

Universidad Técnica Estatal de Quevedo

Gestión de la configuración de software

Docente

Ing. Portilla Olvera Gilberto Elias

Estudiantes

Jorge Steven Gualpa Gia
Valeska Sofia Chica Valfre
Geovanny Alexander Ochoa Gilces
Jordy Alejandro Vilcacundo Chiluisa



Enlace al repositorio: <https://github.com/gochoag/SGFactuacion.git>

Planificación de desarrollo

Planificador de proyecto de Facturación

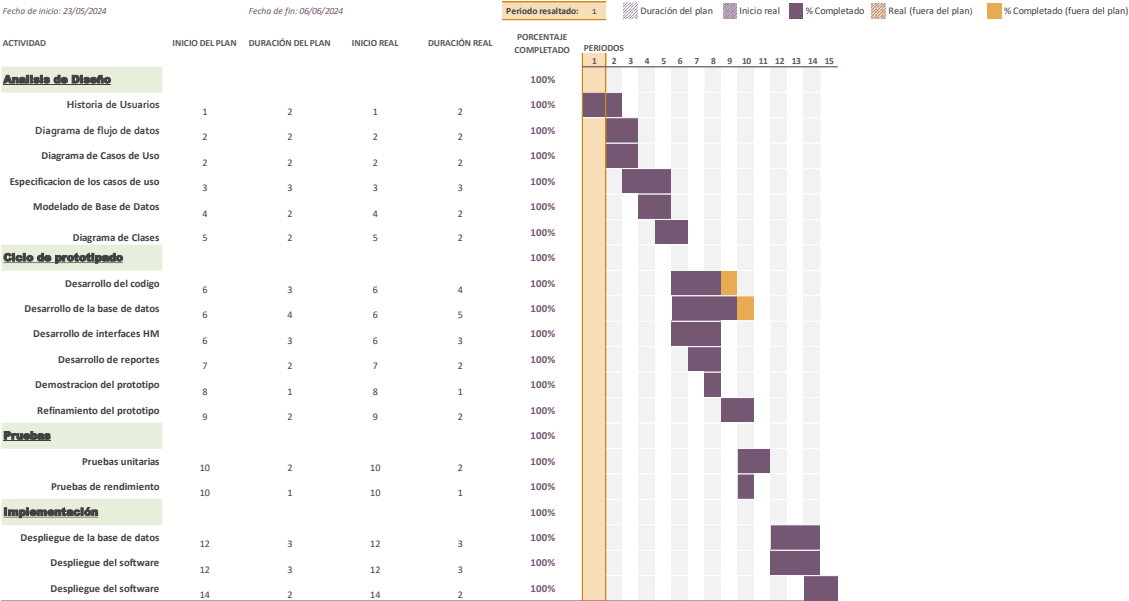

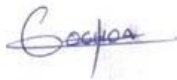


Imagen 1: Planificación del proyecto

Solicitud de cambios en el software

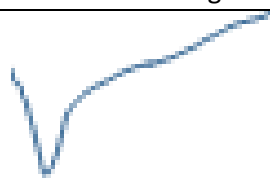
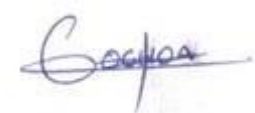
Fecha hora: 22/08/2024	N° Solicitud: 00002_2024
Solicitante: Elías Portilla Olvera (rol de cliente)	Receptor: Geovanny Ochoa (ACC)
A. Descripción del cambio: <ul style="list-style-type: none">Adicionar un nuevo módulo de proveedores al sistema actual.Implementar la funcionalidad para agregar, editar, eliminar y visualizar proveedores.Crear una interfaz de usuario para la gestión de proveedores.Establecer la relación entre proveedores y productos en la base de datos.Implementar una función de búsqueda y filtrado para localizar proveedores específicos.Actualizar los modelos de la base de datos para incluir la entidad de proveedores.Modificar el diagrama de clases para reflejar la nueva entidad y sus relaciones.Actualizar los diagramas de casos de uso para incluir las nuevas funcionalidades.Desarrollar pruebas unitarias para el nuevo módulo de proveedores.Actualizar la documentación del sistema para incluir el nuevo módulo.	
Incluir en la OCI los criterios de revisión y auditoría	
Artefactos a entregar y defender el día del examen: <ol style="list-style-type: none">OCILínea de referencia detallada, resaltando los ICS a cambiar o agregarDiagramas de Casos de Uso actualizadosModelos de la base de datos actualizadosDiagrama de Clases actualizadosBase de datos modificadaPruebas Unitarias para el nuevo módulo.Nueva versión de la aplicación ejecutable.Documentación actualizada	

