Documento de Microdiseño

SRO - DM

proyecto sro

2014

**Historial de Revisiones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** | **Autor** |
| 31/02/2014 | 1.0 | Versión preliminar como una propuesta de desarrollo. | Equipo de Diseño |
| 10/06/2014 | 2.0 | Adición de una tabla de diseño respecto al profesional. | Equipo de Diseño |
| 16/06/2014 | 3.0 | Adición de una tabla de diseño relacionado a los procedimientos | Equipo de Diseño |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Detalle de**  **Diseño N°** | 01 | | |
| **Elaborado Por:** | Cesar Zegarra | | |
| **Descripción General:** | Este documento está orientado a servir de guía en el desarrollo de la funcionalidad:  **CRUD (Creación, Lectura, Actualización, Eliminación) de Paciente**  La aplicación deberá registrar y/o editar los datos básicos de un paciente que entra a cirugía. | | |
| **Referencias:** | **Requerimientos:** | | SRO - DECU – RP.docx  **CU04 - Registrar Paciente** |
| **Diseño de Interfaz de Usuario:** | | SRO – DD.docx  **Pantalla 4 – Listado de Pacientes** |
| **Diseño de Arquitectura:** | | SRO – LR.xls  **Requisitos No Funcionales** |
| **Otros:** | |  |
| **Clases requeridas:** | **Nombre** | **Stereotype** | **Acción** |
| PacienteBean.java | Bean | Crear |
| Paciente.java | Bean | Crear |
| PacienteService.java | Service | Crear |
| IPacienteService.java | Service Interface | Crear |
| PacienteDao.java | Dao | Crear |
| IPacienteDao.java | Dao Inteface | Crear |
| **JSF (xhtml) :** | Nombre del archivo de vista (JSF) | | Acción |
| mantPaciente.xhtml | | Crear |
| **Tablas:** | Nombre de Tabla | | Tipo de Objeto |
| PACIENTE | | Tabla |
| **Detalle de**  **Implementación a**  **Nivel de Clases:** | **org.corp.siro.domain.Paciente**   * Clase de persistencia utilizada para representar la información de la tabla **PACIENTE** y facilitar la programación en un ambiente ***Object Relation Mapping***. Mediante la configuración en anotaciones JAVA presentes en la misma clase, usando el estándar JPA, lo que permitirá realizar operaciones transparentes de Inserción, Actualización y Consulta sobre dicha tabla, sin necesidad de hacer consultas nativas sobre la base de datos.   **Métodos**:   * Dado que se trata de un ***DTO*** que solo será utilizado como un objeto almacenador de información, los únicos métodos a implementar son los accesores: *Getters* y *Setters* por cada propiedad/atributo del mismo.  1. **public TipoDatosAtributo** getNombreAtributo() 2. **public void** setNombreAtributo()   **Acción:**   * Crear   **org.corp.siro.bean.PacienteBean**  Clase ***BackingBean*** utilizada para recuperar los objetos y recursos del contexto. Además nos permite mapear la vista **mantPaciente*.xhtml*** para implementar funcionalidad vía llamadas a los respectivos *Services*.  @ManagedBean(name="PacienteBean")  @ViewScoped  public class PacienteBean implements Serializable  **Métodos:**  Todos los métodos son llamados via ajax.  public void editarEvent()  Llamado cuando un paciente a editar es enviado de la vista al bean.  public void eliminarEvent()  Llamado cuando un paciente a eliminar es enviad de la vista al bean.  public void refrescarPaciente()  Llamado cuando se quiere refrescar la lista de pacientes.  public void validarNuevo(ActionEvent actionEvent)  Método que se llama para validar que no existan campos en blanco o con formato inadecuado antes de insertar.  public void validarEditar(ActionEvent actionEvent)  Método que se llama para validar que no existan campos en blanco o con formato inadecuado antes de editar.  public void validarEliminar(ActionEvent actionEvent)  Método que se llama para validar que no se elimine a algún paciente que tenga relaciones (Foreign Key) con otros registros de la base de datos en uso. | | |
