PRACTICA FINAL

ElReyDelPc

Patrones Software

Realizado por:

Alberto González Martínez 09072311F

Mario Adán Herrero 49226342V

Índice

1. Introducción

ElReyDelPC, es una empresa que se dedica a vender artículos relacionados con la tecnología y la informática, en concreto, ordenadores, portátiles, además de gadgets y demás productos tecnológicos.
Esta aplicación está desarrollada mediante Java y tiene una interfaz sencilla que permite que tanto los empleados como los clientes tengan una buena experiencia a la hora de usarla. La interfaz de los empleados es diferente a la de los clientes, ya que, la interacción de estos con la aplicación es también diferente. Esta interfaz, se ha realizado con la librería Java.Swing.

Dicha aplicación comprende una serie de funcionalidades que variarán en función del perfil del usuario que acceda a la aplicación. De este modo podemos asumir el papel de cliente o el de un empleado de la tienda.
Las funcionalidades del cliente son las siguientes:

- En el menú de inicio se facilitará una interfaz para el registro de nuevos clientes.
- Se dispondrá un menú con los datos del usuario para la modificación de los datos personales del cliente.
- Acceder al catálogo de categorías y productos de la tienda. Dicho catálogo está almacenado en una base de datos de PostgreSQL y consta de placas base, procesadores, tarjetas gráficas, periféricos, portátiles, discos duros, etc. Cada uno con su propia tabla y atributos propios.
- Acceso a la descripción e información de cada artículo, donde se podrá valorar, comprar o añadir a la cesta el producto en concreto.
- Cesta de la compra. En dicha cesta, se irán acumulando los productos que el cliente vaya guardando. Cuando quiera realizar el pedido de todos los artículos guardados puede hacerlo en el menú de “Carrito”, además podrá eliminar artículos de la misma.
- Sistema de seguimiento de pedidos. El cliente dispondrá de la información de los pedidos que ha realizado, así como el estado de los mismos pudiendo confirmar la recepción de estos o rehacer la eliminación de un pedido.
- Búsqueda de productos. El sistema proporciona un sistema de búsqueda, que funcionará por nombre de producto. Se mostrarán los productos que contengan la palabra clave de búsqueda introducida.
- Montaje de ordenador personalizado, el cliente dispondrá de una interfaz con los productos a elegir para montar un ordenador. Podrá realizar el pedido en ese momento o guardar los artículos seleccionados en la cesta.

El empleado dispondrá de una interfaz y alguna funcionalidad similar a la del cliente, pero además dispondrá de una serie de funcionalidades específicas:

- Comparte la funcionalidad de búsqueda y modificación de los datos personales del usuario que ha iniciado sesión.
- Modificación del estado de los pedidos. El empleado modificará el estado de las compras cuyo estado esté “en preparación”. Se podrá variar entre diferentes estados, como por ejemplo enviado o recibido.

- Modificación de los productos de la base de datos. Se dispondrá de una interfaz para la modificación de los atributos de los productos de la base de datos.
- Introducir productos en la base de datos. Se proporcionará una interfaz para introducir productos en la base de datos.

2. Base de datos

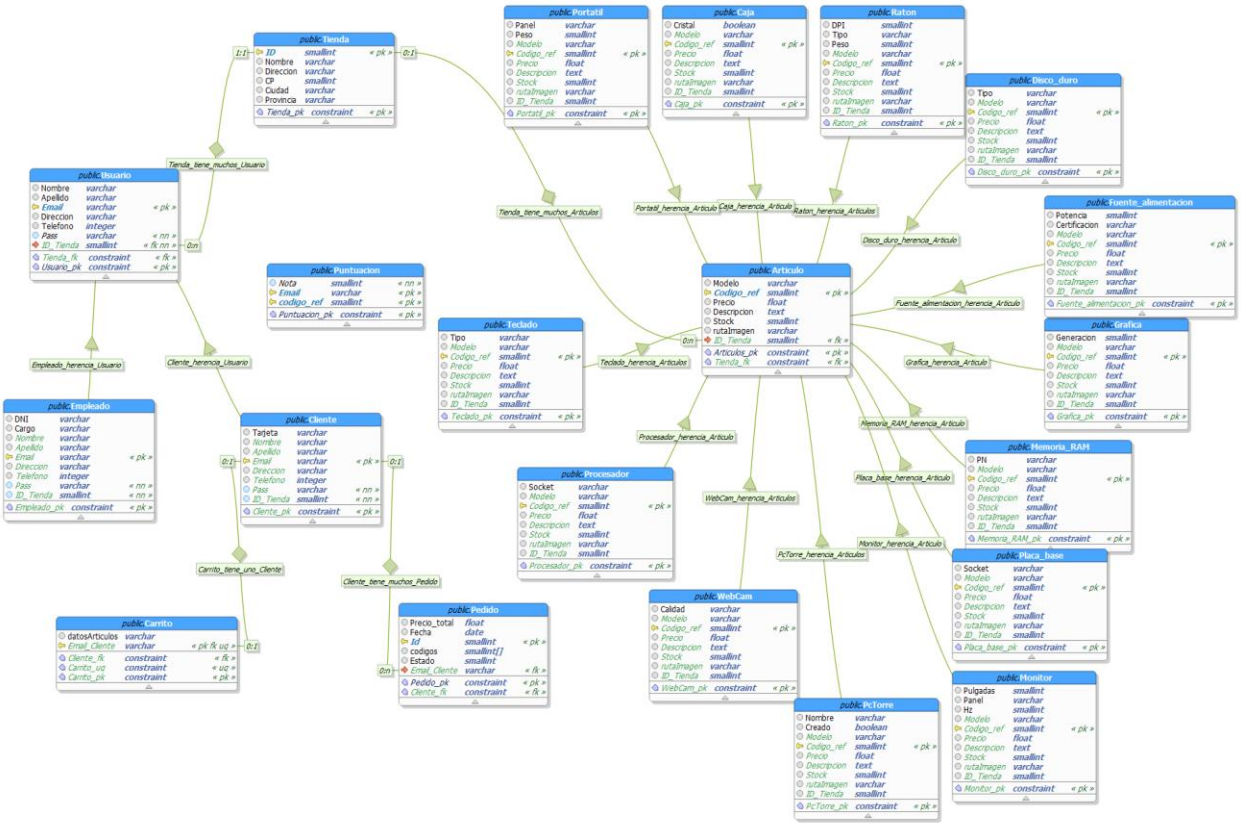

Fig 1. Base de datos

La base de datos cuenta con 21 entidades:

- Entidad Tienda: Representa nuestra tienda “ElReyDelPc” y contiene datos sobre ella como la dirección, provincia, nombre, etc.
- Entidad Usuario: representa cada uno de los usuarios registrados para poder utilizar la aplicación. Almacena información sobre los datos del usuario, como su nombre, apellidos, etc. De esta entidad heredan dos:
    - Entidad Empleado
    - Entidad Cliente
- Entidad Cesta: Representa la cesta que un cliente puede crear para realizar sus compras en la aplicación. Se almacenarán los productos almacenados en la misma (a través de un array String, el cual serán los identificadores de los productos)

- Entidad Pedido: Representa a la compra de uno o varios productos, almacenando el precio total de la compra, los códigos de referencia del artículo, la fecha, el email del cliente que ha realizado el pedido y el estado del pedido (En preparación, enviado o recibido)

- Entidad Artículo: representa cada uno de los productos del catálogo de la tienda. Se almacena toda la información relevante del producto, como su nombre, descripción, imagen a mostrar (ruta relativa en la aplicación), etc. De esta entidad heredan todos los tipos de producto, los cuales tienen alguna característica concreta:
    - Entidad Portátil
    - Entidad Caja
    - Entidad Ratón
    - Entidad Disco duro
    - Entidad Fuente alimentación
    - Entidad Gráfica
    - Entidad Memoria RAM
    - Entidad Placa base
    - Entidad Monitor
    - Entidad PcTorre
    - Entidad WebCam
    - Entidad Procesador
    - Entidad Teclado
- Entidad Puntuación: representa la valoración de los productos en función de una puntuación que va de 0-5 puntos. Se almacena la nota, el email del cliente y el código de referencia del artículo. Cada cliente solo puede valorar una vez un producto.

3. Patrones utilizados

3.1. Patrones de creación

3.1.1. Singleton

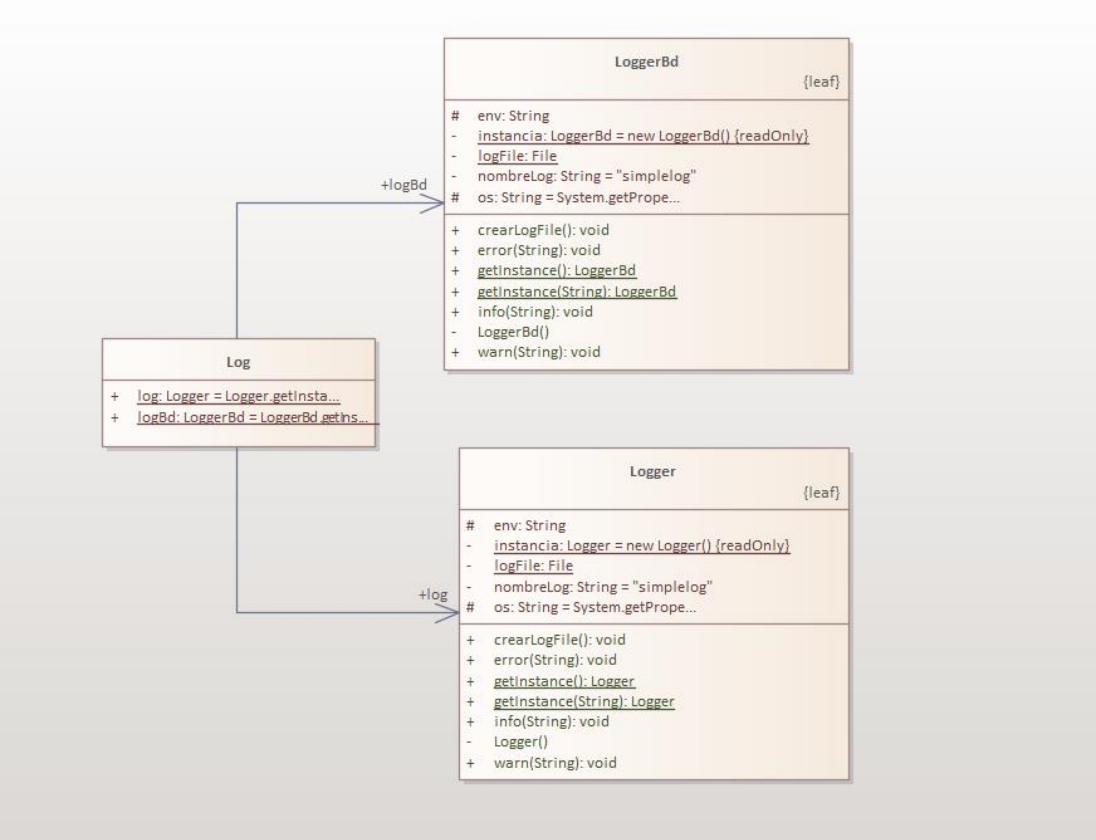

Fig 2. Patrón Singleton

Los singleton garantizan que solo haya una instancia del objeto cuando el programa esté en ejecución y, mediante la clase Log, creamos un punto de acceso global a estos.

Se han creado dos logger, uno para las conexiones y consultas con la base de datos y otro para dar información al usuario sobre la ejecución del programa (eventos, acciones, errores...). La utilización de un singleton para este caso era la más adecuada, ya que, por cada ejecución del programa se genera un único archivo .log con la hora a la que se ha realizado la ejecución de este y en donde encontraremos diversa información precedida por diferentes etiquetas como “[ERROR]”, “[INFO]”, “[WARN]”.

```
[INFO] CONSULTA GetTipoUsuario
[INFO] Inicio de conexion en puerto[5432]
[INFO] Conexion establecida en puerto[5432]
[INFO] Realizada conexion - getTipoUsuario()
[INFO] Realizada consulta - getTipoUsuario()
[INFO] Consulta realizada con éxito - getTipoUsuario()
```

Fig 3. Captura de archivo .log

```
Log.logBd.info("Conexion establecida en puerto["+ puerto +"]");
```

Fig 4. Llamada a log

3.1.2. Builder

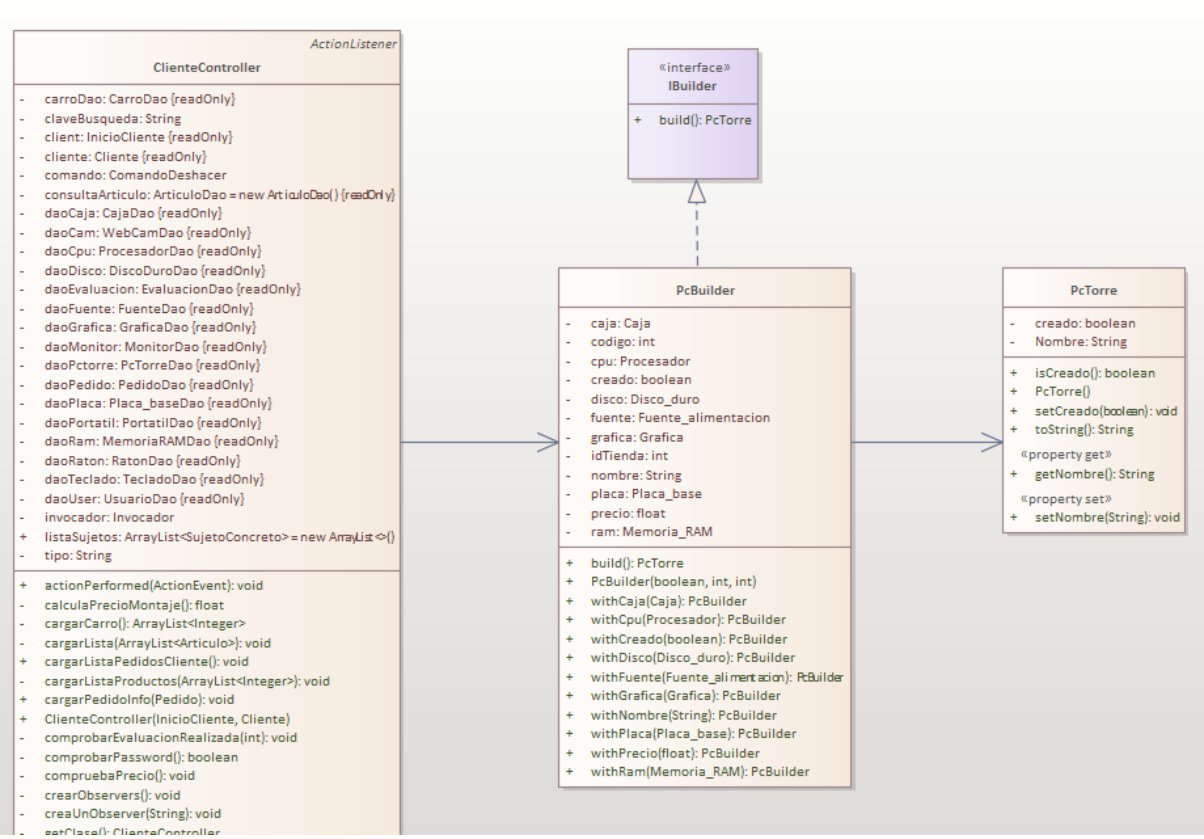

Fig 5. Patrón Builder

El patrón builder separa la construcción de un objeto complejo de su representación, de modo que el mismo proceso de construcción pueda crear diferentes representaciones.

En este caso, el patrón builder se usa para crear una torre de pc personalizada para el cliente de la tienda, en donde este, puede elegir todos y cada uno de los componentes de su ordenador para crear un ordenador a su gusto, en función de preferencias, presupuesto…

La clase “PcTorre” no tiene una instancia de todos los componentes que conforman el ordenador, sino que, en el atributo “descripción” se guarda el nombre del modelo de todos los componentes del ordenador.

En el código el proceso de creación del ordenador se realiza de la siguiente forma:

```
PcBuilder builder = new PcBuilder(true, nuevoCod, 0);
PcTorre pc = builder
        .withCpu(cpu)
        .withCaja(caja)
        .withPlaca(placa)
        .withGrafica(grafica)
        .withDisco(disco)
        .withFuente(fuente)
        .withRam(ram)
        .withPrecio(precioTorre)
        .withNombre("Custom-PC-" + nuevoCod)
        .build();
```

Fig 6. Uso del patrón Builder

3.2. Patrones de comportamiento

3.2.1. Command

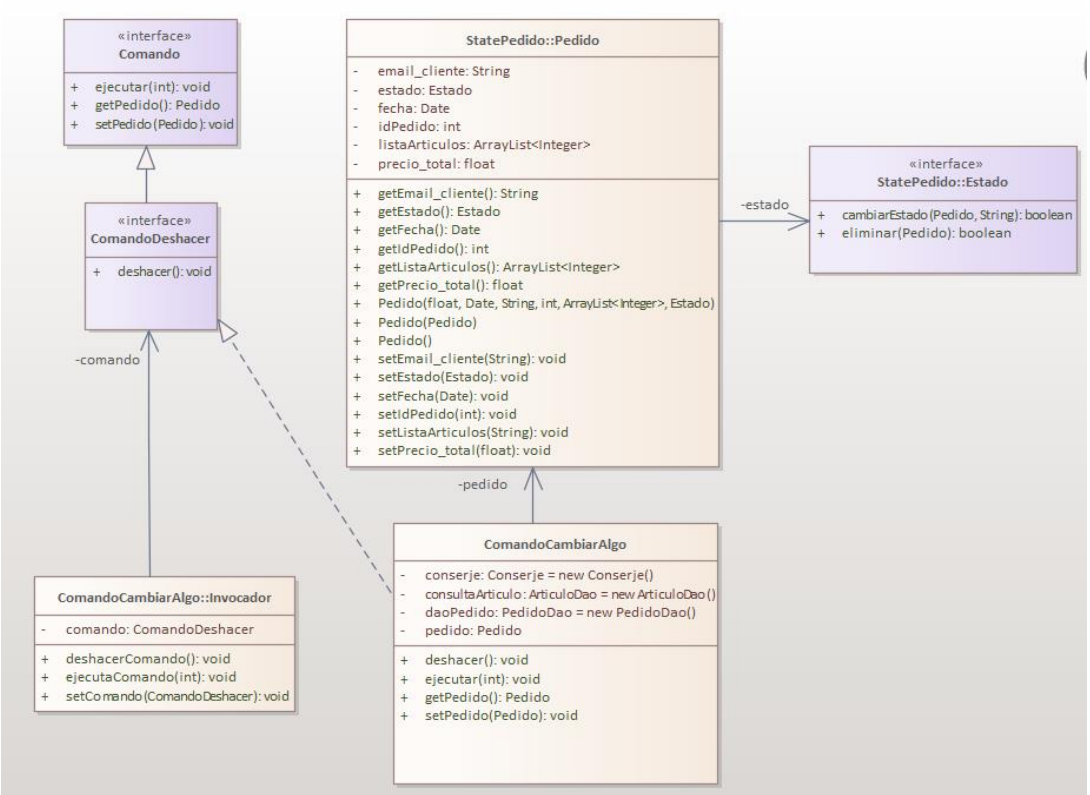

Fig 7. Patrón Command

El patrón command permite encapsular un comando en un objeto. Este objeto contiene el comportamiento y los datos necesarios para una acción específica, como puede ser parametrizar diferentes peticiones, hacer cola, deshacer operaciones…

En este caso, utilizamos el patrón command para poder deshacer un pedido, antes de que este haya sido enviado. Una vez eliminado, en caso de que haya sido un error o el cliente quiera deshacer la cancelación del pedido, podrá volver a recuperar el pedido eliminado, utilizando, como veremos en el siguiente apartado, el patrón memento.

```
int idPedido = comando.getPedido().getIdPedido();
if (JOptionPane.showConfirmDialog(null, "¿Está seguro que quiere recuperar el pedido "+ idPedido +" eliminado?", "WARNING",
            JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {
ago • Uncommitted changes
    invocador.deshacerComando();

    if(comando.getPedido() == null){
        client.panelInfoPedido.setVisible(false);
        client.panelCompras.setVisible(true);
```

Fig 8. Uso de command para deshacer pedido eliminado

```
int idPedido = Integer.parseInt(client.nPedidoInfo.getText());
Pedido pedido = daoPedido.getPedido(idPedido);

//Si el pedido se puede eliminar, es decir, esta actualmente en preparacion
//borramos el pedido en la base de datos
if (pedido.getEstado().eliminar(pedido)) {
    if (JOptionPane.showConfirmDialog(null, "¿Está seguro que quiere eliminar el pedido "+ idPedido +"?", "WARNING",
            JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {}

            comando.setPedido(pedido);
            invocador.setComando(comando);
            invocador.ejecutaComando(idPedido);
```

Fig 9. Uso de Command para guardar pedido eliminado

3.2.2. Memento

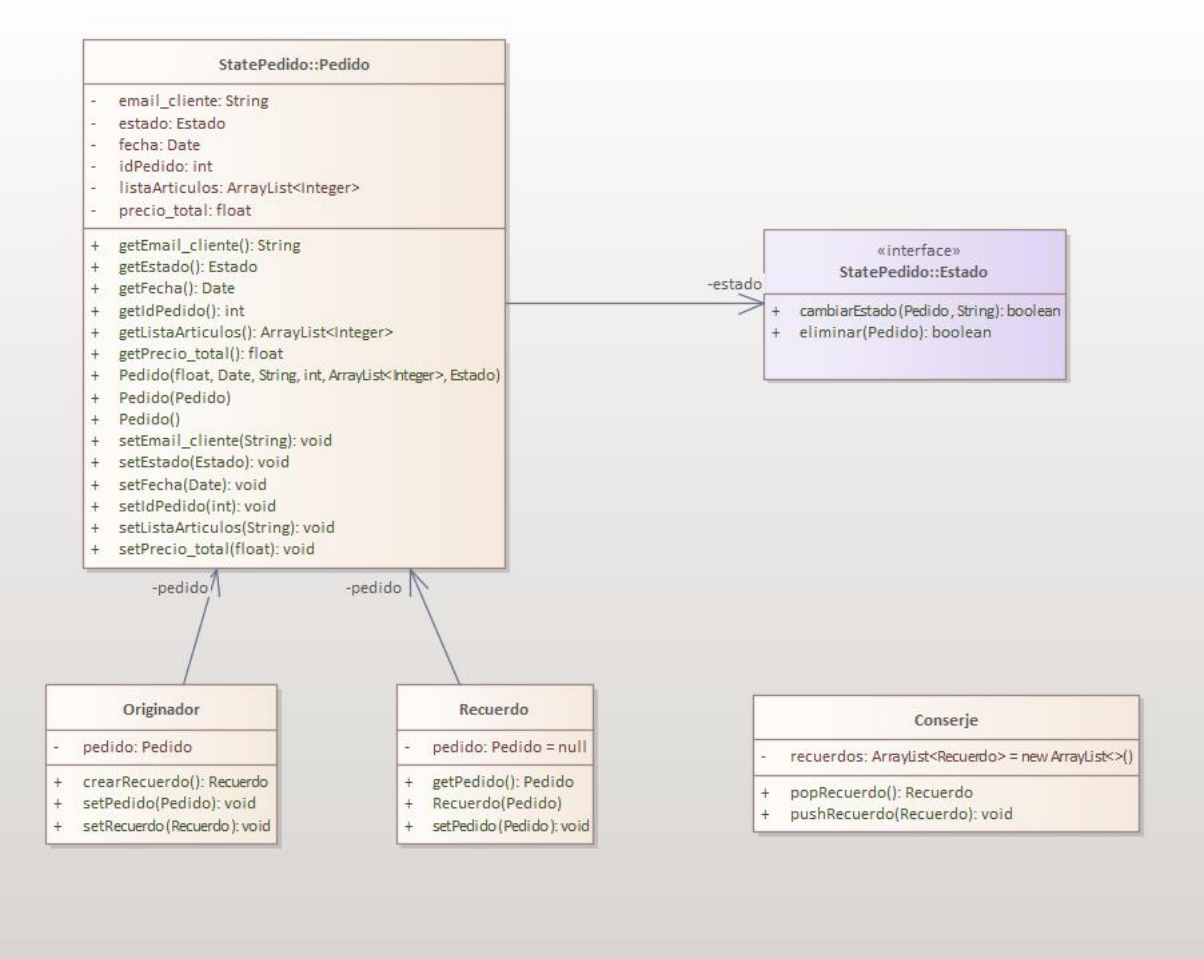

Fig 10. Patrón Memento

El patrón memento sirve para guardar el estado interno de un objeto para que luego se pueda restaurar, sin revelar su contenido al resto del mundo.

Como hemos comentado en el anterior apartado, era necesario guardar la información del pedido eliminado por el usuario, por ello, se ha utilizado este patrón. Para poder recuperar o guardar el pedido eliminado se realiza lo siguiente:

```
Recuerdo recuerdo = conserje.popRecuerdo();
if (recuerdo != null){
    Pedido pedidoRecuerdo = recuerdo.getPedido();
```

Fig 11. Recuperación de pedido eliminado

```
//Creamos el recuerdo con el pedido eliminado
Originador originador = new Originador();
originador.setPedido(pedido);
conserje.pushRecuerdo(originador.crearRecuerdo());
```

Fig 12. Creación de recuerdo para un pedido eliminado.

