# Tecnicatura universitaria en programación.

## Informe práctica profesional.

Municipalidad de Paraná – Área técnica de alumbrado público.

Autor: Eugenia Lucia Main.

Supervisor de práctica: Ignacio Fernández.

Tutor de prácticas: Ernesto Zapata Icart.

Fecha de presentación: 10/05/2024

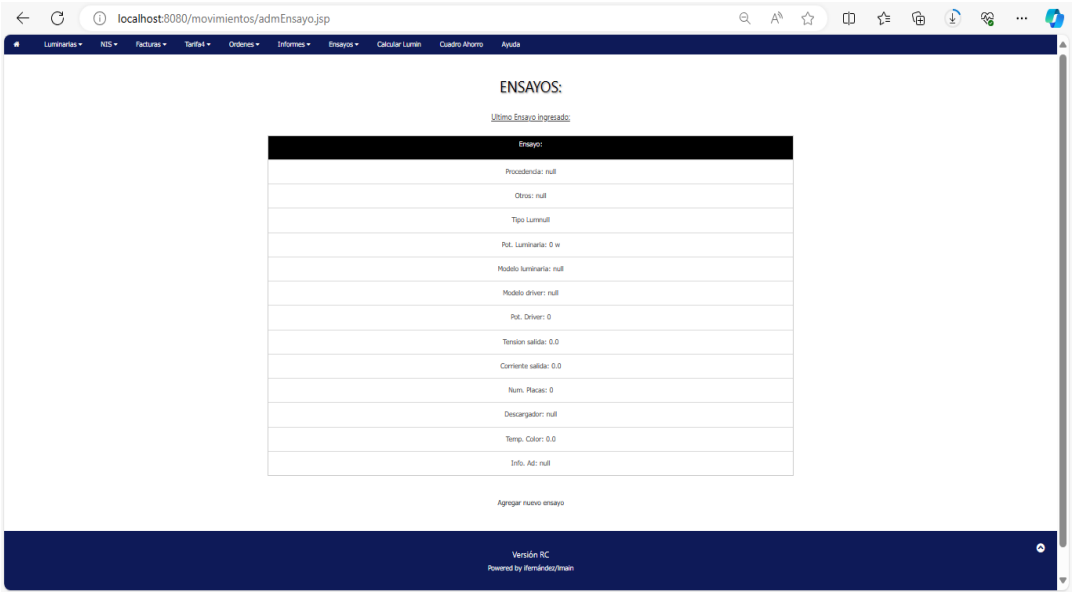
Resumen ejecutivo:

- El informe se centra en las prácticas profesionales desarrolladas en la dirección técnica de alumbrado público.

  Durante este periodo se llevaron a cabo varias actualizaciones, corrección de errores y otras tareas, además de una importante adicción al software ya existente de la dirección. Esta adición permite ahora almacenar, editar y eliminar los ensayos realizados en el laboratorio de alumbrado, lo que ha mejorado significativamente la gestión de datos.

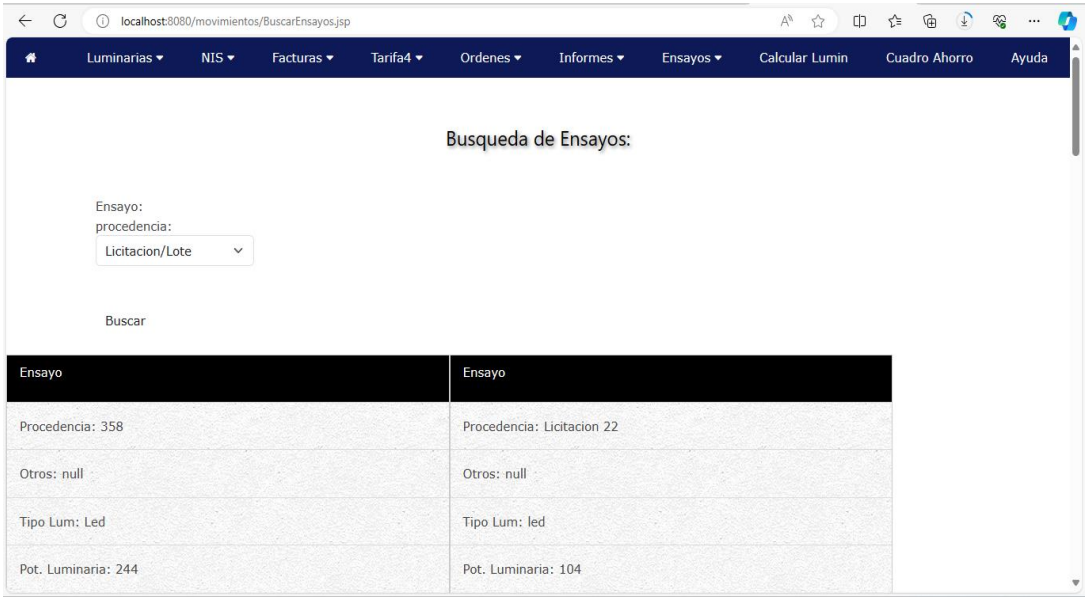
  La problemática surgió debido a la dificultad para localizar en el mapa las luminarias que ya habían sido sometidas a ensayos previos. Esto se volvió crucial para identificar las luminarias por licitación, lote y obra pública al momento de su colocación. Esto facilitaría la gestión de luminarias dentro de alumbrado ya que se podría saber los resultados de los ensayos de las luminarias colocadas en la ciudad. Este problema se aborda mediante una sección dedicada exclusivamente a los ensayos, dividida en tres partes principales:

La primera parte presenta los datos del último ensayo cargado en el sistema, ofreciendo una visión rápida y la opción de agregar uno nuevo.