| **Detalle de**  **implementación a**  **nivel de Servicios** | **org.corp.siro.service.IPacienteService**   * Interface utilizada para desacoplar las capas implementadas en el sistema (capa de *Negocio* de la capa *Controller*) y facilitar el acceso transparente a los métodos de negocio implementadas en las clases de negocio (Services).   **Métodos**:   1. **public void** addPaciente**(Paciente Paciente);** 2. **public void** updatePaciente**(Paciente Paciente);** 3. **public void** deletePaciente**(Paciente Paciente);** 4. **public void** deletePacienteLogico**(Paciente Paciente);** 5. **public Paciente** getPacienteById**(int id);** 6. **public List<Paciente>** getPacientes**();**   **Acción:**  Crear  **org.corp.siro.service.IPacienteService**  Clase de negocio que implementa la interface **IPacienteService**.  Detalla la funcionalidad de los métodos publicados por la interfaz. Son métodos de negocio que realizan operaciones sobre el objeto Paciente.  **Métodos**:   1. **public void** addPaciente**(Paciente Paciente);**   **Registra los datos referentes a un paciente nuevo.**   1. **public void** updatePaciente**(Paciente Paciente);**   **Actualiza los datos referentes a un paciente ya registrado en el sistema.**   1. **public void** deletePaciente**(Paciente Paciente);**   **Elimina fisicamente a un paciente en específico.**   1. **public void** deletePacienteLogico**(Paciente Paciente);**   **Elimina lógicamente (activo=false) a un paciente en específico.**   1. **public Paciente** getPacienteById**(int id);**   **Recupera un paciente por medio de su identificador primario.**   1. **public List<Paciente>** getPacientes**();**   **Recupera todos los registros de pacientes.** | | |
| **Detalle de**  **implementación a**  **nivel de Daos** | **org.corp.siro.dao.IPacienteDao**   * Interface utilizada para desacoplar las capas implementadas en el sistema (capa de *Datos* de la capa *Negocio*).   **Métodos:**   1. **public void addPaciente(Paciente Paciente);** 2. **public void updatePaciente(Paciente Paciente);** 3. **public void deletePaciente(Paciente Paciente);** 4. **public Paciente getPacienteById(int id);** 5. **public List<Paciente> getPacientes();**   **Acción:**   * Crear   **org.corp.siro.dao.PacienteDao**  Clase de acceso a datos implementa la interfase ***IPacienteDao***.  Implementa métodos de acceso usando las funciones disponibles en su clase que controla las sesiones: **sessionFactory**, accediendo de esta forma la tabla PACIENTE sobre la base de datos.   1. **public void addPaciente(Paciente Paciente);**   **Registra los datos referentes a un paciente nuevo en la tabla PACIENTE por medio del método de inserción del sessionFactory (Hibernate).**   1. **public void updatePaciente(Paciente Paciente);**   **Actualiza los datos referentes a un paciente ya registrado en la tabla PACIENTE por medio del método de actualizacion del sessionFactory (Hibernate).**   1. **public void deletePaciente(Paciente Paciente);**   **Elimina físicamente a un paciente en específico de la tabla PACIENTE por medio del método de eliminación del sessionFactory (Hibernate).**   1. **public Paciente getPacienteById(int id);**   **Recupera un paciente de la tabla PACIENTE por medio de su identificador primario usando el método de recuperación de registros por id del sessionFactory (Hibernate).**   1. **public List<Paciente> getPacientes();**   **Recupera todos los registros de pacientes de la tabla PACIENTE usando el método de recuperación de registros por lote del sessionFactory (Hibernate).** | | |
| **Detalle de**  **implementación a**  **nivel de JSF’s** | **mantPaciente.xhtml**  **Ruta:**   * webapp/pages/mantPaciente.xhtml   **Acción:**   * Crear   **Descripción:**  Objeto de vista.  Implementado en formato JSF y utilizado para crear la Interfaz Gráfica de Usuario GUI del mantenimiento de los pacientes, utilizan para ello los tags propios de *JSF* y de la librería Primefaces.  Contiene las siguientes pestañas:  - Cabecera:  ( Historia Clinica, Nombres, Apellidos, Edad )  - Detalle de Pacientes:  (Fecha de Nacimiento) | | |
| **Detalle de**  **Implementación a**  **Nivel de Tablas:** | Script de Creación de Tablas  DROP TABLE IF EXISTS `paciente`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!40101 SET character\_set\_client = utf8 \*/;  CREATE TABLE `paciente` (  `HistClinica` varchar(30) NOT NULL,  `Nombres` varchar(45) NOT NULL,  `Apellidos` varchar(45) NOT NULL,  `Edad` int(11) NOT NULL,  `FechNac` date NOT NULL,  `Eliminacion` tinyint(1) NOT NULL,  PRIMARY KEY (`HistClinica`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/; | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Detalle de**  **Diseño N°** | 02 | | |
