PARCIAL 2

Integrantes:

ANDRES FELIPE RINCON SANCHEZ – 20221978013

SEBASTIAN CAMILO HORTUA ABRIL - 20221978010

DAYANNA ALEJANDRA CLAVIJO PEREZ – 20212578134

Docente:

Noe Arcos Muñoz

Universidad Distrital Francisco José De Caldas

Tecnología En Sistematización De Datos

Programación Avanzada 303

Bogotá D.C. 2023

**CONTROLADOR.SERVLET**

**1.Controlador:** El método processRequest maneja las solicitudes HTTP GET y POST. Analiza los parámetros "menu" y "accion" para determinar la acción a realizar y redirige la solicitud en consecuencia.  
En el manejo de entidades, cada sección (Empleado, Cliente, Producto, NuevaVenta) realiza operaciones específicas utilizando las clases y (DAOS) asociados. Estas operaciones abarcan la lista de entidades, así como la capacidad de agregar, editar, actualizar y eliminar registros. La sección "NuevaVenta" maneja el proceso de venta. Busca clientes y productos, agrega productos a una lista, calcula el total a pagar y genera la venta. Se utiliza un objeto GenerarSerie para gestionar la numeración de series.

Los métodos doGet, doPost y getServletInfo son parte de la implementación de servlet. doGet y doPost llaman a processRequest para manejar las solicitudes HTTP, mientras que getServletInfo proporciona información sobre el servlet.

**2. Validar:** Maneja la autenticación de usuarios, verificando las credenciales ingresadas y redirigiendo a la página principal si son válidas, o volviendo a la página de inicio de sesión si no lo son.

En el método **processRequest**, el servlet recibe solicitudes HTTP GET y POST. Extrae el parámetro "accion" para determinar si la acción es "Ingresar". Si es así, obtiene los parámetros "txtuser" y "txtpass" que representan el nombre de usuario y la contraseña ingresados por el usuario.

Utiliza un objeto **EmpleadoDAO** para validar las credenciales llamando al método **validar** con el nombre de usuario y la contraseña proporcionados. Si la validación es exitosa (si **em.getUser()** no es nulo), se establece un atributo "usuario" en el objeto de solicitud y se redirige a la página principal mediante **Controlador?menu=Principal**. Si la validación falla, se redirige de nuevo a la página de inicio de sesión **(index.jsp).**Principio del formulario

**MODELO.JAVA**

**1. Cliente:** La clase "ClienteDAO" se ha diseñado para gestionar la conexión y las operaciones relacionadas con la base de datos de los clientes. La aplicación debe de gestionar la información de clientes, esta clase es crucial para realizar operaciones como la búsqueda, inserción, actualización y eliminación de registros de clientes en la base de datos.

La clase utiliza la clase "Conexion" para establecer una conexión a la base de datos. Los atributos como "cone" (para la conexión), "ps" (para sentencias preparadas), "rs" (para el conjunto de resultados) y "r" (para el resultado de algunas operaciones) se utilizan para realizar y gestionar consultas SQL.

**Método "buscar":** Este método busca un cliente en la base de datos según su número de identificación (DNI). La información del cliente se recupera de la base de datos y se almacena en un objeto de tipo "Cliente". Este método es útil para obtener detalles específicos de un cliente en particular.

**Método "Listar":** Este método recupera la lista completa de clientes almacenados en la base de datos. Crea una lista de objetos de tipo "Cliente" y los llena con la información de cada cliente recuperada de la base de datos. Es útil para mostrar la lista completa de clientes en la interfaz de usuario.

**Método "Agregar":** Este método se utiliza para agregar un nuevo cliente a la base de datos. Toma un objeto de tipo "Cliente" como parámetro, y los datos del nuevo cliente se insertan en la base de datos mediante una sentencia SQL de inserción.

**Método "listarId":** Recupera la información de un cliente específico según su identificador único (IdCliente) en la base de datos y la almacena en un objeto de tipo "Cliente". Es útil cuando se necesita obtener información detallada de un cliente en particular.

**Método "actualizar":** Actualiza la información de un cliente en la base de datos. Toma un objeto de tipo "Cliente" como parámetro y realiza una actualización en la base de datos utilizando una sentencia SQL de actualización.

