento del registro, cuando se accede a la aplicación se obtienen todas las características del usuario de esta manera el sistema verifica en qué estado se encuentra el motor de inteligencia. Las características son obtenidas desde el modelo de usuario como se muestra en la fracción de código siguiente.

```
  public Usuario obtener(String nom_us, String contrasena)throws
SQLException {
        stmt = conn.abrir();
        String query = "SELECT nombre, apellido_pa, apellido_ma, correo,
motor " + "FROM USUARIO " + "WHERE nom_us='" + nom_us + "' AND
contrasena='" + contrasena + "'";
        res = stmt.executeQuery(query);
        if(res.next() && res.getString(1) != null ){
            String nombre = res.getString(1);
            String apellido_pa = res.getString(2);
            String apellido_ma = res.getString(3);
            String correo = res.getString(4);
            String motor = res.getString(5);
            conn.cerrar();
            return new Usuario(nom_us,nombre,apellido_pa, apellido_ma,
correo, contrasena, motor);
        }
        conn.cerrar();
        return null;
    }
```

Al de acceder a la aplicación el motor verifica el estado actual, de esta manera si esta activado al iniciar de direcciona al recordatorio de las citas más importantes, si no te manda a la vista de las citas de día.

```
if (us.getMotor().equals("false")) {
                    RequestDispatcher rd =
request.getRequestDispatcher("diaPim.jsp");
                    rd.forward(request, response);
                } else {
                    obtenerCitas(request, us);
                    RequestDispatcher rd =
request.getRequestDispatcher("importancia.jsp");
                    rd.forward(request, response);
                }
```

Y se obtienen las citas del día actual y los siguientes cinco días ordenados del más importante al menos importante.

```
  public void obtenerCitas(HttpServletRequest request, Usuario us) {
          ModeloCita mCita = new ModeloCita();
          JspCalendar JspCal = new JspCalendar();

          try {
              String fecha1, fecha2, fecha3, fecha4, fecha5, fecha6;
              fecha1 = JspCal.getCurrentDate();
              fecha2 = JspCal.getNextDate();
              fecha3 = JspCal.getNextDate();
              fecha4 = JspCal.getNextDate();
              fecha5 = JspCal.getNextDate();
              fecha6 = JspCal.getNextDate();
```

```
                ArrayList alta = mCita.obtenerAlta(us.getNom_us(), fecha1,
fecha2, fecha3, fecha4, fecha5, fecha6);
                ArrayList media = mCita.obtenerMedia(us.getNom_us(),
fecha1, fecha2, fecha3, fecha4, fecha5, fecha6);
                ArrayList baja = mCita.obtenerBaja(us.getNom_us(), fecha1,
fecha2, fecha3, fecha4, fecha5, fecha6);
                request.setAttribute("alta", alta);
                request.setAttribute("media", media);
                request.setAttribute("baja", baja);
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
```

Al almacenar una cita, de igual forma el sistema verifica el estado actual del motor, de esta manera si el motor esta desactivado la cita se almacena con la fecha y hora originales, o bien si el motor se encuentra activado la aplicación direcciona al usuario a una nueva ventana con sugerencia de horas disponibles en fechas siguientes a la original.

```
<form method="post" action="Sugerencia">
  …
  …
<%
      if (us.getMotor().equals("true")) {
%>
  <input type="submit" value="Guardar" name="guardar" size="15"
class="boton_fb"/>
<%
} else {
%>
```

```
   <input type="button" value="Guardar" name="guardar" size="15"
class="boton_fb" onclick="almacenarCita()"/>
<%
   }
%>
```

4.7 Vista.

4.7.1 Acceso y registro.

El proyecto inicia con un una página HTML (index.html) para que los usuarios pueden ingresar al administrador. La misma página tiene una liga a registro.jsp para almacenar nuevos usuarios. Las imágenes siguientes muestran la página inicial del sistema y de registro.

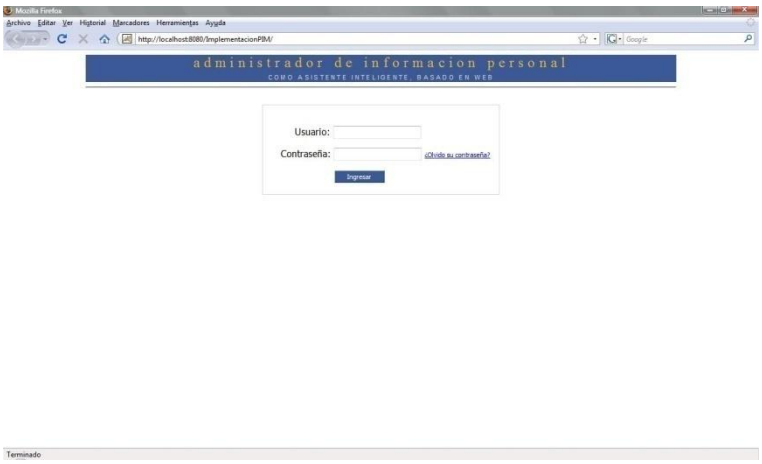

Figura 15. Interfaz de acceso al sistema.