3.2.3. Observer

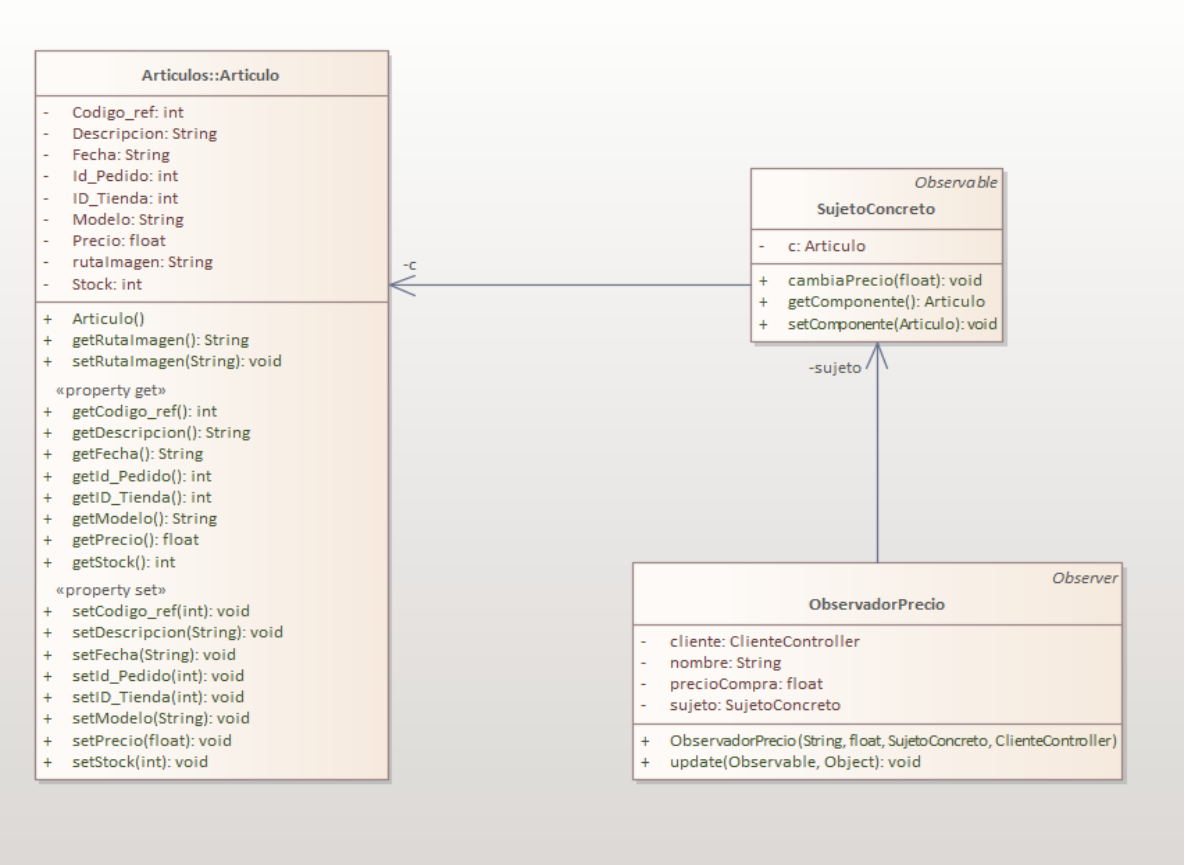

Fig 13. Patrón Observer

El patrón observer permite definir dependencias de 1-N de forma que los cambios en un objeto se comuniquen a los demás objetos que dependan de él.

Para poder avisar al usuario con alertas cada vez que un artículo de su carrito baja de precio, se ha creado un observador por cada artículo que el usuario añade a la cesta y, cada vez que un artículo de la cesta la aplicación notificará al usuario con la rebaja del artículo.

```
private void creaUnObserver(String codigo) {
    Articulo articulo = consultaArticulo.getArticulo(Integer.parseInt(codigo));
    SujetoConcreto sujeto = new SujetoConcreto();
    sujeto.setComponente(articulo);
    listaSujetos.add(sujeto);
    Observer obs = new ObservadorPrecio("obs", articulo.getPrecio(), sujeto, getClase());
}
```

Fig 14. Creación de un observer

```
private void compruebaPrecio() {
    for (int i = 0; i < listaSujetos.size(); i++) {
        SujetoConcreto sujeto = listaSujetos.get(i);
        int codigo = sujeto.getComponente().getCodigo_ref();
        float precio = consultaArticulo.getArticulo(codigo).getPrecio();
        sujeto.cambiaPrecio(precio);
    }
}
```

Fig 15. Cambio de precio de un componente

3.2.4. State

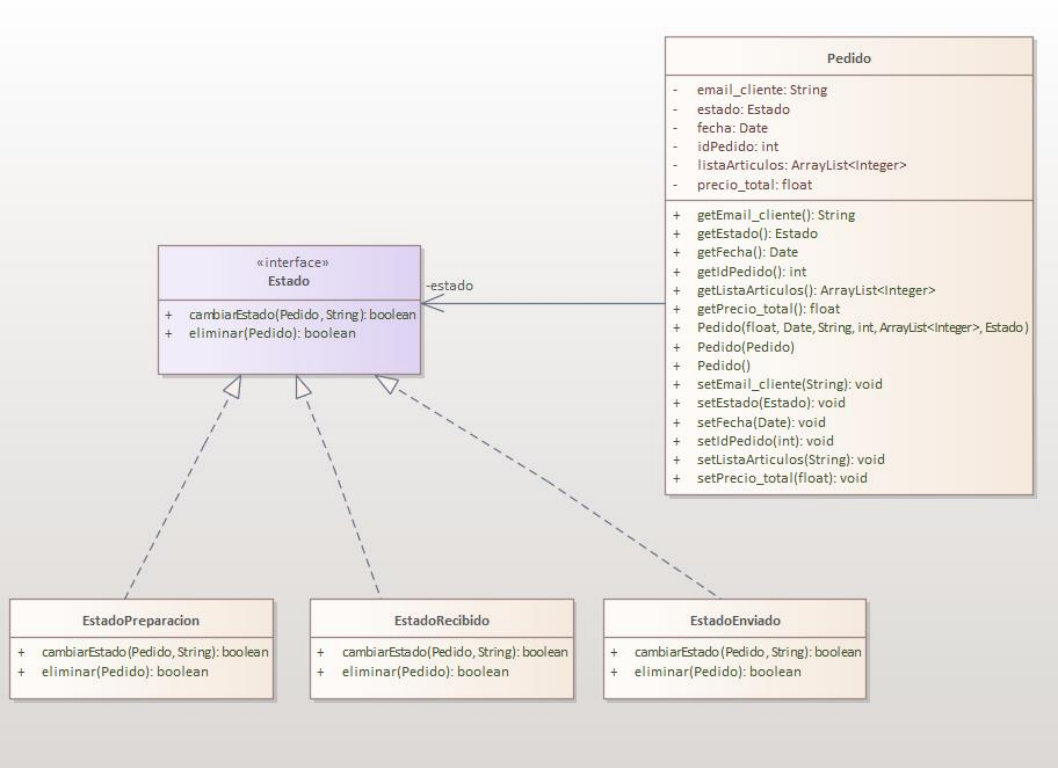

Fig 16. Patrón State

El patrón State permite que un objeto se comporte de distinta forma dependiendo de su estado interno, como si cambiase la clase a la que pertenece. Permite cambiar fácilmente el comportamiento de un objeto en tiempo de ejecución.

En la aplicación, los pedidos tienen 3 estados diferentes; preparación, cuando el pedido ha sido realizado por el cliente y no se ha enviado aún, enviado, cuando el pedido se envía desde la tienda y recibido cuando el cliente o empleado confirman la recepción de este.

Para cambiar cualquiera de los estados se comprueba antes en qué estado está el pedido, es decir, si está en preparación no se puede pasar a recibido. A continuación, se muestran las funciones de cambiarEstado y eliminar para el caso de EstadoPreparacion:

```
@Override
public boolean cambiarEstado(Pedido pedido, String siguienteEstado) {
    // siguienteEstado - [enviado, recibido]
    boolean cambiado = false;

    if(siguienteEstado.equalsIgnoreCase("enviado")){
        EstadoEnviado nuevoEstado = new EstadoEnviado();
        pedido.setEstado(nuevoEstado);
        cambiado = true;
        JOptionPane.showMessageDialog(null, "El estado del pedido ha cambiado a enviado");
    }
    else if(siguienteEstado.equalsIgnoreCase("recibido")){
        JOptionPane.showMessageDialog(null, "ERROR: El pedido está aún en preparación");
    }

    return cambiado;
}

@Override
public boolean eliminar(Pedido pedido) {
    return true;
}
```

Fig 17. Función cambiarEstado en clase EstadoPreparacion

3.3. Patrones estructurales

3.3.1. Adapter

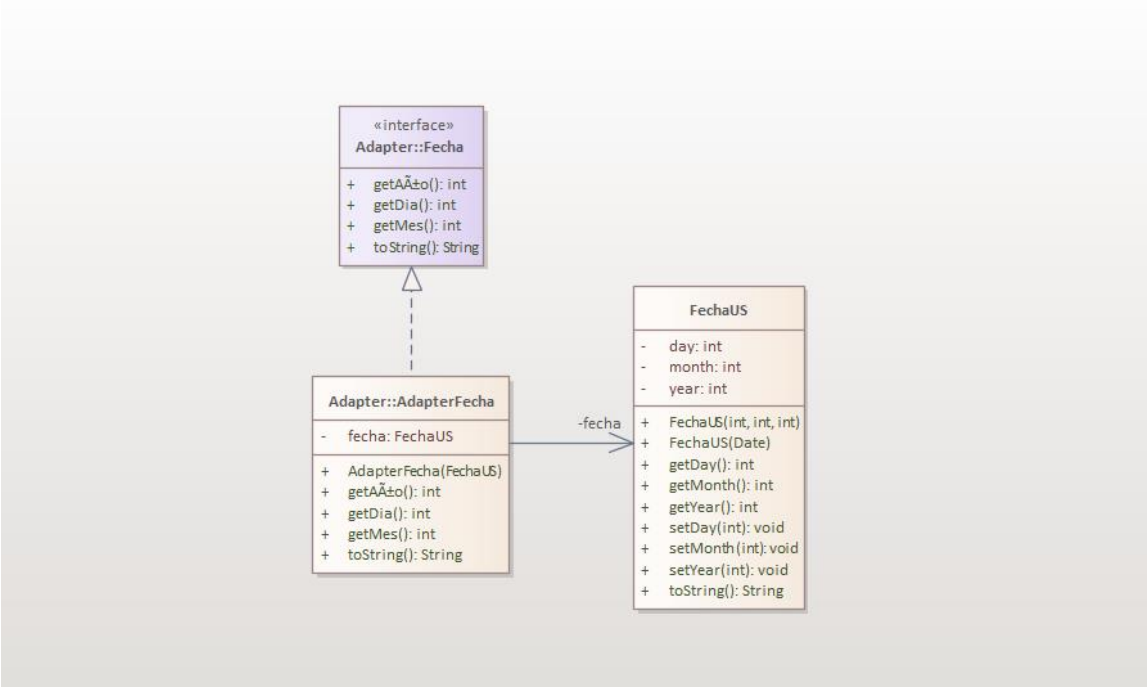

Fig 18. Patrón Adapter

El patrón adapter sirve de intermediario entre dos clases, convirtiendo las interfaces de una clase para que pueda ser utilizada por otra, haciendo que cooperen clases con interfaces incompatibles.

En la aplicación cuando se obtiene la fecha del resultado de ejecutar una consulta, la fecha, como objeto Date, que se obtiene están en formato US (aaaa/mm/dd), pero, en la aplicación se trabaja con un formato diferente (dd/mm/aaaa), por ello se ha optado por la utilización de este patrón.

Para realizar la conversión de la fecha, debido que, dentro de la clase pedido, la fecha, es un objeto Date, se usa el patrón adapter:

```
Fecha fecha = new AdapterFecha(new FechaUS(pedido.getFecha()));
```

Fig 19. Uso de patrón Observer

3.3.2. Facade

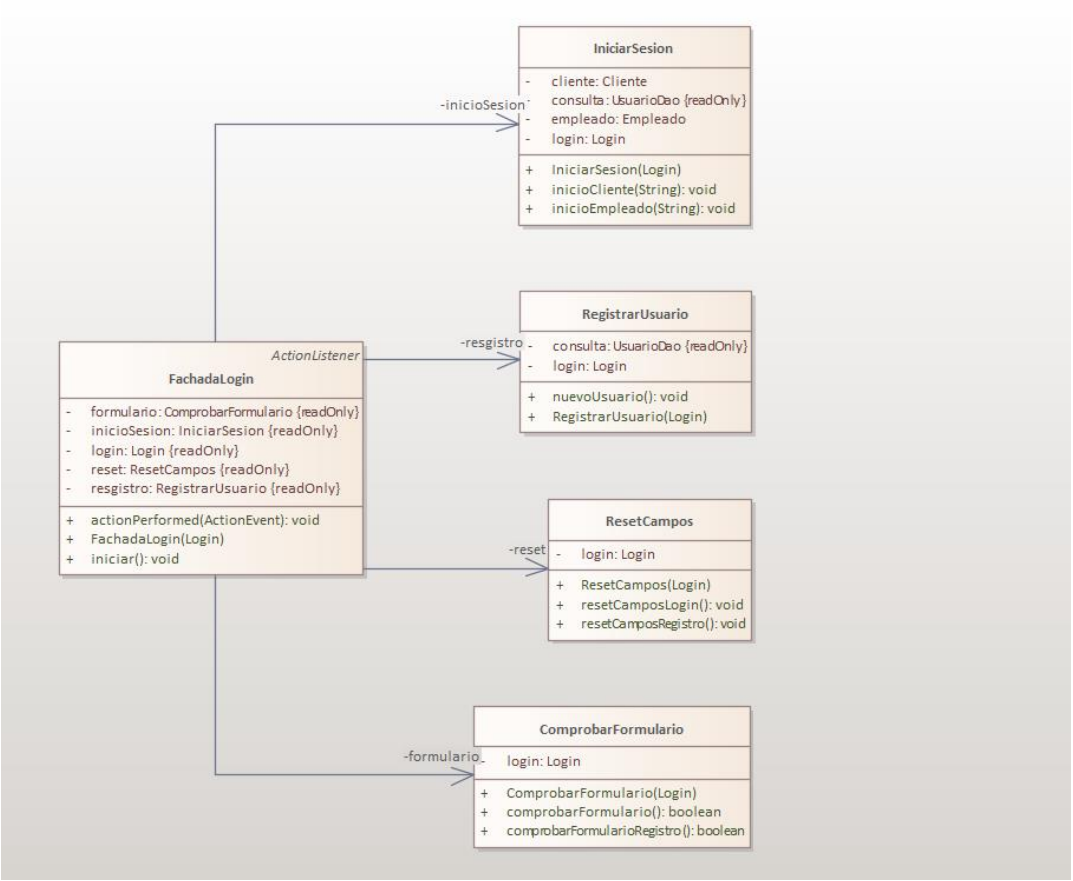

Fig 20. Patrón Facade

El patrón facade simplifica el acceso a un conjunto de subsistemas o un sistema complejo. Representa una única interfaz unificada, que envuelve el subsistema, y es responsable de colaborar con los componentes del subsistema.

La interfaz de login tiene diferentes botones y acciones que se pueden realizar sobre ella, por ello, para simplificar el acceso a los subsistemas que dan funcionalidad a las diferentes partes de la interfaz, se han creado clases que agrupan a estos subsistemas, para así, poder reducir el acoplamiento y la dependencia entre clases.

```
private final ComprobarFormulario formulario;
private final IniciarSesion inicioSesion;
private final RegistrarUsuario resgistro;
private final ResetCampos reset;
```

Fig 21. Declaración en la clase fachada

```
else if(boton.getSource() == login.borrarLogin){
    reset.resetCamposLogin();
    Log.log.info("Vista Login - evento borrarLogin");
}
else if(boton.getSource() == login.borrarRegistro){
    reset.resetCamposRegistro();
    Log.log.info("Vista Login - evento borrarRegistro");
}
else if(boton.getSource() == login.btnDarAlta){
    if(formulario.comprobarFormularioRegistro()){
        resgistro.nuevoUsuario();
        reset.resetCamposLogin();
    }
}
```

Fig 22. Llamada a función desde la clase fachada

3.3.3. Proxy

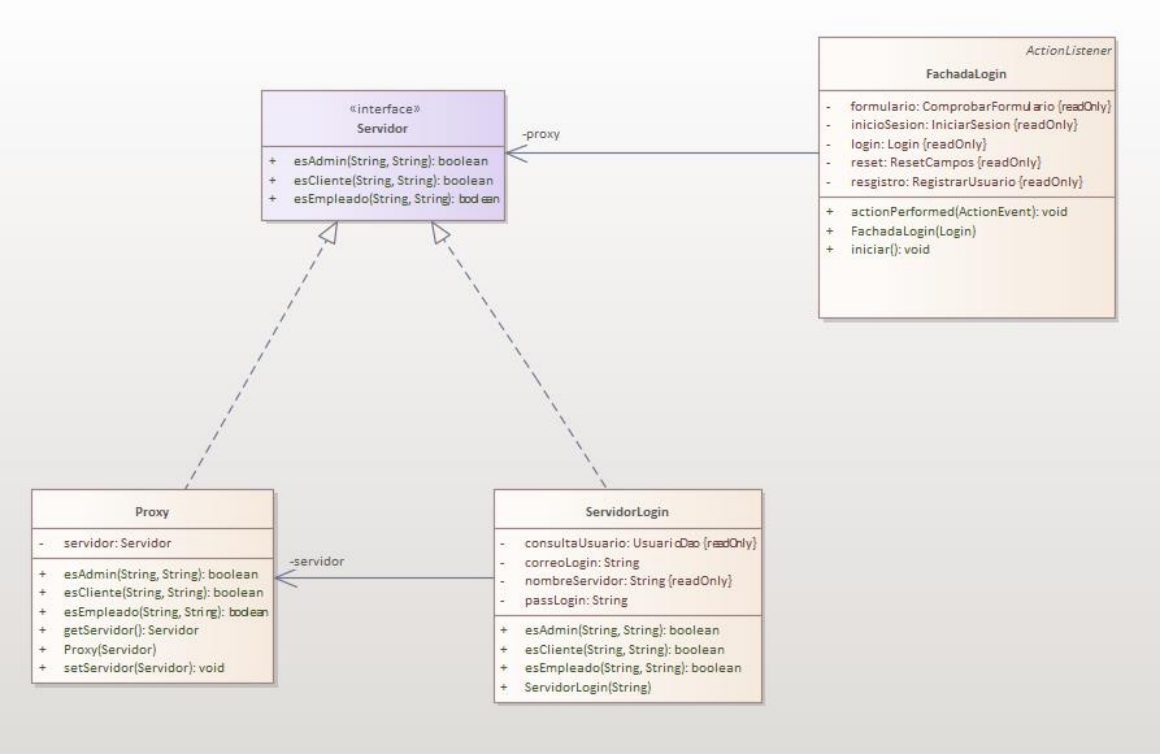

Fig 23. Patrón Proxy

El patrón proxy proporciona un representante o sustituto de otro objeto para controlar el acceso a este.

Cuando un cliente o empleado quiere iniciar sesión el programa comprueba que tipo de usuario está accediendo a esta, para ello, se ha creado un servicio, el cual devuelve qué tipo de usuario es el que está accediendo a la aplicación, para así, poder controlar los derechos de acceso de los usuarios a los objetos, por ejemplo, un empleado puede editar los datos de un artículo mientras que un cliente no.

La instancia y utilización del proxy se realiza en la clase “FachadaLogin” de la siguiente forma:

```
Servidor proxy = new Proxy(new ServidorLogin("ServidorLogin"));

if(proxy.esCliente(usuario, contrasenna)){
    inicioSesion.inicioCliente(usuario);
}
else if(proxy.esEmpleado(usuario, contrasenna)){
    inicioSesion.inicioEmpleado(usuario);
}
else if(proxy.esAdmin(usuario, contrasenna)){
```

Fig 24. Uso del objeto proxy

3.3.4. MVC

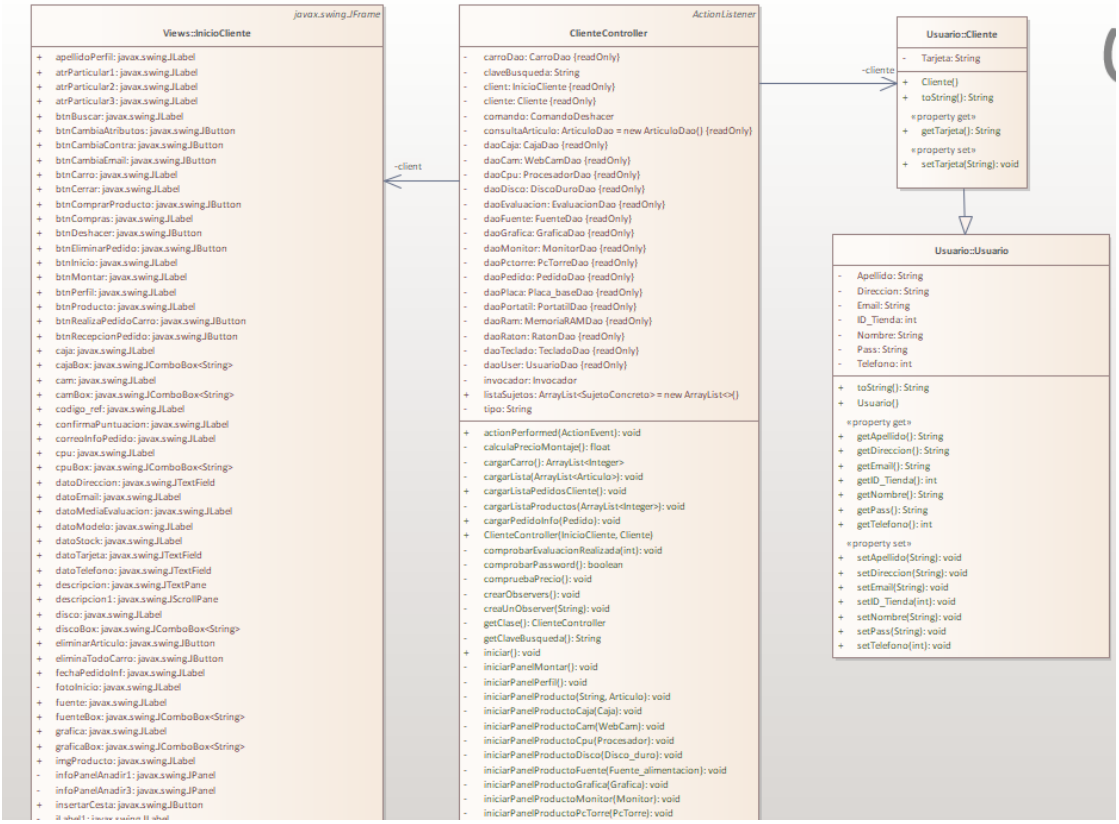

Fig 25. Patrón Modelo Vista Controlador

El patrón modelo vista controlador MVC separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

Se ha utilizado este patrón para separar las vistas, las cuales se han realizado mediante la librería java.Swing, de la lógica de negocio, en donde tenemos las clases que crean los diferentes objetos que componen la aplicación, como por ejemplo Caja, Raton, Procesador…

Se ha optado por utilizar este patrón por diferentes razones:

- Se incrementa la reutilización y flexibilidad del código ya que como se ha explicado anteriormente separa la lógica de las vistas.

- Permite controlar los eventos que se producen en las vistas, como por ejemplo el click en un botón.

- Permite también, gestionar el flujo de control de la aplicación, es decir, en función de los diferentes eventos que realice el usuario, mediante el controlador, generamos una nueva vista, que, en nuestro caso, en vez de una nueva interfaz, se oculta el panel actual y se pone visible el siguiente panel, o realizamos algún tipo de consulta, como cuando, por ejemplo, editamos los datos de un empleado o de un cliente.

Este patrón se usa tanto para la interfaz de cliente como para la interfaz de empleado.