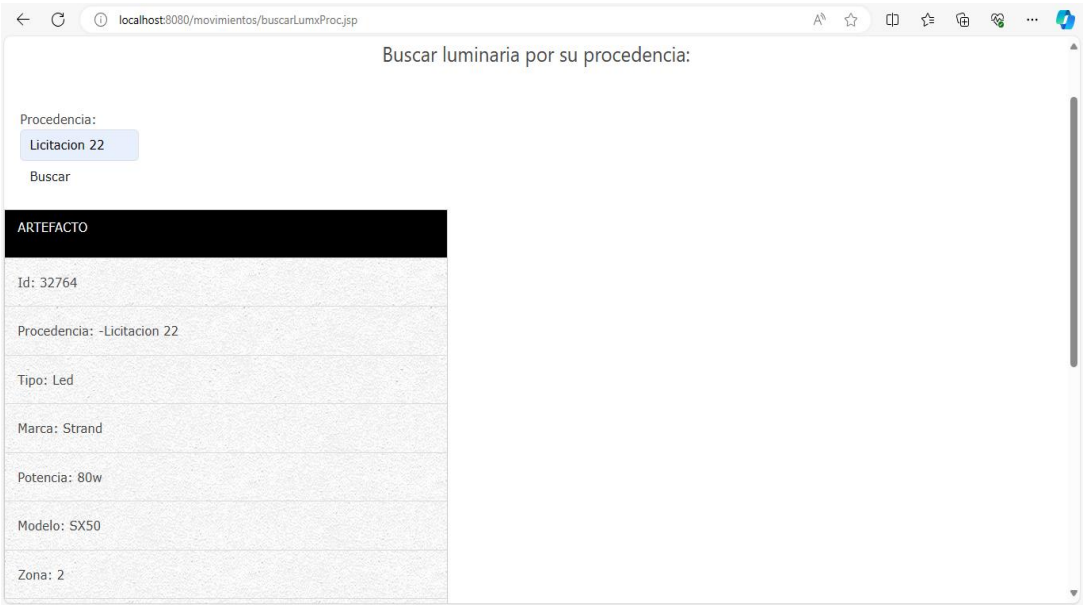

La segunda parte ofrece la opción de agregar un nuevo ensayo al sistema, facilitando la actualización continua de los datos.

Por último, esta sección incluye una función de búsqueda que permite filtrar los ensayos según su procedencia. Los usuarios pueden realizar una búsqueda de ensayos específicos según la licitación, el loteo o la obra pública a la que estén asociados, permite la accesibilidad de la información y facilita la localización de ensayos dentro del sistema.




Al final de cada ensayo filtrado se encuentra la opción de eliminar o editar:

En la sección de luminarias, se ha implementado una función de búsqueda que permite localizar las luminarias según su procedencia. Esta función facilita la identificación de los ensayos realizados.



# Índice

Introducción:

- La dirección técnica de alumbrado público tiene la responsabilidad de mantener un registro actualizado de todas las luminarias de la ciudad de Paraná, además de cumplir con otras actividades importantes. Este departamento cuenta de un laboratorio que tiene la capacidad de desarrollar ensayos y mediciones eléctricas/electrónicas. Con instrumentación compatible para llevar a cabo la introducción a mediciones fotométricas, mejora el funcionamiento y la eficiencia del sistema de luminarias, genera ahorro, detecta fallas y brinda información precisa para la compra de nuevos equipos.

  Además el área técnica implementó el Sistema Integral de Gestión de Alumbrado Público (S.I.G.A.P). Este sistema facilita la manipulación de datos relacionados a puntos georreferenciados en el plano, agilizando el proceso de ingresar información de los puntos de luz de la ciudad tales como: tipo de artefacto, marca, montaje, driver, etc., pudiendo obtener infinidades de estadísticas de forma instantánea de la información que hoy existe y está instalada, con lo que este programa permite crear una base de datos sólida, capaz de almacenar la información precisa del alumbrado público de la ciudad.

  Lo presentado en este informe establece una conexión importante entre el laboratorio del departamento y el software S.I.G.A.P, ya que permite almacenar los ensayos realizados en el laboratorio y transferir esa información de manera efectiva al software para su posterior gestión.

Antecedentes**:**

- Para la realización de este trabajo final, me basé en el proyecto de fin de carrera de Técnico universitario en Programación, desarrollado por Ignacio Fernández, creador del software S.I.G.A.P (2023), en donde se implementó la extensión presentada utilizando las tecnologías, arquitectura y diseño que el empleó, además Ignacio fue el responsable de supervisar mi práctica profesional.

  S.I.G.A.P es un software donde se encuentran diferentes secciones para la manipulación y gestión de datos. Estos datos provienen de un software de código abierto, QGIS, donde las luminarias están georeferenciadas junto con sus atributos correspondientes. El sistema S.I.G.A.P cuenta con una base de datos local donde se almacenan archivos desde el 2019 en adelante permitiendo realizar estadísticas, presupuestos y llevar a cabo otras tareas relacionadas con la gestión de alumbrado público.

Justificación:

- El pasante será capacitado para integrar una nueva funcionalidad a un software ya existente, enfocada en el área del laboratorio dentro del departamento técnico. Esta nueva característica permitirá el almacenamiento de ensayos que detallen el funcionamiento de las luminarias que próximamente serán instaladas en la ciudad., o que fueron extraídas para detectar fallos. Además, se encargará de capacitar al personal para su correcto funcionamiento y estará a disposición para brindar mantenimiento y actualizaciones en el software.

Objetivo General:

- Proporcionar al estudiante la oportunidad de una experiencia práctica mediante la prestación de un servicio de carácter competente a una Institución o Empresa para que desarrolle sus habilidades y destrezas, y conocimiento adquirido en el proceso de aprendizaje durante su carrera universitaria, para complementar su formación integral y profesional.