**Método "delete":** Elimina un cliente de la base de datos según su identificador único (IdCliente). Realiza una operación de eliminación utilizando una sentencia SQL de eliminación.

**2.** **Empleado:** La clase "EmpleadoDAO" se ha creado para gestionar las operaciones relacionadas con la base de datos de empleados. En la aplicación implica la gestión de información de empleados, esta clase es esencial para realizar operaciones como la validación de inicio de sesión, búsqueda, inserción, actualización y eliminación de registros de empleados en la base de datos.

**Método "validar":** Este método se utiliza para validar las credenciales de un empleado durante el inicio de sesión. Toma un nombre de usuario (user) y un número de identificación (DNI) como parámetros, y realiza una consulta SQL para verificar la existencia de un empleado con esas credenciales. Si se encuentra, se crea un objeto de tipo "Empleado" con la información del empleado validado.

**Método "Listar":** Este método recupera la lista completa de empleados almacenados en la base de datos. Crea una lista de objetos de tipo "Empleado" y los llena con la información de cada empleado recuperada de la base de datos. Es útil para mostrar la lista completa de empleados en la interfaz de usuario.

**Método "Agregar":** Este método se utiliza para agregar un nuevo empleado a la base de datos. Toma un objeto de tipo "Empleado" como parámetro, y los datos del nuevo empleado se insertan en la base de datos mediante una sentencia SQL de inserción.

**Método "listarId":** Recupera la información de un empleado específico según su identificador único (IdEmpleado) en la base de datos y la almacena en un objeto de tipo "Empleado". Es útil cuando se necesita obtener información detallada de un empleado en particular.

**Método "actualizar":** Actualiza la información de un empleado en la base de datos. Toma un objeto de tipo "Empleado" como parámetro y realiza una actualización en la base de datos utilizando una sentencia SQL de actualización.

**Método "delete":** Elimina un empleado de la base de datos según su identificador único (IdEmpleado). Realiza una operación de eliminación utilizando una sentencia SQL de eliminación.

**3. Producto:** La clase "ProductoDAO" se ha desarrollado para gestionar las operaciones relacionadas con la base de datos de productos. En la aplicación involucra la gestión de información de productos, esta clase es esencial para realizar operaciones como la búsqueda, actualización, inserción, y eliminación de registros de productos en la base de datos.

**Método "buscar":** Este método busca un producto en la base de datos según su identificador único (idproducto). La información del producto se recupera de la base de datos y se almacena en un objeto de tipo "Producto". Es útil para obtener detalles específicos de un producto en particular.

**Método "actualizarstock"**: Actualiza el stock de un producto en la base de datos. Toma el identificador único del producto (idproducto) y la nueva cantidad de stock como parámetros, y realiza una actualización en la base de datos utilizando una sentencia SQL de actualización.

**Método "Listar":** Recupera la lista completa de productos almacenados en la base de datos. Crea una lista de objetos de tipo "Producto" y los llena con la información de cada producto recuperada de la base de datos. Es útil para mostrar la lista completa de productos en la interfaz de usuario.

**Método "agregar":** Este método se utiliza para agregar un nuevo producto a la base de datos. Toma un objeto de tipo "Producto" como parámetro, y los datos del nuevo producto se insertan en la base de datos mediante una sentencia SQL de inserción.

**Método "listarId":** Recupera la información de un producto específico según su identificador único (IdProducto) en la base de datos y la almacena en un objeto de tipo "Producto". Es útil cuando se necesita obtener información detallada de un producto en particular.

**Método "actualizar":** Actualiza la información de un producto en la base de datos. Toma un objeto de tipo "Producto" como parámetro y realiza una actualización en la base de datos utilizando una sentencia SQL de actualización.

**Método "delete":** Elimina un producto de la base de datos según su identificador único (IdProducto). Realiza una operación de eliminación utilizando una sentencia SQL de eliminación.

**4. Ventas:** La clase "VentaDAO" esta clase es fundamental para realizar operaciones como la generación de números de serie, obtención de identificadores de ventas, y la inserción de información relacionada con las ventas y sus detalles en la base de datos.

**Método "GenerarSerie":** Este método se utiliza para generar el número de serie para una nueva venta. Realiza una consulta SQL para obtener el número de serie máximo de las ventas existentes y luego lo utiliza para generar el nuevo número de serie.