| **Elaborado Por:** | Juan Guerrero | | |
| **Descripción General:** | Este documento está orientado a servir de guía en el desarrollo de la funcionalidad:  **CRUD (Creación, Lectura, Actualización, Eliminación) del Profesional**  La aplicación deberá registrar y/o editar los datos básicos de un profesional. | | |
| **Referencias:** | **Requerimientos:** | | SRO - DECU – RProf.docx  **CU06 - Registrar Profesional** |
| **Diseño de Interfaz de Usuario:** | | SRO – DD.docx  **Pantalla 1 – Listado de Profesionales** |
| **Diseño de Arquitectura:** | | SRO – LR.xls  **Requisitos No Funcionales** |
| **Otros:** | |  |
| **Clases requeridas:** | **Nombre** | **Stereotype** | **Acción** |
| ProfesionalBean.java | Bean | Crear |
| Profesional.java | Bean | Crear |
| ProfesionalService.java | Service | Crear |
| IProfesionalService.java | Service Interface | Crear |
| ProfesionalDao.java | Dao | Crear |
| IProfesionalDao.java | Dao Inteface | Crear |
| **JSF (xhtml) :** | Nombre del archivo de vista (JSF) | | Acción |
| mantProfesional.xhtml | | Crear |
| **Tablas:** | Nombre de Tabla | | Tipo de Objeto |
| PROFESIONAL | | Tabla |
| **Detalle de**  **Implementación a**  **Nivel de Clases:** | **org.corp.siro.domain.Profesional**   * Clase de persistencia utilizada para representar la información de la tabla **PROFESIONAL** y facilitar la programación en un ambiente ***Object Relation Mapping***. Mediante la configuración en anotaciones JAVA presentes en la misma clase, usando el estándar JPA, lo que permitirá realizar operaciones transparentes de Inserción, Actualización y Consulta sobre dicha tabla, sin necesidad de hacer consultas nativas sobre la base de datos.   **Métodos**:   * Dado que se trata de un ***DTO*** que solo será utilizado como un objeto almacenador de información, los únicos métodos a implementar son los accesores: *Getters* y *Setters* por cada propiedad/atributo del mismo.  1. **public TipoDatosAtributo** getNombreAtributo() 2. **public void** setNombreAtributo()   **Acción:**   * Crear   **org.corp.siro.bean.ProfesionalBean**  Clase ***BackingBean*** utilizada para recuperar los objetos y recursos del contexto. Además nos permite mapear la vista **mantProfesional*.xhtml*** para implementar funcionalidad vía llamadas a los respectivos *Services*.  @ManagedBean(name="ProfesionalBean")  @ViewScoped  public class ProfesionalBean implements Serializable  **Métodos:**  Todos los métodos son llamados via ajax.  public void editarEvent()  Llamado cuando un profesional a editar es enviado de la vista al bean.  public void eliminarEvent()  Llamado cuando un profesional a eliminar es enviad de la vista al bean.  public void refrescarProfesional()  Llamado cuando se quiere refrescar la lista de profesionales.  public void validarNuevo(ActionEvent actionEvent)  Método que se llama para validar que no existan campos en blanco o con formato inadecuado antes de insertar.  public void validarEditar(ActionEvent actionEvent)  Método que se llama para validar que no existan campos en blanco o con formato inadecuado antes de editar.  public void validarEliminar(ActionEvent actionEvent)  Método que se llama para validar que no se elimine a algún profesional que tenga relaciones (Foreign Key) con otros registros de la base de datos en uso. | | |
| **Detalle de**  **implementación a**  **nivel de Servicios** | **org.corp.siro.service.IProfesionalService**   * Interface utilizada para desacoplar las capas implementadas en el sistema (capa de *Negocio* de la capa *Controller*) y facilitar el acceso transparente a los métodos de negocio implementadas en las clases de negocio (Services).   **Métodos**:   1. **public void addProfesional(Profesional profesional);** 2. **public void** **updateProfesional(Profesional profesional);** 3. **public void** **deleteProfesional(Profesional profesional);** 4. **public void** **deleteProfesionalLogico(Profesional profesional);** 5. **public Profesional** **getProfesionalById(int id);** 6. **public List<Profesional>** **getProfesionales();**   **Acción:**  Crear  **org.corp.siro.service.IProfesionalService**  Clase de negocio que implementa la interface **IProfesionalService**.  