

# **Distribución de Productos en un Supermercado**

**Segunda entrega: 16/12/2024**

**Versión: 1.1**

**Identificador del equipo: 31.3**

Hug Capdevila: [hug.capdevila@estudiantat.upc.edu](mailto:hug.capdevila@estudiantat.upc.edu)

Levon Asatryan: [levon.asatryan@estudiantat.upc.edu](mailto:levon.asatryan@estudiantat.upc.edu)

Omar Cornejo: [omar.antonio.cornejo@estudiantat.upc.edu](mailto:omar.antonio.cornejo@estudiantat.upc.edu)

Nazarena Ponce: [nazarena.ponce@estudiantat.upc.edu](mailto:nazarena.ponce@estudiantat.upc.edu)

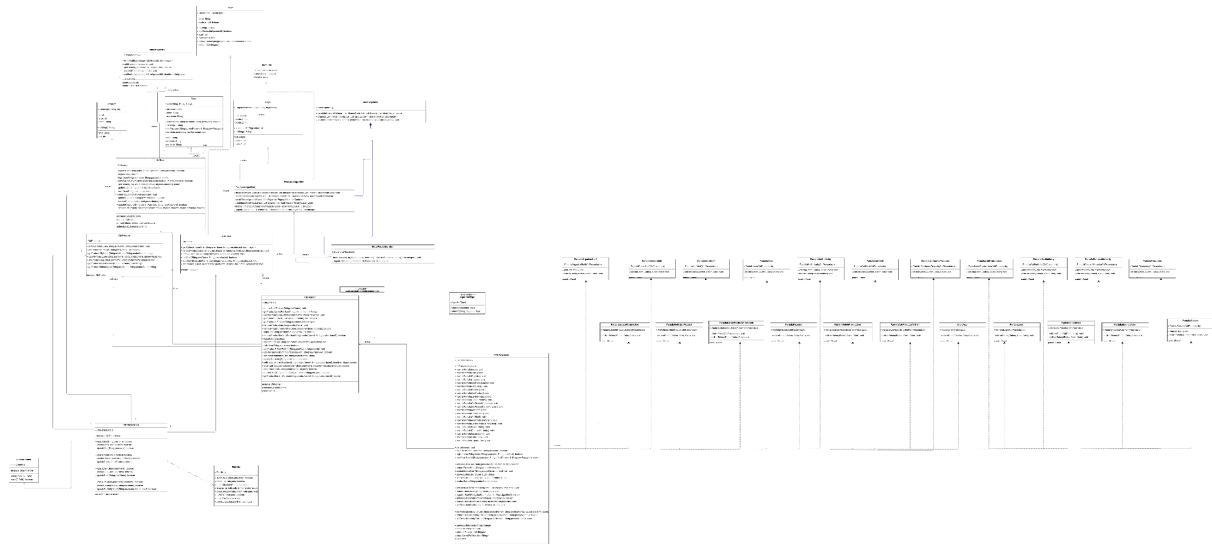


# Índice

<b>1. Diagrama del modelo conceptual.....</b>	<b>3</b>
1.1. Diseño del diagrama del modelo conceptual.....	3
1.2. Cambios más relevantes en la capa de dominio.....	3
<b>2. Diagrama de las clases i controladors de la capa de presentació.....</b>	<b>5</b>
2.1. Explicación de las clases de presentación.....	5
2.1.1. Pantalla Inicio.....	5
2.1.2. Pantalla Registro.....	7
2.1.3. Pantalla Login.....	9
2.1.4. Pantalla User.....	11
2.1.5. Modificar User.....	13
2.1.6. Pantalla Gestión de Estanterías.....	16
2.1.7. Pantalla Añadir Estantería.....	18
2.1.8. Pantalla Consultar Estantería.....	20
2.1.9. Pantalla Eliminar Estantería.....	22
2.1.10. Pantalla Añadir Producto a Estantería.....	24
2.1.11. Pantalla Eliminar Producto de Estantería.....	26
2.1.12. Pantalla Organizar Estantería.....	28
2.1.13. Pantalla Gestión de Productos.....	31
2.1.13. Pantalla Añadir Producto.....	34
2.1.14. Pantalla Eliminar Producto.....	37
2.1.15. Pantalla Actualizar Producto.....	40
2.1.16. Pantalla Consultar Atributos.....	43
2.1.17. Pantalla Productos de Usuario.....	46
2.1.18. Pantalla Gestión de Similitudes.....	48
2.1.19. Pantalla Modificar Similitud.....	50
2.1.20. Pantalla Eliminar Similitud.....	52
2.1.21. Pantalla Consultar Similitud.....	54
2.2. Explicación del controlador de presentación.....	56
2.2.1. Atributos Principales.....	56
2.2.2. Métodos Principales.....	56
2.2.3. Navegación entre Pantallas.....	56
2.2.4. Lógica General.....	58
<b>3. Diagrama de las clases y controladores de la capa de persistencia.....</b>	<b>59</b>
3.1. Explicación de las clases de persistencia.....	59
3.2. Explicación del controlador de persistencia.....	60
<b>4. Descripción de los algoritmos y las estructuras de datos.....</b>	<b>61</b>
4.1. Cambios en Control User.....	61
4.2. Cambios en Control Product.....	62
4.3. Cambios en Control Shelf.....	63
4.4. Cambios en los Algoritmos.....	63

# 1. Diagrama del modelo conceptual

## 1.1. Diseño del diagrama del modelo conceptual



## 1.2. Cambios más relevantes en la capa de dominio

El cambio más relevante y significativo en la aplicación ha tenido lugar en la manera en que se gestiona y almacena la información, introduciendo mejoras sustanciales en comparación con la primera entrega. Inicialmente, la aplicación utilizaba estructuras de datos para conservar en la memoria del programa todo el contenido asociado a cada uno de los usuarios. Este enfoque, aunque funcional, presentaba limitaciones importantes, dado que toda la información debía mantenerse activa en memoria independientemente de su uso inmediato.

Sin embargo, con la implementación de una capa de persistencia, se ha logrado transformar radicalmente este proceso. Ahora, los datos de todos los usuarios se almacenan de manera persistente, lo que permite conservarlos a largo plazo sin depender exclusivamente de la memoria del programa. Esta capa de persistencia no solo asegura la integridad y disponibilidad de los datos, sino que también optimiza el uso de los recursos del sistema. Como resultado, la memoria del programa se utiliza únicamente para mantener información relacionada con el usuario actual, lo que reduce significativamente la carga y mejora el rendimiento general de la aplicación.

En términos prácticos, este cambio no solo incrementa la capacidad de la aplicación para manejar un mayor número de usuarios, sino que también permite una gestión más eficiente de los recursos, facilitando la implementación de futuras funcionalidades relacionadas con el acceso y la manipulación de los datos. Además, al reducir la dependencia de la memoria volátil, se minimiza el riesgo de pérdida de información en caso de fallos en el programa, lo que añade un nivel extra de robustez y fiabilidad al sistema.

[illegible]

### 2.1.1. Pantalla Inicio



5

Esta vista corresponde a la pantalla de inicio de la aplicación. Es la primera pantalla que el usuario verá al iniciar la aplicación, y contiene tres botones principales para permitir el acceso a diferentes funcionalidades: registro de usuario, inicio de sesión y salida de la aplicación.

#### **Atributos:**

- **JPanel mainPanel:** El panel principal de la vista, que organiza todos los elementos gráficos de la pantalla (los botones, el título, etc.) usando un layout de tipo `GridBagLayout`.
- **JButton registerButton:** Un botón que redirige a la pantalla de registro de usuario.
- **JButton loginButton:** Un botón que lleva a la pantalla de inicio de sesión para usuarios ya registrados.
- **JButton exitButton:** Un botón para cerrar la aplicación y salir del programa.

#### **Métodos:**

##### **InicioApp(CtrlPresentacio ctrlPresentacio)**

Constructor de la vista de inicio. Inicializa el panel, los botones, y establece la lógica de navegación para cada uno:

- **registerButton:** Navega a la pantalla de registro mediante la llamada `ctrlPresentacio.mostrarPantallaRegistro()`.
- **loginButton:** Dirige al usuario a la pantalla de inicio de sesión usando `ctrlPresentacio.mostrarPantallaLogin()`.
- **exitButton:** Finaliza la ejecución de la aplicación cuando se hace clic en él, utilizando `System.exit(0)` para cerrar el programa.

##### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (`mouseEntered` y `mouseExited`).

##### **getPanel()**

Retorna el panel principal `panelOrganize` para integrarlo en el contenedor principal de la aplicación.

#### **Flujo de interacción:**

1. **Register:** Al seleccionar este botón el usuario puede crear un perfil en la aplicación.

2. **Login:** Al seleccionar este botón el usuario puede iniciar sesión con un perfil creado previamente.
3. **Salir:** Al pulsar este botón se cierra la aplicación.



### 2.1.2. Pantalla Registro



#### Vista: PantallaRegistro

La vista PantallaRegistro es la pantalla donde los usuarios pueden registrarse proporcionando su nombre completo, nombre de usuario y contraseña. Esta vista permite a los usuarios crear una cuenta para poder acceder a la aplicación.

#### Atributos:

- **JPanel panelRegistro:** El panel principal de la pantalla de registro. Usa un GridBagLayout para organizar los componentes y se le aplica un borde con un título para una apariencia visual atractiva.
- **TextField nameField:** Campo de texto para que el usuario ingrese su nombre completo.
- **TextField usernameField:** Campo de texto para que el usuario ingrese un nombre de usuario.
- **JPasswordField passwordField:** Campo de texto para que el usuario ingrese su contraseña de manera segura.

#### Métodos:

##### PantallaRegistro(CtrlDomain ctrlDomain, CtrlPresentacio ctrlPresentacio)

Constructor de la pantalla de registro. Inicializa los componentes y organiza la interfaz:

- Crea etiquetas (JLabel) para los campos de entrada: nombre completo, nombre de usuario y contraseña.
- Establece estilos para las etiquetas (fuentes en negrita) y los botones de la interfaz.

- Los botones:
  - **saveButton**: Al hacer clic, recoge los valores de los campos de texto, verifica que no estén vacíos y llama al controlador de dominio (`ctrlDomain.registerUser()`) para registrar al usuario. Si el registro es exitoso, muestra un mensaje y limpia los campos de texto, luego vuelve a la pantalla de inicio. Si ocurre algún error (como un nombre de usuario ya existente), muestra un mensaje de error.
  - **volverButton**: Permite volver a la pantalla de inicio (`ctrlPresentacio.mostrarPantallaInicio()`).
- Los componentes (etiquetas, campos de texto, botones) se colocan en el panel usando un `GridBagConstraints` para una disposición flexible.