Objetivos Específicos:

- Proporcionar al estudiante la oportunidad de desarrollar una nueva funcionalidad en el sistema ya integrado, con el fin de permitir el registro de ensayos relacionados con las luminarias de alumbrado público.

Delimitaciones de la práctica profesional:

- La práctica profesional llevada a cabo como proyecto final se está desarrollando en la dirección técnica de alumbrado público de la municipalidad de Paraná, la cual tiene una duración de 1 año comenzando en septiembre del 2023 y concluyendo en agosto del 2024, durante este periodo, se llevan a cabo diversas tareas en un software ya existente, diseñado con el propósito de georeferenciar los puntos de luz de la ciudad de Paraná, generar informes, ordenes, entre otros.

  Las tareas realizadas incluyen actualizaciones, correcciones de errores y adiciones al sistema, incluyendo lo presentado en el informe. La practica fue supervisada por el programador general, creador del software, que proporciono orientación y retroalimentación durante todo el proceso.

Los recursos disponibles incluyen acceso a equipos de desarrollo, herramientas de diseño y documentación. El objetivo principal de la práctica es la mejora del software S.I.G.A.P para aumentar su utilidad y eficacia en el ámbito.

Este enfoque permitió una experiencia completa y significativa en la mejora de un sistema existente, contribuyendo al avance y la eficiencia en la dirección técnica de alumbrado público de la municipalidad de Paraná.

Sección Ensayos en software S.I.G.A.P:

- El ABM fue desarrollado con estructura modelo, vista y controlador.

  Para poder agregar un nuevo ensayo al sistema, es necesario completar los datos en el formulario que se muestra en la interfaz de usuario. Si la información es ingresada correctamente, desde el archivo jsp (servlet) se genera una solicitud la cual es enviada a la vista de ensayo (EnsayoControler) para que así pueda procesarla y mandarla a la base de datos.

```
<form action="ens" method="post">
    <%
        Ensayo e = (Ensayo)session.getAttribute("ensayo");
    %>
    <div class="container ensayosDiv"
```

En la vista de ensayos se corrobora que los datos ingresados sean correctos y se los setea a un objeto que fue creado vacio.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    destino = "admEnsayo.jsp";
    System.out.println("ENTRA AL DOPOST ensayos");

    if (request.getParameter("btnAgregar") != null) {

        System.out.println("ENTRO ACA");
        Ensayo e = new Ensayo();
        if (e != null) {
            try {
            System.out.println("objeto no nulo");
            String nomPro = request.getParameter("txbProcedencia");
            e.setNomProc(nomPro);
            String proc = request.getParameter("SelectProce");
            e.setProce(proc);
            String tipoLum = request.getParameter("txbTipoLum");
            e.setTipoLum(tipoLum);
            System.out.println(tipoLum);
            int potLum = Integer.parseInt(request.getParameter("txbpotLum").trim());
            e.setPotLum(potLum);
            System.out.println(potLum);
            String modelolum = request.getParameter("txbModLum");
            e.setModeloLum(modelolum);
            double fPot = Double.parseDouble(request.getParameter("txbFpot").trim());
            e.setfPot(fPot);
            String modelodrive = request.getParameter("txbModDrv");
            e.setModeloDriver(modelodrive);
            int potdriver = Integer.parseInt(request.getParameter("txbPotDrv").trim());
```

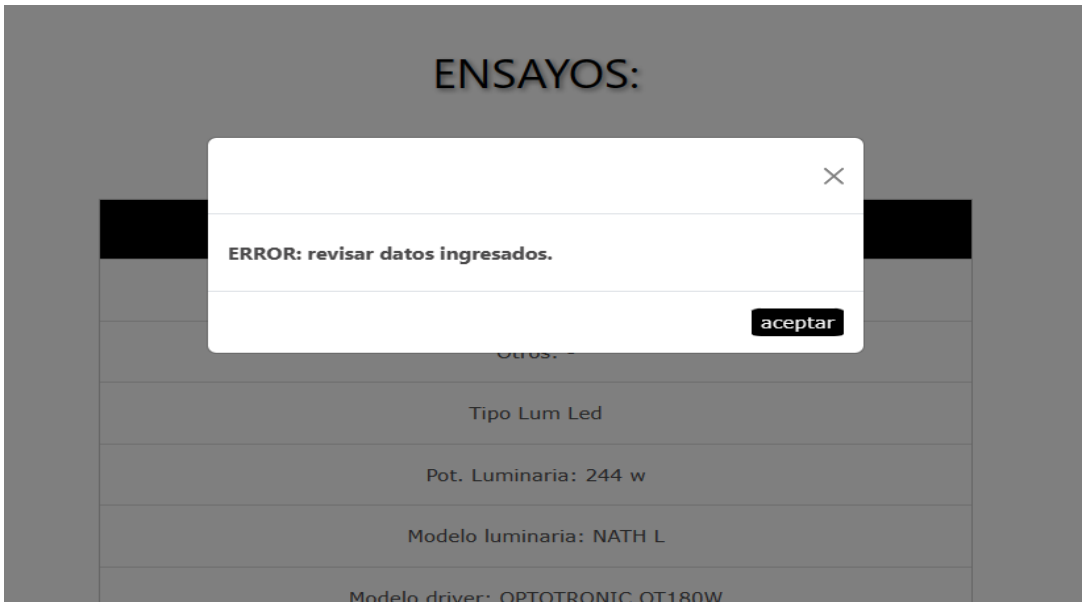
El objeto pasa por una validación y luego se asegura de que es un objeto nuevo, corroborando de que el id sea cero. Si el id no es cero pasa a ser un ensayo editado.