3.3.5. DAO

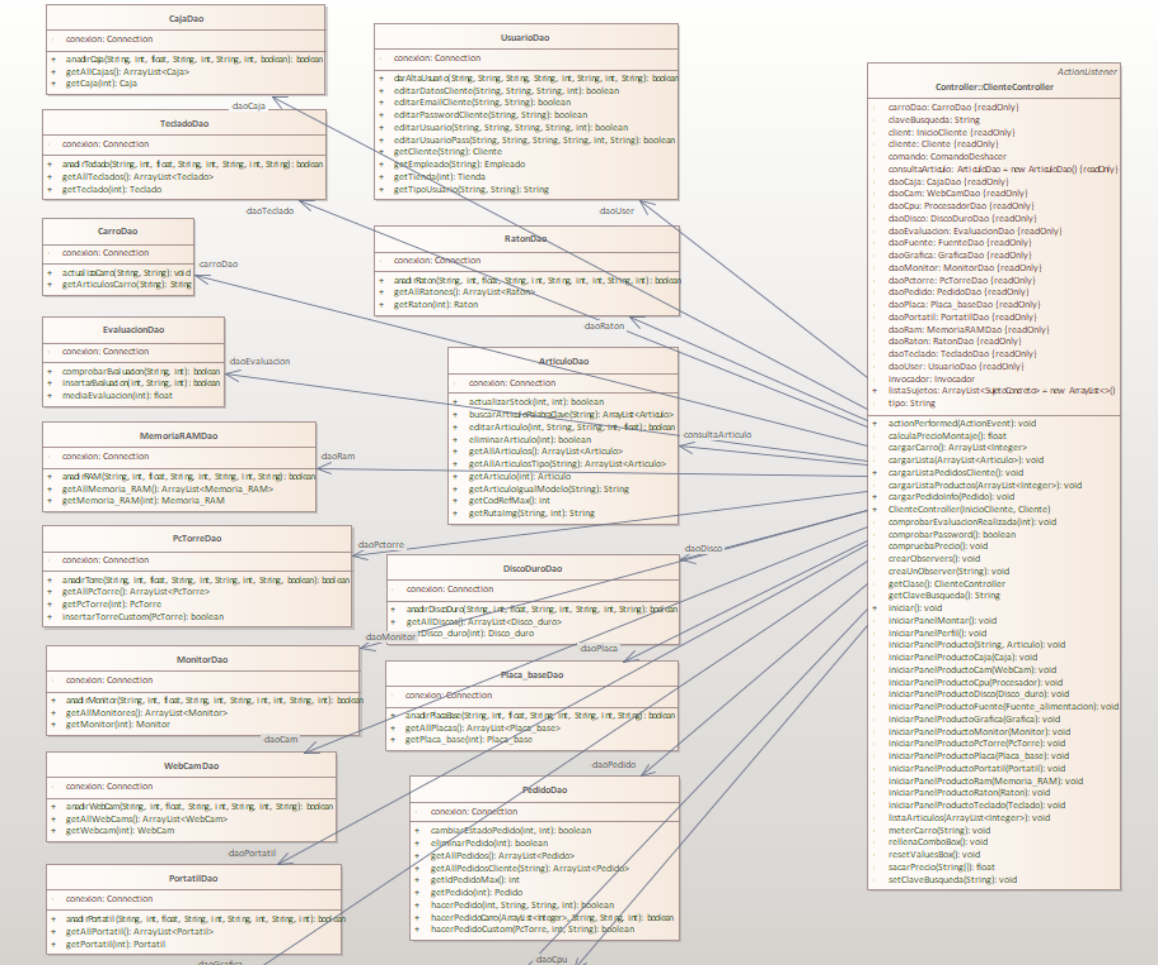

Fig 26. Patrón DAO

El patrón DAO suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, como en este caso, es la base de datos.

Los accesos a la base de datos y las diferentes consultas se han realizado mediante este patrón ya que permite desacoplar las clases de acceso a los datos de las demás clases, como son, en este caso, los controladores o demás clases de la aplicación.

La estructura de este patrón, en la mayoría de los casos, suele definir previamente una interfaz que las clases DAO implementan, pero como hay numerosos DAO, cada uno con consultas diferentes se ha optado por no implementarlas.

```
private final UsuarioDao consultaUsuario = new UsuarioDao();
```

Fig 27. Uso del DAO UsuarioDAO

```
exito = consultaUsuario.editarUsuario(correo, nombre, apellido, direccion, telefono);
```

Fig 28. Llamada a función de UsuarioDao

3.4. Patrones de diseño fundamentales

3.4.1. Interface

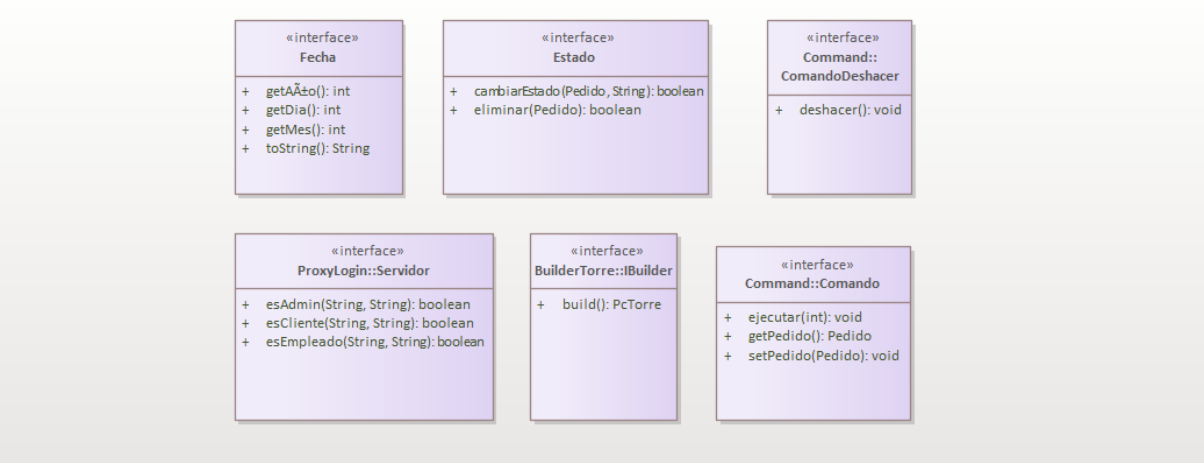

Fig 29. Interfaces del programa

La idea principal por la que hemos utilizado este patrón es para separar las funciones de las implementaciones. Ya que mantiene una clase que usa datos y servicios provistos por otras clases independientes, para proveer un acceso uniforme.

Con esto logramos restar importancia a la clase que realiza la implementación y responde a la llamada, consiguiendo así la capacidad de poder intercambiar fácilmente la clase que implementa la interfaz sin modificar el código donde se hace la llamada a dicha interfaz.

Se ha implementado en los siguientes patrones: Adapter, Builder, Command, Memento, State y Proxy.

4. Paquetes

4.1. Paquete Adapter

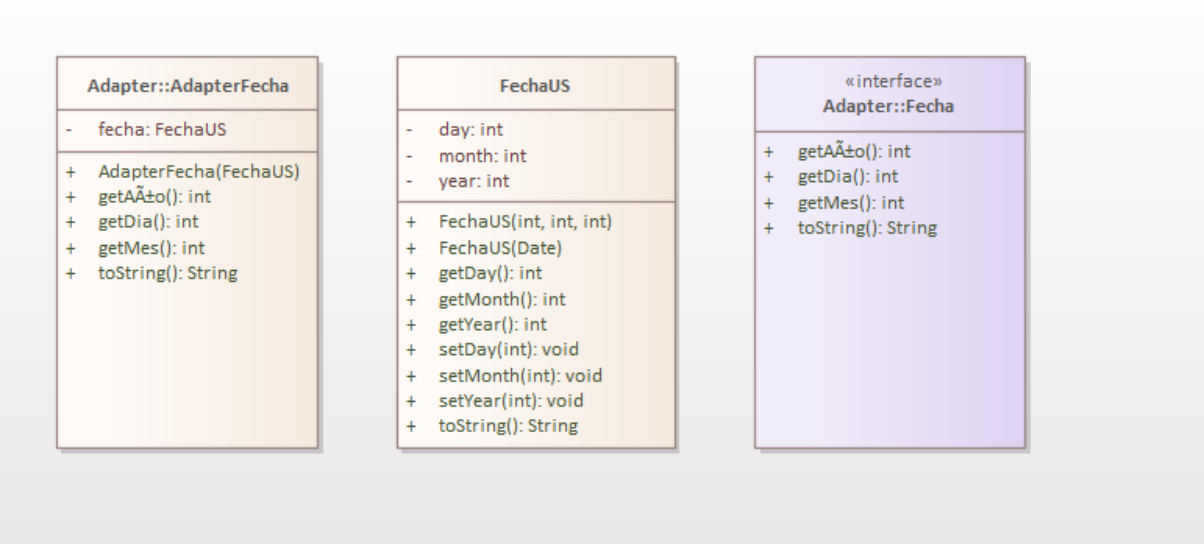

Fig 30. Clases del paquete Adapter

El paquete adapter tiene todas las clases, como su propio nombre indica, que conforman el patrón Adapter. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete adapter tiene las siguientes clases e interfaces:

- Clase AdapterFecha: es la clase que cambia el formato de la fecha de US a el formato que requerimos en nuestra aplicación.

- Clase FechaUS: es la clase que se usa para manejar las fechas en formato US, esta clase recibe la fecha como objeto Date y crea el objeto fecha.

- Interfaz Fecha: esta interfaz define los métodos que implementa la clase AdapterFecha, estos métodos son getters de los atributos año, mes y día.

4.2. Paquete BuilderTorre

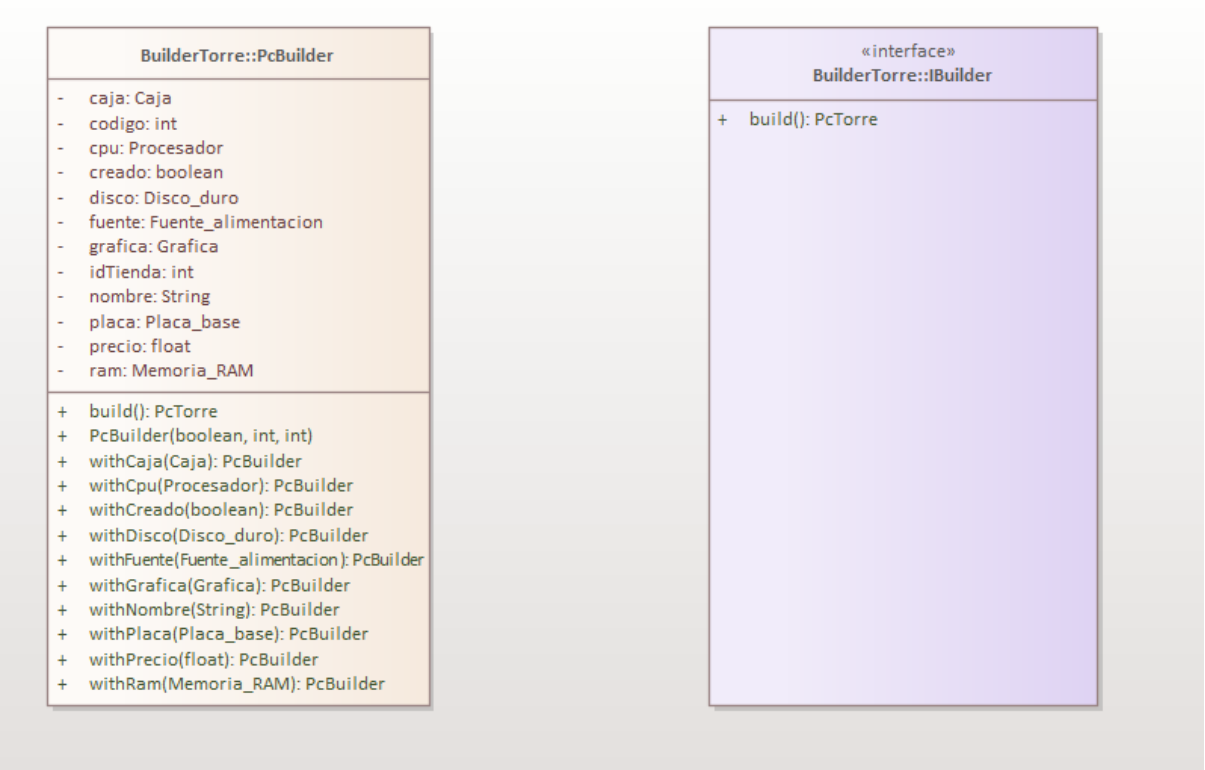

Fig 31. Clases del paquete BuilderTorre

El paquete BuilderTorre tiene todas las clases, como su propio nombre indica, que conforman el patrón Builder. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete BuilderTorre tiene las siguientes clases e interfaces:

- Interfaz IBuilder: es la interfaz que define el método build(), el cual implementa la clase PcBuilder.

- Clase PcBuilder: es la clase la cual contiene tanto el método build(), el cual crea un nuevo objeto PcTorre que guarda el objeto creado con el builder y los métodos que añaden nuevas partes al pc que se está creando.

4.3. Paquete Command y memento

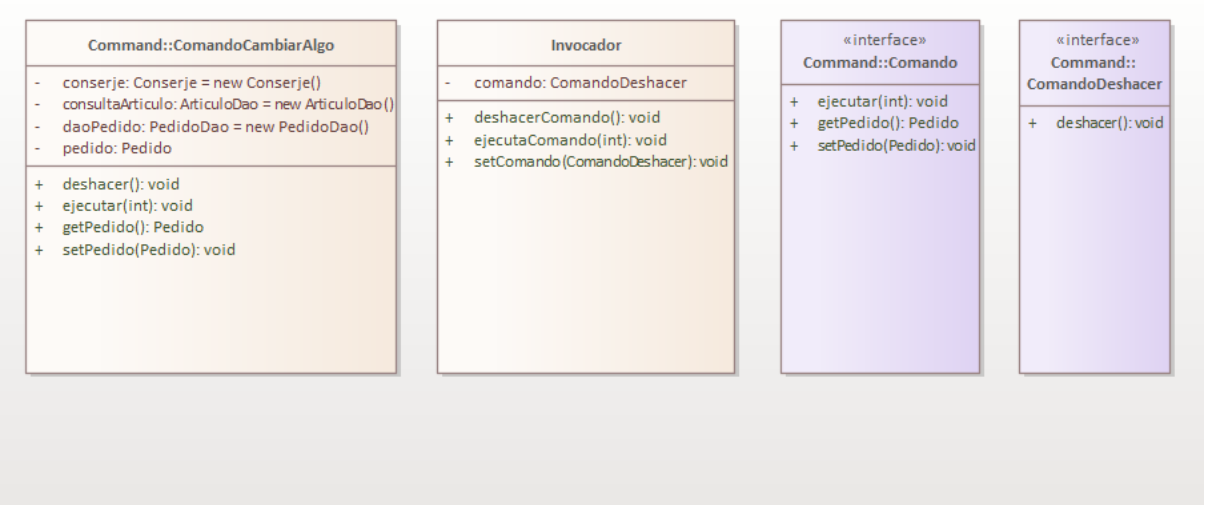

Fig 32. Clases del paquete Command

El paquete Command tiene todas las clases, como su propio nombre indica, que conforman el patrón Command. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete Command tiene las siguientes clases e interfaces:

- Clase ComandoCambiarAlgo: esta clase implementa el comando deshacer proporcionándole los métodos necesarios para poder eliminar, y posteriormente, recuperar el pedido eliminado.

- Clase Invocador: es la clase que guarda el comando y desde la que se puede ejecutar los métodos del comando, deshacer() y ejecutar().

- Interfaz Comando: es la interfaz que define de forma general el comportamiento de los comandos.

- Interfaz ComandoDeshacer: interfaz que extiende a su vez la interfaz comando y en la que se especifica el método para deshacer la eliminación de un pedido.

Fig 33. Clases del paquete Memento

El paquete Command.Memento tiene todas las clases, como su propio nombre indica, que conforman el patrón Memento. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete Command.Memento tiene las siguientes clases e interfaces:

- Clase Conserje: es la clase en la cual se guardan los recuerdos, es decir, todos los pedidos que se han eliminado, para, posteriormente, poder recuperarlos. En esta clase se puede añadir un nuevo recuerdo con pushRecuerdo(), o recuperar el último recuerdo almacenado con popRecuerdo().

- Clase Originador: es la clase que crea un nuevo objeto recuerdo y lo usa para poder recuperar su estado.

- Clase Recuerdo: esta clase almacena el estado interno de un objeto originador.

4.4. Paquete Controller

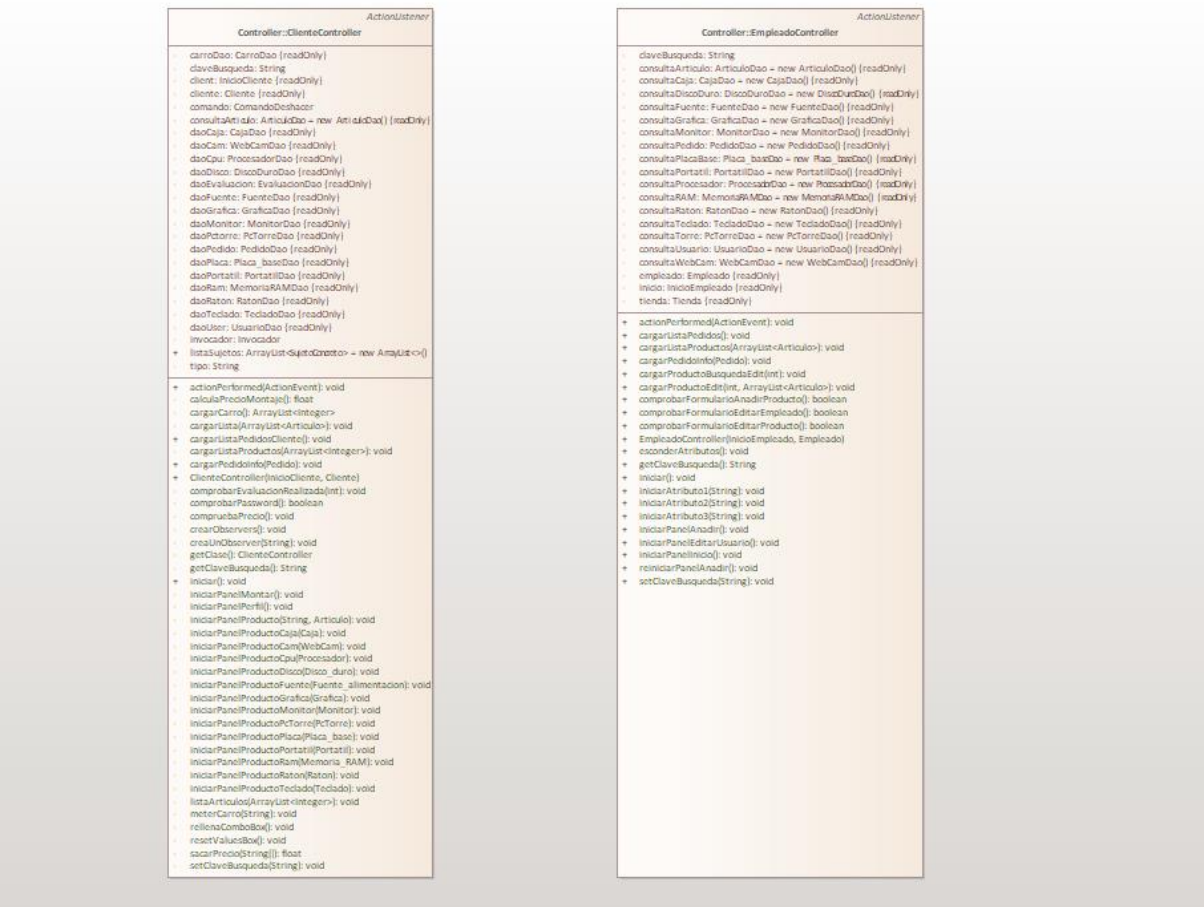

Fig 34. Clases del paquete Controller

El paquete Controller tiene todas las clases, como su propio nombre indica, que conforman el patrón Controller. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete Controller tiene las siguientes clases e interfaces:

- Clase ClienteController: clase que contiene la mayor parte de los métodos que hacen que las interfaces de usuario reaccionen a diferentes eventos tales como pulsar un botón. Se definen eventos, sobretodo Listeners, para cuando el usuario rellena formularios, navega entre menús, etc. Además, en esta clase se gestiona el control de flujo entre las diferentes interfaces y se cargan todos los datos necesarios a mostrar en estas, tales como información de los productos, pedidos, datos personales...

- Clase EmpleadoController: esta clase tiene un funcionamiento y unos métodos bastantes parecidos a la clase ClienteController, pero en vez de gestionar la vista del cliente, esta clase gestiona la vista del empleado.

4.5. Paquete DAO

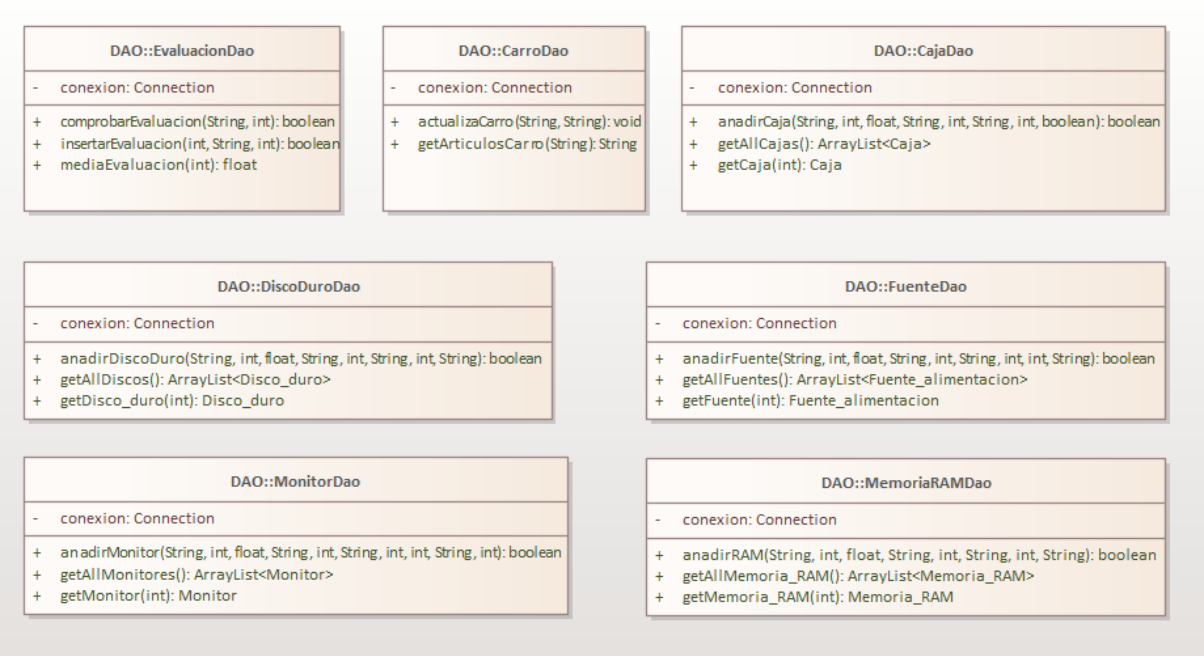

Fig 35. Clases del paquete DAO

El paquete DAO tiene todas las clases, como su propio nombre indica, que conforman el patrón DAO. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

En el paquete DAO todas las clases tiene un objeto Connection los cuales sirven para crear una conexión con la base de datos y poder hacer las consultas, tiene las siguientes clases e interfaces:

- Clase ArticuloDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla articulo de la base de datos.

- Clase CajaDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla caja de la base de datos.

- Clase CarroDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla carro de la base de datos.

- Clase DiscoDuroDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla disco_duro de la base de datos.

- Clase EvaluacionDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla evaluacion de la base de datos.

- Clase FuenteDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla fuente_alimentacion de la base de datos.

- Clase GraficaDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla grafica de la base de datos.

- Clase MemoriaRAMDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla memoria_ram de la base de datos.

- Clase MonitorDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla monitor de la base de datos.

- Clase PcTorreDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla pctorre de la base de datos.

- Clase PedidoDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla pedido de la base de datos.

- Clase Placa_baseDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla placa_base de la base de datos.

- Clase PortatilDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla portatil de la base de datos.

- Clase ProcesadorDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla procesador de la base de datos.

- Clase RatonDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla raton de la base de datos.

- Clase TecladoDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla teclado de la base de datos.

- Clase UsuarioDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla usuario de la base de datos.

- Clase WebCamDAO: esta clase agrupa todas las consultas que se realizan sobre la tabla webcam de la base de datos.

4.6. Paquete Facade

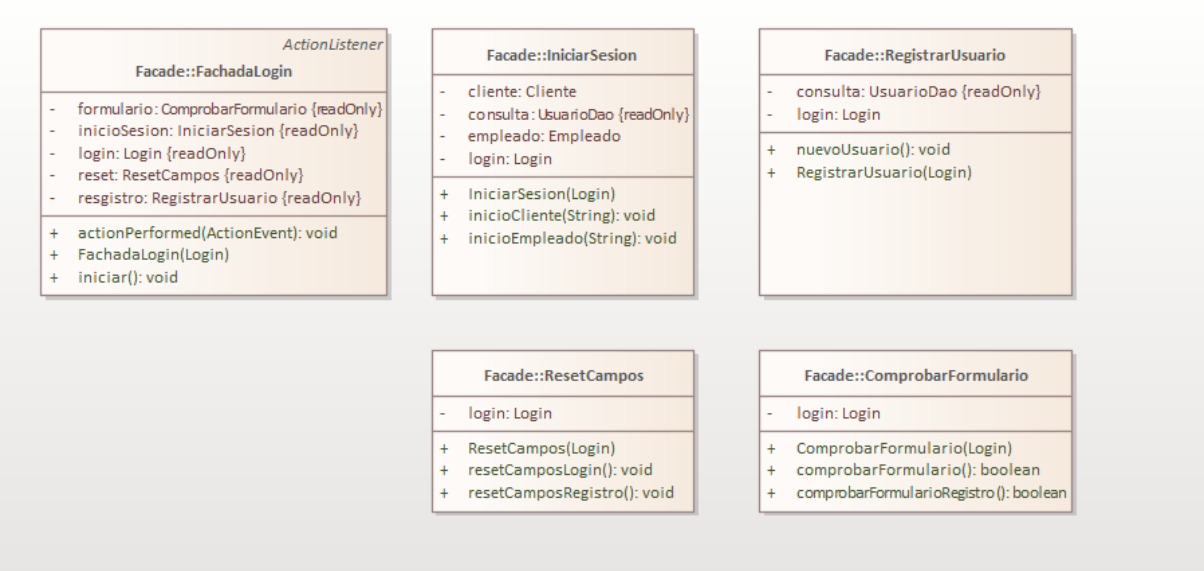

Fig 36. Clases del paquete Facade

El paquete Facade tiene todas las clases, como su propio nombre indica, que conforman el patrón Facade. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete Facade tiene las siguientes clases e interfaces:

- Clase FachadaLogin: esta clase simplifica el acceso a las diferentes funcionalidades que se ponen a disposición del usuario a la hora de acceder a la interfaz de login, tales como, el inicio de sesión, registro de usuario o la comprobación de los diferentes formularios.