**Método "IdVentas":** Retorna el identificador único máximo de las ventas existentes. Realiza una consulta SQL para obtener el máximo valor de la columna "IdVentas" en la tabla de ventas.

**Método "guardarVenta":** Se encarga de guardar la información principal de una venta en la base de datos. Toma un objeto de tipo "Venta" como parámetro y utiliza una sentencia SQL de inserción para agregar la información en la tabla de ventas.

**Método "guardarDetalleventas":** Guarda los detalles de una venta en la base de datos. Toma un objeto de tipo "Venta" que contiene información detallada de la venta, como el identificador de la venta, el identificador del producto, la cantidad y el precio de venta. Utiliza una sentencia SQL de inserción para agregar esta información en la tabla de detalle\_ventas.

**CONFIG**

**1. Conexión**: La clase "Conexion" proporciona una conexión a la base de datos utilizando JDBC para acceder a una base de datos MySQL.

Método "ConexionMethod": Este método es responsable de establecer la conexión a la base de datos. Se utiliza **Class.forName** para cargar el controlador JDBC para interactuar con MySQL.

Se utiliza **DriverManager.getConnection** para establecer la conexión con la base de datos utilizando la URL, nombre de usuario y contraseña proporcionados. Se capturan excepciones de tipo **ClassNotFoundException y SQLException** para manejar posibles errores de conexión.

**2.** **GenerarSerie:** La clase "GenerarSerie" se ha diseñado para generar números de serie basados en un dato proporcionado, tiene atributos los cuales son dato el cual se encarga de generar el número de serie y número es una cadena que almacena el umero de serie

Método "NumeroSerie": Este método toma un parámetro dato y genera un número de serie en función de ese dato e incrementa el dato en 1, también utiliza condicionales para determinar la longitud del número de serie y agregar ceros adicionales en consecuencia, de esta manera devuelve la cadena del número de serie generado.

**VISTAS**

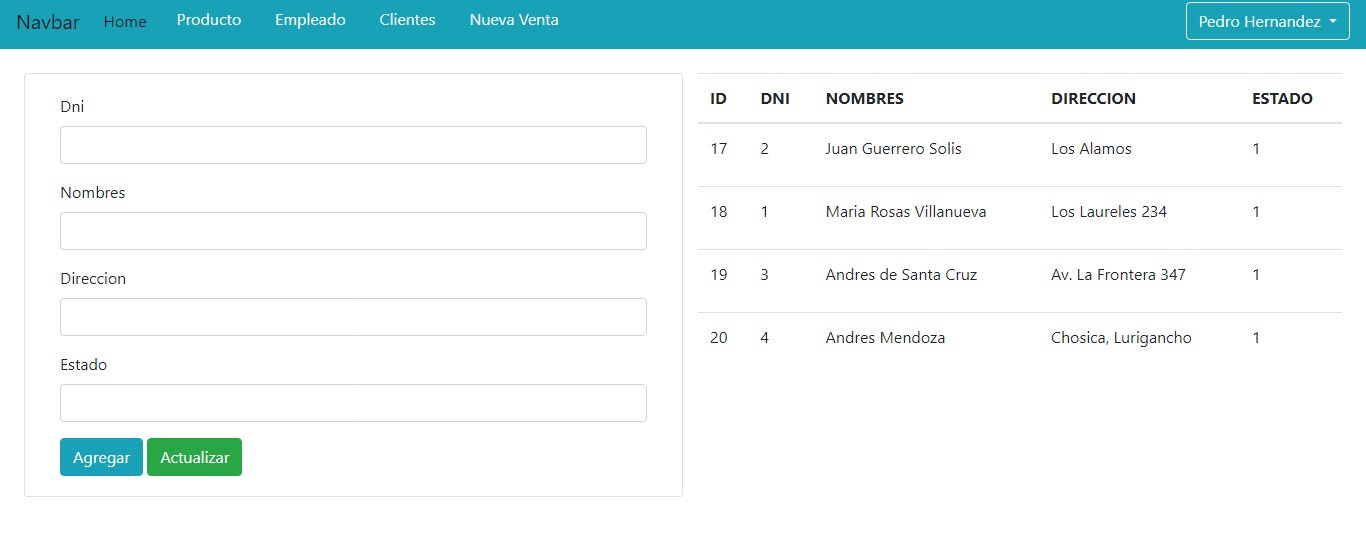
**1. Clientes jsp:** La interfaz de usuario permite la visualización, edición y eliminación de registros. La comunicación con el servidor se realiza a través de los enlaces en el formulario y la URL de los enlaces de acción en la tabla.