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (`mouseEntered` y `mouseExited`).

### **getPanel()**

Retorna el panel principal `panelOrganize` para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Full Name**: En este campo el usuario debe añadir su nombre completo.
2. **Username**: En este campo el usuario debe añadir un nombre de usuario que sea único para identificarlo.
3. **Password**: En este campo el usuario crea su contraseña para poder iniciar sesión en futuras ocasiones.
4. **Save**: La información se valida y se crea el perfil del usuario, si alguno de los campos está vacío se muestra un mensaje de error.
5. **Volver**: Al pulsar este botón el usuario puede regresar a la pantalla inicial.

### 2.1.3. Pantalla Login



The screenshot shows a login window titled "Iniciar Sesión". It features a light blue background. In the center, there are two text input fields. The first is labeled "Nombre de Usuario:" and the second is labeled "Contraseña:". Below these fields are two buttons: a green button labeled "Entrar" and a red button labeled "Volver".

#### Vista: PantallaLogin

La vista PantallaLogin corresponde a la pantalla de inicio de sesión de la aplicación. En ella, los usuarios pueden ingresar su nombre de usuario y contraseña para acceder a la aplicación, y se proporcionan opciones para validar las credenciales.

#### Atributos:

- **JPanel panelLogin:** El panel principal de la pantalla de inicio de sesión, donde se organizan los componentes de la interfaz usando un GridBagLayout. Este panel tiene un borde con un título y un fondo claro para darle una apariencia ordenada y amigable.
- **TextField usernameField:** Campo de texto donde el usuario introduce su nombre de usuario.
- **PasswordField passwordField:** Campo de texto donde el usuario introduce su contraseña de manera segura.

#### Métodos:

##### **PantallaLogin(CtrlDomain ctrlDomain, CtrlPresentacio ctrlPresentacio)**

Constructor de la pantalla de inicio de sesión. Este método configura los componentes visuales y la lógica de la vista:

- Crea etiquetas (JLabel) y campos de texto para que el usuario ingrese su nombre de usuario y contraseña.
- Establece estilos para las etiquetas con fuentes en negrita para mejorar la legibilidad.

- Los botones:
  - **entryButton**: Al hacer clic en este botón, el sistema valida las credenciales ingresadas por el usuario llamando al método `ctrlDomain.loginUser(username, password)`. Si las credenciales son correctas, muestra un mensaje de éxito y navega a la pantalla del usuario (`ctrlPresentacio.mostrarPantallaUsuario()`). Si las credenciales son incorrectas, muestra un mensaje de error. Además, limpia los campos de texto después de la validación.
  - **volverButton**: Permite regresar a la pantalla de inicio (`ctrlPresentacio.mostrarPantallaInicio()`).
- Los componentes se organizan en el panel utilizando `GridBagConstraints` para disponer las etiquetas, los campos de texto y los botones de manera ordenada.

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (`mouseEntered` y `mouseExited`).

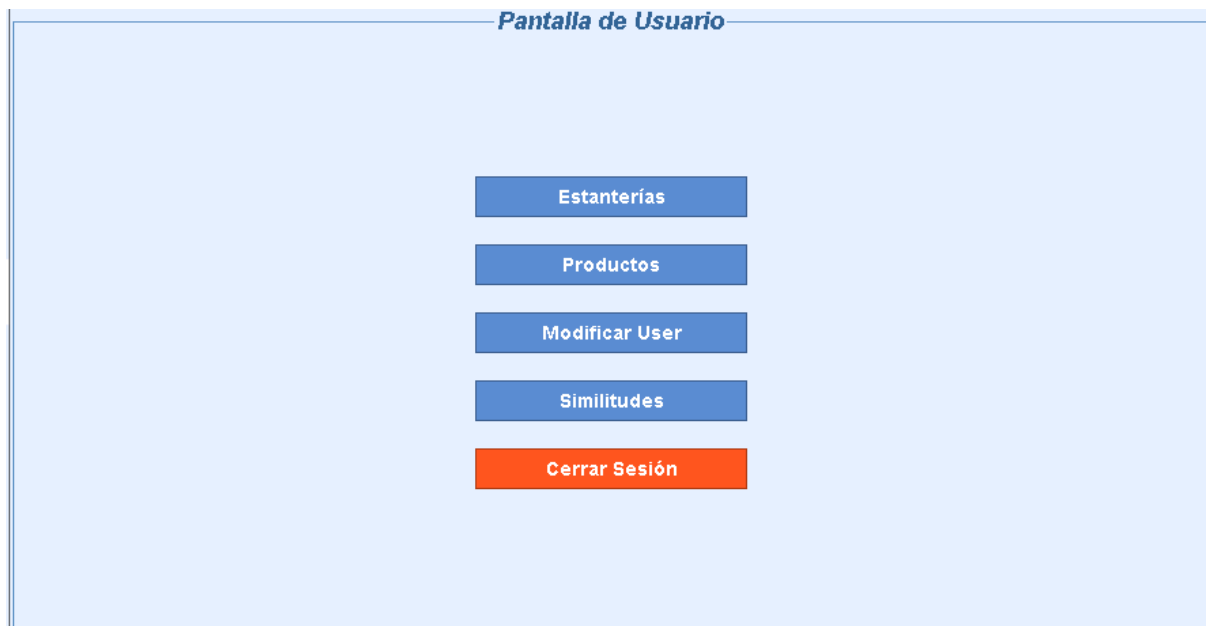
### **getPanel()**

Retorna el panel principal `panelOrganize` para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Username**: En este campo el usuario ingresa su nombre de usuario.
2. **Password**: En este campo el usuario ingresa su contraseña.
3. **Entry**: Al pulsar este botón se validan los datos ingresados comprobando que el perfil exista y, de ser así, se inicia sesión en el perfil autenticado, si alguno de los campos están vacíos o la contraseña es incorrecta se muestra un mensaje de error.
4. **Salir**: Al pulsar este botón se vuelve a la pantalla de inicio.

## 2.1.4. Pantalla User



### Vista: PantallaUser

La vista PantallaUser representa la pantalla principal para un usuario autenticado dentro de la aplicación. En esta vista, el usuario puede interactuar con diversas opciones disponibles, como gestionar estanterías, productos, modificar su información personal, o cerrar sesión.

### Atributos:

- **JPanel panelUser:** El panel principal que contiene todos los componentes visuales de la pantalla. Se organiza utilizando un GridBagLayout para alinear los botones de forma ordenada y flexible. El panel tiene un borde y un fondo claro, lo que mantiene la coherencia visual con otras pantallas.
- **CtrlPresentacio ctrlPresentacio:** Controlador encargado de la presentación de la interfaz de usuario. Se utiliza para manejar la navegación entre pantallas.
- **CtrlDomain ctrlDomain:** Controlador del dominio que gestiona la lógica de negocio de la aplicación. Se usa en los métodos del controlador para acceder a la funcionalidad de backend.

### Métodos:

#### PantallaUser(CtrlPresentacio ctrlPresentacio, CtrlDomain ctrlDomain)

Constructor que inicializa la pantalla del usuario:

- **Bordes y Estilo:** Se establece un borde con un título para el panel, utilizando un estilo consistente con el resto de las vistas de la aplicación. El fondo se define con un color claro para una interfaz más amigable.

- **Botones:**
  - **Estanterías:** Abre la pantalla relacionada con la gestión de estanterías.
  - **Productos:** Permite gestionar los productos.
  - **Modificar User:** Da acceso a la pantalla para modificar los datos del usuario.
  - **Similitudes:** Accede al menú de gestión de similitudes
  - **Cerrar Sesión:** Permite al usuario cerrar sesión y volver a la pantalla de inicio.
- **Acciones de los botones:** Se asignan acciones a los botones, como navegar a otras pantallas o cerrar la sesión.

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

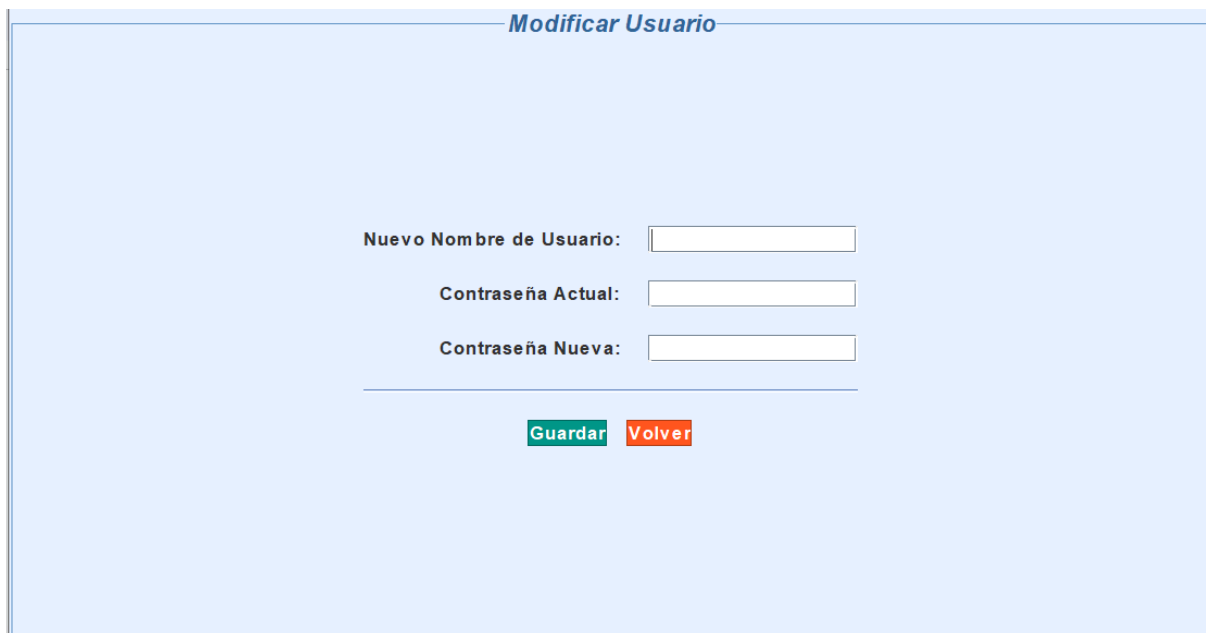
### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Estanterías:** Al seleccionar este botón el usuario puede acceder a la gestión de estanterías de su perfil.
2. **Productos:** Al seleccionar este botón el usuario puede acceder a la gestión de productos de su perfil.
3. **Modificar User:** Al seleccionar este botón el usuario puede acceder a la gestión de su perfil y modificar sus datos.
4. **Similitudes:** Al seleccionar este botón el usuario puede acceder a la gestión de similitudes entre los productos de su perfil.
5. **Cerrar Sesión:** Al pulsar este botón se cierra la sesión del usuario y vuelve a la pantalla inicial.