Figura 16. Interfaz de registro.

4.7.2 Administrador de información personal.

Al iniciar una sesión la primera interfaz que será vista por los usuarios será el administrador de información, con el día actual al inicio de sesión con los horarios de las citas que tiene en el día. El usuario puede navegar a través de esta interfaz teniendo tres vistas disponibles; por día, semana y por mes. En las vistas por día y semana se pueden almacenar las citas correspondientes.

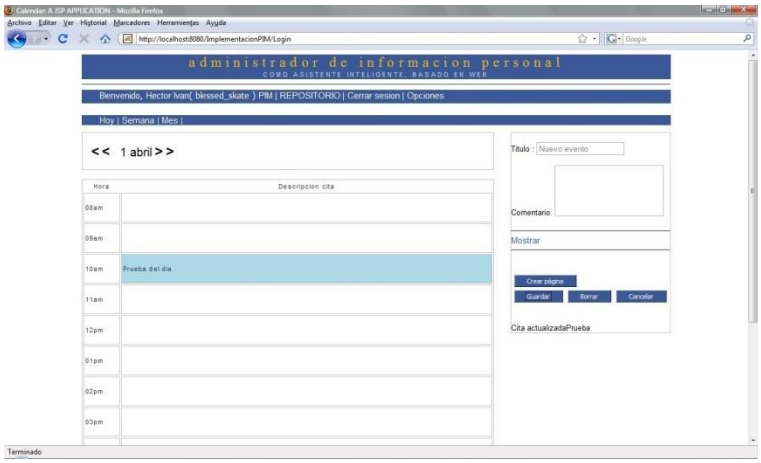

Figura 17. Interfaz del administrador por día.

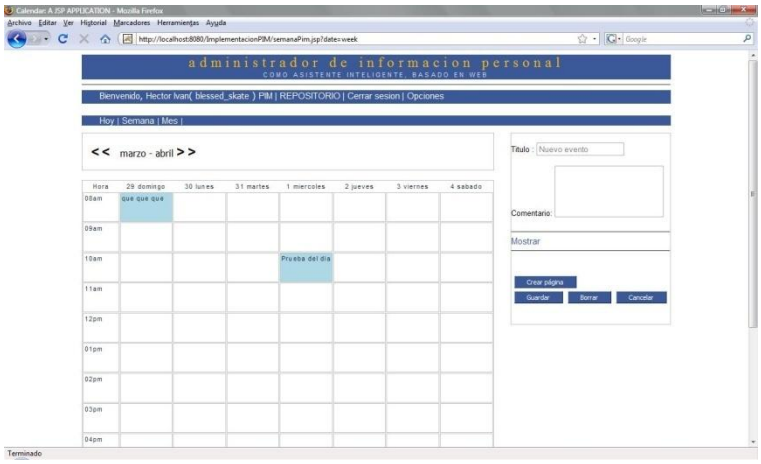

Figura 18. Interfaz del administrador por semana.

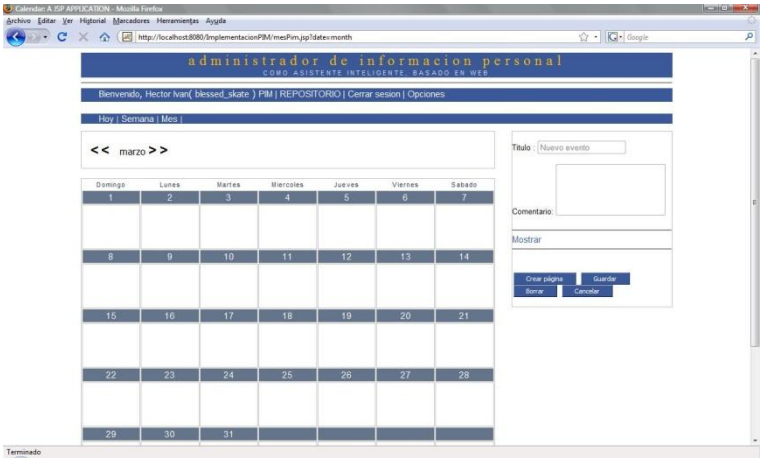

Figura 19. Interfaz del administrador por mes.

Para el administrador de información la vista consta de tres archivos JSP (dia.jsp, semana.jsp y mes.jsp), los cuales hacen un llamado al controlador para recuperar las citas almacenas cada vez que se cargan. Las acciones dinámicas son llamadas desde un archivo Java Script. El archivo JS (datosCitas.js) contiene las acciones para activar los campos para el almacenamiento de citas, obtiene los datos de los campos y los manda al controlador para ser almacenada, borrado o actualizado, en cualquiera de los casos.