Se incluyen enlaces a Bootstrap, una biblioteca popular para el diseño y estilo de interfaces web. La página presenta un formulario que permite agregar y actualizar información de clientes. Los campos del formulario, como DNI, nombres, dirección y estado, se obtiene sus valores del objeto cliente mediante expresiones (${cliente.getDni()}).

El formulario incorpora botones de tipo "submit" para realizar acciones como agregar y actualizar clientes. Así mismo, se incluye una tabla que muestra la lista de clientes, utilizando JSTL para iterar sobre la lista (${clientes}) y mostrar detalles como ID, DNI, nombres, dirección y estado.

Las acciones de "Editar" y "Eliminar" se ejecutan mediante enlaces que incorporan parámetros en la URL (por ejemplo, id=${cli.getId()}) para identificar al cliente correspondiente.

Finalmente, al final del cuerpo de la página, se añaden enlaces a Bootstrap y las bibliotecas de JavaScript (jQuery). Estos enlaces contribuyen a la apariencia y funcionalidad general de la interfaz de usuario.



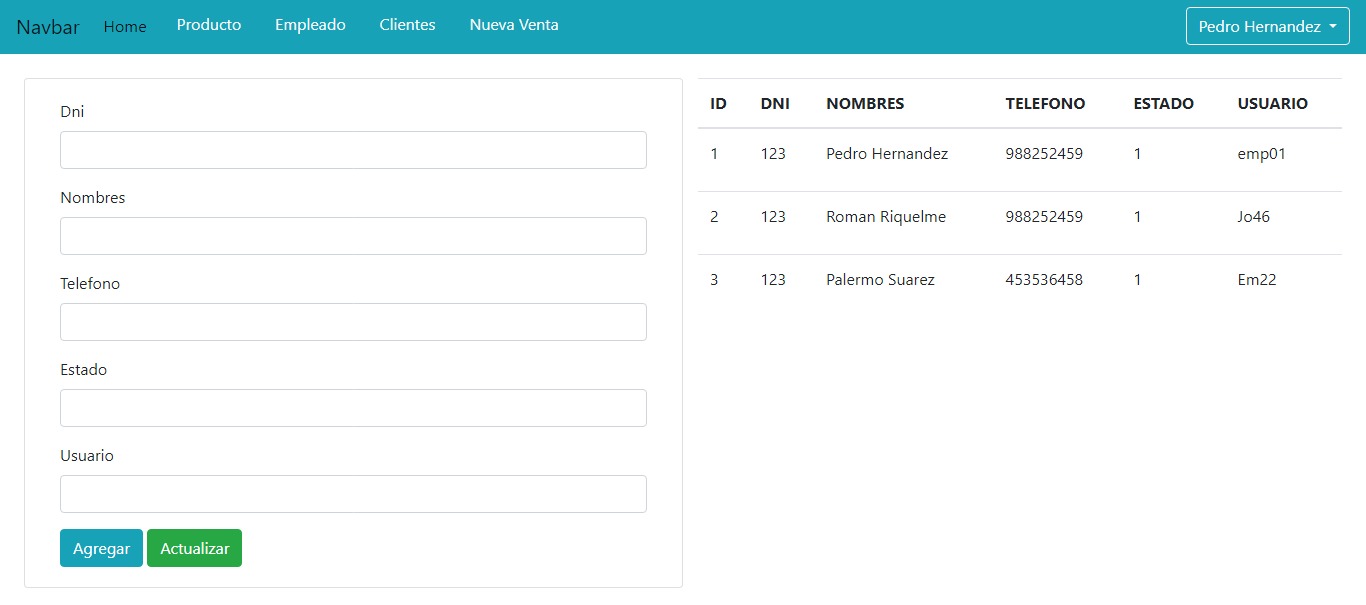
**2. Empleado jsp:** La interfaz de empleado realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar). La página permite agregar nuevos empleados, actualizar la información existente y ver la lista completa de empleados con opciones para editar o eliminar cada registro.