### 2.1.5. Modificar User



**Modificar Usuario**

Nuevo Nombre de Usuario:

Contraseña Actual:

Contraseña Nueva:

#### Vista: PantallaModificarUser

La vista PantallaModificarUser permite a los usuarios modificar su nombre de usuario y su contraseña. Esta pantalla está diseñada para proporcionar una experiencia de usuario clara y simple, con un formulario que incluye campos para ingresar el nombre de usuario y la contraseña actual, así como para establecer una nueva contraseña. Los usuarios pueden guardar los cambios o regresar a la pantalla principal del usuario.

#### Atributos:

- **PantallaModificarUser (JPanel):** El panel principal que contiene todos los elementos de la pantalla. Se organiza utilizando un GridBagLayout, que permite una distribución flexible de los componentes.
- **usernameField (JTextField):** Campo de texto para el nuevo nombre de usuario.
- **passwordField (JPasswordField):** Campo de texto para la nueva contraseña.
- **actualpasswordField (JPasswordField):** Campo de texto para la contraseña actual, que se requiere para realizar modificaciones en la cuenta.

#### Métodos:

##### **PantallaModificarUser(CtrlDomain ctrlDomain, CtrlPresentacio ctrlPresentacio)**

Constructor que configura los componentes de la interfaz:

- **Bordes y Estilo:** El panel tiene un borde con un título que indica la función de la pantalla (modificar usuario), y un fondo claro para mantener una interfaz amigable y coherente.

- **Campos de texto:**
  - **Nuevo Nombre de Usuario:** Un campo de texto que permite al usuario ingresar un nuevo nombre de usuario. Cuando el campo recibe foco, se limpia el texto por defecto ("New Username") y se restablece si el campo se deja vacío.
  - **Contraseña Actual:** Un campo de contraseña para ingresar la contraseña actual. Este campo es obligatorio para poder realizar cambios.
  - **Nueva Contraseña:** Un campo de contraseña donde el usuario puede establecer una nueva contraseña.
- **Botones:**
  - **Guardar:** Al hacer clic en este botón, el sistema valida los campos e intenta realizar los cambios mediante el CtrlDomain (controlador de dominio).
  - **Volver:** Regresa a la pantalla de usuario sin realizar cambios.
- **Acciones de los botones:**
  - Si el campo "Contraseña Actual" está vacío, muestra un mensaje de error.
  - Si ambos campos "Nuevo Nombre de Usuario" y "Nueva Contraseña" están vacíos, también muestra un mensaje de error.
  - Si se intentan realizar cambios, se valida si el nombre de usuario es único y si la contraseña actual es correcta. Si todo es válido, se realizan los cambios y se muestra un mensaje de éxito.
  - Si ocurre algún error (como contraseña incorrecta o nombre de usuario ya existente), se muestra un mensaje de error.
- **Estilo de los botones:** Los botones están estilizados con un fondo y color de texto personalizados, además de un efecto visual cuando el ratón pasa sobre ellos (el color de fondo se ilumina).

#### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

#### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.



### **Flujo de interacción:**

1. **New Username:** El usuario debe ingresar un nuevo nombre de usuario.
2. **Actual Password:** El usuario debe ingresar su contraseña actual para poder realizar los cambios.
3. **New Password User:** Si desea modificar la contraseña, el usuario debe ingresar la contraseña nueva.
4. **Save:** Al pulsar este botón se validan los datos y, de ser correctos, se realizan los cambios en el perfil. En caso contrario se muestra un mensaje de error y no realiza ningún cambio.
5. **Volver:** Al pulsar este botón se vuelve a la pantalla de usuario.

### 2.1.6. Pantalla Gestión de Estanterías



#### Vista: PantallaShelves

La vista PantallaShelves permite a los usuarios modificar o consultar su inventario de estanterías. Esta pantalla está diseñada para proporcionar una experiencia de usuario clara y simple, con siete botones que permiten seleccionar la funcionalidad buscada, incluyendo volver a la pantalla principal de Usuario.

#### Atributos:

- **JPanel panelShelves:**  
Panel principal que contiene todos los componentes visuales de la pantalla. Utiliza un GridBagLayout para organizar los elementos de forma flexible y ordenada. Este panel tiene un borde general y un fondo de color claro para mantener la coherencia visual con otras pantallas.
- **CtrlPresentacio ctrlPresentacio:**  
Controlador encargado de manejar la presentación de la interfaz de usuario. Se utiliza para navegar entre las diferentes pantallas de la aplicación.

#### Métodos:

##### PantallaShelves(CtrlPresentacio ctrlPresentacio)

- Se inicia la pantalla de gestión de estanterías.
- **Estilo visual:**

- Se establece un borde con un espacio uniforme y un fondo claro para una interfaz limpia y agradable.
- Incluye un título centrado que identifica la funcionalidad de la pantalla.
- **Elementos principales:**
  - Un panel de botones organizado con GridBagLayout.
  - Botones estilizados y uniformes para acceder a diferentes funcionalidades.
- **Botones de acción:**
  - **Añadir Estantería:** Navega a la pantalla para agregar una nueva estantería.
  - **Consultar Estantería:** Permite consultar información sobre una estantería específica.
  - **Borrar Estantería:** Abre la funcionalidad para eliminar una estantería existente.
  - **Añadir Productos:** Navega a la pantalla para agregar productos a una estantería.
  - **Eliminar Productos:** Permite gestionar la eliminación de productos en una estantería.
  - **Organizar Estantería:** Abre una pantalla para reorganizar el contenido de una estantería.
  - **Volver:** Regresa a la pantalla principal del usuario autenticado.
- **Acciones de los botones:** Cada botón está vinculado a una acción específica mediante un ActionListener. Estas acciones llaman a métodos en CtrlPresentacio para navegar a otras pantallas.

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Nombre de la estantería:** El usuario introduce un nombre en el campo de texto. Si no se proporciona un nombre, se muestra un mensaje de error.
2. **Aceptar:** La información se valida y se registra la estantería.
3. **Cancelar:** El usuario regresa a la pantalla de gestión de estanterías sin realizar cambios.

### 2.1.7. Pantalla Añadir Estantería



#### Vista: PantallaAddShelf

La vista PantallaAddShelf permite a los usuarios añadir una estantería a su inventario. Esta pantalla cuenta con un campo para ingresar el nombre de la estantería que se desea crear. En la parte inferior de la pantalla hay un botón de aceptar y otro de cancelar que permite volver a la pantalla principal de Usuario.

#### Atributos:

- **JPanel panelAddShelf:**  
Panel principal que contiene todos los componentes visuales de la pantalla. Se organiza utilizando un GridBagLayout para mantener un diseño flexible y ordenado. Este panel tiene un fondo claro para mantener la consistencia visual con el resto de la aplicación.

#### Métodos:

##### PantallaAddShelf(CtrlPresentacio ctrlPresentacio)

Inicializa la pantalla de adición de estanterías:

- **Título:** Incluye un título centrado que identifica la funcionalidad de la pantalla.
- **Formulario:**
  - Campo de texto para ingresar el nombre de la estantería.
  - Un JComboBox para seleccionar un producto existente asociado a la estantería.
- **Botones y acciones:** Botones estilizados para aceptar o cancelar la acción.

- **Aceptar:**
  - Valida que el nombre de la estantería no esté vacío.
  - Recupera el producto seleccionado (si lo hay) y lo vincula a la nueva estantería.
  - Llama al método `addShelfForUser()` de `CtrlPresentacio` para registrar la estantería en el sistema.
  - Muestra mensajes de éxito o error dependiendo del resultado de la operación.
- **Cancelar:**
  - Regresa a la pantalla anterior de gestión de estanterías (`PantallaShelves`).

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (`mouseEntered` y `mouseExited`).

### **getPanel()**

Retorna el panel principal `panelOrganize` para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Nombre de la estantería:** El usuario introduce un nombre en el campo de texto. Si no se proporciona un nombre, se muestra un mensaje de error.
2. **Seleccionar producto:** El usuario selecciona uno de los productos guardados en su inventario para añadirlo en la estantería creada.
3. **Aceptar:** La información se valida y se registra la estantería.
4. **Cancelar:** El usuario regresa a la pantalla de gestión de estanterías sin realizar cambios.

### 2.1.8. Pantalla Consultar Estantería



#### Vista: PantallaGetShelf

La vista `PantallaGetShelf` permite a los usuarios consultar una estantería de su inventario. Esta pantalla cuenta con un campo para ingresar el nombre de la estantería que se desea consultar y un visor que muestra los productos que se han guardado previamente en la estantería ingresada. En la parte inferior de la pantalla hay un botón de aceptar y otro de cancelar que permite volver a la pantalla principal de Usuario.

#### Atributos:

- **JPanel `panelGetShelf`:**  
Panel principal que contiene todos los componentes visuales de la pantalla. Utiliza un diseño `GridBagLayout` para organizar los elementos de manera flexible y ordenada.

#### Métodos:

##### **`PantallaGetShelf(CtrlPresentacio ctrlPresentacio)`**

Configura la pantalla de consulta de estanterías con los siguientes elementos:

- **Título:** Un título centrado que indica la funcionalidad de la pantalla.
- **Formulario:**
  - Campo de texto (`TextField`) para ingresar el nombre de la estantería.
  - Un área de texto (`TextArea`) no editable para mostrar los productos asociados, con un scroll en caso de desbordamiento.
- **Botones y acciones:** Botones estilizados para consultar o cancelar.

- **Consultar:** Valida que el campo del nombre de la estantería no esté vacío y llama al método `getProductsInShelf()` de `CtrlPresentacio` para obtener los productos asociados.
  - Si no existen productos o la estantería no está registrada, informa al usuario mediante un cuadro de diálogo.
  - Si existen productos, los lista en el área de texto, con un identificador por línea.
- **Cancelar:** Regresa a la pantalla anterior de gestión de estanterías (`PantallaShelves`).

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño visual consistente a los botones:

- Define colores de fondo y texto, fuentes, y un borde uniforme.
- Añade un efecto visual de cambio de color al pasar el ratón.