```
// VALIDACION
boolean valid = true;
System.out.println("VALIDACION");
try {
    if (request.getParameter("txbProcedencia").equals("")) {
        destino = "admEnsayo.jsp";
        valid = false;
        System.out.println("NO PASO LA VALIDACION----------------");

    } else {
        destino = "admEnsayo.jsp";
        System.out.println("PASO LA VALIDACION");
        System.out.println("ENTRO ACA Tambien");
    }
    try {
    if (valid == true) {

        if (request.getParameter("id").equals("0")) {

            EnsayoCR.setE(e);

            EnsayoCR.AgregarEnsayo(e);

        }else {
            e.setId(Integer.parseInt(request.getParameter("id")));

            EnsayoCR.setE(e);
            EnsayoCR.actualizarEnsayo(e);

        }
```

Si el objeto pasa la validación y es nuevo es enviado como parámetro hacia el controlador de ensayos (EnsayoCR), en el cual pasa por otra validación y lo agrega a la base de datos.

```
public static void AgregarEnsayo(Ensayo e) {
    VerificacionEnsayo();
    boolean verificar = true;
    EntityManager em = null;
    try {
        em = PersistenceManager.createEntityManager();
        em.getTransaction().begin();
        if (e != null) {
            for (Ensayo ens : ensayos) {
                if (ens.getId() == e.getId()) {
                    verificar = false;
                    SetError(1);
                }
            }
            if (verificar == true) {
                ensayos.add(e);
                em.persist(e);
                SetError(0);
            }
        }
        em.getTransaction().commit();
    }catch(org.hibernate.exception.DataException er) {
        SetError(3);
    }
    catch (PersistenceException exP) {
        SetError(2);
    }catch (Exception ex) {
        SetError(3);
        if (em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        } else {
            System.out.println(ex.getMessage());
            ex.printStackTrace();
        }
```

Cuando el objeto es agregado a la base de datos el destino de la página pasa a ser de AmbEnsayos.jsp el cual muestra los datos cargados. En caso de que el objeto no fue guardado correctamente, muestra los datos pero con el id en cero y una ventana emergente marcando el error.




Si quisiéramos editar o borrar un objeto de la base de datos, debemos realizar una búsqueda primero: Desde la interfaz filtramos los ensayos por su procedencia.

Al apretar el botón buscar enviamos una solicitud a la vista para realizar la búsqueda de ensayos, la cual devuelve una lista de tipos ensayos con los que fueron encontrados en la base de datos.

Desde la vista:

```
System.out.println("CODIGO BUSCAR ENSAYOS");
if (request.getParameter("btnbuscarEns") != null) {
    HttpSession session = request.getSession();
    List<Ensayo> ens = new ArrayList<Ensayo>();
    if (request.getParameter("selectBuscarEnsayo") != null && !request.getParameter("selectBuscarEnsayo").equals("")) {
        ens = EnsayoCR.BuscarEnsayos(request.getParameter("selectBuscarEnsayo"));
        session.setAttribute("BuscarEnsayos", ens);
    }
    destino = "BuscarEnsayos.jsp";
}
```

Desde el controlador:

```
public static List<Ensayo> BuscarEnsayos(String proc) {
    System.out.println("Buscar ensayos en EnsayoCR ");
    EntityManager em = null;
    List<Ensayo> ens = new ArrayList<Ensayo>();
    try {
        em = PersistenceManager.createEntityManager();
        em.getTransaction().begin();
        TypedQuery<Ensayo> qEnsayo = em.createQuery("select e from Ensayo e where e.proce like '%" + proc + "%'",
                Ensayo.class);
        ens = qEnsayo.getResultList();
        System.out.println(ens.size());
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
        ex.printStackTrace();
    } finally {
        if (em != null) {
            em.close();
        }
    }
    return ens;
}

public static void SetError(int error) {
    Errorr = error;

}
```

Lo recibe y maneja la lista: BuscarEnsayos.jsp:

```
<%
List<Ensayo> ens = new ArrayList<Ensayo>();

if (session.getAttribute("BuscarEnsayos") != null) {
    ens = (List<Ensayo>) session.getAttribute("BuscarEnsayos");
}
if (ens != null) {
%>
<table class="listado">
    <%
        for (Ensayo e : ens) {
    %>
<div class="w3-col m5"
        style="background: url(http://www.pixeden.com/media/k2/galleries/131/003-subtle-light-pattern-background-texture
        <ul class="w3-ul w3-border w3-hover-shadow">
            <li style="background-color: #000000;">
                <p style="color: white;">Ensayo</p>
            </li>
            <li class="w3-padding-16"><b>Procedencia:</b> <%=e.getNomProc()%></li>
            <li class="w3-padding-16"><b>Otros:</b> <%=e.getOtros()%></li>
            <li class="w3-padding-16"><b>Tipo Lum:</b> <%=e.getTipoLum()%></li>
            <li class="w3-padding-16"><b>Pot. Luminaria:</b> <%=e.getPotLum()%></li>
            <li class="w3-padding-16"><b>Modelo luminaria:</b> <%=e.getModeloLum()%></li>
            <li class="w3-padding-16"><b>Modelo driver:</b> <%=e.getModeloDriver()%></li>
            <li class="w3-padding-16"><b>Pot. Driver:</b> <%=e.getPotDriver()%></li>
            <li class="w3-padding-16"><b>Tension salida:</b> <%=e.getTenSalida()%></li>
            <li class="w3-padding-16"><b>Corriente salida:</b> <%=e.getCorrSalida()%></li>
            <li class="w3-padding-16"><b>Num. Placas:</b> <%=e.getNumPlacas()%></li>
            <li class="w3-padding-16"><b>Descargador:</b> <%=e.getDescargador()%></li>
```

Lo muestra así:


Acá podemos borrar o editar el ensayo seleccionado.

Para borrar un ensayo apretamos el icono de borrar el cual tiene la acción de borrar

