Manual Técnico - Sistema de Gestión de Tickets

1. Descripción General

Este sistema de gestión de tickets fue desarrollado en Java utilizando JavaFX para la interfaz gráfica y PostgreSQL como sistema de gestión de base de datos. El sistema permite gestionar usuarios, departamentos, roles, permisos, tickets, colas de atención, flujos de trabajo y bitácora de actividades. Está orientado a organizaciones que necesiten registrar, controlar y resolver incidencias reportadas por usuarios.

2. Estructura del Proyecto

- Modelo: Clases como Ticket, Departamento, EstadoTicket, FlujoTrabajo, etc.
- Controladores: Manejan eventos y lógica de interacción en las vistas JavaFX.
- DAO: Acceso a datos mediante JDBC y sentencias SQL.
- Vistas: Archivos FXML con componentes JavaFX.
- Utilidades: Clases auxiliares como Sesion, Utils, ServicioColas, etc.

3. Base de Datos

Se utiliza PostgreSQL. Entre las tablas más relevantes están:

- persona
- ticket
- estado_ticket
- flujo_trabajo
- flujo_estado
- transicion_estado
- departamento
- cola_atencion_ticket
- ticket_historial
- ticket_archivo
- bitacora

Se usa ON DELETE CASCADE en varias relaciones para mantener la integridad referencial.

4. Gestión de Tickets

- Creación: Un usuario puede crear tickets con título, descripción, archivo adjunto, prioridad y departamento.
- Cola de atención: El ticket se encola automáticamente según el departamento asignado.
- Asignación: Técnicos pueden tomar tickets de la cola.
- Actualización: Cambio de estado, asignación de técnico, carga de archivos adicionales.

- Historial: Se registra cada cambio en la tabla ticket_historial.
- Cierre: El estado cambia a 'Resuelto' o 'Cerrado'.

5. Gestión de Flujos de Trabajo

- Configuración por administradores.
- Se definen estados involucrados, transiciones, restricciones y acciones automáticas.
- Cada ticket sigue un flujo de trabajo preestablecido basado en su estado actual.

6. Roles y Permisos

- Cada rol puede tener múltiples permisos asociados.
- Se implementa control de visibilidad y acciones mediante UtilsPermisos.tienePermiso().
- Acceso restringido a ciertos módulos como Usuarios, Configuración y Bitácora.

7. Bitácora del Sistema

- Registro de toda acción significativa.
- Incluye: usuario que ejecuta, módulo afectado, acción realizada, tipo y fecha.
- Visualizable en diferentes módulos a través de un componente reutilizable.
- Útil para auditoría y seguimiento de actividades.

8. Exportación e Importación

- Exportación de tickets a archivos .bin usando ObjectOutputStream.
- Función de importar archivos con listado de tickets (implementación opcional).
- Carga diferida de colas desde la base al iniciar sesión o reiniciar sistema.

9. Configuración de Empresa

- Se configura nombre, logo, idioma, zona horaria, días de vencimiento, niveles de prioridad.
- Se guarda en tabla config_empresa y tabla separada para nivel_prioridad.
- El logo cargado se usa dinámicamente en la interfaz (menú principal).

10. Observaciones Finales

- Se debe evitar eliminar niveles de prioridad una vez asignados.
- Validar correctamente los roles al hacer login para instanciar correctamente Técnico, Usuario o Admin.
- La cola en memoria se sincroniza con la base al iniciar.
- Las acciones deben registrar bitácora siempre que sea relevante.

7. Ejemplos de Código

7.1 Insertar un Ticket

```
public static int insertarTicket(Ticket ticket) {
   String sql = "INSERT INTO ticket (titulo, descripcion, fecha_creacion, adjunto,
nombre_archivo, estado_id, prioridad, departamento_id, creado_por) "
            + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?) RETURNING id";

   try (Connection conn = Conexion.getConexion();
       PreparedStatement ps = conn.prepareStatement(sql)) {

      ps.setString(1, ticket.getTitulo());
      ps.setString(2, ticket.getDescripcion());
      ps.setTimestamp(3, Timestamp.valueOf(ticket.getFechaCreacion()));
      ps.setBytes(4, ticket.getArchivoAdjunto());
      ps.setString(5, ticket.getNombreArchivo());
      ps.setInt(6, ticket.getEstado().getId());
      ps.setInt(7, ticket.getPrioridad().getId());
      ps.setInt(8, ticket.getDepartamento().getId());
      ps.setInt(9, ticket.getCreador().getIdentificacion());

      ResultSet rs = ps.executeQuery();
      if (rs.next()) {
         int id = rs.getInt("id");
         ticket.setId(id);
         return id;
      }
   } catch (SQLException e) {
      System.err.println("Error al insertar ticket: " + e.getMessage());
   }
   return -1;
}
```

7.2 Agregar Ticket a Cola

```
public static void agregarTicketACola(int ticketId, int departamentoId, LocalDateTime
fecha) {
   String sql = "INSERT INTO cola_atencion_ticket (ticket_id, departamento_id,
fecha_ingreso) VALUES (?, ?, ?)";

   try (Connection conn = Conexion.getConexion();
       PreparedStatement ps = conn.prepareStatement(sql)) {

      ps.setInt(1, ticketId);
```

```
            ps.setInt(2, departamentoId);
            ps.setTimestamp(3, Timestamp.valueOf(fecha));
            ps.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Error al agregar ticket a la cola: " + e.getMessage());
        }
    }
```

7.3 Cambiar Estado del Ticket (Controlador)

```
@FXML
private void actualizarEstado(ActionEvent event) {
    Ticket ticketSeleccionado = tablaTickets.getSelectionModel().getSelectedItem();
    EstadoTicket nuevoEstado = comboEstado.getValue();

    if (ticketSeleccionado == null || nuevoEstado == null) {
        Utils.mostrarAlerta(Alert.AlertType.ERROR, "Error", "Debe seleccionar un ticket y un
nuevo estado.");
        return;
    }

    Dao.actualizarEstadoTicket(ticketSeleccionado.getId(), nuevoEstado);
    Dao.insertarHistorialEstado(ticketSeleccionado.getId(), nuevoEstado, "Cambio de estado
por administrador");
    Dao.registrarBitacora(Sesion.getUsuarioActual(), "Cambio de estado del ticket " +
ticketSeleccionado.getId() + " a " + nuevoEstado.getNombre(), "Gestión de Tickets",
"Actualización");

    Utils.mostrarAlerta(Alert.AlertType.INFORMATION, "Éxito", "Estado actualizado.");
    cargarTickets();
}
```