### **getPanel()**

Devuelve el panel principal (`panelGetShelf`) para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Nombre de la estantería:** El usuario introduce el nombre en el campo de texto. Si está vacío, se muestra un mensaje de error.
2. **Consultar productos:** Al presionar "Consultar", la aplicación verifica si la estantería existe:
  - a. Si no se encuentra, informa al usuario mediante un cuadro de diálogo.
  - b. Si se encuentra, lista los productos asociados en el área de texto.
3. **Cancelar:** Al presionar "Cancelar", el usuario regresa a la pantalla de gestión de estanterías.

### 2.1.9. Pantalla Eliminar Estantería



#### Vista: PantallaDeleteShelf

La vista `PantallaDeleteShelf` permite a los usuarios eliminar una estantería de su inventario. Esta pantalla cuenta con un campo para ingresar el nombre de la estantería que se desea consultar. En la parte inferior de la pantalla hay un botón de aceptar y otro de cancelar que permite volver a la pantalla principal de Usuario.

#### Atributos

- **JPanel `panelDeleteShelf`:**  
Panel principal que contiene todos los componentes de la pantalla. Utiliza un diseño `GridBagLayout` para la organización de elementos.

#### Métodos

##### **`PantallaDeleteShelf(CtrlPresentacio ctrlPresentacio)`**

Configura los elementos visuales y funcionales de la pantalla, incluyendo:

- **Título:** Texto centrado que describe la acción principal ("Eliminar Estantería").
- **Campo de entrada (JTextField):** Permite al usuario ingresar el nombre de la estantería que desea eliminar.
- **Botones y acciones:**
  - **Aceptar:** Comprueba que el campo de texto no esté vacío y llama al método `removeShelfForUser` de `CtrlPresentacio` con el nombre proporcionado. Si está vacío, muestra un mensaje de error.



- Si la eliminación es exitosa, informa mediante un cuadro de diálogo y regresa a la pantalla de gestión de estanterías.
  - Si ocurre algún error (por ejemplo, la estantería no existe), muestra un mensaje de error.
- **Cancelar:** Regresa a la pantalla principal de gestión de estanterías llamando a `mostrarPantallaShelves`.

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño visual consistente a los botones:

- Define colores de fondo y texto, fuentes, y un borde uniforme.
- Añade un efecto visual de cambio de color al pasar el ratón.

### **getPanel()**

Devuelve el panel principal `panelDeleteShelf` para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción**

1. **Nombre de la estantería:** El usuario introduce el nombre en el campo de texto. Si está vacío, se muestra un mensaje de error.
2. **Eliminar estantería:** Al presionar el botón Aceptar:
  - a. Valida el nombre ingresado.
  - b. Llama al método correspondiente en el controlador para intentar eliminar la estantería.
  - c. Muestra un mensaje de confirmación si se elimina con éxito o un error si no es posible.
3. **Cancelar:** Al presionar el botón Cancelar, el usuario vuelve a la pantalla principal sin realizar ninguna acción.

### 2.1.10. Pantalla Añadir Producto a Estantería



#### Vista: PantallaAddProductToShelf

La vista `PantallaAddProductToShelf` permite a los usuarios añadir un producto a una estantería de su inventario. Esta pantalla cuenta con un campo para ingresar el ID del producto que se desea añadir y otro para ingresar el nombre de la estantería donde se quiere añadir. En la parte inferior de la pantalla hay un botón de aceptar y otro de cancelar que permite volver a la pantalla principal de Usuario.

#### Atributos

- **JPanel `panelAddProductsToShelf`:**  
Contenedor principal de la vista, con un diseño `GridBagLayout` para organizar los elementos visuales.

#### Métodos

##### **`PantallaAddProductsToShelf(CtrlPresentacio ctrlPresentacio)`**

Inicializa la pantalla con los siguientes elementos:

- **Título:** Muestra el propósito de la vista ("Añadir Producto a Estantería") con un diseño destacado.
- **Campos de entrada:** Nombre de la estantería (`shelfNameField`): Permite al usuario introducir el nombre de la estantería.
- **ID del producto (`productIdField`):** Campo numérico donde el usuario introduce el ID del producto a añadir.
- **Botones y acciones:**

- **Aceptar:** Comprueba que ambos campos estén llenos y valida que el ID del producto sea numérico. También llama al método `addProductToShelf` del controlador (`CtrlPresentacio`) con los valores proporcionados.
  - En caso de éxito, muestra un mensaje de confirmación y regresa a la pantalla principal.
  - Si la operación falla, informa al usuario con un mensaje de error.
- **Cancelar:** Llama a `mostrarPantallaShelves` para regresar a la pantalla de gestión de estanterías sin realizar cambios.

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño visual consistente a los botones:

- Define colores de fondo y texto, fuentes, y un borde uniforme.
- Añade un efecto visual de cambio de color al pasar el ratón.

### **getPanel()**

Retorna el panel principal `panelAddProductsToShelf` para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción**

1. **Introducir datos:** El usuario introduce el nombre de la estantería y el ID del producto.
2. **Añadir producto:** Con el botón **Aceptar**:
  - a. Valida los datos y llama al método del controlador para intentar añadir el producto.
  - b. En caso de éxito, muestra un mensaje y regresa a la pantalla principal.
  - c. Si ocurre un error, informa al usuario.
3. **Cancelar operación:** El botón **Cancelar** regresa al menú principal sin realizar ninguna acción.

### 2.1.11. Pantalla Eliminar Producto de Estantería



Nombre de la estantería:

ID del producto:

**Aceptar** **Cancelar**

#### Vista: `PantallaDeleteProductFromShelf`

La vista `PantallaDeleteProductFromShelf` permite a los usuarios eliminar un producto de una estantería en su inventario. Esta pantalla cuenta con un campo para ingresar el ID del producto que se desea eliminar y otro para ingresar el nombre de la estantería donde está guardado. En la parte inferior de la pantalla hay un botón de aceptar y otro de cancelar que permite volver a la pantalla principal de Usuario.

#### Atributos

- **JPanel `panelDeleteProductsFromShelf`:**  
Contenedor principal de la vista, con un diseño `GridBagLayout` para organizar los elementos visuales.

#### Métodos

##### **`PantallaDeleteProductsFromShelf(CtrlPresentacio ctrlPresentacio)`**

- Inicializa la pantalla con los siguientes elementos:
  - **Título:** Muestra el propósito de la vista ("Eliminar Producto de Estantería") con un diseño destacado.
  - **Campos de entrada:**
  - **Nombre de la estantería (`shelfField`):** Permite al usuario introducir el nombre de la estantería.
  - **ID del producto (`productField`):** Campo numérico donde el usuario introduce el ID del producto a eliminar.
- **Botones y acciones:**

- **Aceptar:** Comprueba que el campo del nombre de la estantería no esté vacío y valida que el ID del producto sea numérico. Llama al método `removeProductFromShelf` del controlador (`CtrlPresentacio`) con los valores proporcionados.
  - En caso de éxito, muestra un mensaje de confirmación y limpia los campos de entrada.
  - Si la operación falla, informa al usuario con un mensaje de error.
  - Si el ID del producto no es numérico, muestra un mensaje adecuado.
  - Captura y muestra cualquier excepción lanzada por el controlador.
- **Cancelar:** Llama a `mostrarPantallaShelves` para regresar a la pantalla de gestión de estanterías sin realizar cambios.

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (`mouseEntered` y `mouseExited`).

### **getPanel()**

- Retorna el panel principal `panelDeleteProductsFromShelf` para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción**

1. **Introducir datos:** El usuario introduce el nombre de la estantería y el ID del producto.
2. **Eliminar producto:** Con botón Aceptar:
  - a. Valida los datos y llama al método del controlador para intentar eliminar el producto.
  - b. En caso de éxito, muestra un mensaje de confirmación y limpia los campos de entrada.
  - c. Si ocurre un error, informa al usuario mediante un mensaje de error.
3. **Cancelar:** El botón Cancelar regresa al menú principal sin realizar ninguna acción.

### 2.1.12. Pantalla Organizar Estantería



#### Vista: PantallaOrganizeShelf

La vista PantallaOrganizeShelf permite a los usuarios organizar una estantería existente con los productos guardados en esta. Esta pantalla cuenta con un campo para ingresar el nombre de la estantería y dos botones que permiten seleccionar el algoritmo con el que se desea hacer la organización. En la parte inferior de la pantalla hay un botón de volver que permite regresar a la pantalla principal de Usuario.

#### Atributos

- **CtrlPresentacio ctrlPresentacio:**  
Controlador principal que maneja la lógica de la aplicación. Se utiliza para organizar la estantería.
- **JPanel panelOrganize:**  
Contenedor principal de la vista, diseñado con un GridBagLayout para organizar los componentes de manera estructurada.

#### Métodos

##### PantallaOrganizeShelf(CtrlPresentacio ctrlPresentacio)

- Inicializa la pantalla con los siguientes elementos:
  - **Título:** Muestra el propósito de la vista ("Organizar Estantería") con un diseño destacado.
  - **Campos de entrada:**

- **Nombre de la estantería (shelfField):** Campo de texto donde el usuario introduce el nombre de la estantería a organizar.
- **Botones y acciones:**
  - **Two-Approximation:** Comprueba que el campo de nombre de estantería no esté vacío y llama al método organizeShelf del controlador con el algoritmo TWO\_APPROXIMATION.
    - Si la operación es exitosa, muestra un mensaje de confirmación. En caso de error, muestra un mensaje de error.
  - **Brute Force:** Comprueba que el campo de nombre de estantería no esté vacío y llama al método organizeShelf del controlador con el algoritmo BRUTE\_FORCE.
    - Si la operación es exitosa, muestra un mensaje de confirmación. En caso de error, muestra un mensaje de error.
  - **Volver:** Llama al método mostrarPantallaShelves para regresar a la pantalla principal de gestión de estanterías.

#### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

#### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

#### **Flujo de interacción**

1. **Introducir datos:** El usuario introduce el nombre de la estantería en el campo correspondiente.
2. **Organizar estantería:**
  - a. Si el usuario hace clic en el botón "Two-Approximation", se valida que el nombre de la estantería no esté vacío y se llama al método correspondiente del controlador con el algoritmo de aproximación.

- b. Si el usuario hace clic en el botón "Brute Force", se valida que el nombre de la estantería no esté vacío y se llama al método correspondiente del controlador con el algoritmo de fuerza bruta.
  - c. En ambos casos, si la operación es exitosa, se muestra un mensaje de confirmación. Si ocurre un error, se informa al usuario.
- 3. **Cancelar:** El botón "Volver" regresa al menú principal sin realizar ninguna acción en la estantería.