- Clase ComprobarFormulario: clase que agrupa las funcionalidades para la comprobación de los formularios, tiene funciones para comprobar tanto el formulario de inicio de sesión y el de registro y mostrar alertas en caso de que se detecte algún error a la hora de enviar el formulario.

- Clase ResetCampos: clase que agrupa las funcionalidades para los diferentes botones de borrar los campos de los formularios.

- Clase RegistrarUsuario: clase que agrupa las funcionalidades para el registro de nuevos clientes.

- Clase IniciarSesion: clase que agrupa las funcionalidades para la comprobación del tipo de usuario que está accediendo a la aplicación, pudiendo ser este un cliente o un empleado.

4.7. Paquete Articulos

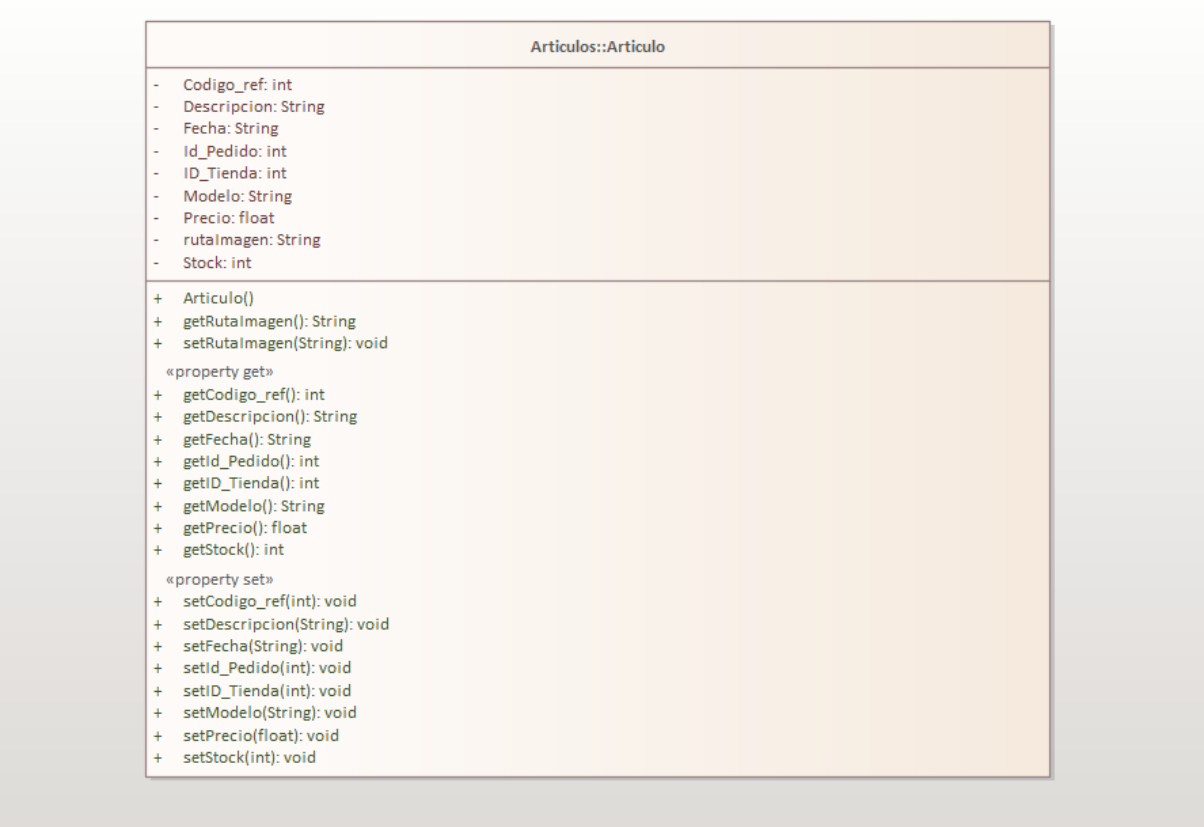

Fig 37. Clase Articulo

La clase Articulo es la clase padre de todos los productos que se pueden comprar en la tienda y tiene atributos, los cuales compartirán todos los demás productos de la tienda, como son Codigo_ref que es el código único que tiene un artículo, Descripcion, Fecha, ID_Tienda, Modelo, Precio, rutaImagen que es la imagen que se mostrará en la interfaz a la hora de cargar el producto y Stock.

Los métodos de esta clase son únicamente los getter y setter para poder manejar desde otra clase el valor de los atributos.

Como hemos explicado, habrá diferentes productos los cuales heredan de la clase Articulo, cada uno de estos productos, a parte de los atributos comunes anteriormente citados, tienen otros atributos específicos de para ese artículo

A continuación, se muestran las demás clases de este paquete, en donde podemos observar los atributos específicos de cada uno de los productos:

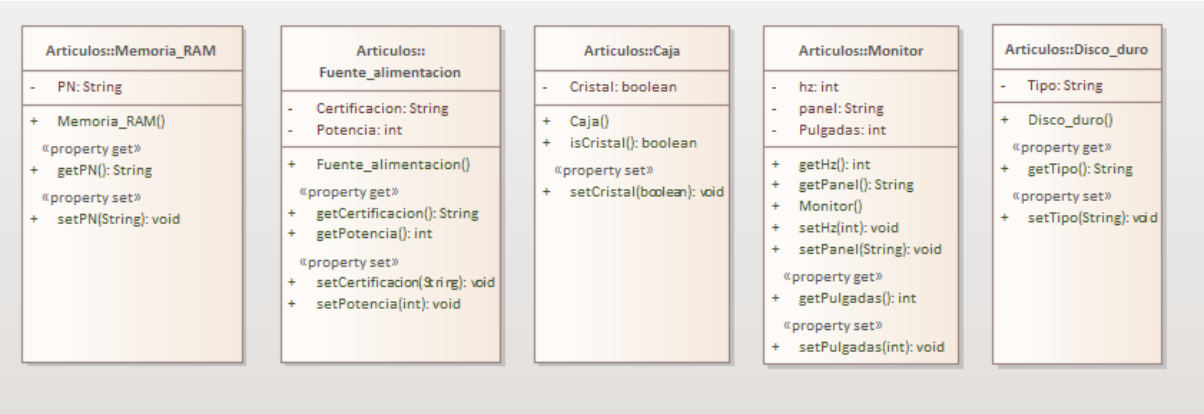

Fig 38. Clases RAM, Fuente, Caja, Monitor, Disco_Duro

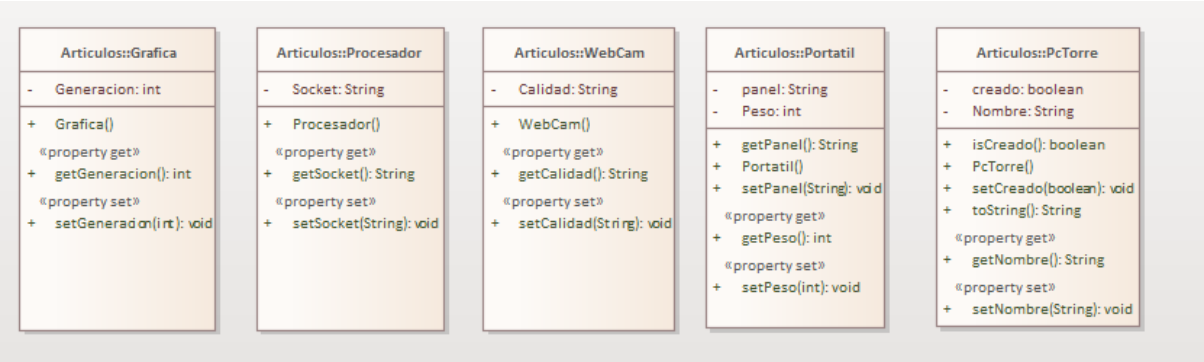

Fig 39. Clases Grafica, Procesador, WebCam, Portatil, PcTorre

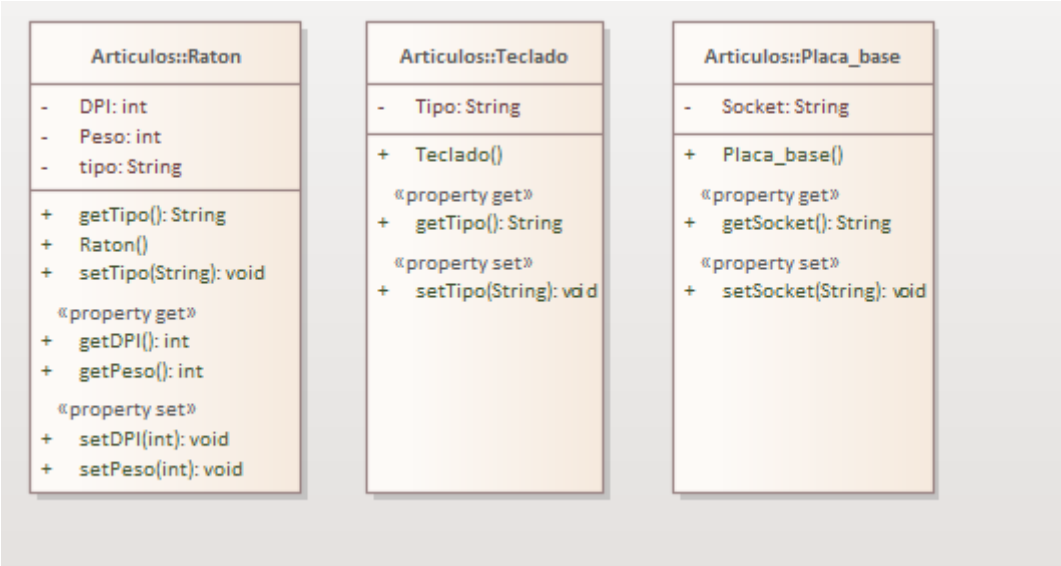

Fig 40. Clases Raton, Teclado, PlacaBase

4.8. Paquete Usuario

Fig 41. Clase Usuario

La clase Usuario es la clase padre de todos los usuarios que pueden acceder a la aplicación, los cuales son empleados y clientes. Tiene atributos comunes a los dos usuarios como son Nombre, Apellido, Direccion, Email, ID_Tienda que es la tienda a la que se conecta el usuario o en la tienda en la que trabaja el empleado, Pass que es la contraseña con la cual accede el usuario a la aplicación y Telefono.

Tiene métodos getter y setter para poder acceder y operar desde otra clase con los valores de las variables de los objetos, ya que, como se puede ver en la imagen, los atributos de la clase Usuario son privados.

Fig 42. Clase Cliente

La clase Cliente hereda los métodos y atributos de la clase Usuario, añade el atributo Tarjeta y sus getter y setter correspondientes.

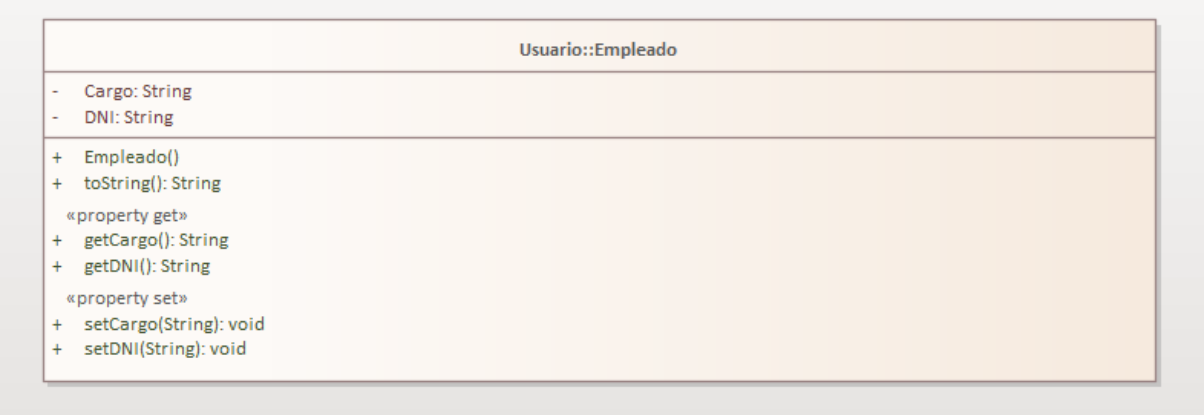

Fig 43. Clase Empleado

La clase Empleado hereda los métodos y atributos de la clase Usuario, añade el atributo DNI y Cargo el cual guarda el cargo que desempeña el empleado en la tienda, y sus getter y setter correspondientes.

Fig 44. Clase Tienda

La clase Tienda guarda la información referente a la tienda a la que el usuario está accediendo bien sea para consultar el catálogo o comprar algún producto, en el caso de cliente, o para gestionar los productos y catálogo, en el caso del empleado.

Esta clase tiene los atributos codigo_postal, ciudad, dirección, id_tienda que será un identificador único para la tienda en específico, nombre y provincia, además de los diferentes métodos getter y setter para poder acceder y modificar los valores de los atributos de la clase, ya que estos son privados.

4.9. Paquete Observer

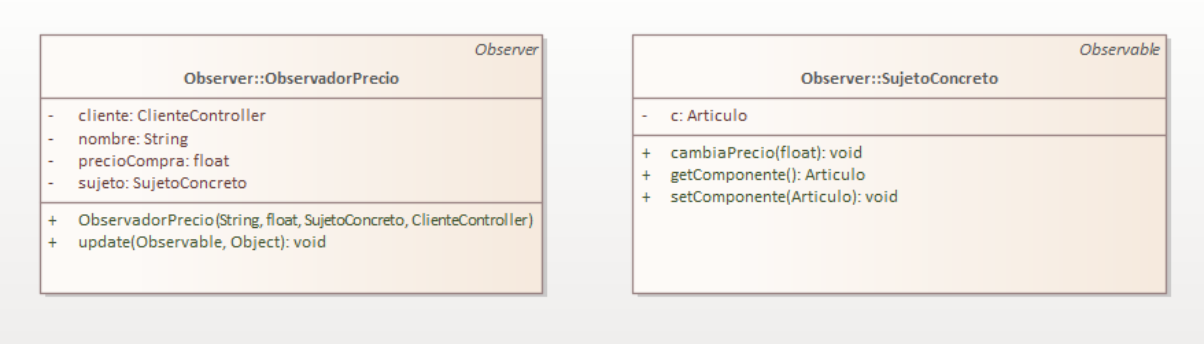

Fig 45. Clases del paquete Observer

El paquete Observer tiene todas las clases, como su propio nombre indica, que conforman el patrón Observer. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete Observer tiene las siguientes clases e interfaces:

- Clase ObservadorPrecio: implementa la interfaz Observer y mantiene una referencia al objeto SujetoConcreto. Tiene el método update() de la interfaz Observer que en el caso de detectar que se cumple la condición impuesta para el observador de ese artículo, mostrará una alerta.

- Clase SujetoConcreto: extiende la clase Observable y se encarga de gestionar los observadores de un componente en concreto. Tiene los métodos getter y setter para poder establecer o recuperar el componente y el método cambiarPrecio() que cada vez que se llama con un nuevo precio notifica a los observers de este cambio, para en caso de cumplir la condición que se active el observador.

4.10. Paquete ProxyLogin

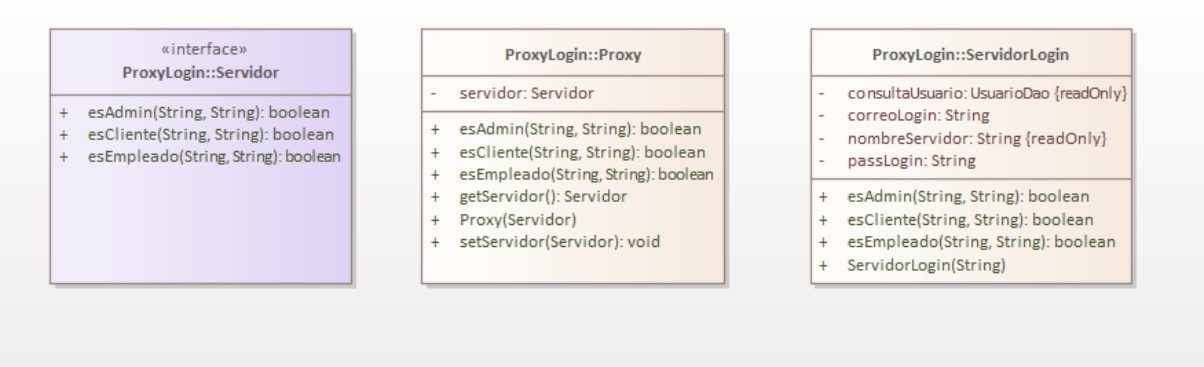

Fig 46. Clases del paquete ProxyLogin

El paquete ProxyLogin tiene todas las clases, como su propio nombre indica, que conforman el patrón Proxy. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete ProxyLogin tiene las siguientes clases e interfaces:

- Interfaz Servidor: esta interfaz define los métodos que usarán las clases Proxy y ServidorLogin, estos métodos son esCliente(), esEmpleado(), esAdmin() y sirven para comprobar que tipo de usuario está accediendo a la aplicación.

- Clase Proxy: implementa la interfaz Servidor y redirige las llamadas de los métodos al objeto real, el cual es el creado por la clase ServidorLogin.

- Clase Servidorlogin: implementa la interfaz Servidor e implementa los servicios reales que ofrece la interfaz Servidor, los cuales son, como hemos dicho anteriormente, esAdmin(), esCliente(), esEmpleado().

4.11. Paquete SingletonLog

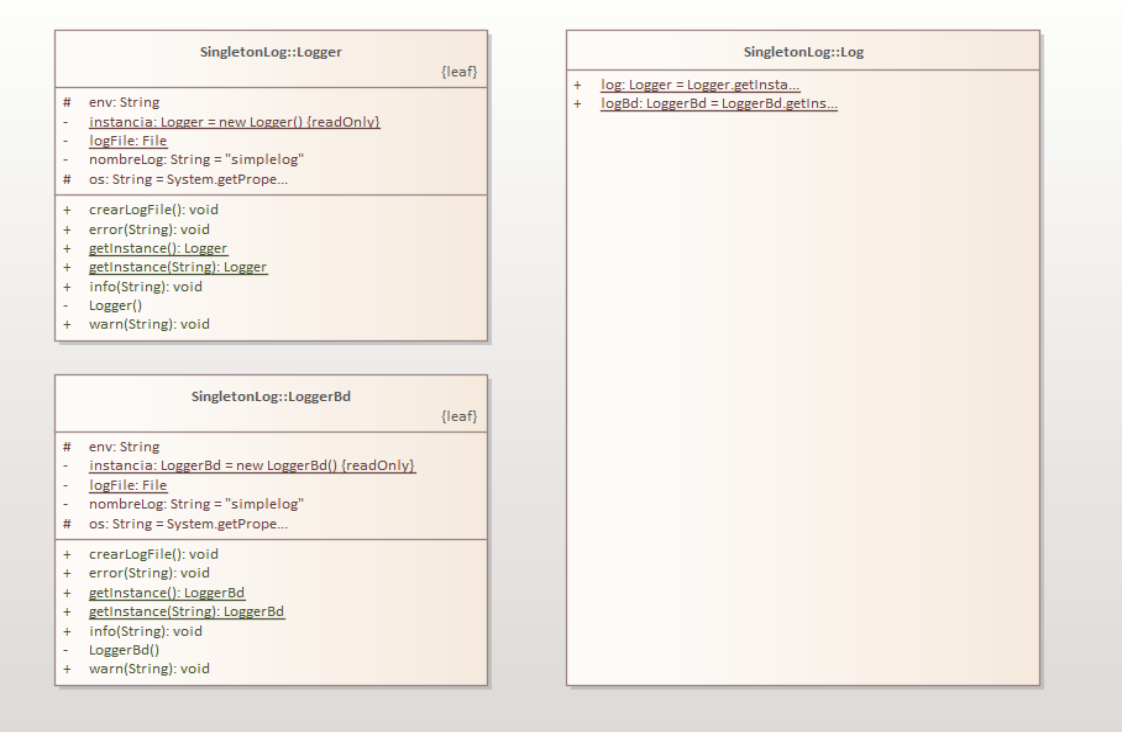

Fig 47. Clases del paquete SingletonLog

El paquete SingletonLog tiene todas las clases, como su propio nombre indica, que conforman el patrón Singleton. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete SingletonLog tiene las siguientes clases e interfaces:

- Clase Log: esta clase guarda las instancias de los logs, logger y loggerBd, para que puedan ser accedidos de forma global por cualquier clase del programa.

- Clase Logger: clase que crea el singleton log para la traza de la aplicación.

- Clase LoggerBd: clase que crea el singleton log para la base de datos.

4.12. Paquete StatePedido

CLASE PEDIDO

Fig 48. Clase Pedido

La clase Pedido guarda la información un pedido realizado por el cliente, con los atributos email_cliente el email del cliente que realiza el pedido, estado que es un objeto Estado el cual implementa el patrón State y ya se ha explicado con anterioridad, fecha, idPedido que es el id único para ese pedido, listaArticulos que es un ArrayList con los id de los artículos que componen el pedido y precio_total.

Además de los diferentes métodos getter y setter para poder acceder y modificar los valores de los atributos de la clase, ya que estos son privados.

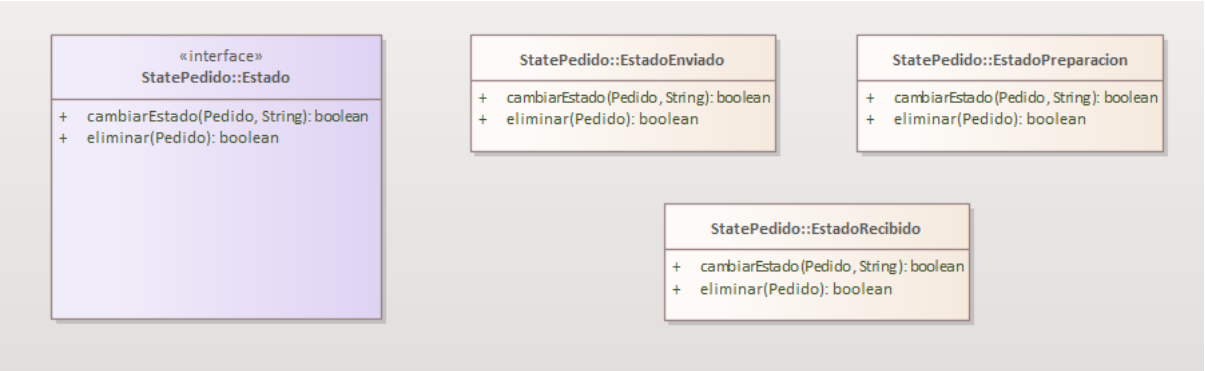

Fig 49. Otras Clases del paquete StatePedido

El paquete StatePedido tiene todas las clases, como su propio nombre indica, que conforman el patrón State. El funcionamiento general de las clases y del patrón ya ha sido explicado en el anterior apartado.

El paquete StatePedido tiene las siguientes clases e interfaces:

- Interfaz Estado: es la interfaz que define los métodos que dependen del estado del objeto, estos métodos son cambiarEstado(pedido, siguienteEstado) el cual cambia el estado, si se puede, de un pedido al siguiente estado y eliminar() que elimina, si se puede, el pedido seleccionado.

- Clase EstadoEnviado: implementa la interfaz Estado y además implementa el comportamiento para el estado enviado.

- Clase EstadoPreparacion: implementa la interfaz Estado y además implementa el comportamiento para el estado preparacion.

- Clase EstadoRecibido: implementa la interfaz Estado y además implementa el comportamiento para el estado recibido.

4.13. Paquete Util

CLASE CONEXION

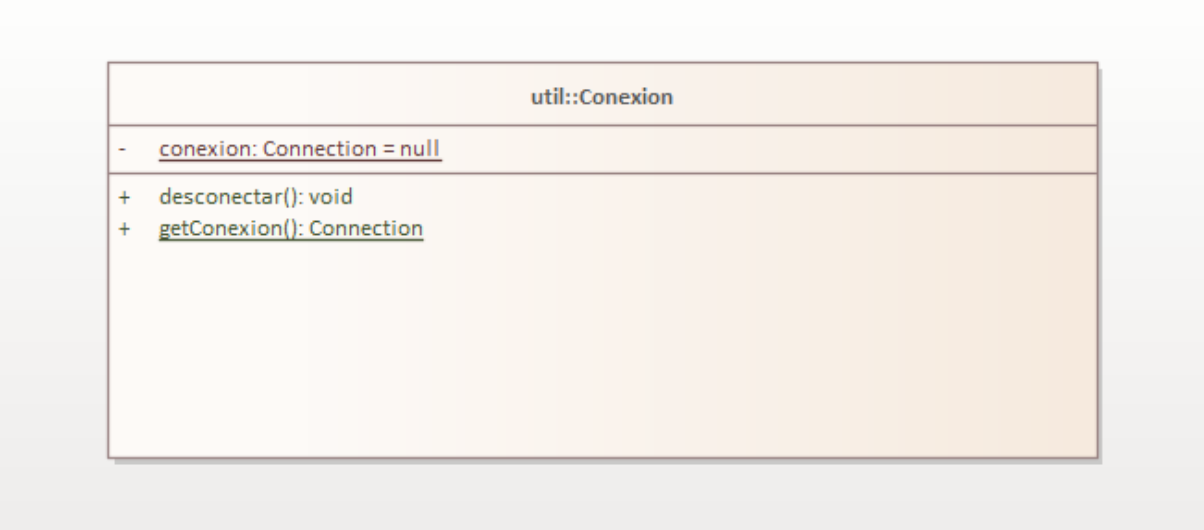

Fig 50. Clase Conexion

Esta clase es la encargada de gestionar la conexión con la base de datos, implementa dos métodos que son, getConexion() el cual si ya hay una conexión activa retorna la conexión actual, pero si no hay ninguna conexión crea una nueva y desconectar() que elimina la conexión actual.