```
<li class="w3-padding-16 icono"><b>Borrar</b> <a
    href="EnsayoControler?accion=borrar&id=<%=e.getId()%>"
    style="color: black;"><span class="fa fa-trash fa-2x"></span></a></li>
```

La cual manda una solicitud y ejecuta el método borrar en la vista.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    HttpSession session = request.getSession();
    accion = request.getParameter("accion");
    referer = request.getHeader("Referer");
    session.setAttribute("PagAnterior", referer);
    destino = "admEnsayo.jsp";
    session.setAttribute("PagAnterior", referer);
    if (accion == null) {
        response.sendError(500, "No se encuentra la accion");
        return;
    }
    if (accion.equals("agregar")) {
        agregar(request, response);
    }
    if (accion.equals("cancelar")) {
        cancelar(request, response);
    }
    if (accion.equals("editar")) {
        editar(request, response);
    }
    if (accion.equals("borrar")) {
        borrar(request, response);
    }
    if (accion.equals("BEnsayos")) {
        BEnsayos(request, response);
    }
    response.sendRedirect(destino);
```

El método llama al controlador y setea el destino.

```
private void borrar(HttpServletRequest request, HttpServletResponse response) {
    String strId = request.getParameter("id");
    int id = Integer.parseInt(strId);
    EnsayoCR.EliminarEnsayo(id);
    destino = "BuscarEnsayos.jsp";
}
```

Desde el controlador:

```
public static void EliminarEnsayo(int id) {
    EntityManager em = null;
    List<Ensayo> ens = new ArrayList<Ensayo>();
    try {
        em = PersistenceManager.createEntityManager();
        em.getTransaction().begin();
        TypedQuery<Ensayo> qEnsayos = em.createQuery("select e from Ensayo e where e.id = " + id + "",
                Ensayo.class);
        ens = qEnsayos.getResultList();
        em.remove(ens.get(0));
        em.getTransaction().commit();
    } catch (Exception ex) {
        if (em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        } else {
            System.out.println(ex.getMessage());
            ex.printStackTrace();
        }

    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

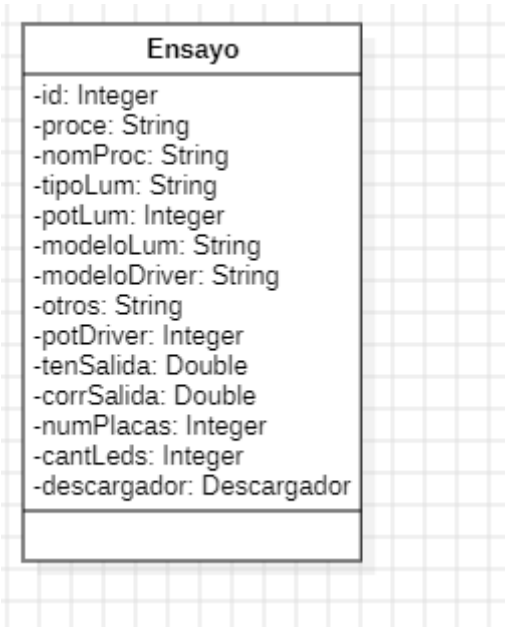
Si queremos editar el ensayo envía la misma solicitud con la acción de editar la cual setea el destino en el mismo jsp que usamos para agregar un ensayo nuevo, pero este se carga con el id del objeto y no en cero, por lo cual pasa por el método del controlador que actualiza la base de datos:

```
public static void actualizarEnsayo(Ensayo e) {
    EntityManager em = null;
    try {
        em = PersistenceManager.createEntityManager();
        em.getTransaction().begin();
        em.merge(e);
        em.getTransaction().commit();
    } catch (Exception ex) {
        if (em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        } else {
            System.out.println(ex.getMessage());
            ex.printStackTrace();
        }

    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

Programas y arquitecturas utilizados para el desarrollo:

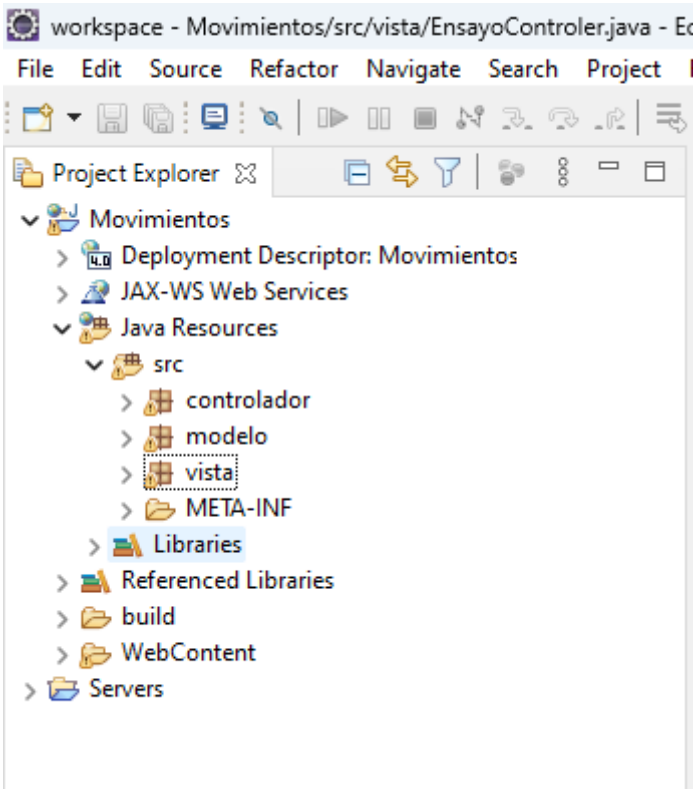
1. **Diagrama de Clases:** UML modelado de datos.
2. **Eclipse:** entorno de desarrollo integrado.
3. **Mvc**: modelo para que la aplicación sea escalable y sostenible.
4. **Java:** como el lenguaje principal de la aplicación.
5. **Hmtl, Css, Javascript:** para desarrollar la vista con las JSP.
6. **Servlets:** vistas.
7. **Jsp:** Java Server Pages.
8. **Apache Tomcat9.0:** contenedor web.
9. **Hibernate:** mapeador objeto relacional.
10. **JPA:** con Hibernate.
11. **PostgreSQL:** conexión con hibernate (base de datos relacional).
12. **pgAdmin 4:** gestión base de datos.