### 2.1.13. Pantalla Gestión de Productos



#### Vista: PantallaProductos

La vista **PantallaProductos** es la pantalla principal para gestionar los productos dentro de la aplicación. Proporciona al usuario varias opciones relacionadas con la administración de productos, como añadir, eliminar, modificar y consultar información de productos. Además, cuenta con la opción de volver a la pantalla principal.

#### Atributos:

- **JPanel panelProducts:** El panel principal de la pantalla de productos. Utiliza un BorderLayout para organizar los componentes principales, como el título en la parte superior y los botones en la parte central.
- **JButton addProductButton:** Botón para acceder a la funcionalidad de añadir un nuevo producto.
- **JButton deleteProductButton:** Botón para acceder a la funcionalidad de eliminar un producto existente.
- **JButton updateProductButton:** Botón para acceder a la funcionalidad de modificar productos existentes.
- **JButton getAttributesProductButton:** Botón para consultar los atributos de un producto.
- **JButton getNamesProductsButton:** Botón para consultar los nombres de los productos.
- **JButton backButton:** Botón para regresar a la pantalla principal de usuario.

## Métodos:

**PantallaProductos(CtrlPresentacio ctrlPresentacio):** Constructor de la clase que inicializa y organiza la pantalla de gestión de productos.

- **Título:** Un JLabel en la parte superior del panel (BorderLayout.NORTH) con el texto "Gestión de Productos", estilizado con una fuente grande y en negrita.
- **Panel de Botones:** Se utiliza un GridBagLayout dentro de un panel central para organizar los botones verticalmente con márgenes uniformes.
- **Estilo:** Los botones son estilizados con el método privado estilizarBoton, que define colores de fondo, texto y efectos visuales al pasar el cursor sobre los botones.
- **Botones y acciones**
  - **Añadir Producto:** Muestra la pantalla para añadir un producto mediante el controlador ctrlPresentacio.mostrarPantallaAddProduct().
  - **Eliminar Producto:** Muestra la pantalla para eliminar un producto mediante ctrlPresentacio.mostrarPantallaDeleteProduct()
  - **Modificar Producto:** Permite modificar productos existentes mediante ctrlPresentacio.mostrarPantallaUpdateProduct().
  - **Consultar Producto:** Accede a la pantalla de consulta de atributos de productos mediante ctrlPresentacio.mostrarPantallaGetProductAttributes().
  - **Consultar Nombre de Productos:** Accede a la pantalla de consulta de nombres de productos mediante ctrlPresentacio.mostrarPantallaGetProductNamesForUser().
  - **Volver:** Regresa a la pantalla principal del usuario mediante ctrlPresentacio.mostrarPantallaUsuario().

**estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

**getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

## Flujo de interacción

1. **Añadir Producto:** El usuario hace clic en el botón "Añadir Producto" para acceder a la pantalla donde puede introducir los datos de un nuevo producto.
2. **Eliminar Producto:** Al pulsar "Eliminar Producto", el usuario es redirigido a una pantalla para seleccionar y eliminar un producto existente.
3. **Modificar Productos:** El usuario selecciona "Modificar Productos" para editar las características de productos en una pantalla específica.
4. **Ver Atributos de Producto:** El botón "Ver Atributos de Producto" lleva al usuario a una pantalla donde puede consultar los atributos detallados de un producto.
5. **Ver Nombres de Productos:** Al hacer clic en "Ver Nombres de Productos", el usuario accede a una pantalla con una lista de los nombres de sus productos.
6. **Volver:** El botón "Volver" regresa al usuario al menú principal sin realizar ninguna acción en la gestión de productos.

### 2.1.13. Pantalla Añadir Producto



#### Vista: PantallaAddProductUser

La vista **PantallaAddProductUser** permite a los usuarios añadir un producto proporcionando su nombre y precio. Es una funcionalidad clave para gestionar productos dentro de la aplicación, asegurando una experiencia sencilla e intuitiva para el usuario. Además, incluye opciones para regresar a la pantalla anterior.

#### Atributos:

- **JPanel panelAddProduct:** El panel principal de la pantalla de añadir producto. Utiliza un GridBagLayout para organizar los componentes de manera flexible y uniforme.
- **JLabel titleLabel:** Etiqueta principal que muestra el texto "Añadir Producto". Está estilizada con una fuente grande y en negrita para destacar.
- **JLabel productNameLabel:** Etiqueta que describe el campo donde se debe ingresar el nombre del producto.
- **TextField productNameField:** Campo de texto donde el usuario puede ingresar el nombre del producto que desea añadir.
- **JLabel productPriceLabel:** Etiqueta que describe el campo donde se debe ingresar el precio del producto.
- **TextField productPriceField:** Campo de texto para ingresar el precio del producto. Solo acepta valores numéricos válidos.
- **Button anadirButton:** Botón que activa la funcionalidad de añadir un producto al sistema.
- **Button backButton:** Botón que permite regresar a la pantalla anterior de gestión de productos.

## Métodos:

### **PantallaAddProductUser(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza la pantalla de añadir productos.

- **Título:** Un JLabel en la parte superior del panel con el texto "Añadir Producto", estilizado con una fuente grande y en negrita.
- **Panel de Componentes:** Los elementos (etiquetas, campos de texto y botones) están organizados con un GridBagLayout, que asegura una disposición ordenada y centrada en el panel.
- **Estilo:** Los componentes tienen un fondo azul claro (Color(230, 240, 255)) para proporcionar un diseño limpio y profesional.
- **Botones y acciones:**
  - **Añadir:**
    - Recupera el nombre y el precio ingresados por el usuario desde los campos de texto.
    - Valida que ambos campos no estén vacíos y que el precio sea un número válido.
    - Llama al método ctrlPresentacio.addProductToUser() para registrar el producto.
    - Muestra un mensaje de éxito o error dependiendo del resultado.
    - Limpia los campos después de un registro exitoso.
  - **Volver:** Permite regresar a la pantalla de gestión de productos mediante el método ctrlPresentacio.mostrarPantallaProductos().

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

## **Flujo de interacción:**

1. **Introducir nombre del producto:** El usuario escribe el nombre del producto en el campo correspondiente.
2. **Introducir precio del producto:** El usuario introduce el precio del producto en el campo correspondiente.
3. **Añadir Producto:**
  - a. El usuario hace clic en el botón "Añadir".
  - b. Si algún campo está vacío, se muestra un mensaje de error.
  - c. Si el precio no es un número válido, se muestra un mensaje de error.
  - d. Si los datos son válidos, se llama al controlador para añadir el producto al usuario.
    - i. Si la operación es exitosa, se muestra un mensaje de confirmación y se regresa a la pantalla principal de productos.
    - ii. Si ocurre un error, se informa al usuario con un mensaje adecuado.
4. **Cancelar:** El usuario hace clic en el botón "Volver" para regresar a la pantalla principal de productos sin añadir nada.

### 2.1.14. Pantalla Eliminar Producto



#### Vista: PantallaDeleteProductUser

La vista **PantallaDeleteProductUser** proporciona la funcionalidad para eliminar un producto específico de la aplicación. Permite al usuario ingresar el nombre del producto y confirmar su eliminación, con opciones claras y un diseño intuitivo.

#### Atributos:

- **JPanel panelDelProduct:**  
El panel principal de la pantalla de eliminación de productos. Utiliza un `GridBagLayout` para organizar los componentes de manera ordenada.
- **JLabel titleLabel:**  
Etiqueta principal con el texto "**Eliminar Producto**", estilizada con una fuente grande y en negrita para destacar el propósito de la pantalla.
- **JLabel productNameLabel:**  
Etiqueta que describe el campo donde el usuario debe ingresar el nombre del producto que se desea eliminar.
- **JTextField productNameField:**  
Campo de texto donde el usuario introduce el nombre del producto.
- **JButton eliminarButton:**  
Botón que ejecuta la acción de eliminar el producto ingresado.
- **JButton backButton:**  
Botón que permite al usuario regresar a la pantalla anterior de gestión de productos.

#### Métodos:

### **PantallaDeleteProductUser(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza los componentes de la pantalla.

- **Título:** Un JLabel en la parte superior del panel con el texto "**Eliminar Producto**".
- **Panel de Componentes:** Los elementos (etiquetas, campo de texto y botones) están organizados con GridBagLayout, que garantiza una disposición centrada y espaciada.
- **Estilo:** El fondo del panel es de color azul claro, proporcionando una apariencia profesional y agradable.
- **Botones y Acciones:**
  - **Eliminar:**
    - Recupera el nombre del producto desde el campo de texto.
    - Valida que el campo no esté vacío.
    - Llama al método ctrlPresentacio.removeProductUser() para eliminar el producto del sistema.
    - Muestra mensajes de éxito o error dependiendo del resultado.
    - Limpia el campo de texto después de la operación.
  - **Volver:** Permite regresar a la pantalla principal de gestión de productos mediante el método ctrlPresentacio.mostrarPantallaProductos()

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Introducir nombre del producto:** El usuario escribe el nombre del producto que desea eliminar en el campo correspondiente.
2. **Eliminar Producto:**
  - a. El usuario hace clic en el botón "Eliminar".
  - b. Si el campo está vacío, se muestra un mensaje de error.
  - c. Si el nombre es válido, se llama al controlador para eliminar el producto del usuario.



- i. Si la operación es exitosa, se muestra un mensaje de confirmación y se regresa a la pantalla principal de productos.
  - ii. Si ocurre un error, se informa al usuario con un mensaje adecuado.
- 3. **Cancelar:** El usuario hace clic en el botón "Volver" para regresar a la pantalla principal de productos sin realizar ninguna acción.

### 2.1.15. Pantalla Actualizar Producto



**Actualizar Producto**

Nombre del Producto:

Nuevo Precio del Producto:

**Actualizar**

**Volver**

#### Vista: PantallaUpdateProductUser

La vista **PantallaUpdateProductUser** permite actualizar la información de un producto existente, específicamente su precio. El usuario puede ingresar el nombre del producto y su nuevo precio para realizar la actualización de manera sencilla y rápida.