B. Estudio de viabilidad realizada por los desarrolladores sobre el reporte del estado de la configuración	
Desarrolladores que realizan el estudio: Geovanny Ochoa Vilcacundo Jordy Gualpa Steven Chica Valeska	Resumen del estudio de viabilidad: Resultado: A: Se aprueba totalmente B: Se aprueba parcialmente C: Se rechaza totalmente D: Otra _____
 Firma del solicitante (Elías Portilla Olvera)	 Geovanny Ochoa - ACC

Orden de cambio de Ingeniería (OCI)

OCI – Software de Facturación de línea de referencia	
Identificación: OCI_002_2024	Referencia de solicitud de cambio: 002_2024
Fecha de emisión de la OCI: 23/08/2024	Fecha de la solicitud de cambio: 22/08/2024
Proyecto: Línea de referencia – Sistema defacturación UTEQ v2.1. Version resultante: Sistema de facturación UTEQ v3.	Solicitando del cambio: Elías Portilla Olvera (rolde cliente)
Descripción de los cambios: <ul style="list-style-type: none"> • Crear un nuevo módulo de proveedores que permita la adición, edición, eliminación y visualización de proveedores. • Implementar una función de búsqueda y filtrado para localizar proveedores específicos. • Establecer la relación entre proveedores y productos en la base de datos. • Actualizar los modelos de la base de datos para incluir la entidad de proveedores. • Modificar el diagrama de clases para reflejar la nueva entidad de proveedores y sus relaciones • Actualizar los diagramas de casos de uso para incluir las nuevas funcionalidades de gestión de proveedores. • Desarrollar pruebas unitarias para el nuevo módulo de proveedores • Actualizar la documentación del sistema para incluir el nuevo módulo de proveedores. 	
Razón de cambio: El sistema de facturación actual no cuenta con un módulo de gestión de proveedores, lo cual genera las siguientes problemáticas: <ul style="list-style-type: none"> • Falta de un registro centralizado y organizado de los proveedores del negocio. • Dificultad para vincular productos con sus respectivos proveedores, lo que complica la gestión de inventario • Reducción de la eficiencia operativa al no tener centralizada la información de proveedores en el sistema de facturación. 	