CLASE LISTADINAMICA Y LISTADINAMICAIMAGEN

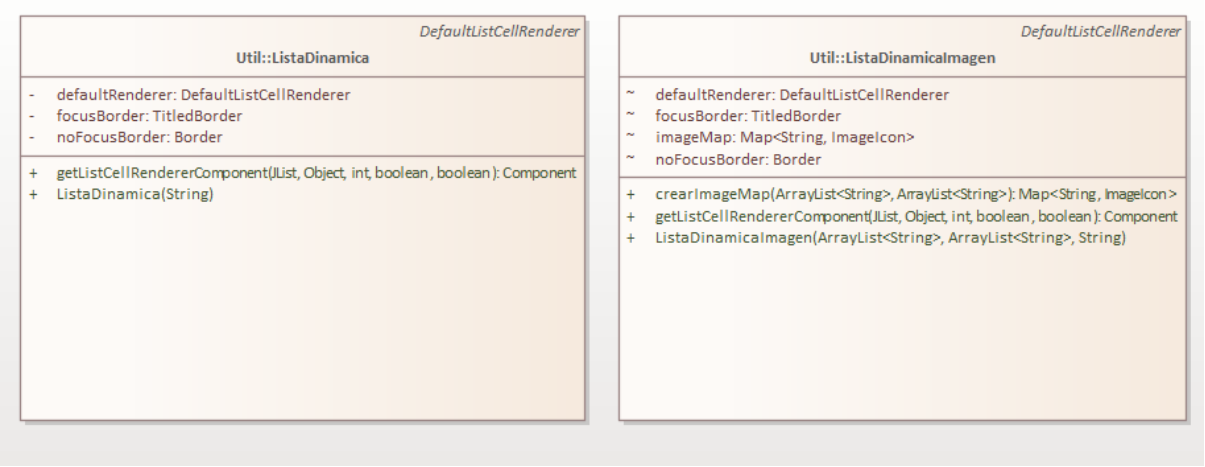

Fig 51. Clase ListaDinamica y ListaDinamicaImagen

Las clases ListaDinamica y ListaDinamicaImagen su funcionalidad es la de renderizar un componente jList de la librería java. Swing con componentes jLabel en cada uno de los ítems del componente jList.

La clase ListaDinamica tendrá únicamente en el componente jLabel información que queremos mostrar, ya sea, en el caso de la lista de pedidos, información referente al pedido.

Fig 52. Lista pedidos de cliente con ListaDinamica

Por otro lado, la clase ListaDinamicaImagen renderizará el jLabel, aparte de con la información que queremos mostrar, con una imagen elegida por nosotros. Para guardar tanto la información como el objeto ImagenIcon con la imagen a mostrar, se utiliza el método crearImageMap() que crea un Map con la información y el ImageIcon.

Fig 53. Lista de articulos de la tienda con ListaDinamicaImagen

4.14. Paquete Views

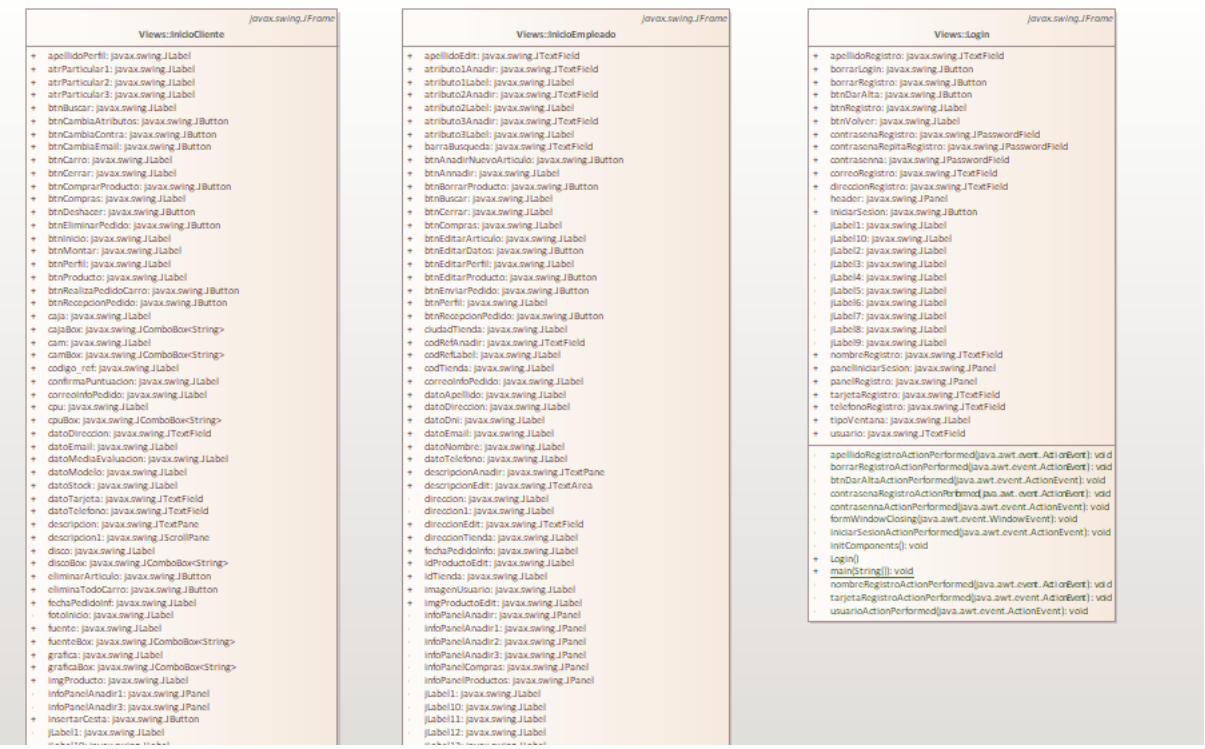

Fig 54. Clases del paquete Views

En el paquete Views se encuentran todas las interfaces de usuario que hay en la aplicación estas son la interfaz de login, la del cliente y la del empleado.

El funcionamiento de las interfaces se explicará más adelante.

5. Diagramas

5.1. Diagrama de componentes

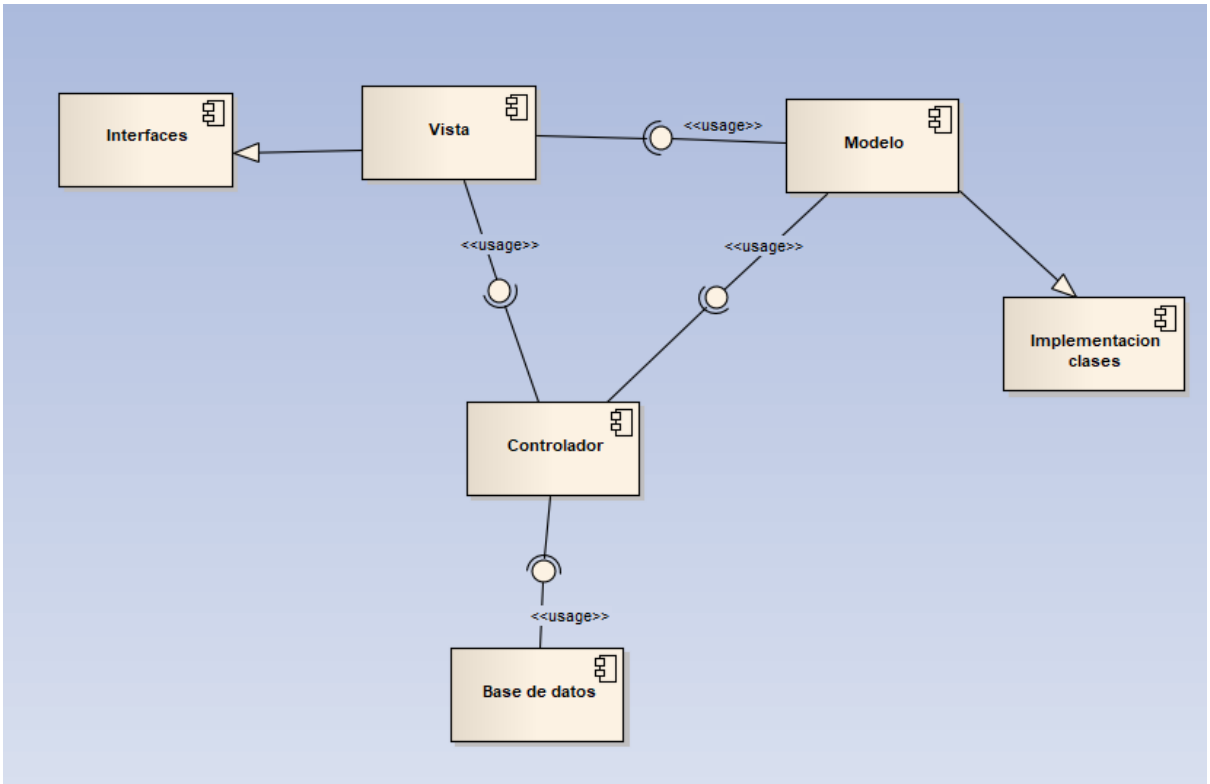

Fig 55. Diagrama de componentes de la aplicación

El modelo está basado en el patrón estructural MVC. Por ello, dispondrá de tres componentes:

- Modelo: Es la representación de la información con la cual el sistema opera, el modelo encapsula el estado de la aplicación sin saber nada de las categorías Vista y Controlador. Las peticiones de acceso o modificación de la información llegan a través del controlador.

- Vista: Presenta el modelo en un formato adecuado para el usuario, por lo general, son las interfaces de usuario, representando la apariencia de la aplicación.

- Controlador: Es quien responde a los eventos, es decir, las acciones del usuario y es el encargado de crear y asignar valores al Modelo para su funcionamiento dado que es el intermediario entre la vista y el modelo.

Se ha utilizado este patrón ya que permite separar las vistas, realizadas mediante la librería java. Swing, de la lógica de negocio, incrementando así la reutilización y flexibilidad del código. Además, permite gestionar el flujo de control de la aplicación en función de las acciones que realice el usuario. Por lo que obtendremos un mantenimiento de la aplicación más sencillo que modelos de una sola capa.

5.2. Diagramas de casos de uso

5.2.1. Caso de uso Iniciar sesión

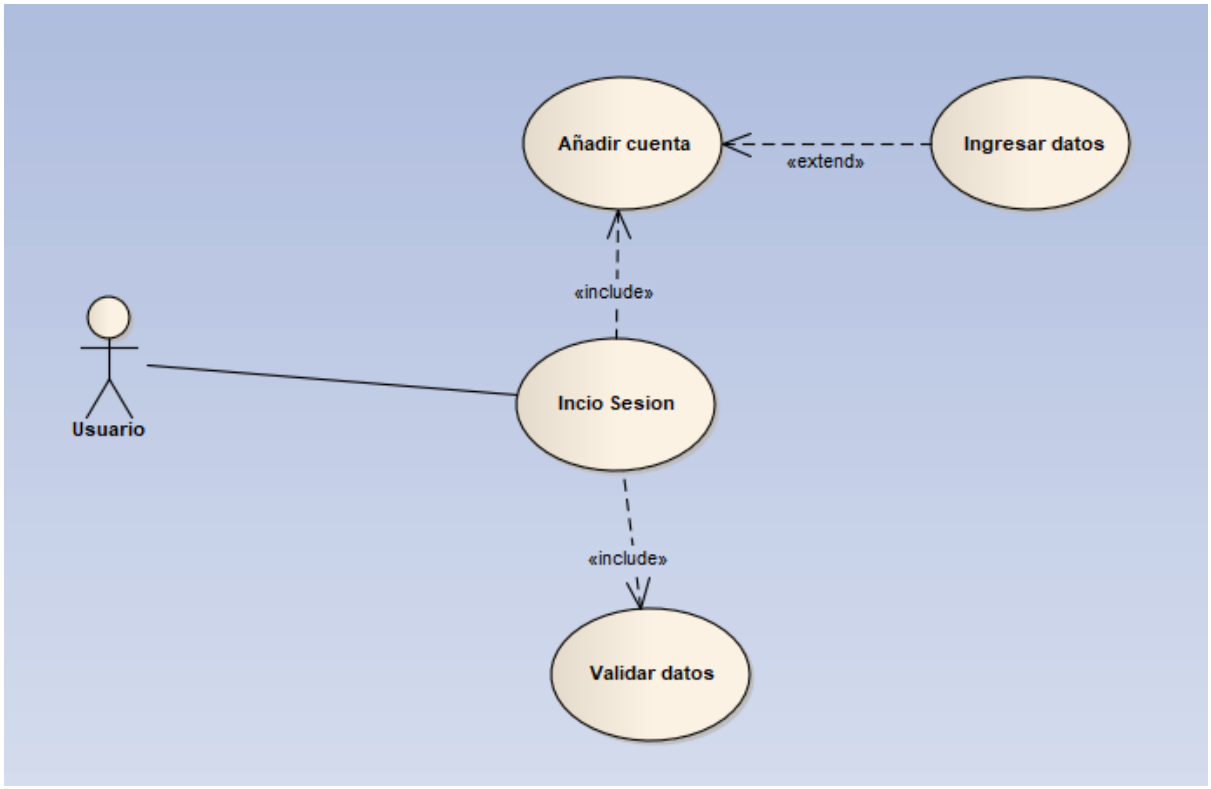

Fig 56. Caso de uso Iniciar sesión

5.2.2. Caso de uso Comprar

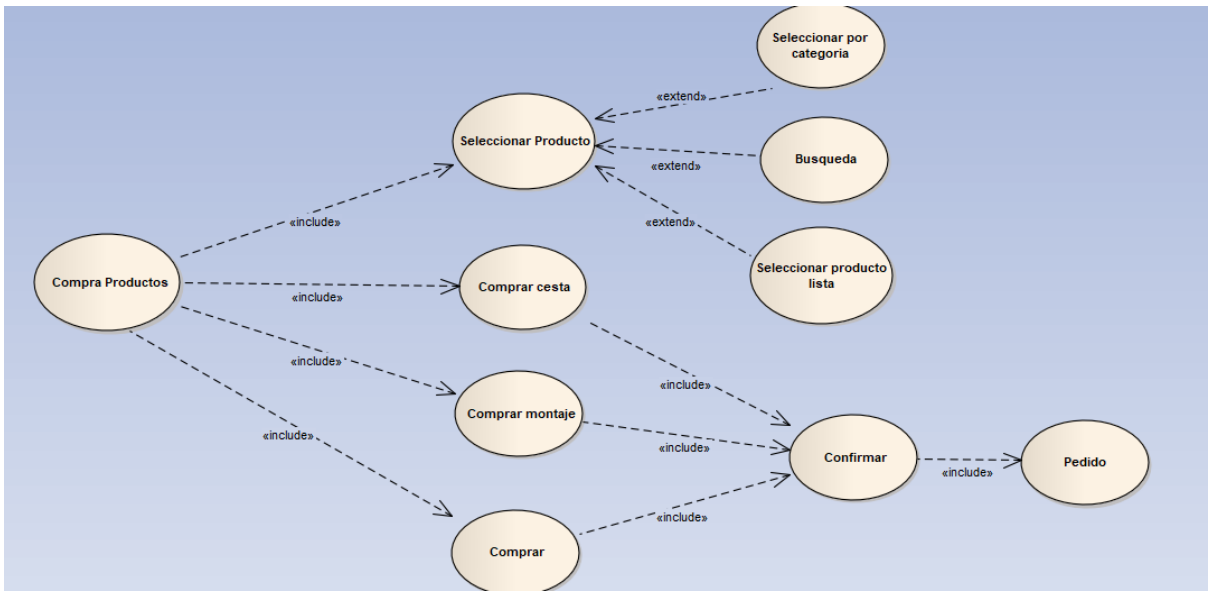

Fig 57. Caso de uso de Comprar

5.2.3. Caso de uso añadir a la cesta

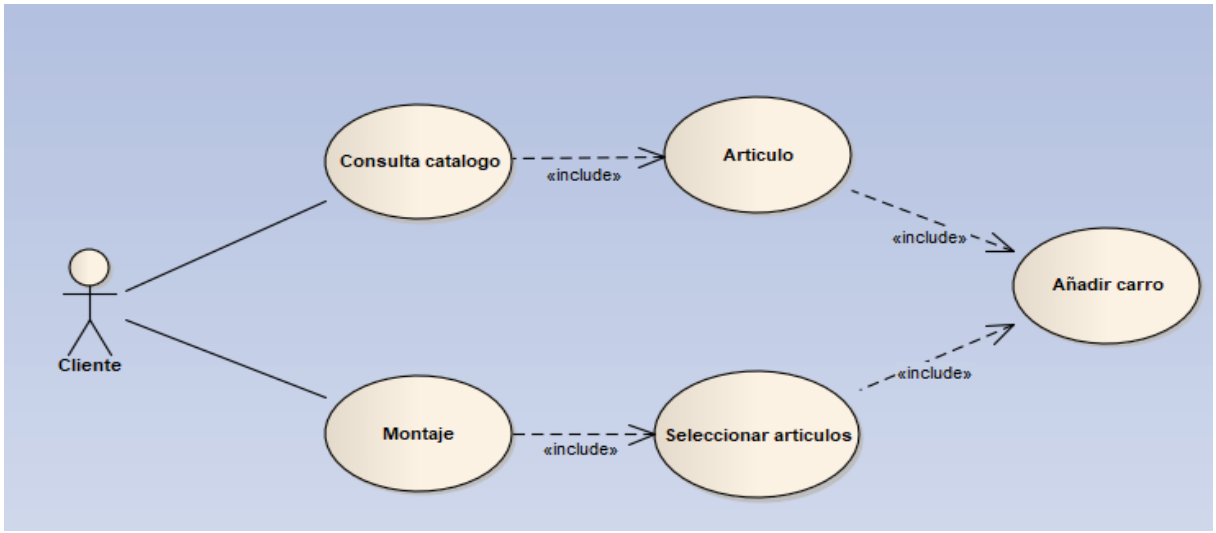

Fig 58. Caso de uso Añadir a la cesta

5.2.4. Caso de uso visualización articulo

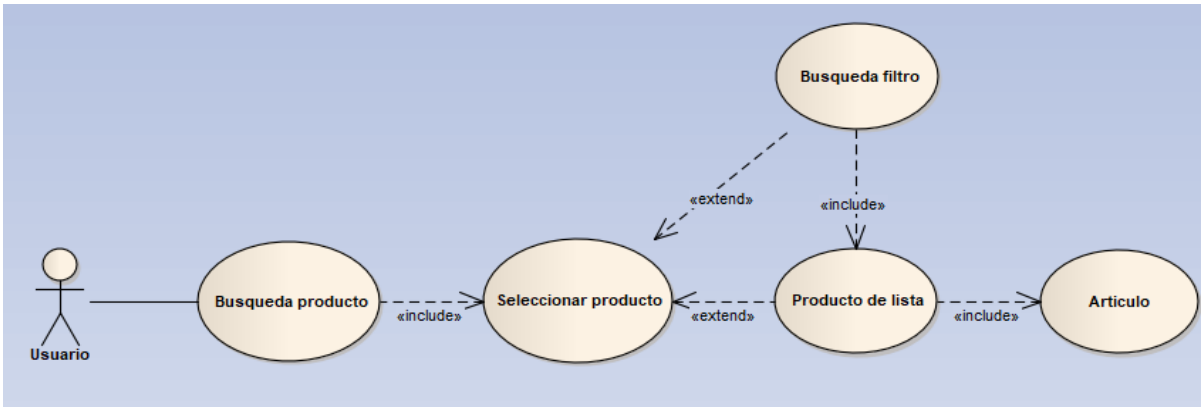

Fig 59. Caso de uso Búsqueda/selección de un artículo

5.2.5. Caso de uso editar perfil

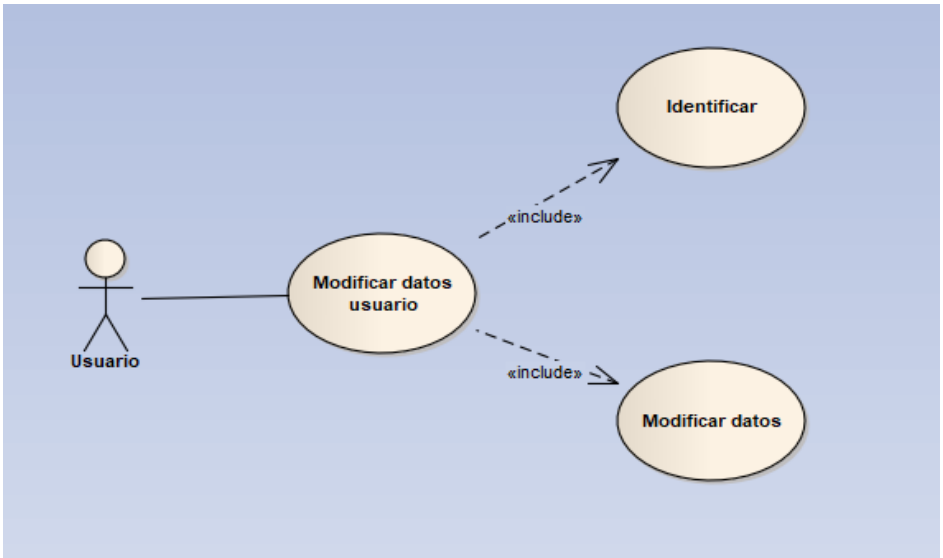

Fig 60. Caso de uso de Editar perfil

5.2.6. Caso de uso modificación artículo

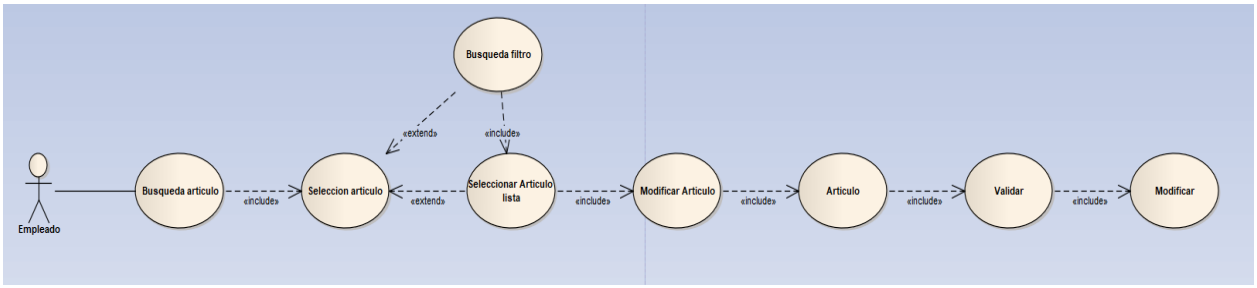

Fig 61. Caso de uso Modificación de artículo

5.2.7. Caso de uso estado de pedido

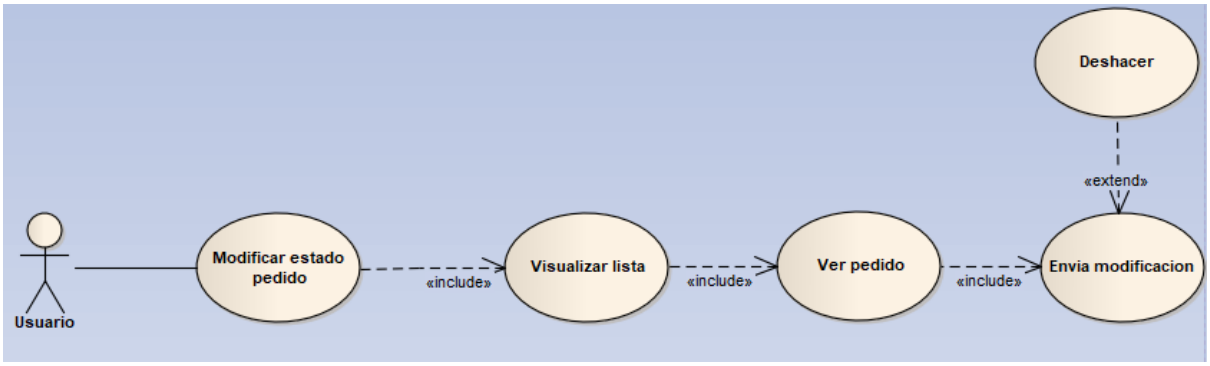

Fig 62. Caso de uso Estado de pedido

5.2.8. Caso de uso añadir artículo

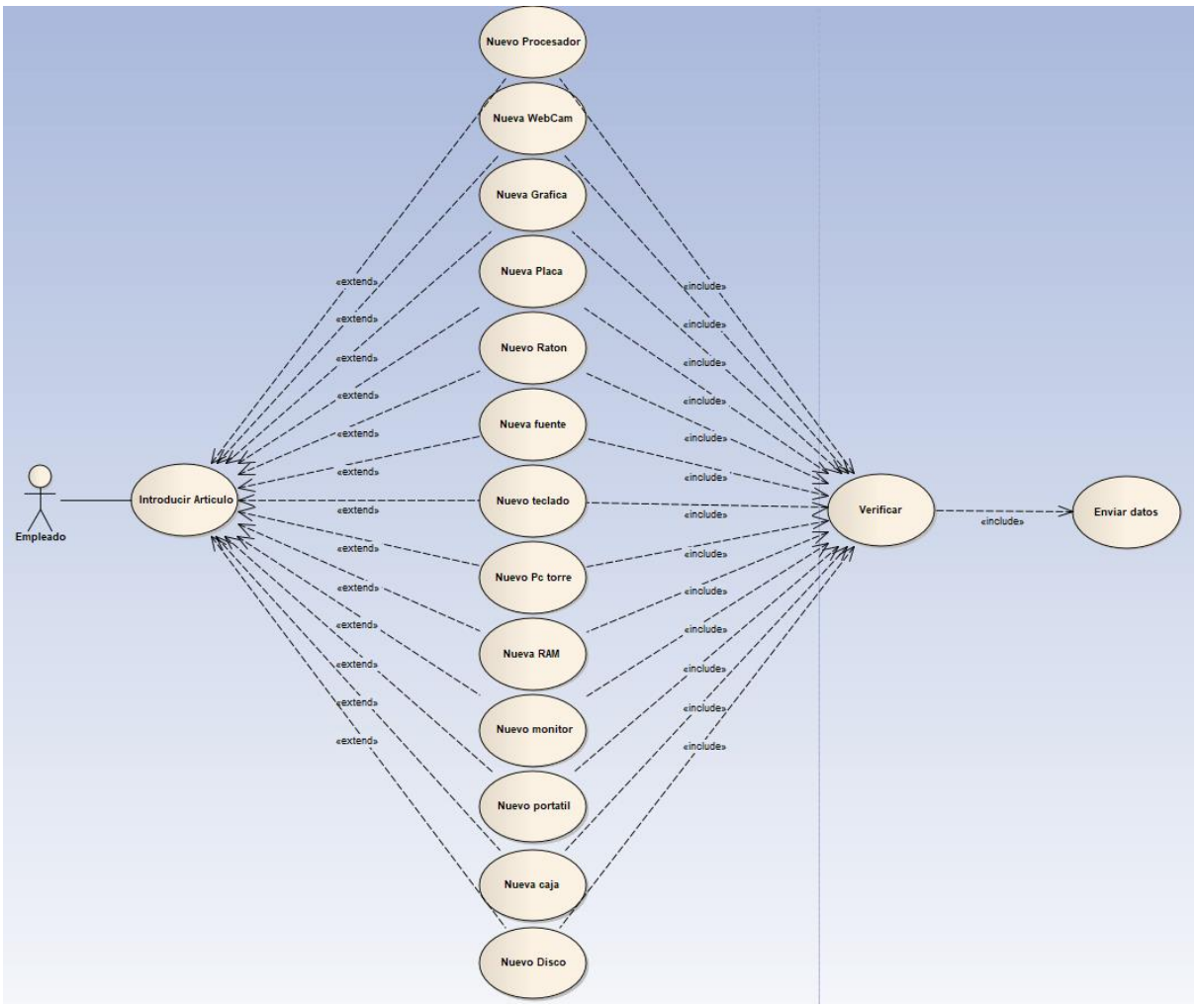

Fig 63. Caso de uso Introducir nuevo artículo

6. Instalación y manual de usuario de la aplicación

6.1. Base de datos

Esta aplicación emplea una base de datos PostgreSQL, para almacenar tanto los usuarios registrados en el sistema, como el catálogo de productos disponibles, así como las cestas y pedidos registrados para cada usuario.