#### Atributos:

- **JPanel panelUpdateProduct:**  
El panel principal de la pantalla de actualización de productos. Utiliza un `GridBagLayout` para organizar los componentes de forma centrada y ordenada.
- **JLabel titleLabel:**  
Etiqueta principal con el texto "**Actualizar Producto**", estilizada con una fuente grande y en negrita para destacar el propósito de la pantalla.
- **JLabel productNameLabel:**  
Etiqueta descriptiva para el campo donde se ingresa el nombre del producto.
- **TextField productNameField:**  
Campo de texto donde el usuario escribe el nombre del producto a actualizar.
- **JLabel newPriceLabel:**  
Etiqueta descriptiva para el campo donde se ingresa el nuevo precio del producto.
- **TextField newPriceField:**  
Campo de texto donde el usuario introduce el nuevo precio del producto.
- **Button actualizarButton:**  
Botón que ejecuta la acción de actualizar el producto.
- **Button backButton:**  
Botón que permite regresar a la pantalla principal de gestión de productos.

## Métodos:

### **PantallaUpdateProductUser(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza los componentes de la pantalla.

- **Título:** Un JLabel en la parte superior del panel con el texto "**Actualizar Producto**", alineado al centro y estilizado con Font.BOLD.
- **Panel de Componentes:** Los componentes (etiquetas, campos de texto y botones) están organizados verticalmente con GridBagLayout y márgenes uniformes (Insets).
- **Estilo:** El fondo del panel tiene un color azul claro (Color(230, 240, 255)), proporcionando una interfaz agradable y profesional.
- **Botones y acciones:**
  - **Actualizar:**
    - Recupera el nombre del producto y el nuevo precio desde los campos de texto.
    - Valida que el nuevo precio sea un valor numérico.
    - Llama al método ctrlPresentacio.updateProductUser() para actualizar el producto en el sistema.
    - Muestra mensajes de éxito o error dependiendo del resultado.
    - Limpia los campos de texto después de realizar la operación.
  - **Volver:** Permite regresar a la pantalla principal de gestión de productos mediante el método ctrlPresentacio.mostrarPantallaProductos().

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

### **getPanel()**


Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

## **Flujo de interacción:**

1. **Introducir datos del producto:** El usuario escribe el nombre del producto que desea actualizar en el campo correspondiente.
2. **Introducir nuevo precio:** El usuario introduce el nuevo precio del producto en el campo correspondiente.
3. **Actualizar producto:**

- a. El usuario hace clic en el botón "Actualizar".
  - b. Si el nuevo precio no es un valor numérico, se muestra un mensaje de error.
  - c. Si el nombre del producto y el nuevo precio son válidos, se llama al controlador para actualizar el producto del usuario.
    - i. Si la operación es exitosa, se muestra un mensaje de confirmación y se regresa a la pantalla principal de productos.
    - ii. Si ocurre un error, se informa al usuario con un mensaje adecuado.
4. **Cancelar:** El usuario hace clic en el botón "Volver" para regresar a la pantalla principal de productos sin realizar ninguna acción.

### 2.1.16. Pantalla Consultar Atributos



#### Vista: **PantallaGetAttribProduct**

La vista **PantallaGetAttribProduct** permite a los usuarios consultar los atributos de un producto específico proporcionando su nombre. Es una funcionalidad clave para obtener información detallada de los productos dentro de la aplicación. Además, incluye opciones para regresar a la pantalla anterior de gestión de productos.

#### Atributos:

- **JPanel panelUpdateProduct:**  
El panel principal de la pantalla de consulta de atributos. Utiliza un GridBagLayout para organizar los componentes de manera ordenada y centrada.
- **JLabel titleLabel:**  
Etiqueta principal que muestra el texto "Ver Atributos". Está estilizada con una fuente grande, en negrita, y centrada para destacar.
- **JLabel productNameLabel:**  
Etiqueta que indica el campo donde se debe ingresar el nombre del producto.
- **TextField productNameField:**  
Campo de texto donde el usuario puede ingresar el nombre del producto que desea consultar.
- **Button consultarButton:**  
Botón que activa la funcionalidad para obtener y mostrar los atributos del producto ingresado.
- **Button backButton:**  
Botón que permite regresar a la pantalla anterior de gestión de productos.

## Métodos:

### **PantallaGetAttribProduct(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza la pantalla de consulta de atributos.

- **Título:** Un JLabel en la parte superior del panel con el texto "Ver Atributos", estilizado con una fuente grande y en negrita.
- **Panel de Componentes:** Los elementos (etiquetas, campos de texto y botones) están organizados con un GridBagLayout, asegurando una disposición centrada.
- **Estilo:** Los componentes tienen un fondo azul claro (Color(230, 240, 255)) para proporcionar un diseño limpio y profesional.
- **Botones y acciones:**
  - **Consultar:**
    - Recupera el texto ingresado en productNameField.
    - Valida que el campo no esté vacío.
    - Llama al método ctrlPresentacio.getProductAttributesUser(productName) para obtener los atributos del producto.
    - Muestra los atributos en un cuadro de diálogo si la operación es exitosa o un mensaje de error en caso contrario.
    - Limpia el campo de texto después de ejecutar la operación.
  - **Volver:** Permite regresar a la pantalla de gestión de productos llamando al método ctrlPresentacio.mostrarPantallaProductos().

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

## Flujo de interacción:

1. **Introducir datos del producto:** El usuario escribe el nombre del producto en el campo correspondiente.
2. **Consultar atributos del producto:**
  - a. El usuario hace clic en el botón "Consultar".

- b. Si el campo del nombre del producto está vacío, se muestra un mensaje de error.
  - c. Si el nombre del producto es válido, se llama al controlador para obtener los atributos del producto.
    - i. Si la operación es exitosa, los atributos se muestran al usuario en un mensaje emergente.
    - ii. Si ocurre un error, se informa al usuario con un mensaje adecuado.
3. **Cancelar:** El usuario hace clic en el botón "Volver" para regresar a la pantalla principal de productos sin realizar ninguna acción.

### 2.1.17. Pantalla Productos de Usuario



#### Vista: PantallaGetNamesProducts

La vista **PantallaGetNamesProducts** permite a los usuarios visualizar una lista de los productos asociados a su cuenta. Es una funcionalidad clave para mostrar los productos disponibles para el usuario en la aplicación, facilitando la navegación y la gestión de los productos.

#### Atributos:

- **JPanel panelNameProducts:**  
El panel principal de la pantalla de productos. Utiliza un `GridBagLayout` para organizar los componentes de manera flexible y uniforme.
- **JLabel titleLabel:**  
Etiqueta principal que muestra el texto "Productos de Usuario". Está estilizada con una fuente grande y en negrita para destacar.
- **JList<String> productList:**  
Lista que muestra los nombres de los productos disponibles para el usuario, cargados desde el controlador. Cada producto se presenta como un ítem de la lista.
- **JScrollPane scrollPane:**  
Contenedor de la lista de productos (`productList`) con un tamaño preferido para asegurar la visibilidad adecuada de los elementos en la interfaz.
- **JButton backButton:**  
Botón que permite regresar a la pantalla anterior de gestión de productos.

#### Métodos:



### **PantallaGetNamesProducts(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza la pantalla de productos del usuario.

- **Título:** Un JLabel en la parte superior del panel con el texto "Productos de Usuario", estilizado con una fuente grande y en negrita.
- **Carga de Productos:** Se obtiene la lista de nombres de productos del controlador (ctrlPresentacio.getProductNamesForUser()) y se muestra en una JList, que a su vez está contenida en un JScrollPane.
- **Estilo:** Los componentes tienen un fondo azul claro para proporcionar un diseño limpio y profesional.
- **Botones y acciones:**
  - **Volver:** Permite regresar a la pantalla de gestión de productos mediante el método ctrlPresentacio.mostrarPantallaProductos().

### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

1. **Cargar lista de productos:** El sistema obtiene los nombres de los productos asociados al usuario y los muestra en una lista desplegable.
2. **Consultar productos:**
  - a. El sistema muestra la lista de productos en un componente desplazable (JList).
  - b. Si ocurre un error al obtener los nombres de los productos, se muestra un mensaje de error.
3. **Cancelar:** El usuario hace clic en el botón "Volver" para regresar a la pantalla principal de productos sin realizar ninguna acción.

### 2.1.18. Pantalla Gestión de Similitudes



#### Vista: PantallaSimilitudes

La vista **PantallaSimilitudes** permite gestionar las similitudes dentro del sistema, ofreciendo opciones para modificar, eliminar o consultar similitudes. También incluye una funcionalidad para regresar al menú principal, asegurando una navegación fluida y una experiencia intuitiva para el usuario.

#### Atributos:

- **JPanel panelSimilitudes:** El panel principal de la pantalla de gestión de similitudes. Está diseñado con un `GridBagLayout` para proporcionar una disposición flexible y centrada.
- **JLabel titleLabel:** Etiqueta principal que muestra el texto "Gestión de Similitudes". Está estilizada con una fuente grande y en negrita para destacar su importancia.
- **JButton setSimilarityButton:** Botón que permite acceder a la funcionalidad de modificar una similitud.
- **JButton removeSimilarityButton:** Botón que facilita la eliminación de una similitud.
- **JButton getSimilarityButton:** Botón que muestra la funcionalidad de consultar una similitud.
- **JButton backButton:** Botón para regresar al menú principal. Está diseñado con un fondo rojo para diferenciarlo de las demás acciones.

#### Métodos:

**PantallaSimilitudes(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza todos los componentes de la pantalla, aplicando estilos y configurando las acciones de los botones.

**initializePanel(CtrlPresentacio ctrlPresentacio):**

Método privado que organiza los componentes visuales y configura las acciones de los botones. Estiliza el panel principal con un fondo azul claro y añade bordes internos para mejorar la presentación.

**estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

**getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

**Flujo de interacción:**

1. **Modificar Similitud:** El usuario hace clic en el botón "Modificar Similitud" y se muestra la pantalla correspondiente para modificar la similitud entre productos.
2. **Eliminar Similitud:** El usuario hace clic en el botón "Eliminar Similitud" y se muestra la pantalla correspondiente para eliminar una similitud existente entre productos.
3. **Consultar Similitud:** El usuario hace clic en el botón "Consultar Similitud" y muestra la pantalla correspondiente para consultar las similitudes existentes.
4. **Volver:** El usuario hace clic en el botón "Volver" y regresa a la pantalla principal del usuario sin realizar ninguna acción.

### 2.1.19. Pantalla Modificar Similitud



#### Vista: **PantallaSetSimilarity**

La vista **PantallaSetSimilarity** permite a los usuarios modificar la similitud entre dos productos dentro del sistema. Esta funcionalidad es esencial para gestionar relaciones entre productos de manera personalizada. También incluye una opción para regresar a la pantalla anterior de gestión de similitudes.