**Diagrama de clases uml:**



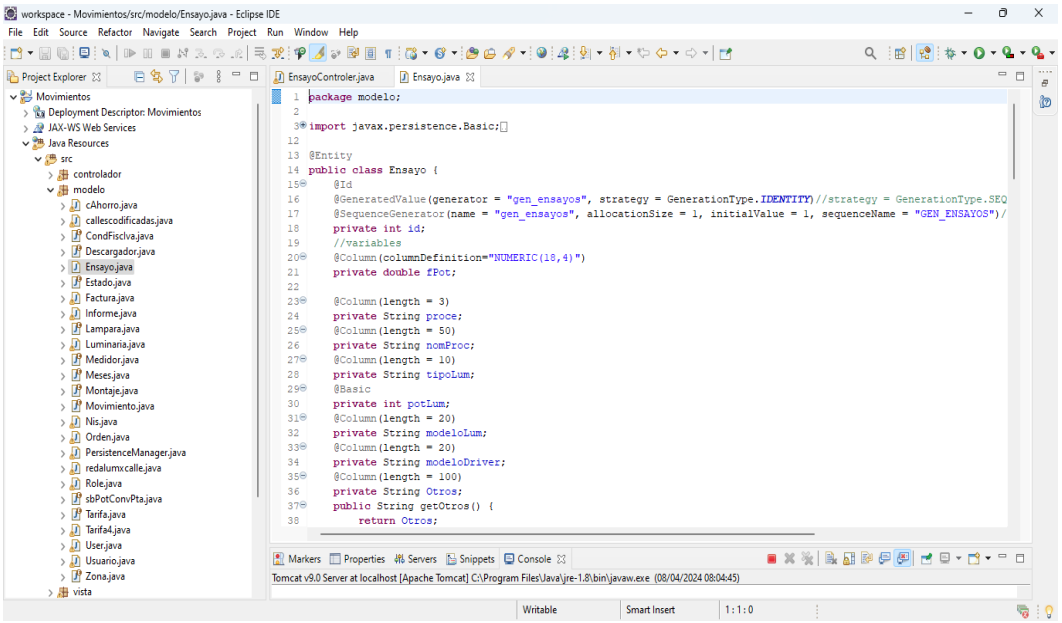

**ERD con pgAdmin:**

**Eclipse:** es el IDE más usado para el desarrollo en Java. Es de código abierto y gratuito. Eclipse, permite extender sus funciones, mediante el desarrollo de plugins. Existen muchas versiones de Eclipse para programar en diferentes lenguajes e incluso muchas versiones para Java.