Detalla la funcionalidad de los métodos publicados por la interfaz. Son métodos de negocio que realizan operaciones sobre el objeto Profesional.  **Métodos**:   1. **public void addProfesional(Profesional profesional);**   **Registra los datos referentes a un profesional nuevo.**   1. **public void updateProfesional(Profesional profesional);**   **Actualiza los datos referentes a un profesional ya registrado en el sistema.**   1. **public void deleteProfesional(Profesional profesional);**   **Elimina físicamente a un profesional en específico.**   1. **public void deleteProfesionalLogico(Profesional profesional);**   **Elimina lógicamente (activo=false) a un profesional en específico.**   1. **public Profesional** **getProfesionalById(int id);**   **Recupera un profesional por medio de su identificador primario.**   1. **public List<Profesional> getProfesionales();**   **Recupera todos los registros de profesionales.** | | |
| **Detalle de**  **implementación a**  **nivel de Daos** | **org.corp.siro.dao.IProfesionalDao**   * Interface utilizada para desacoplar las capas implementadas en el sistema (capa de *Datos* de la capa *Negocio*).   **Métodos:**   1. **public void addProfesional(Profesional profesional);** 2. **public void updateProfesional(Profesional profesional);** 3. **public void deleteProfesional(Profesional profesional);** 4. **public Profesional getProfesionalById(int id);** 5. **public List<Profesional> getProfesionales();**   **Acción:**   * Crear   **org.corp.siro.dao.ProfesionalDao**  Clase de acceso a datos implementa la interfase ***IProfesionalDao***.  Implementa métodos de acceso usando las funciones disponibles en su clase que controla las sesiones: **sessionFactory**, accediendo de esta forma la tabla PROFESIONAL sobre la base de datos.   1. **public void addProfesional(Profesional profesional);**   **Registra los datos referentes a un profesional nuevo en la tabla PROFESIONAL por medio del método de inserción del sessionFactory (Hibernate).**   1. **public void updateProfesional(Profesional profesional);**   **Actualiza los datos referentes a un profesional ya registrado en la tabla PROFESIONAL por medio del método de actualización del sessionFactory (Hibernate).**   1. **public void deleteProfesional(Profesional profesional);**   **Elimina físicamente a un profesional en específico de la tabla PROFESIONAL por medio del método de eliminación del sessionFactory (Hibernate).**   1. **public Profesional getProfesionalById(int id);**   **Recupera un Profesional de la tabla PROFESIONAL por medio de su identificador primario usando el método de recuperación de registros por id del sessionFactory (Hibernate).**   1. **public List<Profesional> getProfesionales();**   **Recupera todos los registros de profesionales de la tabla PROFESIONAL usando el método de recuperación de registros por lote del sessionFactory (Hibernate).** | | |
| **Detalle de**  **implementación a**  **nivel de JSF’s** | **mantProfesional.xhtml**  **Ruta:**   * webapp/pages/mantProfesional.xhtml   **Acción:**   * Crear   **Descripción:**  Objeto de vista.  Implementado en formato JSF y utilizado para crear la Interfaz Gráfica de Usuario GUI del mantenimiento de los profesionales, utilizan para ello los tags propios de *JSF* y de la librería Primefaces.  Contiene las siguientes pestañas:  - Cabecera:  ( Nombres, Apellidos, Profesión )  - Detalle de profesionales:  ( Colegio ) | | |
| **Detalle de**  **Implementación a**  **Nivel de Tablas:** | Script de Creación de Tablas  DROP TABLE IF EXISTS `profesional`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!40101 SET character\_set\_client = utf8 \*/;  CREATE TABLE ` profesional ` (  `IDPROFESIONAL` int NOT NULL,  `COLEGIO` varchar(45),  `NOMBRES` varchar(45),  `APELLIDOS` varchar(45),  `PROFESION` varchar(45),  PRIMARY KEY (` IDPROFESIONAL `)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/; | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Detalle de**  **Diseño N°** | 03 | | |
| **Elaborado Por:** | Alexander Huaman | | |