#### Atributos:

- **JPanel panelSetSimilarity:** El panel principal de la pantalla para modificar similitudes. Diseñado con un `GridBagLayout` para garantizar una disposición ordenada y centrada.
- **JLabel titleLabel:** Etiqueta principal que muestra el texto "Modificar Similitud". Estilizada con una fuente grande y en negrita para destacar.
- **JLabel product1Label:** Etiqueta que describe el campo donde se debe ingresar el nombre del primer producto.
- **TextField product1Field:** Campo de texto para ingresar el nombre del primer producto.
- **JLabel product2Label:** Etiqueta que describe el campo donde se debe ingresar el nombre del segundo producto.
- **TextField product2Field:** Campo de texto para ingresar el nombre del segundo producto.
- **JLabel similarityLabel:** Etiqueta que describe el campo donde se debe ingresar el valor de similitud entre los productos.
- **TextField similarityField:** Campo de texto para ingresar el valor de similitud (entre 0 y 1).

- **JButton submitButton**: Botón que activa la funcionalidad para modificar la similitud.
- **JButton backButton**: Botón que permite regresar a la pantalla anterior.

### **Métodos:**

#### **PantallaSetSimilarity(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza todos los componentes de la pantalla.

#### **initializePanel(CtrlPresentacio ctrlPresentacio):**

Método privado que organiza los componentes visuales y configura las acciones de los botones. Aplica un fondo azul claro (**Color(230, 240, 255)**) al panel principal.

#### **estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

#### **getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

### **Flujo de interacción:**

#### **1. Introducir datos de los productos:**


- a. El usuario escribe el nombre del "Producto 1" y "Producto 2" en los campos correspondientes.
- b. El usuario introduce la similitud deseada entre los productos en el campo "Similitud (0-1)", que debe ser un valor numérico entre 0 y 1.

#### **2. Modificar similitud:** El usuario hace clic en el botón "Modificar".

- a. Si los productos no existen o los datos ingresados son incorrectos, se muestra un mensaje de error. En estos casos, se validan los productos y la similitud:
- b. Si algún producto no existe, se muestra un mensaje de error.
- c. Si la similitud no está en el rango [0, 1], se muestra un mensaje de error.
- d. Si la similitud no es un número válido, se muestra un mensaje de error.
- e. Si la similitud se ha modificado correctamente, se muestra un mensaje de éxito indicando que la similitud se ha modificado con éxito.

3. **Si ocurre un error:** Si el proceso de modificación de la similitud no se puede completar, se muestra un mensaje de error indicando que hubo un problema al modificar la similitud.
4. **Cancelar:** El usuario hace clic en el botón "Volver" para regresar a la pantalla principal de similitudes sin realizar ninguna acción.

### 2.1.20. Pantalla Eliminar Similitud



#### Vista: **PantallaRemoveSimilarity**

La vista **PantallaRemoveSimilarity** permite a los usuarios eliminar la similitud entre dos productos. Es una funcionalidad clave para gestionar relaciones entre productos, facilitando su modificación o eliminación según sea necesario. Además, incluye una opción para regresar a la pantalla de gestión de similitudes.

#### Atributos:

- **JPanel panelRemoveSimilarity:** Panel principal de la pantalla para eliminar similitudes. Diseñado con un `GridBagLayout` para garantizar una disposición ordenada y centrada.
- **JLabel titleLabel:** Etiqueta principal que muestra el texto "Eliminar Similitud". Está estilizada con una fuente grande y en negrita para destacar.
- **JLabel product1Label:** Etiqueta que describe el campo donde se debe ingresar el nombre del primer producto.
- **TextField product1Field:** Campo de texto para ingresar el nombre del primer producto.
- **JLabel product2Label:** Etiqueta que describe el campo donde se debe ingresar el nombre del segundo producto.
- **TextField product2Field:** Campo de texto para ingresar el nombre del segundo producto.
- **JBUTTON submitButton:** Botón que activa la funcionalidad para eliminar la similitud.
- **JBUTTON backButton:** Botón que permite regresar a la pantalla anterior.

#### Métodos:

**PantallaRemoveSimilarity(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza los componentes visuales y las acciones de la pantalla.

**initializePanel(CtrlPresentacio ctrlPresentacio):**

Método privado que organiza los componentes visuales y configura las acciones de los botones. Aplica un fondo azul claro (**Color(230, 240, 255)**) al panel principal.

**estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

**getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

**Flujo de interacción:**

1. **Introducir productos:** El usuario escribe el nombre de los productos ("Producto 1" y "Producto 2") en los campos correspondientes.
2. **Eliminar Similitud:**
  - a. El usuario hace clic en el botón "Eliminar".
  - b. Si uno o ambos productos no existen: Se muestra un mensaje de error diciendo que los productos no están registrados.
  - c. Si ambos productos existen:
    - i. Se llama al controlador para eliminar la similitud entre los productos.
    - ii. Se muestra un mensaje de confirmación y se regresa a la pantalla principal de similitudes.
  - d. Si ocurre un error: Se informa al usuario con un mensaje adecuado.
3. **Cancelar:** El usuario hace clic en el botón "Volver" para regresar a la pantalla de gestión de similitudes sin realizar ninguna acción.



### 2.1.21. Pantalla Consultar Similitud



#### Vista: PantallaGetSimilarity

La vista **PantallaGetSimilarity** permite a los usuarios consultar la similitud registrada entre dos productos. Es una funcionalidad clave para visualizar las relaciones definidas entre productos en el sistema. Además, incluye una opción para regresar a la pantalla de gestión de similitudes.

#### Atributos:

- **JPanel panelGetSimilarity:** Panel principal de la pantalla para consultar similitudes. Diseñado con un GridBagLayout para garantizar una disposición clara y ordenada.
- **JLabel titleLabel:** Etiqueta principal que muestra el texto "Consultar Similitud". Está estilizada con una fuente grande y en negrita para destacar.
- **JLabel product1Label:** Etiqueta que describe el campo donde se debe ingresar el nombre del primer producto.
- **JTextField product1Field:** Campo de texto para ingresar el nombre del primer producto.
- **JLabel product2Label:** Etiqueta que describe el campo donde se debe ingresar el nombre del segundo producto.
- **JTextField product2Field:** Campo de texto para ingresar el nombre del segundo producto.
- **JButton submitButton:** Botón que activa la funcionalidad para consultar la similitud.
- **JButton backButton:** Botón que permite regresar a la pantalla anterior.

#### Métodos:

**PantallaGetSimilarity(CtrlPresentacio ctrlPresentacio):**

Constructor de la clase que inicializa y organiza los componentes visuales y las acciones de la pantalla.

**initializePanel(CtrlPresentacio ctrlPresentacio):**

Método privado que organiza los componentes visuales y configura las acciones de los botones. Aplica un fondo azul claro (**Color(230, 240, 255)**) al panel principal.

**estilizarBoton(JButton boton, Color colorFondo, Color colorTexto)**

Aplica un diseño personalizado a los botones para:

- Cambiar colores de fondo y texto.
- Aplicar bordes estilizados.
- Añadir efectos visuales al interactuar con el ratón (mouseEntered y mouseExited).

**getPanel()**

Retorna el panel principal panelOrganize para integrarlo en el contenedor principal de la aplicación.

**Flujo de interacción:**

1. El usuario escribe el nombre de los productos ("Producto 1" y "Producto 2") en los campos correspondientes.
2. **Consultar Similitud:**
  - a. El usuario hace clic en el botón "Consultar".
  - b. Si uno o ambos productos no existen: Se muestra un mensaje de error diciendo que los productos no están registrados.
  - c. Si ambos productos existen:
    - i. Se llama al controlador para obtener la similitud entre los productos.
    - ii. Si existe una similitud registrada: Se muestra un mensaje indicando la similitud entre los productos.
  - d. Si no existe similitud registrada: Se muestra un mensaje informando que no hay similitud entre los productos.
3. **Cancelar:**El usuario hace clic en el botón "Volver" para regresar a la pantalla de gestión de similitudes sin realizar ninguna acción.

## 2.2. Explicación del controlador de presentación

### 2.2.1. Atributos Principales

- **CtrlDomain domain:**  
Controlador de la capa de dominio que maneja la lógica central del sistema. CtrlPresentacio se comunica con CtrlDomain para realizar operaciones como consultas, actualizaciones o eliminaciones en el sistema.
- **CardLayout cardLayout:**  
Un diseño que permite cambiar dinámicamente entre diferentes pantallas en un único contenedor (mainPanel). Cada pantalla se identifica por un nombre único.
- **JPanel mainPanel:**  
Panel principal que contiene todas las vistas del sistema. Es el contenedor gestionado por CardLayout.
- **JFrame mainFrame:**  
Ventana principal de la aplicación, donde se visualizan las pantallas.

### 2.2.2. Métodos Principales

#### Inicialización

- **iniPresentacio():**  
Es el punto de entrada para inicializar la presentación. Este método:
  - Crea la ventana principal (mainFrame) y la configura.
  - Establece un CardLayout para gestionar las vistas.
  - Añade varias pantallas (Inicio, Registro, Login, etc.) al mainPanel.
  - Muestra la pantalla de inicio al cargar.

### 2.2.3. Navegación entre Pantallas

Los métodos mostrarPantallaX() permiten cambiar entre las vistas gestionadas por el CardLayout. Ejemplos:

- **mostrarPantallaRegistro():** Muestra la pantalla de registro y actualiza el título de la ventana.
- **mostrarPantallaLogin():** Muestra la pantalla de inicio de sesión.
- **mostrarPantallaSimilitudes():** Verifica si el usuario ha iniciado sesión antes de mostrar la pantalla de gestión de similitudes.
- **Pantallas relacionadas con estanterías:**
  - Métodos como mostrarPantallaAddShelf() o mostrarPantallaDeleteShelf() gestionan vistas específicas de estanterías y verifican que el usuario esté autenticado antes de permitir el acceso.

## Gestión de Productos y Similitudes

- **productExists(String productName):**  
Verifica si un producto específico existe en la lista de productos del usuario autenticado.
- **setProductSimilarityForUser(String product1, String product2, double similarity):**  
Actualiza la similitud entre dos productos del usuario actual.
- **removeProductSimilarityForUser(String product1, String product2):**  
Elimina la similitud registrada entre dos productos.
- **getProductSimilarityForUser(String product1, String product2):**  
Recupera la similitud registrada entre dos productos.