El cuerpo de la página se compone de dos secciones principales. La primera sección es un formulario que permite agregar y actualizar información de empleados. Utiliza campos de entrada para el Dni, nombres, teléfono, estado y usuario. Los valores se obtienen del objeto empleado mediante expresiones (${empleado.get...()}).

El formulario incluye botones de tipo "submit" para realizar acciones como agregar y actualizar empleados.

La segunda sección presenta una tabla que muestra la lista de empleados. Utiliza JSTL para iterar sobre la lista (${empleados}) y muestra detalles como ID, DNI, nombres, teléfono, estado y usuario. La tabla también incluye enlaces para "Editar" y "Eliminar" empleados, con parámetros en la URL para identificar al empleado correspondiente.

Al final del cuerpo de la página, se incluyen enlaces a las bibliotecas de Bootstrap y jQuery para mejorar el diseño y la funcionalidad de la interfaz de usuario.

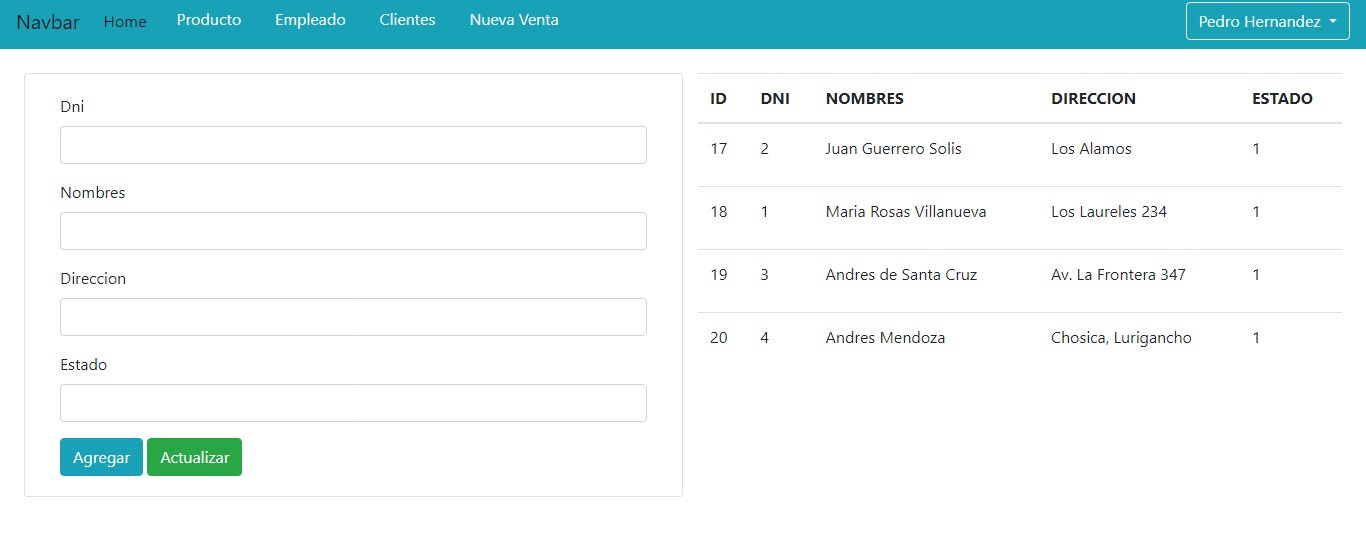


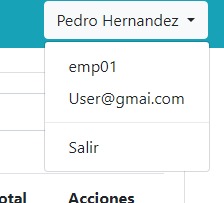
**3. Principal jsp:** En la barra de navegación, se observan enlaces intuitivos, como "Home", "Producto", "Empleado", "Clientes" y "Nueva Venta". Estos enlaces están diseñados para actualizar dinámicamente un iframe denominado "myFrame", permitiendo una navegación fluida entre las diferentes secciones de la aplicación.

Además, la barra de usuario exhibe el nombre del usuario en un botón desplegable. Al hacer clic en este botón, se despliega un menú que muestra información adicional del usuario, como el nombre de usuario y una dirección de correo electrónico. Se incluye la opción "Salir" para cerrar sesión, implementada mediante un formulario POST hacia el controlador.