| **Descripción General:** | Este documento está orientado a servir de guía en el desarrollo de la funcionalidad:  **CRUD (Creación, Lectura, Actualización, Eliminación) del Procedimiento**  La aplicación deberá registrar y/o editar los datos básicos de un procedimientol. | | |
| **Referencias:** | **Requerimientos:** | | SRO - DECU – RProc.docx  **CU06 - Registrar Procedimiento** |
| **Diseño de Interfaz de Usuario:** | | SRO – DD.docx  **Pantalla 3 – Listado de Procedimientos** |
| **Diseño de Arquitectura:** | | SRO – LR.xls  **Requisitos No Funcionales** |
| **Otros:** | |  |
| **Clases requeridas:** | **Nombre** | **Stereotype** | **Acción** |
| ProcedimientoBean.java | Bean | Crear |
| Procedimiento.java | Bean | Crear |
| ProcedimientoService.java | Service | Crear |
| IProcedimientoService.java | Service Interface | Crear |
| ProcedimientoDao.java | Dao | Crear |
| IProcedimientoDao.java | Dao Inteface | Crear |
| **JSF (xhtml) :** | Nombre del archivo de vista (JSF) | | Acción |
| mantProcedimiento.xhtml | | Crear |
| **Tablas:** | Nombre de Tabla | | Tipo de Objeto |
| PROCEDIMIENTO | | Tabla |
| **Detalle de**  **Implementación a**  **Nivel de Clases:** | **org.corp.siro.domain.Profcedimiento**   * Clase de persistencia utilizada para representar la información de la tabla **PROCEDIMIENTO** y facilitar la programación en un ambiente ***Object Relation Mapping***. Mediante la configuración en anotaciones JAVA presentes en la misma clase, usando el estándar JPA, lo que permitirá realizar operaciones transparentes de Inserción, Actualización y Consulta sobre dicha tabla, sin necesidad de hacer consultas nativas sobre la base de datos.   **Métodos**:   * Dado que se trata de un ***DTO*** que solo será utilizado como un objeto almacenador de información, los únicos métodos a implementar son los accesores: *Getters* y *Setters* por cada propiedad/atributo del mismo.  1. **public TipoDatosAtributo** getNombreAtributo() 2. **public void** setNombreAtributo()   **Acción:**   * Crear   **org.corp.siro.bean.ProcedimientoBean**  Clase ***BackingBean*** utilizada para recuperar los objetos y recursos del contexto. Además nos permite mapear la vista **mantProcedimiento*.xhtml*** para implementar funcionalidad vía llamadas a los respectivos *Services*.  @ManagedBean(name="ProcedimientoBean")  @ViewScoped  public class ProcedimientoBean implements Serializable  **Métodos:**  Todos los métodos son llamados via ajax.  public void editarEvent()  Llamado cuando un procedimiento a editar es enviado de la vista al bean.  public void eliminarEvent()  Llamado cuando un procedimiento a eliminar es enviad de la vista al bean.  public void refrescarProfesional()  Llamado cuando se quiere refrescar la lista de procedimiento.  public void validarNuevo(ActionEvent actionEvent)  Método que se llama para validar que no existan campos en blanco o con formato inadecuado antes de insertar.  public void validarEditar(ActionEvent actionEvent)  Método que se llama para validar que no existan campos en blanco o con formato inadecuado antes de editar.  public void validarEliminar(ActionEvent actionEvent)  Método que se llama para validar que no se elimine a algún procedimiento que tenga relaciones (Foreign Key) con otros registros de la base de datos en uso. | | |
| **Detalle de**  **implementación a**  **nivel de Servicios** | **org.corp.siro.service.IProcedimientoService**   * Interface utilizada para desacoplar las capas implementadas en el sistema (capa de *Negocio* de la capa *Controller*) y facilitar el acceso transparente a los métodos de negocio implementadas en las clases de negocio (Services).   **Métodos**:   1. **public void addProcedimiento(Procedimiento procedimiento);** 2. **public void** **updateProcedimiento(Procedimiento procedimiento);** 3. **public void** **deleteProcedimiento(Procedimiento procedimiento);** 4. **public void** **deleteProcedimientoLogico(Procedimiento procedimiento);** 5. **public Procedimiento** **getProcedimientoById(int id);** 6. **public List<Procedimiento>** **getProcedimientos();**   **Acción:**  Crear  **org.corp.siro.service.ProcedimientoService**  Clase de negocio que implementa la interface I**ProcedimientoService**.  