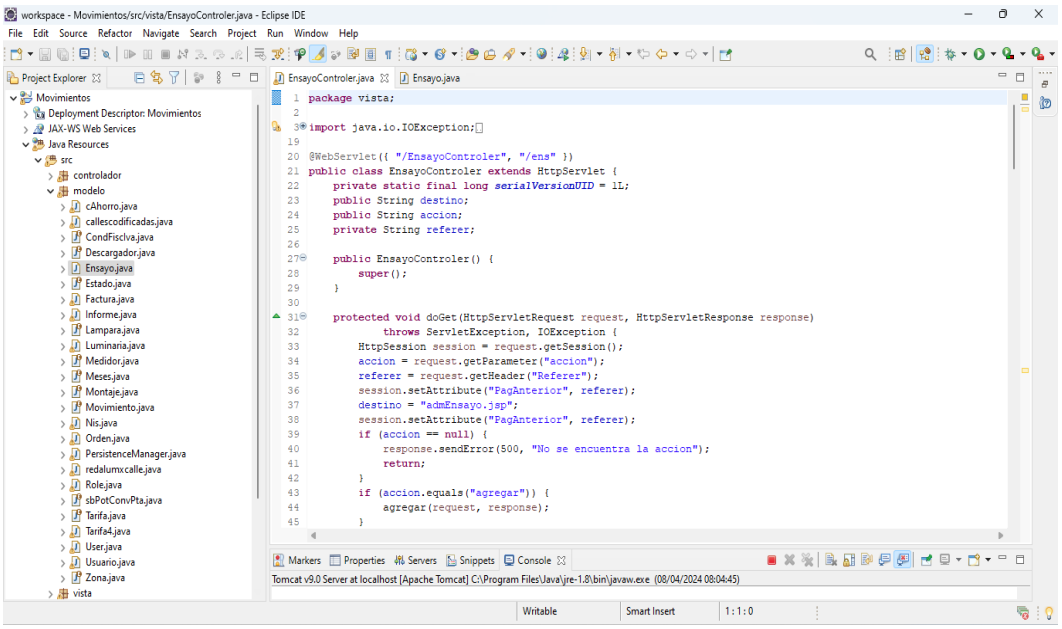


**Mvc:** Modelo-Vista-Controlador.

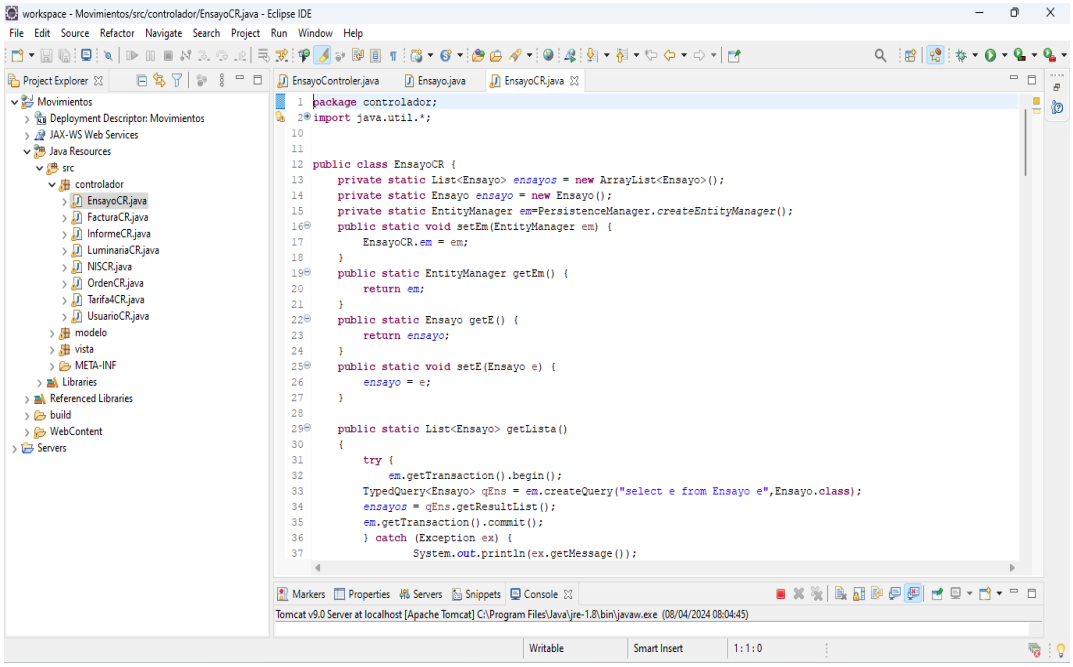
**Modelo**: clase raíz.




**Vista**: Presenta el modelo en un formato adecuado para interactuar



**Controlador:** hace de intermediario entre la ‘vista’ y el ‘modelo’.




**Java:** es una plataforma informática de lenguaje de programación

**HTML, CSS y Javascript:**

**HTML:** busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado. Nos permite indicar la estructura de nuestro documento mediante etiquetas y nos ofrece una gran adaptabilidad y es fácil de interpretar tanto por humanos como por máquinas.

**CSS:** Está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas o layouts, los colores y las fuentes.

**JavaScript:** es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico

**Servlets/Vistas:** Un servlet es un objeto en ejecución en un servidor que se encarga de interceptar peticiones enviadas por un cliente y generar una respuesta. El tipo de servlet más popular es HttpServlet que se encarga de los métodos HTTP como GET y POST, entre otros. Son clases java que se ejecutan en un contenedor de Servlets (Servlet container), implementan la interfaz Servlet. Los que usaremos nosotros son descendientes de HttpServlet. JSP (Java Server Pages) se traducen en un Servlet que es generado y compilado por el contenedor.

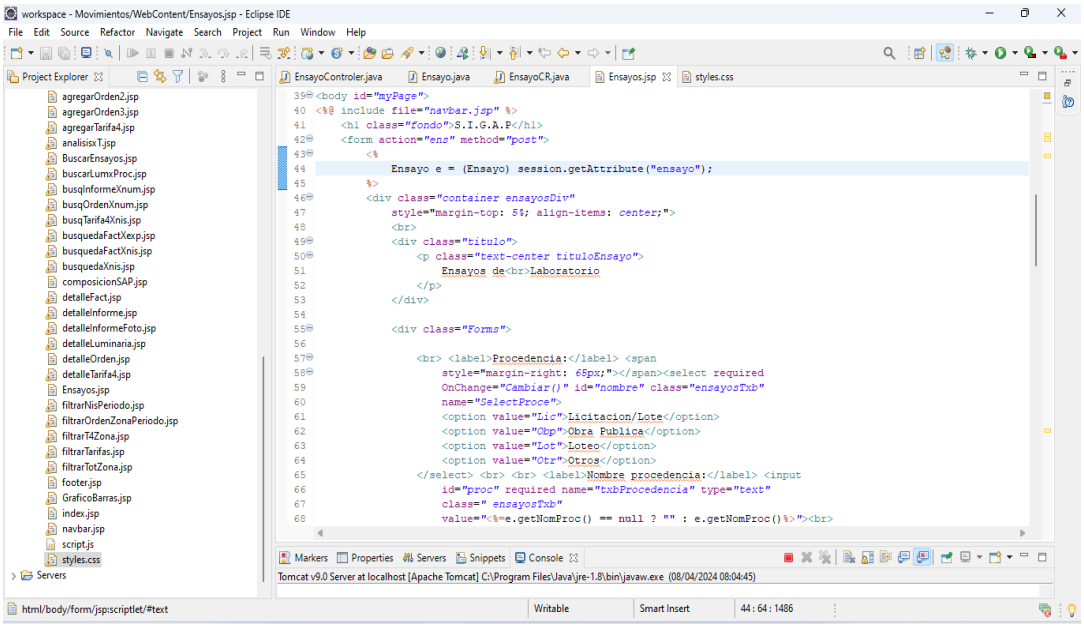
```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    destino = "admEnsayo.jsp";
    System.out.println("ENTRA AL DOPOST ensayos");
```

**JSP:** Java Server Pages

Para desplegar y correr JavaServerPages, se requiere un servidor web compatible con contenedores servlet como Apache Tomcat.