Para su instalación es necesario ir a la página https://www.postgresql.org/download/ seleccionar el sistema operativo de su ordenador y descargar el instalador adecuado.

Fig 64. Instalación de PostgreSQL 1

Debemos seguir los pasos, dando a siguiente y marcando como mínimo PostgreSQL server y PgAdmin 4:

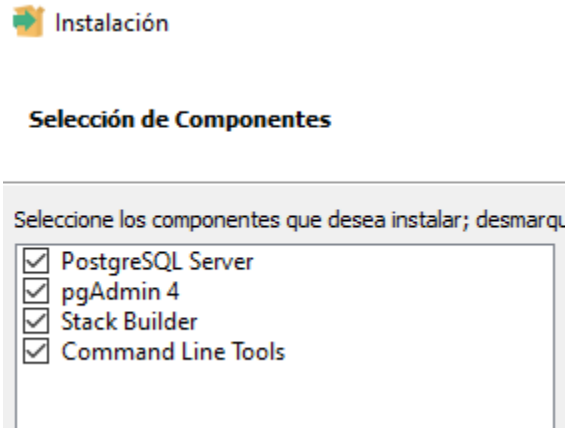

Fig 65. Instalación de PostgreSQL 2

Ingresamos la contraseña que queremos utilizar para acceder al servidor postgres más adelante, que en nuestro caso es “postgres” sin las comillas, y la contraseña “postgres” sin las comillas.
Una vez terminada la instalación, ejecutaremos PgAdmin 4 para acceder a ella con las credenciales que empleamos previamente.

Fig 66. Inicio de pgAdmin 4

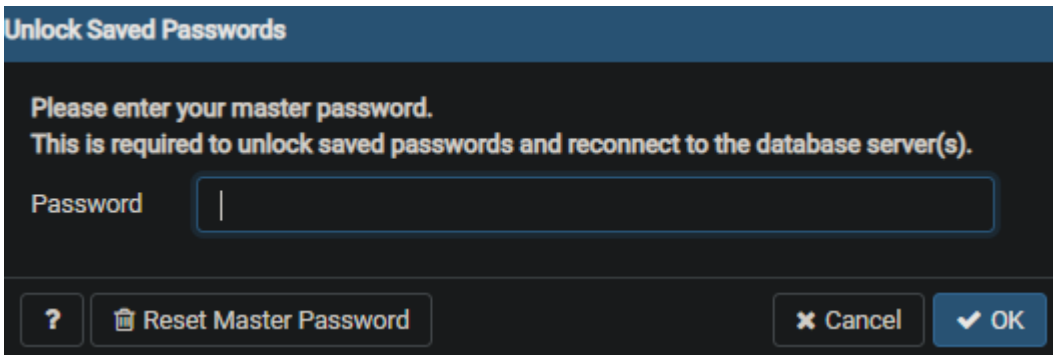

Fig 67. Ingreso de contraseña para inicio de servidor

Es importante que estos campos usuario y contraseña tengan estos mismos valores, ya que son los que utilizará la base de datos.

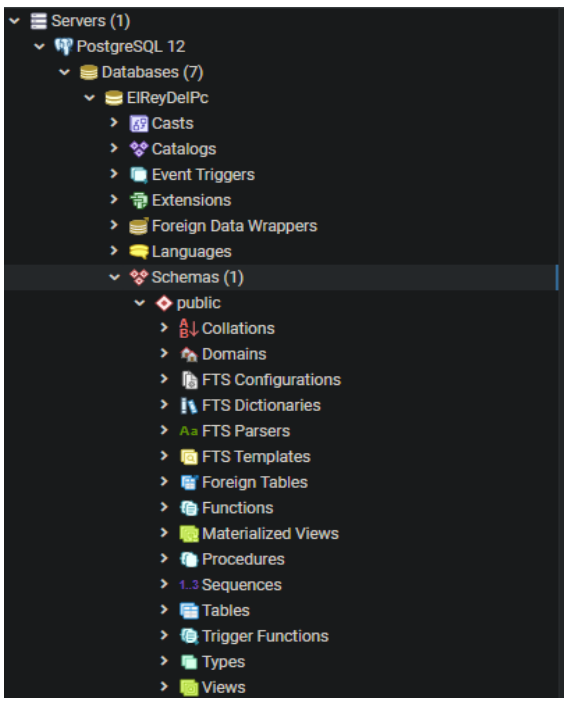

Fig 68. Estructura de base de datos

A continuación, crearemos una base de datos llamada “ElReyDelPc”, con una codificación UTF8.
Junto con el código de la práctica se añade el archivo SQL que emplearemos para crear nuestra base de datos, el cual contiene el esquema con las tablas y relaciones y además una serie de datos de muestra ya introducidos para la prueba del programa.

Para abrirlo, debemos hacer click derecho sobre nuestra base de datos y seleccionar la opción “Query tool...” o bien dar al botón señalado en la siguiente imagen:

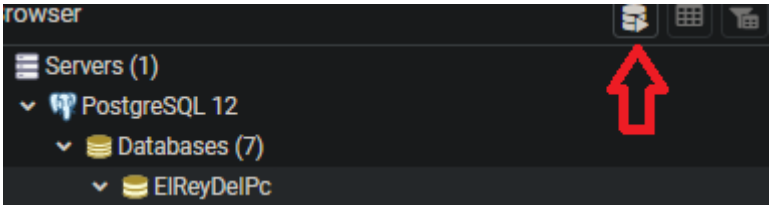

Fig 69. Iniciar nueva query

Posteriormente podremos arrastrar el archivo SQL a la ventana abierta o abrirlo seleccionando el botón de la carpeta de la parte superior y buscarlo en el directorio que esté guardado:

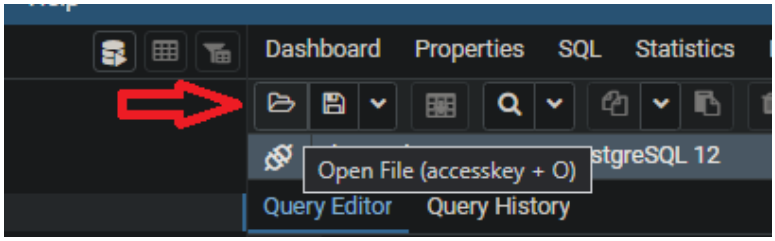

Fig 70. Importación de archivo .SQL

```sql
-- INSERTS --


-- TIENDA Y USUARIOS --

INSERT INTO Tienda VALUES(0, 'ElReyDelPc', 'Direccion1', 28073, 'Ciudad1', 'Provincia1');
INSERT INTO Empleado VALUES('Juan', 'Apellido2', 'email@email.com', 'Direccion2', 621512233, 'pass1',0,'DNI1','Empleado1');
INSERT INTO Cliente VALUES('Pedro', 'Apellido1', 'email2@email.com', 'Direccion1', 621424230, 'pass1',0,'tarjeta1');

INSERT INTO Empleado VALUES('Maria', 'Apellido2', 'prueba@gmail.com', 'DireccionX', 671512237, 'lll',0,'DNI2','Empleado2');
INSERT INTO Cliente VALUES('Sandra', 'Apellido1', 'cliente1@gmail.com', 'Direccion3', 631426239, 'lll',0,'tarjeta2');
INSERT INTO Cliente VALUES('Marta', 'Apellido1', 'cliente2@gmail.com', 'Direccion4', 641424731, 'lll',0,'tarjeta3');
INSERT INTO Cliente VALUES('Julio', 'Apellido1', 'cliente3@gmail.com', 'Direccion5', 651424732, 'lll',0,'tarjeta4');
INSERT INTO Cliente VALUES('Javier', 'Apellido1', 'cliente4@gmail.com', 'Direccion6', 656624234, 'lll',0,'tarjeta5');

-- ARTICULOS --

INSERT INTO Portatil VALUES('Portatil Acer ChromeBook', 1, 350, 'Descripcion1', 2,'/images/portatil1.png',0,'IPS',2);
INSERT INTO Portatil VALUES('Portatil ASUS ChromeBook', 123, 399, 'Descripcion123', 2,'/images/portatil1.png',0,'IPS',2);
INSERT INTO Portatil VALUES('Portatil ASUS TUF Gaming', 124, 899, 'Descripcion124', 3,'/images/portatil1.png',0,'IPS',3);
INSERT INTO Portatil VALUES('Portatil Lenovo iDeaPad 4', 125, 420, 'Descripcion125', 5,'/images/portatil1.png',0,'IPS',2);
INSERT INTO Portatil VALUES('Portatil HP ChromeBook', 126, 500, 'Descripcion126', 1,'/images/portatil1.png',0,'IPS',1);
INSERT INTO Portatil VALUES('Portatil HP Pavilion Gaming', 127, 905, 'Descripcion127', 2,'/images/portatil1.png',0,'IPS',4);
```

Fig 71. Script de creación de la base de datos

Finalmente pulsamos F5 para ejecutar el SQL y crear la base de datos.

6.2. Aplicación

Para la ejecución del programa es necesario tener instalado Java. Podemos descargarlo en la página oficial de Oracle https://www.java.com/es/

Fig 72. Descarga de Java

Posteriormente instalamos NetBeans 8.2 descargándolo desde la página oficial: https://netbeans.org/downloads/old/8.2/ , seleccionando la primera opción.
Tras seguir los pasos y completar la instalación, ejecutamos el programa y abrimos el proyecto proporcionado en File > Open Proyect, situado en la esquina izquierda de la ventana.

Finalmente, hacemos click derecho sobre el proyecto “ElReyDelPc” y damos a “Run”.

A pesar de que la mejor manera de ejecutar la aplicación es la ya mencionada, existe otro modo con la que podemos ejecutar la aplicación desarrollada. Sería mediante el .JAR ubicado en la carpeta “Dist” del proyecto. Pero destacar, que, al ejecutar la aplicación de esta manera, se pierde la imagen de vista previa del producto en las listas.

En caso de que la configuración de postgreSQL no tenga la contraseña y usuario especificados en el apartado anterior, debemos ir al proyecto en NetBeans y modificar la clase “Conexión.java” ubicada en el paquete “Util”. Y cambiar el valor de la variable user y password con su valor correspondiente.

```
public static Connection getConexion() {
    String driver = "org.postgresql.Driver";
    String nombreBd = "ElReyDelPc";
    String puerto = "5432";
    String user = "postgres";
    String password = "postgres";
    String url = "jdbc:postgresql://localhost:"+ puerto +"/"+ nombreBd
```

Fig 73.1. Conexión base datos java

A continuación, se va a proceder a explicar el uso de la aplicación desarrollada.

La primera página mostrada al ejecutar la aplicación es la interfaz de inicio de sesión, donde el usuario debe introducir sus datos (email y contraseña) para acceder a su correspondiente menú.

Fig 74. Interfaz login de usuario

Si el usuario es un cliente y no se ha registrado previamente, deberá dirigirse al formulario de registro pulsando el texto “Registrate aquí” de la parte inferior de la ventana.

Fig 75. Interfaz registro de nuevo cliente

El usuario debe completar los campos correctamente y pulsar “Registrarse”. Tras esto, volverá a la pantalla de inicio de sesión y deberá rellenar los campos con los datos que uso en el formulario anterior.

Si se desea acceder al programa con un usuario de pruebas ya creado, y comprobar el funcionamiento con algunos datos de ejemplo ya introducidos, se han utilizado los siguientes usuarios:

- El usuario de tipo cliente:
  - Email: email2@email.com
  - Contraseña: pass1
- El usuario de tipo empleado:
  - Email: email@email.com
  - Contraseña: pass1

Si iniciamos sesión correctamente podremos acceder a una serie de funcionalidades añadidas en función de qué tipo de usuario sea.

6.2.1. Interfaz cliente

6.2.1.1. Inicio

Cuando nos conectamos como cliente se nos muestra el menú de cliente en la ventana de inicio o bienvenida.

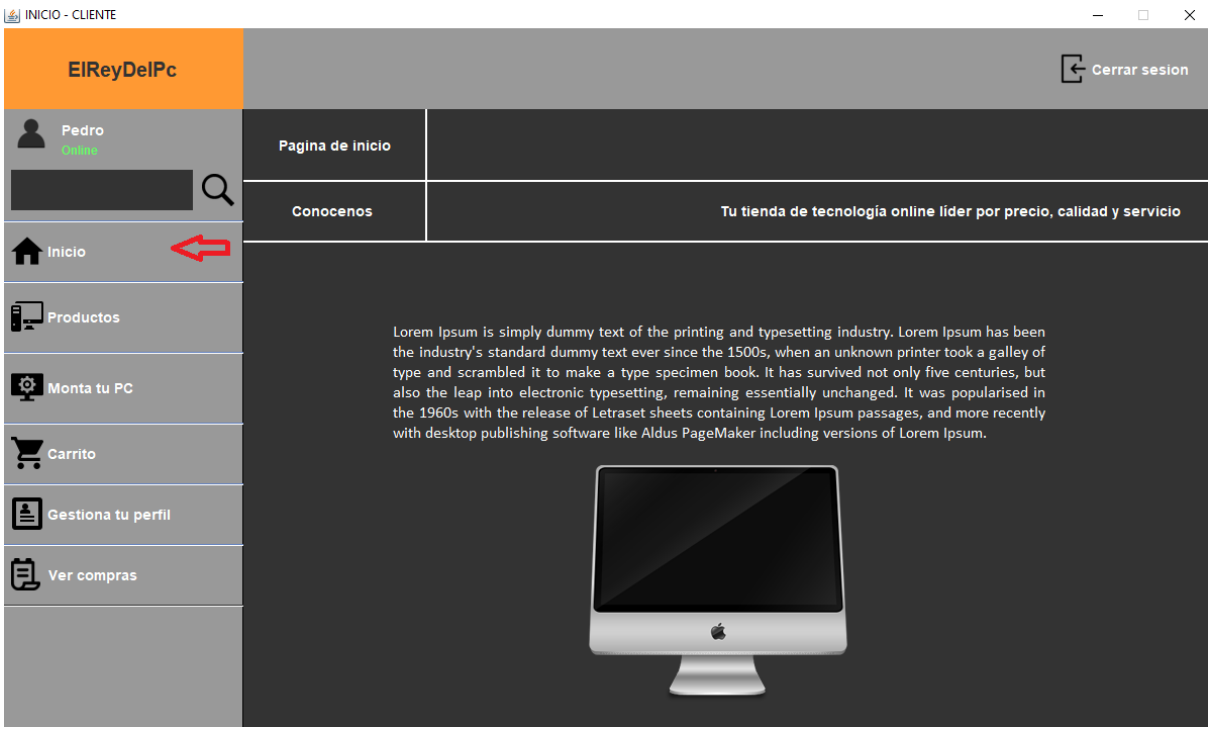

Fig 76. Panel de inicio en interfaz cliente

Podemos navegar por la interfaz con los botones de la parte izquierda de la ventana seleccionando diferentes vistas:

6.2.1.2. Productos

En esta vista podemos encontrar todo el catálogo de artículos disponibles en la tienda clasificados por categorías. Aquí podemos encontrar Placas base, procesadores, graficas, cajas, monitores, teclados, ratones, etc.

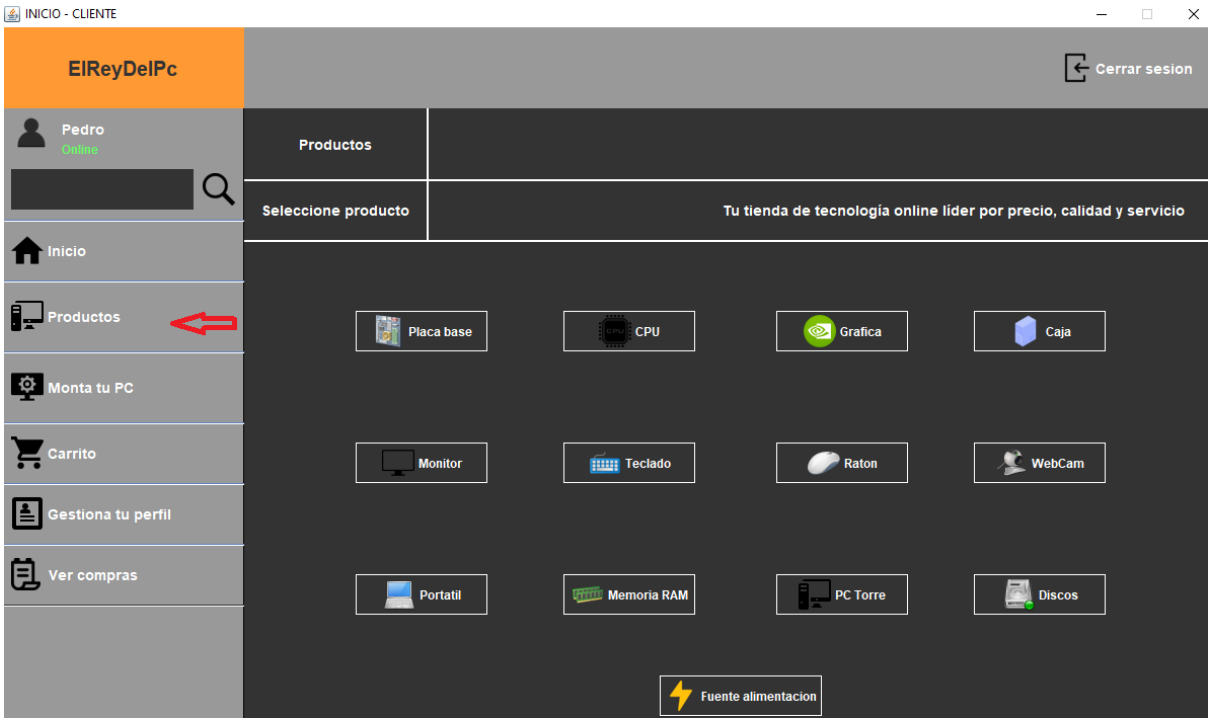

Fig 77. Panel de productos en interfaz cliente

Para acceder a los artículos debemos seleccionar una categoría pulsando sobre ella y se nos mostrará una lista de todos los artículos que hay de ese tipo (ejemplo con placa base):

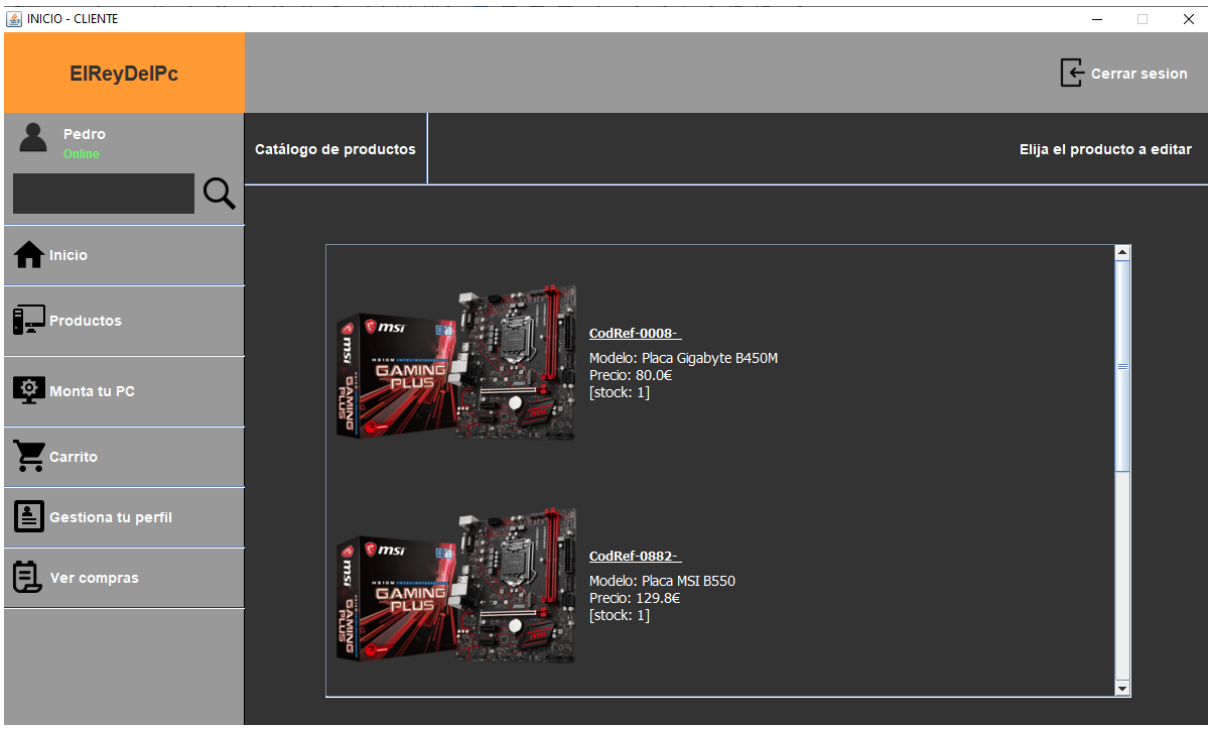

Fig 78. Panel de catálogo productos en interfaz cliente

Seleccionamos un elemento de la lista pulsando sobre él para dirigirnos a la vista del artículo más en detalle.
En la siguiente ventana se nos muestra las características del producto:

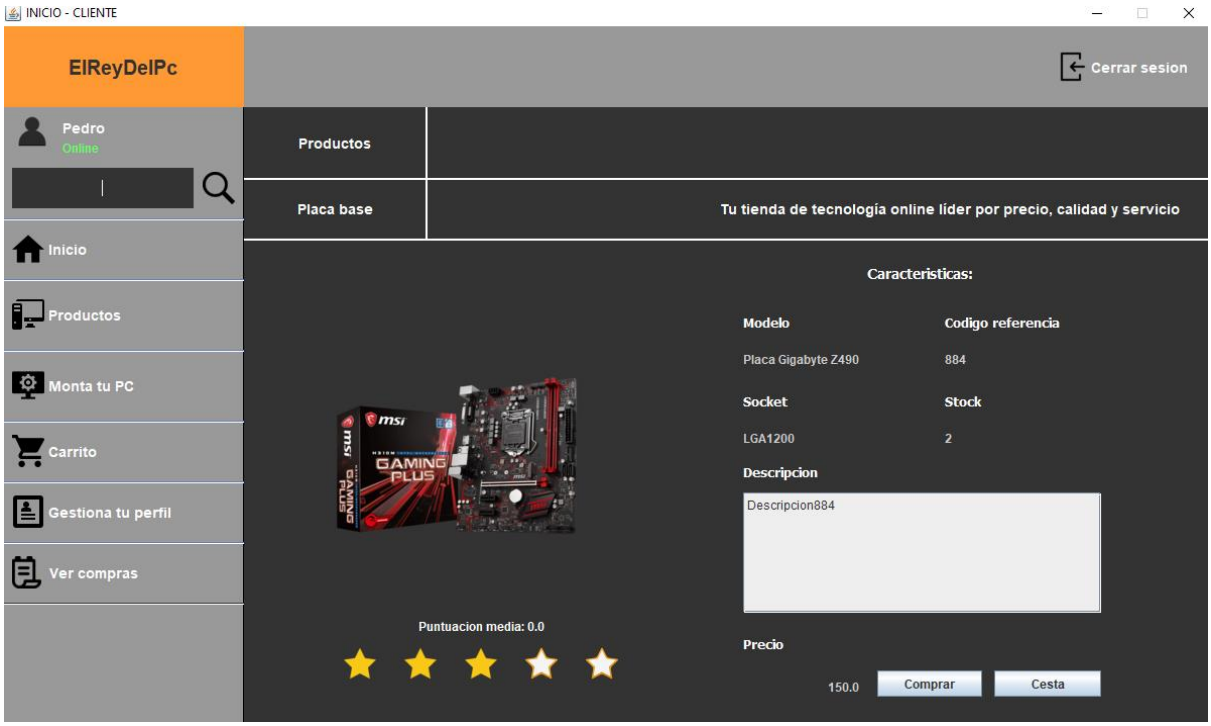

Fig 79. Panel de información de producto en interfaz cliente

En esta ventana podemos observar diferentes opciones o maneras de interactuar con el producto.
Podemos evaluarlo (si no se ha calificado este producto anteriormente) pulsando sobre las estrellas en función de la nota o puntuación que se le quiera otorgar (1-5).