Detalla la funcionalidad de los métodos publicados por la interfaz. Son métodos de negocio que realizan operaciones sobre el objeto Procedimiento.  **Métodos**:   1. **public void addProcedimiento(Procedimiento procedimiento);**   **Registra los datos referentes a un procedimiento nuevo.**   1. **public void** **updateProcedimiento(Procedimiento procedimiento);**   **Actualiza los datos referentes a un procedimiento ya registrado en el sistema.**   1. **public void** **deleteProcedimiento(Procedimiento procedimiento);**   **Elimina físicamente a un procedimiento en específico.**   1. **public void** **deleteProcedimientoLogico(Procedimiento procedimiento);**   **Elimina lógicamente (activo=false) a un procedimiento en específico.**   1. **public Procedimiento** **getProcedimientoById(int id);**   **Recupera un procedimiento por medio de su identificador primario.**   1. **public List<Procedimiento>** **getProcedimientos();**   **Recupera todos los registros de procedimientos.** | | |
| **Detalle de**  **implementación a**  **nivel de Daos** | **org.corp.siro.dao.IProcedimientoDao**   * Interface utilizada para desacoplar las capas implementadas en el sistema (capa de *Datos* de la capa *Negocio*).   **Métodos:**   1. **public void addProcedimiento(Procedimiento procedimiento);** 2. **public void updateProcedimiento(Procedimiento procedimiento);** 3. **public void deleteProcedimiento(Procedimiento procedimiento);** 4. **public Procedimiento getProcedimientoById(int id);** 5. **public List<Procedimiento> getProcedimientos();**   **Acción:**   * Crear   **org.corp.siro.dao.ProcedimientoDao**  Clase de acceso a datos implementa la interfase ***IProcedimientoDao***.  Implementa métodos de acceso usando las funciones disponibles en su clase que controla las sesiones: **sessionFactory**, accediendo de esta forma la tabla PROCEDIMIENTO sobre la base de datos.   1. **public void addProcedimiento(Procedimiento procedimiento);**   **Registra los datos referentes a un procedimiento nuevo en la tabla PROCEDIMIENTO por medio del método de inserción del sessionFactory (Hibernate).**   1. **public void updateProcedimiento(Procedimiento procedimiento);**   **Actualiza los datos referentes a un procedimiento ya registrado en la tabla PROCEDIMIENTO por medio del método de actualización del sessionFactory (Hibernate).**   1. **public void deleteProcedimiento(Procedimiento procedimiento);**   **Elimina físicamente a un procedimiento en específico de la tabla PROCEDIMIENTO por medio del método de eliminación del sessionFactory (Hibernate).**   1. **public Procedimiento getProcedimientoById(int id);**   **Recupera un Procedimiento de la tabla PROCEDIMIENTO por medio de su identificador primario usando el método de recuperación de registros por id del sessionFactory (Hibernate).**   1. **public List<Procedimiento> getProcedimientos();**   **Recupera todos los registros de procedimiento de la tabla PROCEDIMIENTO usando el método de recuperación de registros por lote del sessionFactory (Hibernate).** | | |
| **Detalle de**  **implementación a**  **nivel de JSF’s** | **mantProcedimiento.xhtml**  **Ruta:**   * webapp/pages/mantProcedimiento.xhtml   **Acción:**   * Crear   **Descripción:**  Objeto de vista.  Implementado en formato JSF y utilizado para crear la Interfaz Gráfica de Usuario GUI del mantenimiento de los profesionales, utilizan para ello los tags propios de *JSF* y de la librería Primefaces.  Contiene las siguientes pestañas:  - Cabecera:  ( Nombres, Apellidos, Profesión )  - Detalle de profesionales:  ( Colegio ) | | |
| **Detalle de**  **Implementación a**  **Nivel de Tablas:** | Script de Creación de Tablas  DROP TABLE IF EXISTS `procedimiento`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!40101 SET character\_set\_client = utf8 \*/;  CREATE TABLE `procedimiento` (  `idProcedimiento` int(11) NOT NULL AUTO\_INCREMENT,  `CIE` varchar(30) DEFAULT NULL,  `Nombre` varchar(45) NOT NULL,  `Descripcion` varchar(45) DEFAULT NULL,  `Eliminacion` tinyint(1) NOT NULL,  PRIMARY KEY (`idProcedimiento`),  UNIQUE KEY `CIE\_UNIQUE` (`CIE`)  ) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/; | | |