## Gestión de Usuario

- **registerUser(String name, String username, String password):**  
Llama a la capa de dominio para registrar un nuevo usuario.
- **loginUser(String username, String password):**  
Autentica a un usuario en el sistema a través de CtrlDomain.
- **modificarUsuario(String username, String actualPassword, String newPassword):**  
Permite al usuario modificar su nombre de usuario y/o contraseña.
- **cerrarSesion():**  
Cierra la sesión del usuario actual y redirige a la pantalla de inicio.

## Gestión de Estanterías

- Métodos como `organizeShelf()`, `addShelfForUser()`, y `removeShelfForUser()` permiten gestionar estanterías de productos, como organizar, añadir, eliminar productos o eliminar estanterías completas.

## Interacciones con la Vista

- **setTitulo(String title):**  
Cambia el título de la ventana principal (`mainFrame`).

- **mainPanel.add(vista.getPanel(), "NombreVista"):**  
Añade una nueva vista al panel principal para que sea accesible a través del CardLayout.

#### 2.2.4. Lógica General

1. **Cambio de Pantallas:**

Cada vista tiene su propio método para mostrarla, que verifica condiciones previas (como que el usuario esté autenticado) y luego la carga en el mainPanel.

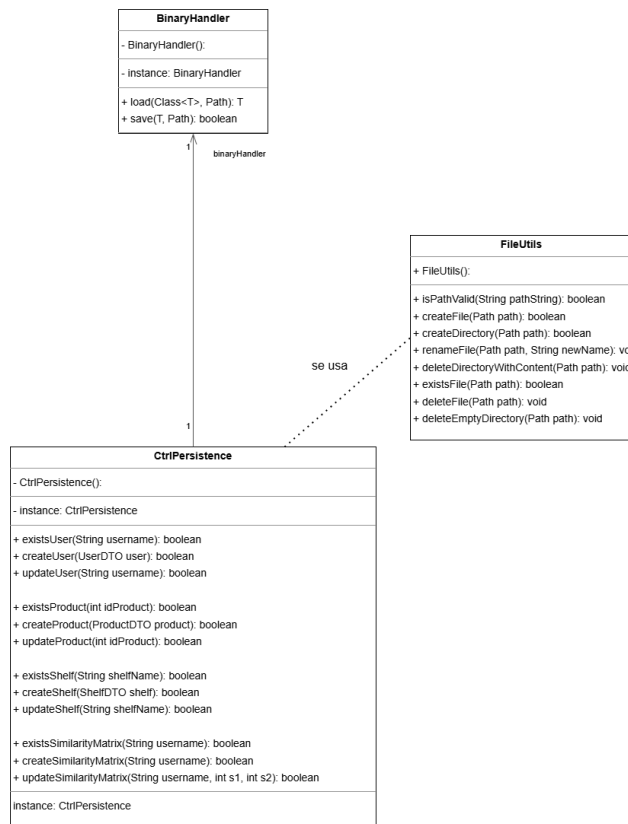
2. **Comunicación con el Modelo (Dominio):**

La clase interactúa con CtrlDomain para realizar operaciones de consulta, modificación o eliminación en los datos del sistema.

3. **Gestión de Errores:**

Si ocurre algún error en las operaciones del dominio, muestra mensajes al usuario mediante JOptionPane.

### 3. Diagrama de las clases y controladores de la capa de persistencia



#### Clase CtrlPersistencia

Esta clase es responsable de la gestión de la capa de persistencia en la arquitectura del proyecto. Actualmente, no todas las funciones han sido implementadas.

La capa de persistencia está sujeta a cambios y mejoras, incluyendo:

- Implementación de nuevas funciones según los requisitos adicionales.

#### 3.1. Explicación de las clases de persistencia

Las clases principales de la capa de persistencia incluyen un controlador, que actúa como intermediario directo con los *subcontroladores* y se encarga de organizar las diversas funcionalidades que deseamos proporcionar. Además, contamos con una clase estática que ofrece métodos de utilidad accesibles para todo el marco de trabajo, y, por último, un gestor de archivos binarios cuya responsabilidad principal es la serialización y deserialización de

archivos, siendo este el componente que interactúa directamente con el contenido que queremos guardar.

El objetivo principal de nuestro proyecto ha sido implementar un proceso eficiente de mantenimiento y actualización de datos en tiempo real para nuestra aplicación. Para lograrlo, hemos diseñado y estructurado la capa de persistencia siguiendo cuidadosamente los principios de este paradigma. En otras palabras, nuestra meta ha sido garantizar que la capa de datos permanezca actualizada en todo momento, ofreciendo así un sistema altamente fiable y robusto frente a posibles errores o desconexiones inesperadas del sistema.

Esta arquitectura permite mantener una sincronización constante entre los datos en memoria y los almacenados de manera persistente, lo cual resulta esencial para proporcionar una experiencia de usuario consistente.

Adicionalmente, esta estructura modular facilita la escalabilidad del sistema, permitiendo que se puedan añadir o modificar funcionalidades en el futuro sin comprometer la integridad de los datos. El controlador principal coordina todas las operaciones de la capa, asegurando que cada componente cumpla su función específica de manera eficiente.

En definitiva, el diseño de esta capa de persistencia ha sido clave para garantizar un sistema sólido, adaptable y preparado.

### 3.2. Explicación del controlador de persistencia

La clase **CtrlPersistence** es un controlador de la capa de persistencia que centraliza las operaciones relacionadas con el manejo de datos persistentes en la aplicación. Utiliza un patrón singleton para garantizar una única instancia activa y hace uso de un **BinaryHandler** y utilidades de archivos (FileUtils) para gestionar la creación, almacenamiento y actualización de usuarios, productos, estanterías y matrices de similitud en el sistema. Entre sus responsabilidades principales se incluyen:

- Validar la existencia de usuarios, productos, estanterías y matrices de similitud en el sistema de archivos.
- Crear y actualizar entidades persistentes usando Data Transfer Object (DTO) como usuarios (UserDTO), productos (ProductDTO) y estanterías (ShelfDTO).
- Garantizar la correcta organización y almacenamiento de los datos en archivos binarios.

Esta clase actúa como un puente entre la lógica del dominio (en este caso los sub controladores) y la capa de persistencia, asegurando la integridad y fiabilidad de los datos.

## 4. Descripción de los algoritmos y las estructuras de datos

Las principales estructuras de datos que han cambiado han sido aquellas destinadas a almacenar información de cada uno de los usuarios. Inicialmente, estas estructuras se diseñaron para mantener los datos en memoria durante la ejecución del programa, lo que permitía un acceso directo y rápido a la información. Sin embargo, este enfoque presentaba ciertas limitaciones.

### 4.1. Cambios en Control User

#### ENTREGA 1:

```
public class CtrlUser { 5 usages  omar.antonio.cornejo *  
  
    private static Map<String, User> users; // Almacena los usuarios 10 usages  
    private static Map<String, SimilarityMatrix> similarityMatrix; 9 usages  
    private static String usrSelected; // Usuario actual autenticado 22 usages  
    private static CtrlUser singletonObject; 3 usages
```

Los atributos **users** y **similarityMatrix** desempeñaban un papel fundamental en el diseño inicial del sistema, ya que se utilizaban para establecer una asociación directa entre los nombres de usuario (usernames), que actuaban como claves únicas dentro de un mapa, y las respectivas clases que contenían toda la información vinculada a cada usuario.

#### ENTREGA 2 y 3:

```
public class CtrlUser { 5 usages  omar.antonio.cornejo +2 *  
  
    private static SimilarityMatrix similarityMatrix; 9 usages  
    private static User user; // Usuario actual autenticado 2 usages  
    private static CtrlUser singletonObject; 3 usages  
    private final CtrlPersistence ctrlPersistence; 3 usages
```

Como se puede observar, en la nueva versión del sistema únicamente se almacena en memoria el usuario autenticado en ese momento, junto con su correspondiente matriz de similitudes asociada a los productos registrados.



## 4.2. Cambios en Control Product

### ENTREGA 1:

```
public class CtrlProduct { 5 usages  📄 omar.antonio.cornejo +1 *  
  
    /* Asocia cada usuario a un mapa <nombre_producto -> producto */  
    private Map<String, Map<String, Product>> products; 8 usages  
  
    private static CtrlProduct singletonObject; 3 usages  
    private int productIdCounter; 2 usages
```

En esta captura de pantalla extraída del código antiguo, se puede apreciar cómo el atributo **products** era responsable de vincular cada usuario de la aplicación con su colección de productos. Este atributo funcionaba como un mapa que relacionaba los identificadores de usuario con sus respectivos conjuntos de productos, permitiendo gestionar esta información de forma directa dentro del sistema.

### ENTREGA 2 y 3:

```
public class CtrlProduct { 5 usages  📄 omar.antonio.cornejo +1 *  
  
    private Map<String, Product> products; 8 usages  
    private static CtrlProduct singletonObject; 3 usages  
    private int productIdCounter; 2 usages
```

En cuanto a las últimas entregas, analizando la fotografía del código actual es claro que la funcionalidad del atributo **products** se centrará en guardar el conjunto de productos vinculados exclusivamente al usuario autenticado en ese instante.

### 4.3. Cambios en Control Shelf

#### ENTREGA 1:

```
public class CtrlShelf { 5 usages  👤 omar.antonio.cornejo *  
  
    // Almacena las estanterías de cada usuario  
    private Map<String, Map<String, Shelf>> shelves; 7 usages  
    private static CtrlShelf singletonObject; 3 usages
```

En este controlador, el atributo **shelves** se utilizaba para almacenar las estanterías asociadas a cada usuario del sistema. Este atributo funcionaba como un mapa que vinculaba una clave única, correspondiente al identificador del usuario, con su respectiva colección de estanterías.

#### ENTREGA 2 y 3:

```
public class CtrlShelf { 5 usages  👤 omar.antonio.cornejo *  
  
    // Almacena las estanterías de cada usuario  
    private Map<String, Shelf> userShelves; 2 usages  
    private static CtrlShelf singletonObject; 3 usages
```

En cuanto al modelo de datos definitivos para **CtrlShelf**, es evidente, como se observa en la fotografía del código actual, que la funcionalidad del atributo **userShelves** se centrará en guardar el conjunto de estanterías vinculadas exclusivamente al usuario autenticado en ese momento.

### 4.4. Cambios en los Algoritmos

En lo referente a las funciones/métodos y los algoritmos que se utilizan en ellas, cabe destacar que no han sido modificadas de forma sustancial. Principalmente hemos añadido algún paso extra para las funciones de autenticación de usuario debido a la nueva capa de persistencia y por otra parte también se ha reducido la complejidad de muchas de ellas debido a las nuevas estructuras de datos simplificadas usadas en esta nueva versión.