Fig 80. Evaluación de producto y media de evaluación

Cuando se envíe la puntuación no se podrá volver a valorar el artículo.

También, podemos hacer un pedido del producto directamente pulsando sobre el botón de comprar, mostrándose un mensaje de confirmación previo.

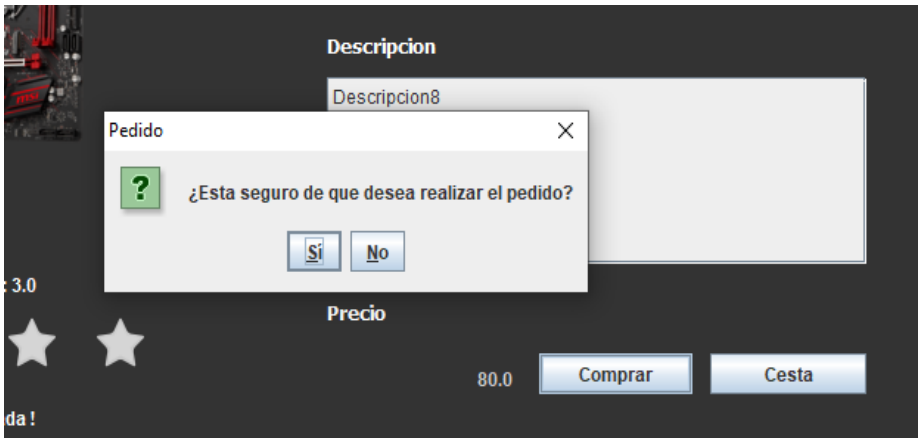

Fig 81. Confirmación de realización de nuevo producto

Por último, en esta ventana podemos guardar el artículo en la cesta o carrito para realizar el pedido en un futuro.

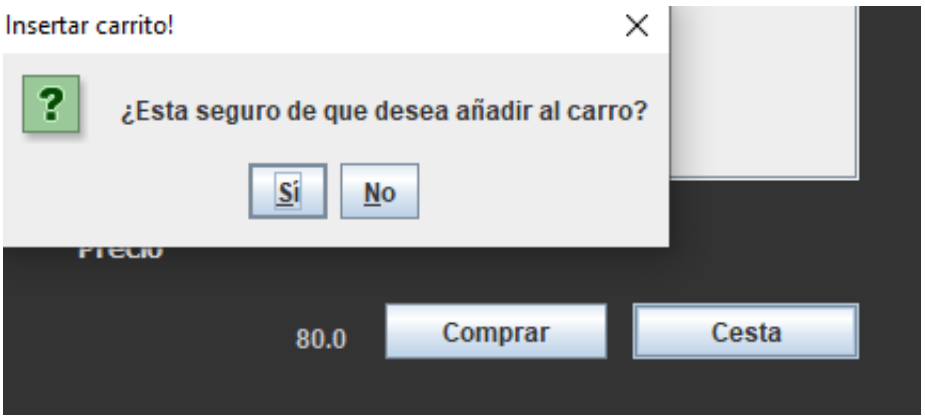

Fig 82. Confirmación de añadir al carro nuevo producto

6.2.1.3. Monta tu PC

En esta vista podemos montar un ordenador de torre completando los campos con la opción seleccionada. Se deben seleccionar todos los campos de la parte izquierda, es decir, son campos obligatorios para la construcción. Después, se pueden añadir los periféricos al pedido, pero son tratados como extras.

Para elegir un producto se debe desplegar el combobox y pulsar una.

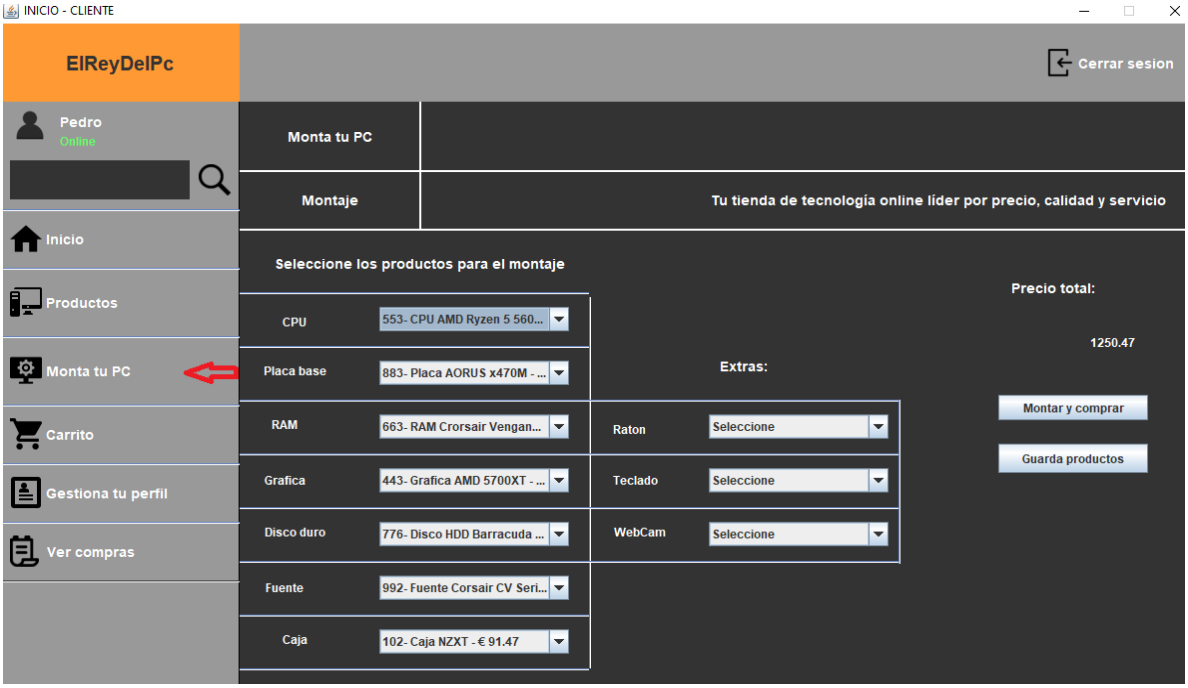

Fig 83. Panel de montar PC en interfaz cliente

A continuación, tenemos dos opciones la primera es realizar el pedido del montaje personalizado pulsando “Montar y comprar”

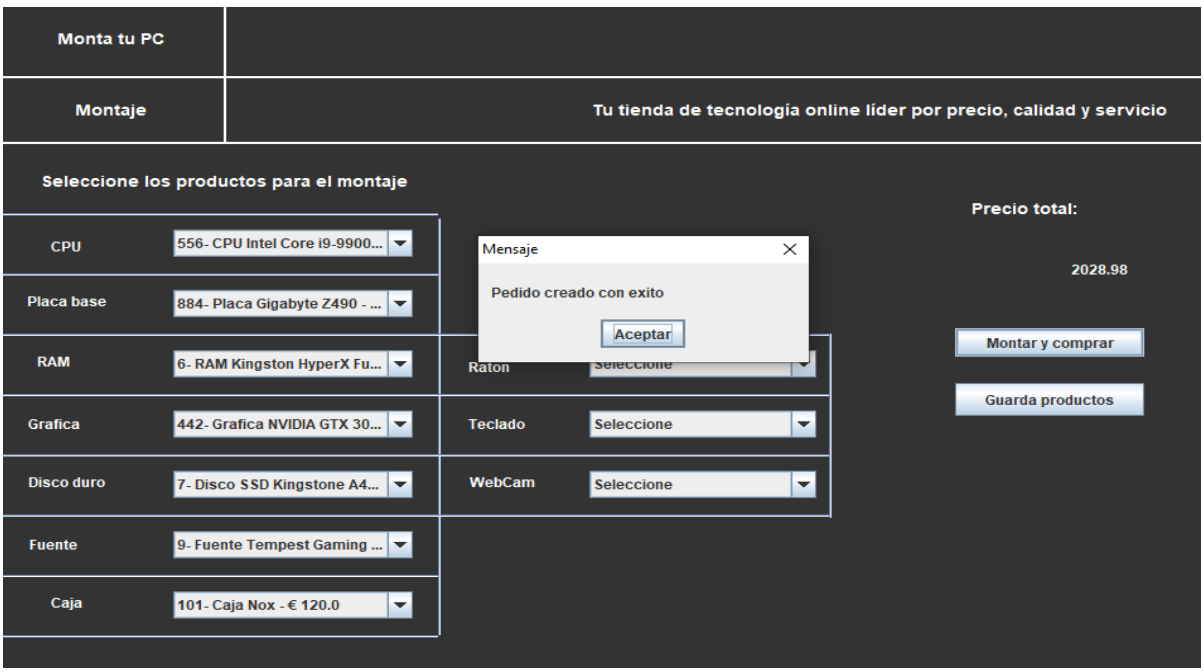

Fig 84. Confirmación de realización de nuevo pedido

Y la otra opción es guardar los productos del montaje en la cesta pulsando “Guarda productos”.

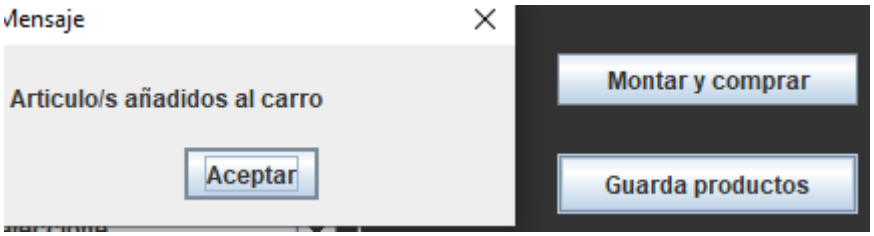

Fig 85. Guardar productos de pc montado

6.2.1.4. Carrito

En esta vista se muestra la cesta o el carrito del cliente, por lo que podemos ver una lista de todos los artículos que se han añadido a la cesta.

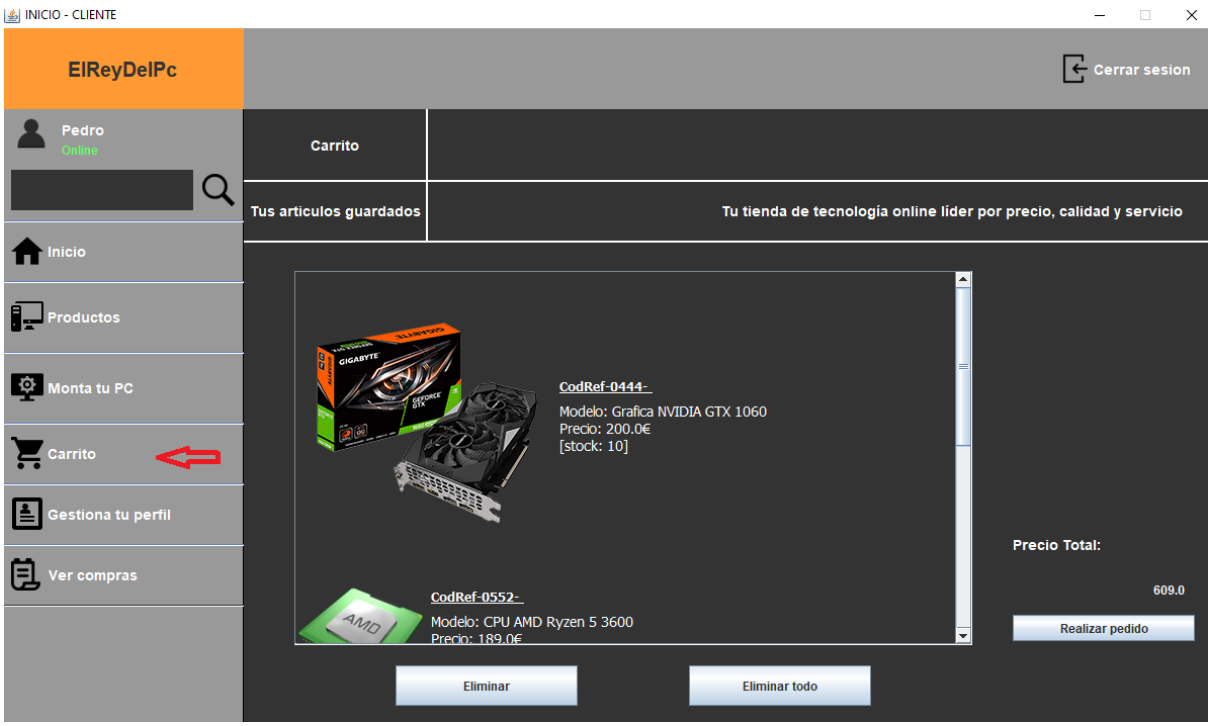

Fig 86. Panel de carrito en interfaz cliente

Disponemos de diferentes opciones: podemos eliminar un producto en concreto de la lista seleccionando y pulsando el botón de eliminar como se muestra a continuación

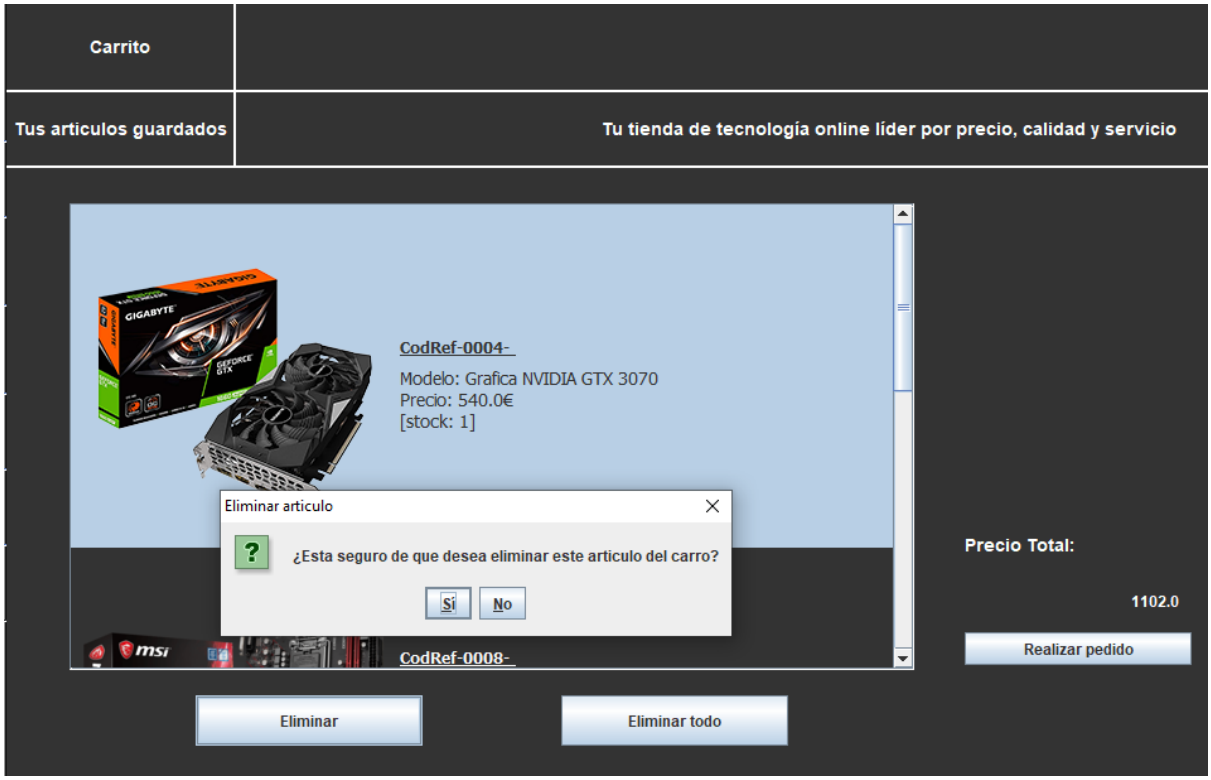

Fig 87. Eliminación de un artículo del carrito

También, podemos eliminar todos los artículos de la cesta para vaciarla con un solo click. Para ello debemos pulsar el botón “Eliminar todo”.
Y finalmente, podemos realizar el pedido de todos los productos de la cesta pulsando el botón de “Realizar el pedido”.

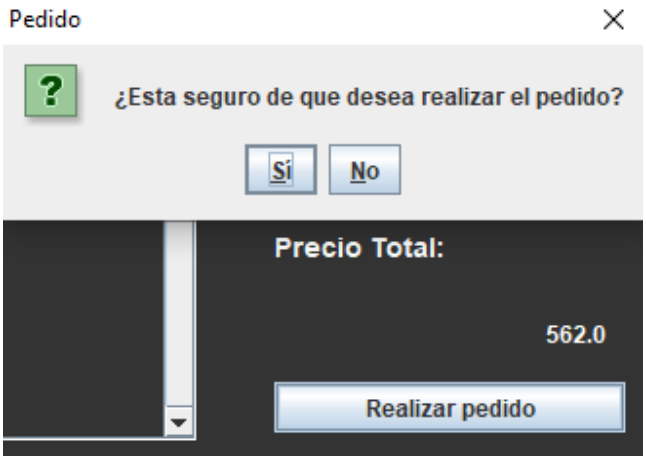

Fig 88. Realización de un pedido desde el carrito

En caso de que no haya stock de algún producto de la cesta, es necesario, eliminarlo para poder hacer el pedido.

6.2.1.5. Gestiona tu perfil

En esta vista podemos ver los datos del usuario que ha iniciado sesión, además podemos editarlos en caso de ser necesario.
Para editar los datos, es necesario hacerlo por partes:

- La primera parte consta de tarjeta, teléfono y dirección. Para editarla y confirmar la edición se debe pulsar el botón “guardar” situado en la parte inferior.
- La segunda parte está compuesta por el email. Para editarla debemos poner un email válido y pulsar el botón “cambiar” situado al lado del textField para introducir el correo.
- La tercera parte es para la contraseña. Para editarla debemos introducir la misma contraseña dos veces y pulsar el botón “cambiar” situado debajo de los campos de nueva contraseña.

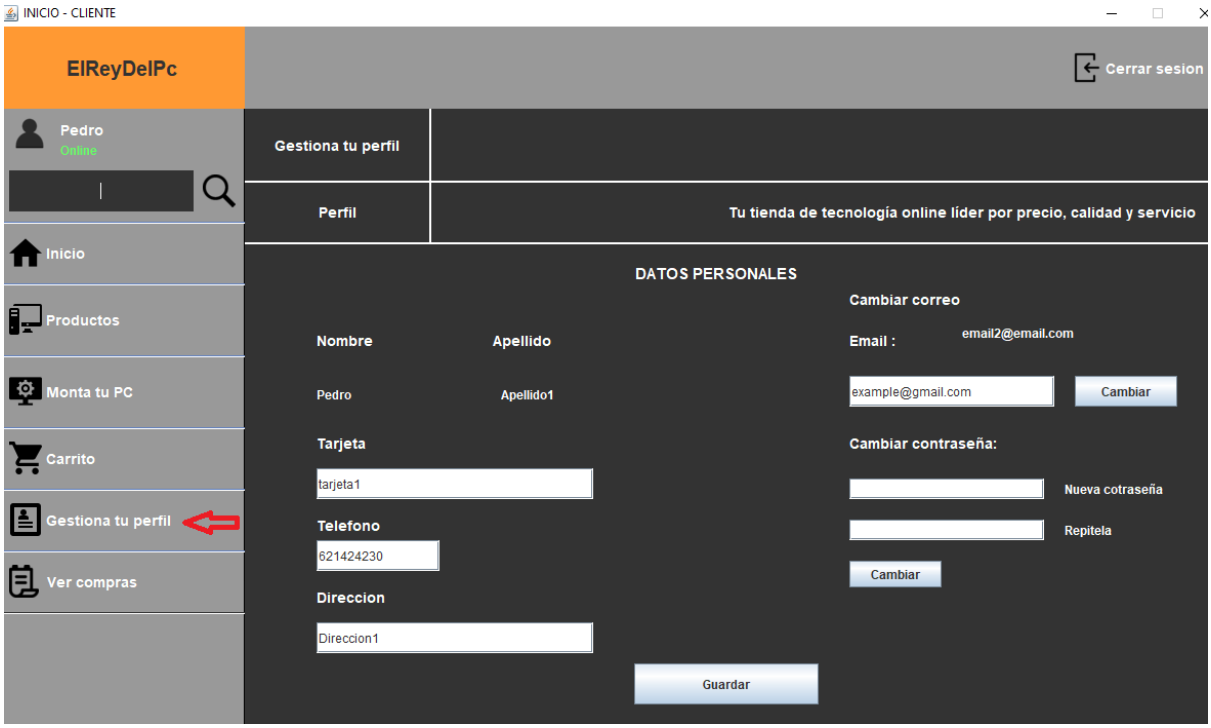

Fig 89. Panel de gestiona perfil en interfaz cliente

6.2.1.6. Ver compras

En esta vista se puede observar una lista con los pedidos que ha realizado el cliente. En estos podemos ver una vista previa de la descripción y el estado de los mismos.

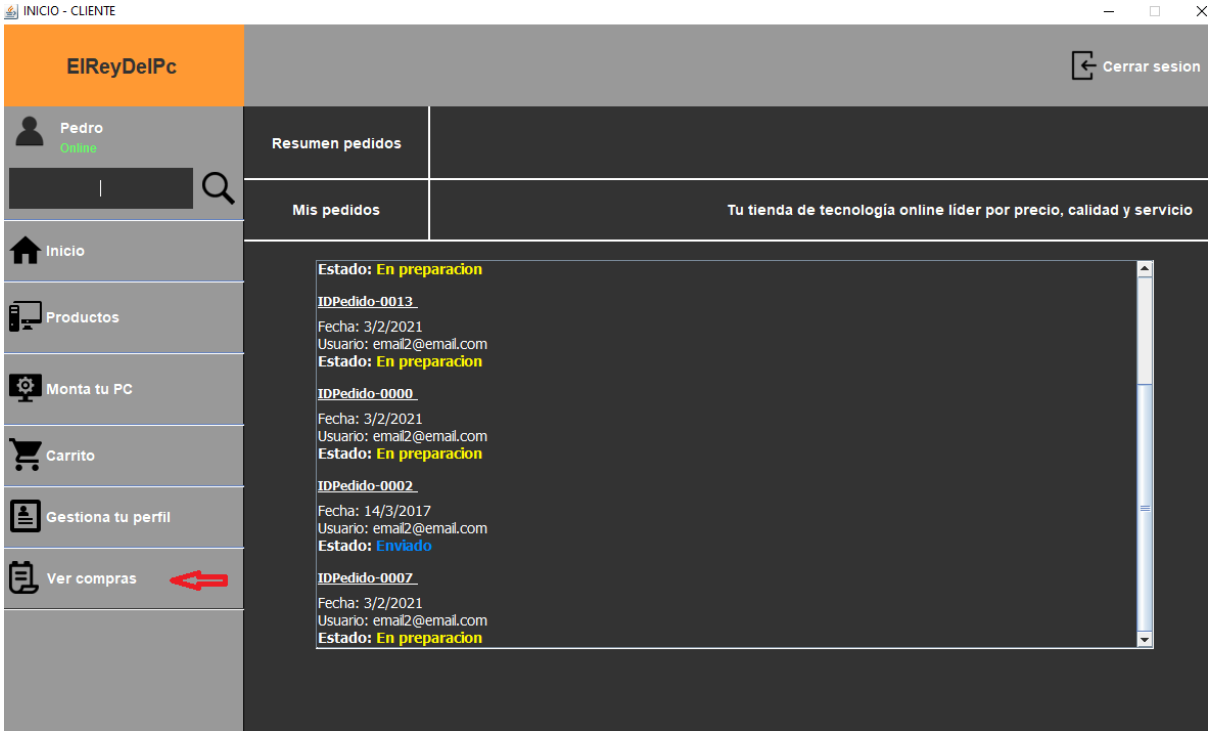

Fig 90. Panel de ver compras en interfaz cliente

Se puede acceder a una vista más detallada del pedido pulsando sobre él.