4.7.3 Motor de inteligencia.

El motor de inteligencia puede ser activado o desactivado en opciones. Un usuario que acceda al sistema con el motor activo vera inmediatamente un recordatorio de las citas más importantes, teniendo en segundo plano las de mediana y baja importancia.

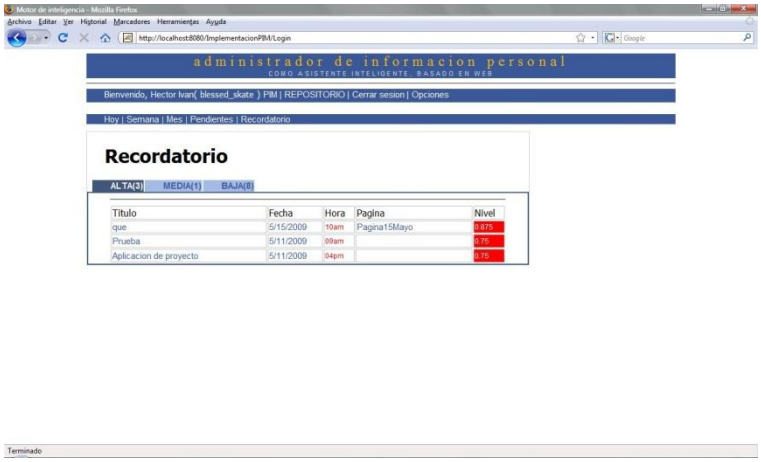

Figura 20. Interfaz de recordatorio.

Al almacenar una cita, con el motor activado, se muestra una lista con posibles fechas y sugerencias de hora de días siguientes, dependiendo de la importancia de las citas en los días siguientes. A su vez se muestra la cita como fue dada de alta en el administrador, teniendo la opción de elegir otra fecha y hora, o almacenar la cita original.

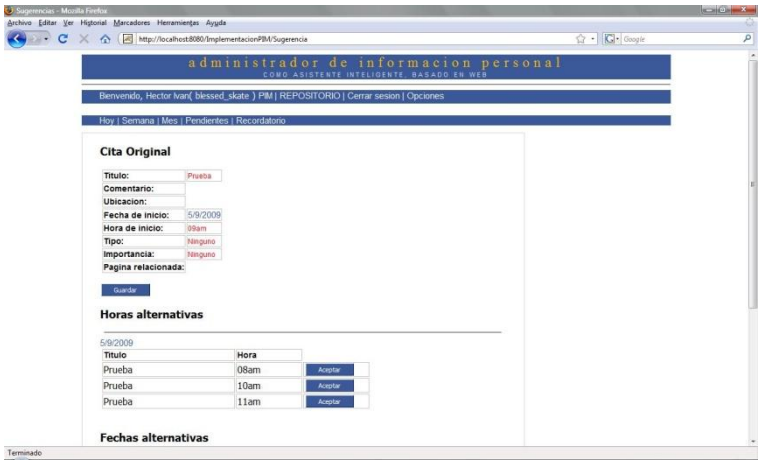

Figura 21. Interfaz de sugerencias.

4.7.4 Repositorio.

El repositorio cuenta con una página principal de bienvenida, anida las páginas existentes para el fácil acceso. Las páginas contienen los archivos, notas, listas y fotos almacenadas, ordenados dándole prioridad a los archivos, enseguida las notas, listas y finalmente las fotos. Las siguientes imágenes muestran la interfaz del repositorio y la página.

Figura 22. Interfaz del repositorio.

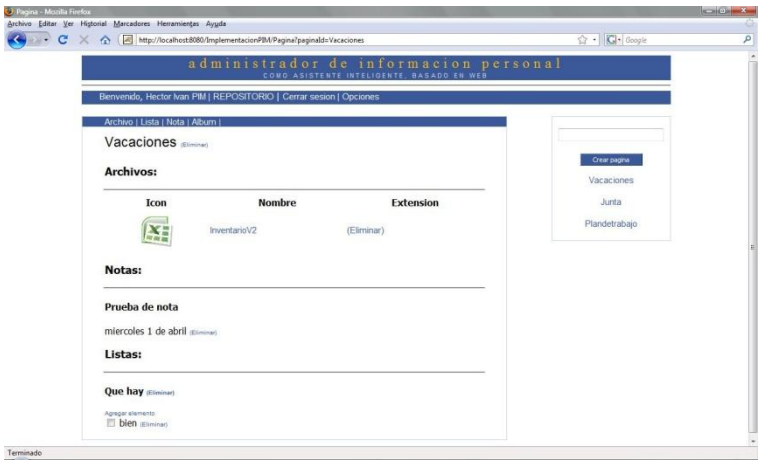

Figura 23. Interfaz de página.