La disposición del contenido dinámico se logra mediante el uso de un elemento **<iframe>** llamado "myFrame". Este enfoque posibilita la carga de contenido de manera dinámica sin necesidad de recargar toda la página.

Cabe destacar que se han integrado enlaces a las bibliotecas de Bootstrap y jQuery al final de la página, asegurando una presentación consistente y una funcionalidad mejorada gracias a estas herramientas.

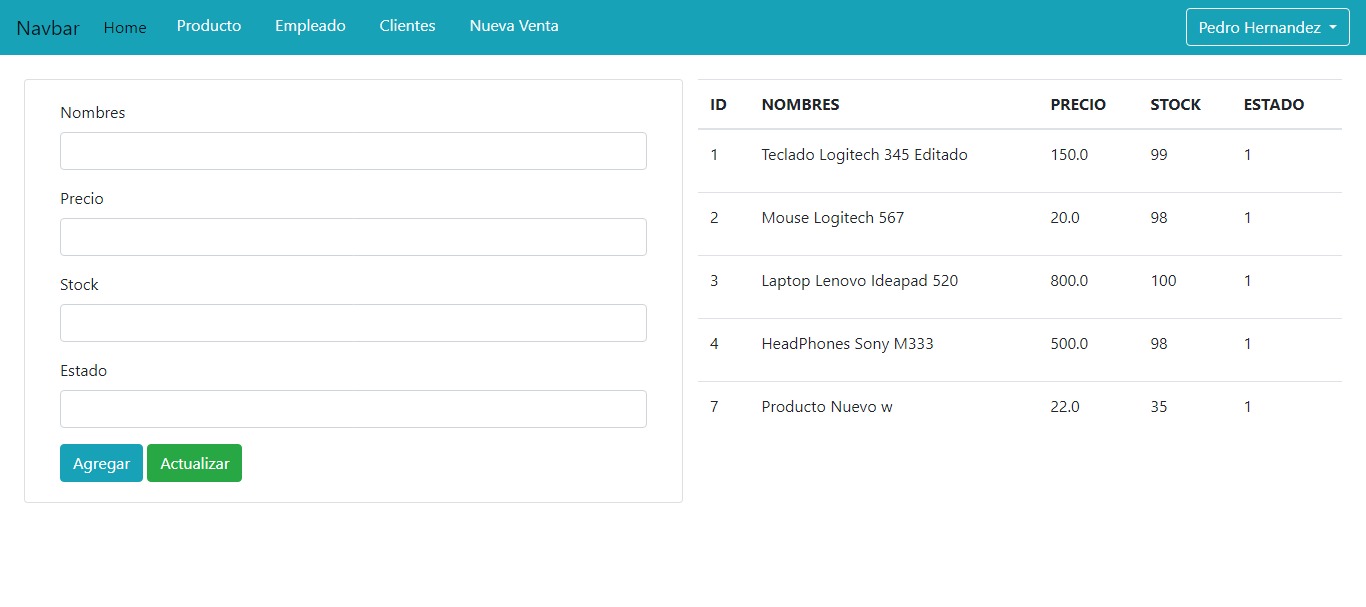




**4. Producto jsp:** Se incorporan enlaces a la biblioteca Bootstrap para mejorar el diseño de la interfaz. El formulario de gestión de productos permite agregar y actualizar información, utilizando campos de entrada que se llenan con los valores del objeto producto los cuales son nombre, precio, strock y estado a través de expresiones **(${producto.get...()}).** Los botones "submit" ejecutan acciones de agregar y actualizar.

La visualización de la lista de productos se realiza mediante una tabla, donde la etiqueta JSTL **<c:forEach>** itera sobre la lista de productos **(${productos}),** mostrando información como ID, nombres, precio, stock y estado. Las opciones de "Editar" y "Eliminar" se ejecutan mediante enlaces con parámetros en la URL, como **id=${pro.getId()},** para identificar el producto correspondiente.

Al final del cuerpo de la página, se incluyen enlaces a las bibliotecas de Bootstrap y jQuery para mejorar la apariencia y la funcionalidad general de la interfaz. En resumen, la página ofrece una interfaz efectiva y estéticamente agradable para la gestión de productos, aprovechando tecnologías como JSTL y Bootstrap.