Fig 91. Información de productos de un pedido

Una vez dentro del pedido seleccionado, se puede confirmar la recepción (únicamente si el pedido está en el estado enviado) o eliminar (si el este está en preparación) mostrándose un mensaje de confirmación de la acción.

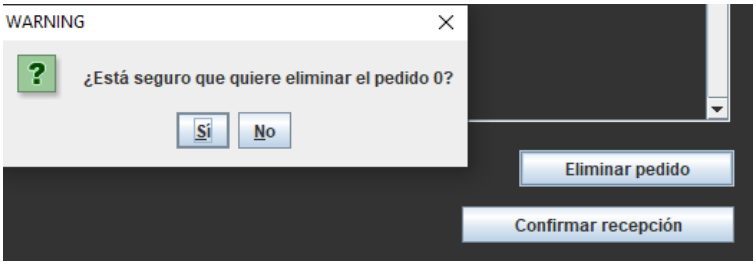

Fig 92. Eliminación de pedido

Una vez eliminado se nos abre la opción de deshacer la eliminación con el botón “deshacer” para restablecer el pedido.

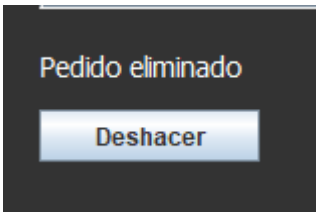

Fig 93. Botón Deshacer eliminación de pedido

6.2.1.7. Búsqueda

La búsqueda podemos realizarla desde cualquier vista, escribiendo la palabra clave el en recuadro de arriba a la izquierda. Luego debemos pulsar el botón de la lupa para buscar.

Esta búsqueda encuentra el nombre del modelo que contenga la palabra clave dada. Si la búsqueda tiene algún resultado se muestra una lista de los artículos obtenidos.

Fig 94. Lista de productos tras realizar búsqueda

Por último, podemos volver al menú de inicio de sesión pulsando el botón de la parte superior derecha para cerrar sesión.

Fig 95. Botón Cerrar Sesión

6.2.2. Interfaz empleado

6.2.2.1. Perfil

Al iniciar sesión como empleado, la primera vista que vemos es el perfil. Donde encontramos todos los datos del usuario y la tienda.

Fig 96. Panel de perfil en interfaz empleado

6.2.2.2. Editar perfil

En esta vista el empleado puede cambiar sus datos introduciendo los nuevos. Para finalizar y confirmar los nuevos datos debe pulsar el botón de “Editar”.
En caso de querer modificar la contraseña solo se cambiará si el usuario introduce la actual, si no la introduce no comprobará la nueva.

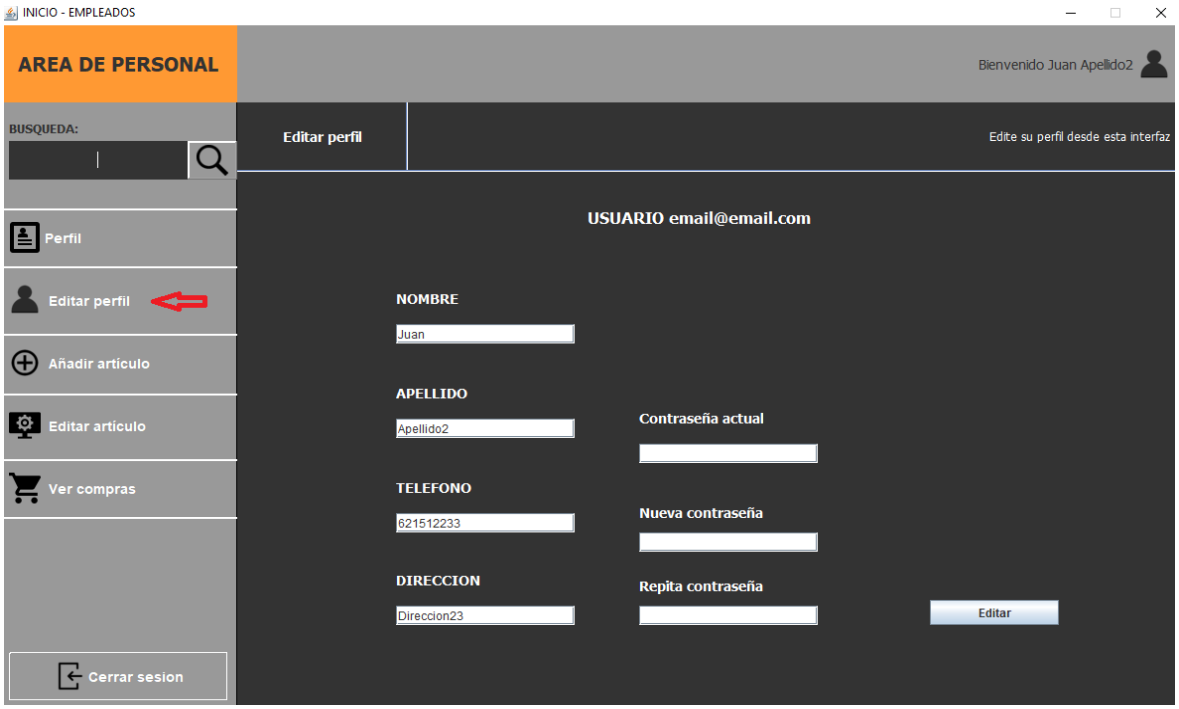

Fig 97. Panel de editar perfil en interfaz empleado

6.2.2.3. Añadir artículo

En esta vista podemos insertar un nuevo artículo al catálogo.

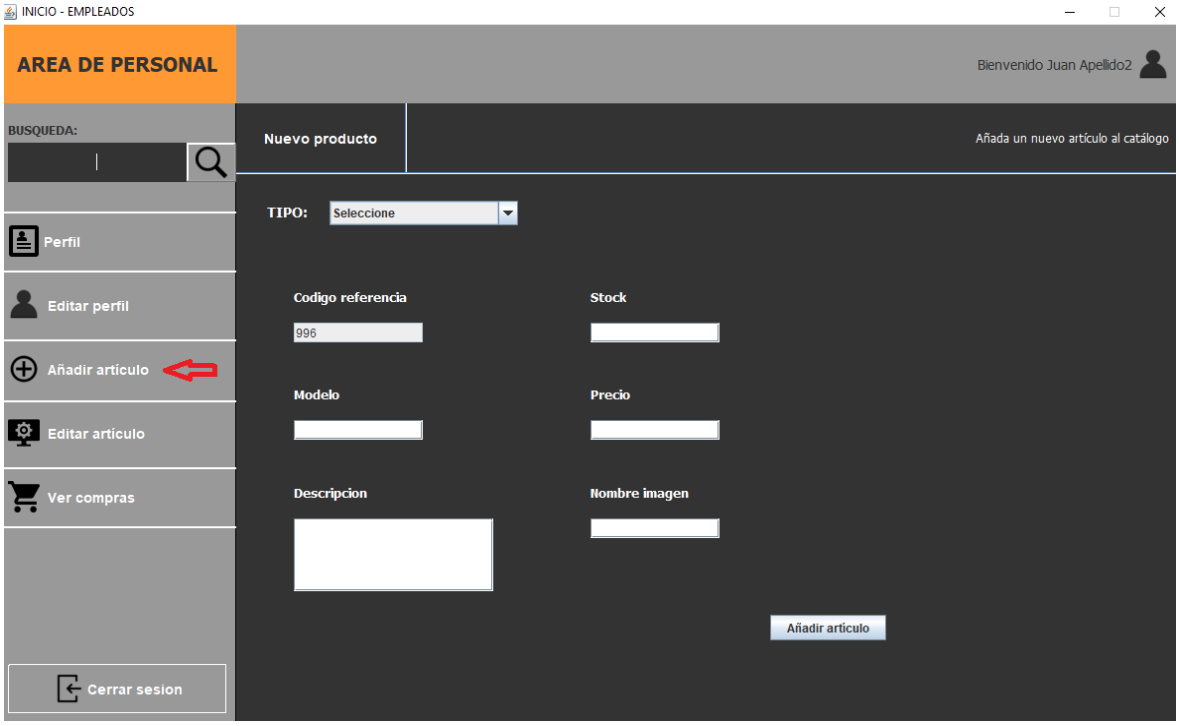

Fig 98. Panel de añadir artículo en interfaz empleado

Para ello, debemos seleccionar el tipo de artículo a introducir para que se desplieguen todos los atributos correspondientes.

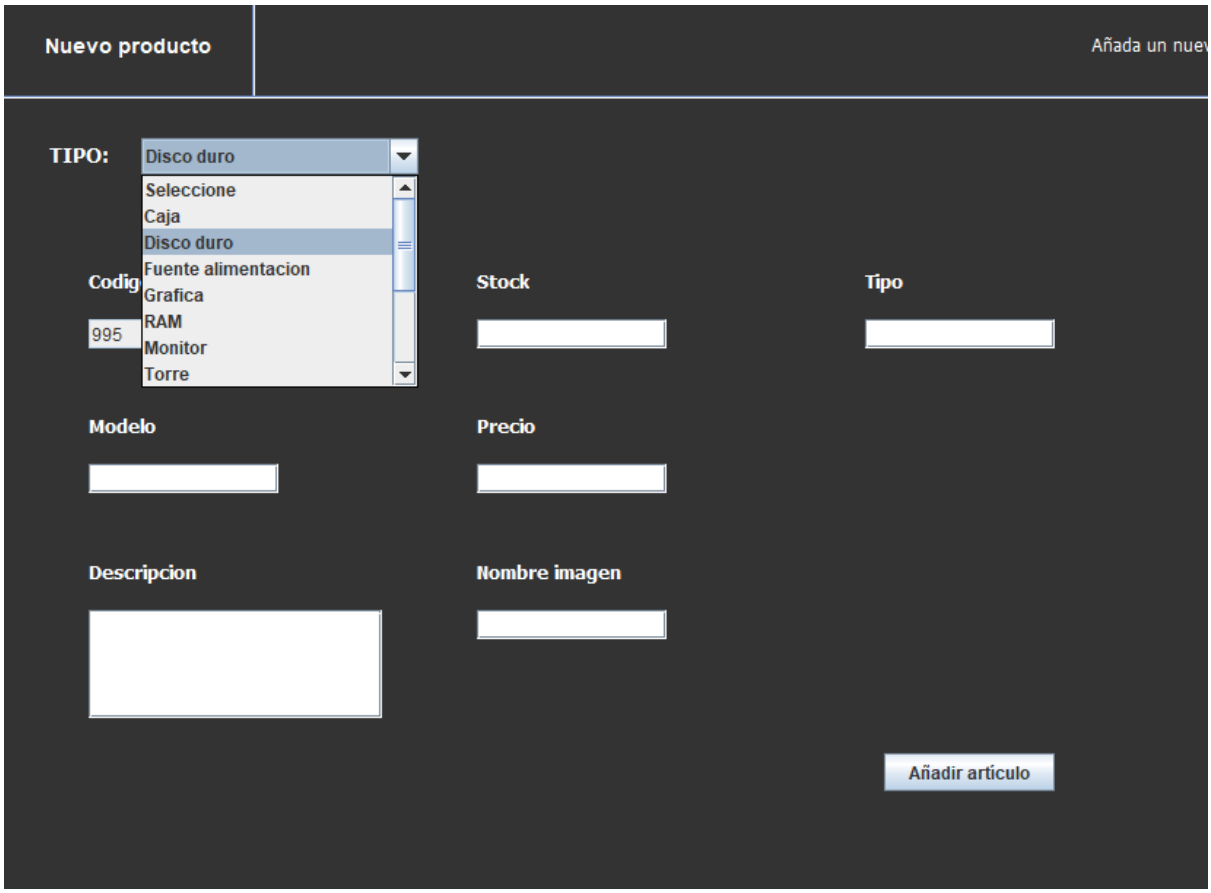

Fig 99. Panel para añadir un nuevo artículo

Después, rellenaremos los campos y pulsaremos “añadir artículo” y el artículo se insertará en la base de datos.

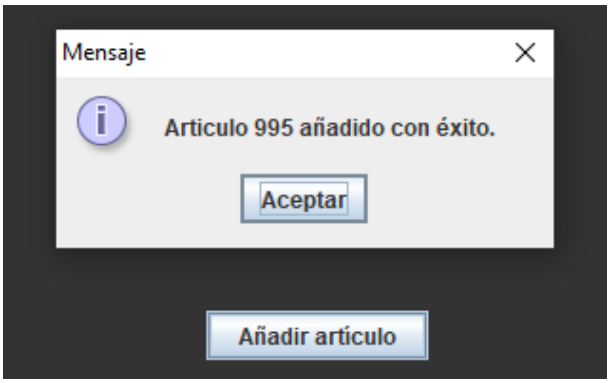

Fig 100. Confirmación de adición de nuevo artículo

6.2.2.4. Editar artículo

En esta vista el empleado puede ver una lista de todos los artículos que hay en el catálogo para poder editarlos.

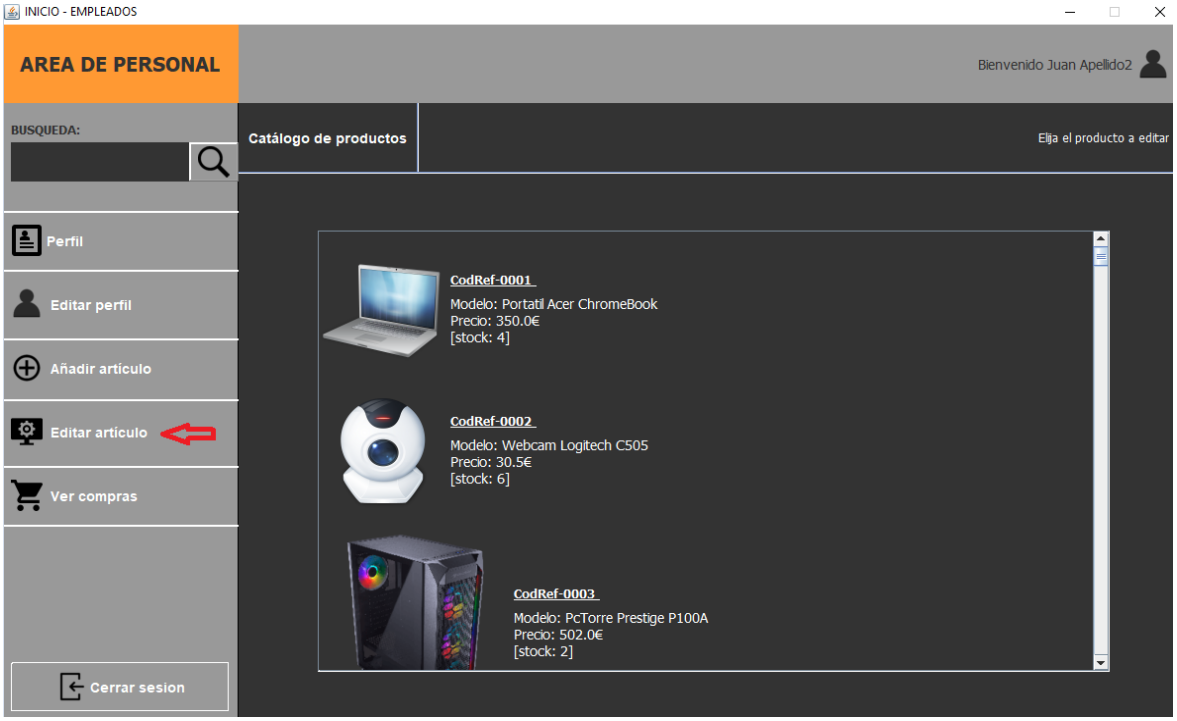

Fig 101. Panel de editar artículo en interfaz empleado

Para modificar uno, debe pulsar sobre él y se mostrarán sus características.
Una vez que estamos en la vista de las características del producto, podemos editarlas (cambiando los datos y pulsando “editar”) o borrar el producto (pulsando “borrar”).

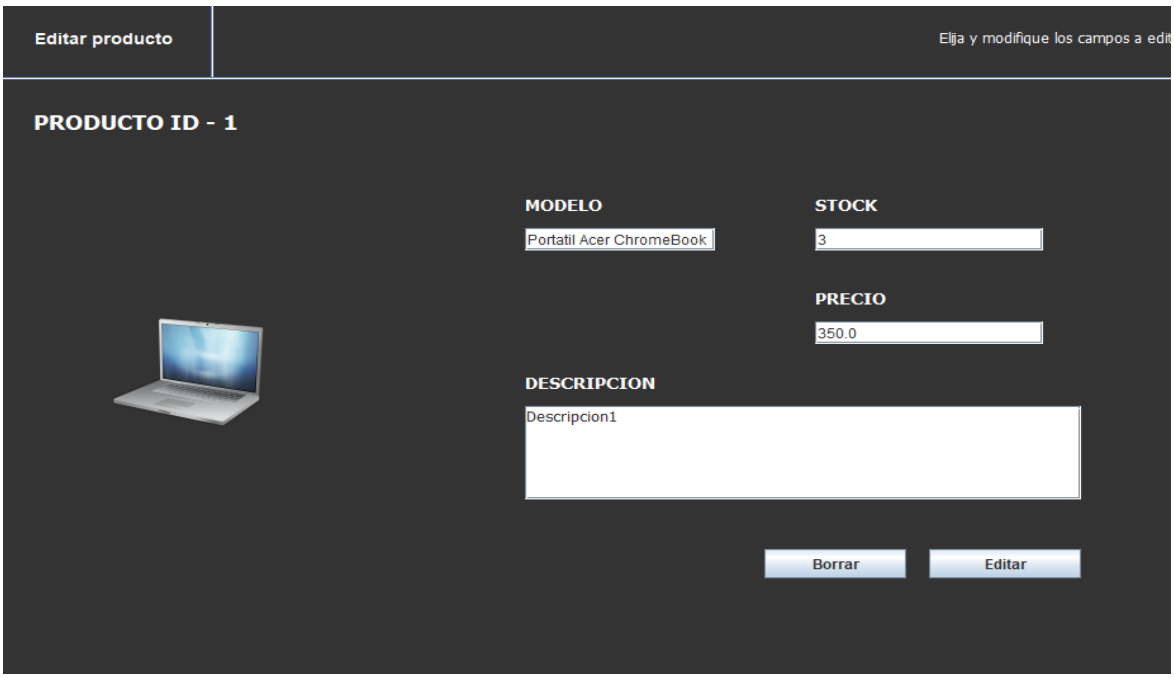

Fig 102. Panel para editar un artículo específico

6.2.2.5. Ver compras

En esta vista podemos observar todos los pedidos que se han realizado en la tienda, con una pequeña vista previa del pedido y su estado.

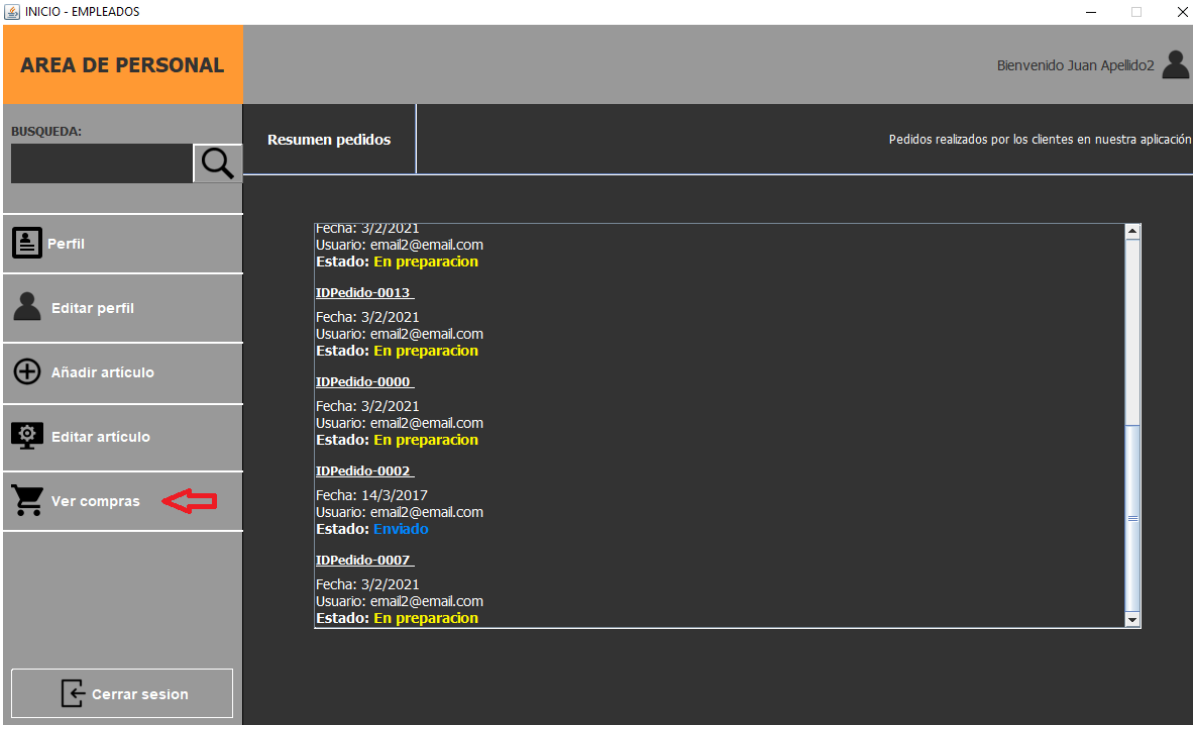

Fig 103. Panel de ver compras interfaz empleado

Para acceder a un pedido y verlo más en detalle debemos pulsar sobre él y se nos mostrará la información del pedido, así como los artículos que lo componen.

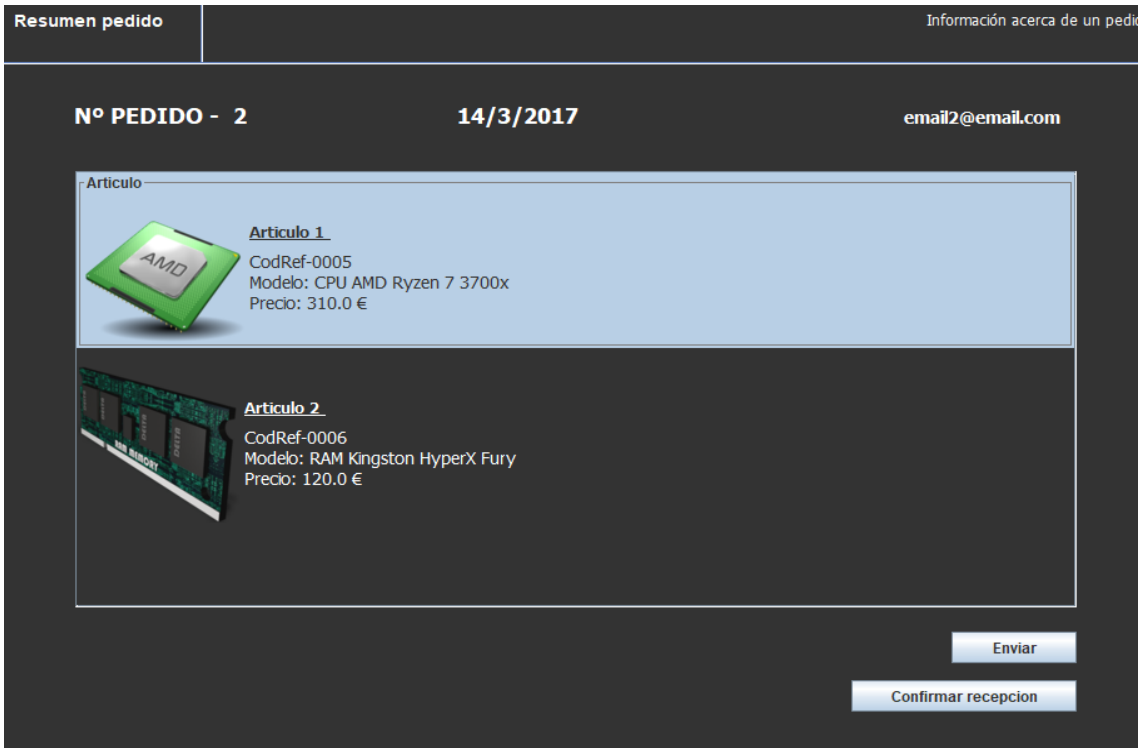

Fig 104. Información de lista de artículo de pedido

También se nos muestran dos opciones para cambiar el estado del producto, pudiendo enviarlo si este está “en preparación” o confirmar la recepción si el estado está en “enviado".

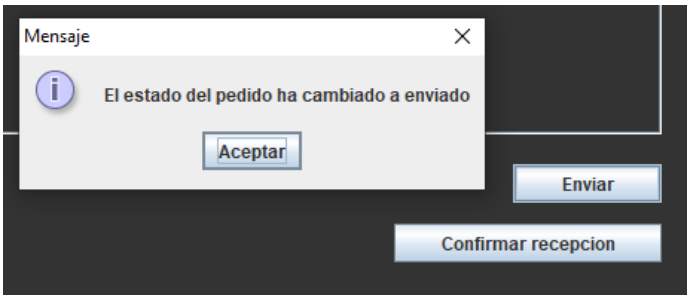

Fig 105. Envío de producto

Fig 106. Cambio de estado de producto

Del mismo modo que en el menú del cliente podemos buscar un artículo con la barra de búsqueda y la lupa, pero en este caso nos sirve para buscar un producto y editarlo.

Fig 107. Barra búsqueda en interfaz empleado

Para finalizar, también disponemos de la misma funcionalidad para cerrar sesión pulsando el botón de cerrar sesión situado en la parte inferior izquierda.

Fig 108. Botón cerrar sesión en interfaz empleado