4.8 Modelo relacional.

A continuación se muestra el esquema relacional para la base de datos, describiendo el tipo, las restricciones y las llaves para cada tabla. Además de la inserción automática de los valores para el tipo e importancia de cada cita.

```
drop table USUARIO;

drop table ENTRIES;

drop table PAGINA;

drop table ARCHIVO;

drop table NOTA;

drop table LISTA;

drop table OBJETOLISTA;

drop table TIPO;

drop table IMPORTANCIA;


create table USUARIO(
               id         NUMBER(2)    NOT NULL,
               nom_us         VARCHAR2(20) NOT NULL,
               nombre         VARCHAR2(20) NOT NULL,
               apellido_pa    VARCHAR2(10),
               apellido_ma    VARCHAR2(10),
               correo         VARCHAR2(50),
               contrasena     VARCHAR2(12) NOT NULL,
               PRIMARY KEY (nom_us,contrasena));
create table ENTRIES(
               nom_us              VARCHAR2(20) NOT NULL,
               titulo              VARCHAR2(30) NOT NULL,
               comentario          VARCHAR2(50),
               ubicacion           VARCHAR2(20),
```

```
                fecha_inicio   VARCHAR2(10) NOT NULL,
                hora_inicio          VARCHAR2(4)  NOT NULL,
                fecha_fin            VARCHAR2(10) ,
                hora_fin       VARCHAR2(4),
                tipo                 VARCHAR2(15),
                importancia    VARCHAR2(15),
                pagina         VARCHAR2(30),
                PRIMARY KEY (nom_us, titulo, fecha_inicio, hora_inicio));

create table PAGINA(
                id             VARCHAR2(30) NOT NULL,
                nom_us               VARCHAR2(20) NOT NULL,
                fecha_inicio   VARCHAR2(10) NOT NULL,
                hora_inicio          VARCHAR2(4),
                PRIMARY KEY (id, nom_us, fecha_inicio));
create table ARCHIVO(
                id       VARCHAR2(30)  NOT NULL,
                id_pagina      VARCHAR2(30)  NOT NULL,
                nom_us         VARCHAR2(20)  NOT NULL,
                url_archivo    VARCHAR2(100) NOT NULL,
                extension      VARCHAR2(4)   NOT NULL,
                PRIMARY KEY (id,id_pagina,nom_us,extension));
create table NOTA(
                id       VARCHAR2(30)  NOT NULL,
                id_pagina      VARCHAR2(30)  NOT NULL,
                nom_us         VARCHAR2(20)  NOT NULL,
                nota           VARCHAR2(100) NOT NULL,
                PRIMARY KEY (id,id_pagina,nom_us));
create table LISTA(
```

```
                id       VARCHAR2(30)  NOT NULL,
                id_pagina      VARCHAR2(30)  NOT NULL,
                nom_us         VARCHAR2(20)  NOT NULL,
                PRIMARY KEY (id,id_pagina,nom_us));
create table OBJETOLISTA(
                id       VARCHAR2(30)  NOT NULL,
                id_lista VARCHAR2(30)  NOT NULL,
                id_pagina      VARCHAR2(30)  NOT NULL,
                nom_us   VARCHAR2(20)  NOT NULL,
                valor    VARCHAR2(5),
                PRIMARY KEY (id, id_lista,id_pagina,nom_us));
create table FOTO(
                id       VARCHAR2(30)  NOT NULL,
                id_pagina      VARCHAR2(30)  NOT NULL,
                nom_us         VARCHAR2(20)  NOT NULL,
                PRIMARY KEY (id,id_pagina,nom_us));
create table TIPO(
                id       VARCHAR2(15)  NOT NULL,
                valor    NUMBER(3,2)   NOT NULL,
                PRIMARY KEY (id,valor));

create table IMPORTANCIA(
                id       VARCHAR2(15)  NOT NULL,
                valor    NUMBER(3,2)   NOT NULL,
                PRIMARY KEY (id,valor));
insert into TIPO values ('Ninguno' , 0.0);
insert into TIPO values ('Familiar', 0.75);
insert into TIPO values ('Trabajo' , 0.50);
insert into TIPO values ('Personal', 0.25);
```

```
insert into IMPORTANCIA values ('Ninguno' , 0.0);
insert into IMPORTANCIA values ('Muy alta', 1.0);
insert into IMPORTANCIA values ('Alta'    , 0.80);
insert into IMPORTANCIA values ('Media'   , 0.60);
insert into IMPORTANCIA values ('Baja'    , 0.40);
insert into IMPORTANCIA values ('Muy baja', 0.20);
```

Resumen.

Una vez que se implementó el sistema se fueron encontrando errores que no fueron tomados en cuenta, estos fueron resueltos durante la etapa de pruebas.