Los servlets y Java Server Pages (JSPs) son dos métodos de creación de páginas web dinámicas en servidor usando el lenguaje Java. En ese sentido son similares a otros métodos o lenguajes tales como el PHP, ASP o los CGIs, programas que generan páginas web en el servidor.

JSPs y servlets se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él.




**Apache TomCat:**

Es un contenedor web con soporte de servlets y JSPs; no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

**Hibernate:** Mapeo Objeto/Relacional

El mapeo relacional de objetos (ORM) es una técnica de mapeo de dichos objetos y sus atributos en la base de datos a través de mapeadores relacionales de objetos. Esta técnica también nos ayuda a convertir datos entre sistemas incompatibles utilizando aplicaciones de programación orientadas a objetos.

La clase a mapear se anota con @Entity

El atributo identificador de la clase se anota con @Id

Opciones:

Nombre de la tabla: @Table a nivel de clase

- Nombre/propiedades de los atributos: @Column

- Generadores: @GeneratedValue, @XXXGenerator

```
@Entity
public class Ensayo {
    @Id
    @GeneratedValue(generator = "gen_ensayos", strategy = GenerationType.IDENTITY)//strategy = GenerationType.SEQ
    @SequenceGenerator(name = "gen_ensayos", allocationSize = 0, initialValue = 1, sequenceName = "GEN_ENSAYOS")/
    private int id;
    //variables
    @Column(columnDefinition="NUMERIC(18,4)")
    private double fPot;

    @Column(length = 3)
    private String proce;
    @Column(length = 50)
    private String nomProc;
    @Column(length = 10)
    private String tipoLum;
    @Basic
    private int potLum;
    @Column(length = 20)
    private String modeloLum;
    @Column(length = 20)
    private String modeloDriver;
    @Column(length = 100)
    private String Otros;
    public String getOtros() {
        return Otros;
```

**Java Persistence API (JPA):**

JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB).

**API (Application programming interface):**

Biblioteca para ser utilizada por otro software como una capa de abstracción.

Por ejemplo para encontrar un objeto en la BD y cargarlo al Contexto de persistencia, se usa el método find del EntityManagerobj = em.find(); Find devuelve la instancia o null si no la pudo encontrar.

Recuperar un conjunto de objetos: Hibernate provee un lenguaje completo de consultas HQL o HibernateQueryLanguage.

```
public static void AgregarEnsayo(Ensayo e) {
    VerificacionEnsayo();
    boolean verificar = true;
    EntityManager em = null;
    try {
        em = PersistenceManager.createEntityManager();
        em.getTransaction().begin();
        if (e != null) {
            for (Ensayo ens : ensayos) {
                if (ens.getId() == e.getId()) {
                    verificar = false;
                    SetError(1);
                }
            }
            if (verificar == true) {
                ensayos.add(e);
                em.persist(e);
                SetError(0);
            }
        }
        em.getTransaction().commit();
    } catch (PersistenceException exP) {
        SetError(2);
    } catch (Exception ex) {
        if (em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        } else {
            System.out.println(ex.getMessage());
            ex.printStackTrace();
```

**JPA CON HIBERNATE:**

**Entity Manager Factory:**

El EntityManagerFactory es la clase que se encarga de abrir la conexión a la base de datos y pone a nuestra disposición los distintos “manejadores de entidades” que usemos

Un EntityManager está ligado habitualmente aunque no siempre (Extended EntityManagers) a una transacción y tiene de vida lo que dura esa transacción.

Desde la clase persistence manager se crea una fábrica de manejadores de entidad, a partir de esto, cada vez que se necesita una instancia de el manejador de persistencia, se llama a la función createEntityManager().

```
public class PersistenceManager {
   protected static PersistenceManager me = null;
   // private public EntityManagerFactory emf = null;
   private static EntityManagerFactory emf = null;

   public static EntityManager createEntityManager() {
       try {
      if(emf == null) {
         emf = Persistence.createEntityManagerFactory("movimientos");
      }
       }catch(Exception e){
           System.out.println(e.getMessage());
       }
       return emf.createEntityManager();
   }
```

**PostgreSQL:**

es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto

Es una base de datos 100% ACID (Atomicity, Consistency, Isolation, Durability).

Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP...), cadenas de bits, etc.

**pgAdmin 4:**

4 nos permite acceder a todas las funcionalidades de la base de datos, consulta, manipulación y gestión de datos y puede ser ejecutado en múltiples plataformas

Su interfaz gráfica soporta todas las características de PostgreSQL y facilita de gran manera la administración ya que nos permite desde hacer búsquedas SQL hasta desarrollar toda nuestra base de datos de forma muy fácil e intuitiva: directamente desde la interfaz gráfica.

Conclusiones:

A continuación mencionaremos las conclusiones generales:

- Esta adición brinda al software una nueva herramienta destinada al laboratorio de la dirección técnica el cual no había sido integrada hasta ahora.
- Ayuda a los integrantes del laboratorio a tener un historial de ensayos y facilitar su búsqueda.
- Los objetivos planteados al inicio de la práctica profesional se han cumplido satisfactoriamente, brindando al estudiante una experiencia práctica en el desarrollo de la extensión al software y permitiéndole aplicar los conocimientos adquiridos durante su formación académica.

En resumen, este proyecto ha demostrado cómo la implementación de nuevas funcionalidades puede mejorar la gestión de la información técnica para el área.