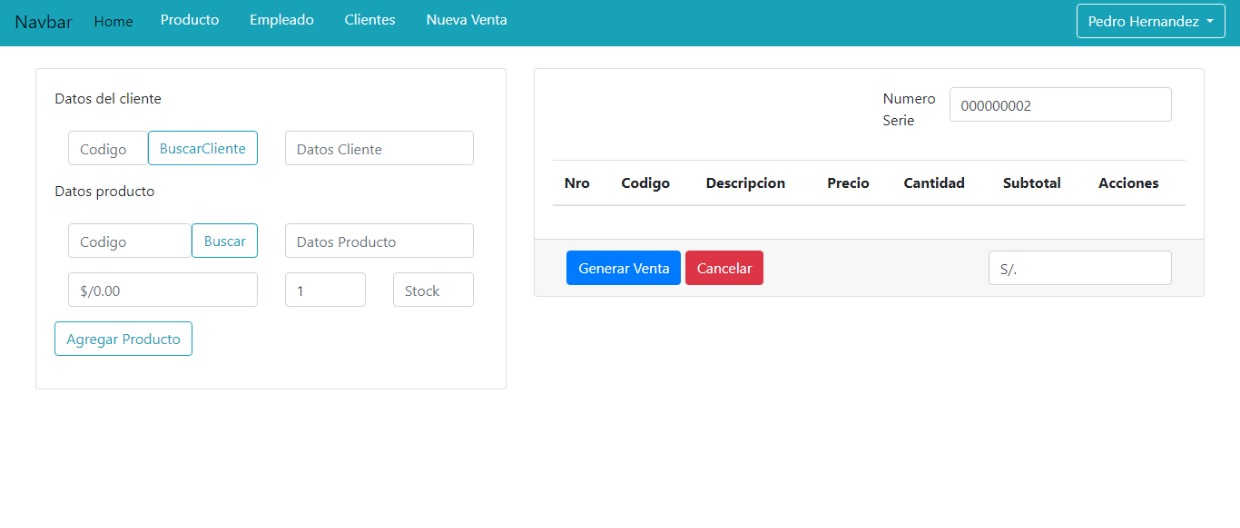
**5. RegistrarVenta.jsp**

La página incorpora un estilo de impresión (`@media print`) que oculta ciertos elementos al imprimir. Luego, se presenta un formulario dividido en dos secciones. La primera parte permite buscar y seleccionar un cliente, así como buscar y agregar productos al carrito de ventas.

El formulario incluye campos que se llenan automáticamente con información del cliente y del producto, utilizando expresiones EL (`${cli.getDni()}`, `${producto.getId()}`, etc.). Se proporcionan botones para buscar clientes y productos, y para agregar productos al carrito.

La segunda parte de la página muestra un resumen de la venta en un formato de tabla. Utiliza la etiqueta JSTL `<c:forEach>` para iterar sobre la lista de productos en el carrito (`${lista}`) y mostrar detalles como el código, descripción, precio, cantidad, subtotal y acciones de edición y eliminación. Se incluye un campo para el número de serie y un área para el total a pagar. Al final de la página, se encuentran enlaces a las bibliotecas Bootstrap y jQuery para mejorar la presentación y la funcionalidad general.

En el pie de página, se presenta un área para generar la venta, imprimir y cancelar la transacción, junto con la visualización del monto total a pagar. En resumen, la página proporciona una interfaz intuitiva y completa para el proceso de venta, aprovechando la funcionalidad de JSTL y el diseño de Bootstrap.



**6. Index.jsp**: La sección principal del contenido presenta una disposición centrada y adaptada, lograda mediante el uso de tarjetas (**card**) en Bootstrap. El formulario, identificado como **form-sign,** utiliza el método POST y está configurado para enviar datos al servlet "Validar" al hacer clic en el botón "Ingresar".

Dentro del formulario, se encuentran campos para el nombre de usuario **(txtuser)** y la contraseña **(txtpass).** El campo de contraseña utiliza el tipo "password" para ocultar los caracteres ingresados, mejorando la seguridad de la información sensible.

Además, se proporciona un título y una etiqueta de bienvenida para mejorar la presentación y la claridad del propósito del formulario.