<ul style="list-style-type: none"> Ineficiencia en el proceso de reabastecimiento de productos al no tener la información de proveedores integrada en el sistema.
<p>Impacto en el proyecto: El impacto que tendrán estos cambios en el software es significativo ya que se agregará una nueva funcionalidad principal al sistema. Los siguientes elementos se verán afectados:</p> <ul style="list-style-type: none"> Se creará una nueva entidad (Proveedores) en la base de datos, lo que requerirá modificaciones en la estructura actual. Se desarrollará una nueva interfaz de usuario para la gestión de proveedores, lo que implicará cambios en el diseño general de la aplicación. Se modificará la lógica de negocio para incluir las operaciones relacionadas con los proveedores y su vinculación con los productos. Se actualizarán los diagramas de casos de uso y de clases para reflejar las nuevas funcionalidades de gestión de proveedores. Se implementarán nuevas pruebas unitarias para asegurar el correcto funcionamiento del módulo de proveedores. Se actualizará la documentación del sistema para incluir información sobre el nuevo módulo de proveedores y sus funcionalidades.
<p>Restricciones:</p> <ol style="list-style-type: none"> En todos los formularios con DataGridView: <ol style="list-style-type: none"> Restricción: No cargar todos los datos por defecto en el DataGridView; cargar los datos solo cuando el usuario busque algún registro. En los formularios que listan Clientes, Productos, Proveedores, empleados y Facturas: <ol style="list-style-type: none"> Restricción: Mejorar la barra de búsqueda permitiendo al usuario seleccionar el campo por el cual buscar. La búsqueda debe activarse mediante un botón y no en base al evento de cambio de texto. En el proceso de selección de usuarios: <ol style="list-style-type: none"> Restricción: La selección de un usuario debe realizarse mediante el evento de doble clic, eliminando el botón para seleccionar. En el formulario de impresión de factura: <ol style="list-style-type: none"> Restricción: Permitir la impresión de factura una vez guardada en la base de datos. Agregar el botón de impresión en el mismo formulario.
<p>Criterio para revisar y auditar:</p> <ul style="list-style-type: none"> Revisar la nueva versión del software con los cambios solicitados. Verificar que la búsqueda de clientes, empleados, productos y proveedores funcione correctamente mediante los campos de texto. Verificar la funcionalidad de los módulos Cliente, Empleados, Productos y proveedores en base al registro, edición, visualización y eliminación. Asegurarse de que la búsqueda y filtrado de usuarios y proveedores funcionen correctamente. Comprobar el login y control de acceso con usuarios y contraseñas funcione correctamente. Evaluar la usabilidad del software, considerando la experiencia del usuario. Evaluar el código fuente de la nueva versión y la versión anterior Analizar los modelos de la nueva versión y la versión anterior
<p>Número de sprint que se tomarán: 2 sprint (cada sprint tiene una duración de una semana).</p>
<p>Responsables de la implementación:</p> <ul style="list-style-type: none"> Valeska Chica Jorge Gualpa Jordy Vilcacundo Geovanny Ochoa

Recursos necesarios: <ul style="list-style-type: none">• SQL Server Management Studio 2020.• SQL Server 2019 Express.• IDE – Visual Studio 2022 Community 17.10.3• PowerDesigner 16.7• Lenguaje de programación C#• Microsoft .NET Framework 4.7.2• 4 ordenadores con los softwares instalados.• Git y GitHub.	
Aprobado por: Geovanny Ochoa Gilces (ACC)	
Fecha Inicio de la Implementación: 24/08/2024	
Estatus: Aprobado	
Comentarios adicionales: ninguno.	
 Firma del solicitante (Elías Portilla Olvera)	 Firma del aprobador – ACC (Geovanny Ochoa)

Línea de referencia

Identificación del Proyecto:	Línea de Referencia – Sistema de Facturación v2.1.0
Referencia de solicitud de cambio:	SC_002_2024
Referencia de solicitud de OCI:	OCI_003_2024
Versiones del proyecto:	Versión 1 (13/06/2024) Versión 2 (15/07/2024) Versión 2.1 (24/07/2024) Versión 3 (24/08/2024)
Fecha a entregar última versión:	24/07/2024
Auditorías realizadas:	Auditoria a la versión 2 (09/28/2024)
Enlace a código fuente:	https://github.com/gochoag/SGFactuacion

ICS modificados:

- Diagramas de Casos de Uso v3
- Modelo Conceptual de la Base de Datos v3
- Modelo Lógico de la Base de Datos v3
- Modelo Físico de la Base de Datos v3
- Diagrama de Clases v3
- Procedimientos almacenados:
 - - Sp_Insert_Proveedor (Crear)
 - -Sp_Buscar_Proveedor (Crear)
 - -Sp_Delete_Proveedor (Crear)
 - -Sp_Update_Proveedor (Crear)
 - -Sp_Listado_Proveedor (Crear)
 - -Sp_Insert_Producto (Modificado)
 - -Sp_Buscar_Producto (Modificado)
 - -Sp_Listar_Producto (Modificado)
 - -Sp_Update_Producto (Modificado)
 - -Sp_Delete_Producto (Modificado)
- Base de datos física v3
- Código Fuente v3
- Documentación del proyecto v3

Modelado de la base de datos

1.1. Modelo conceptual

Modelo:	Modelo Conceptual
Autores:	Valeska Chica, Gualpa Steven, Geovanny Ochoa y Jordy Vilcacundo

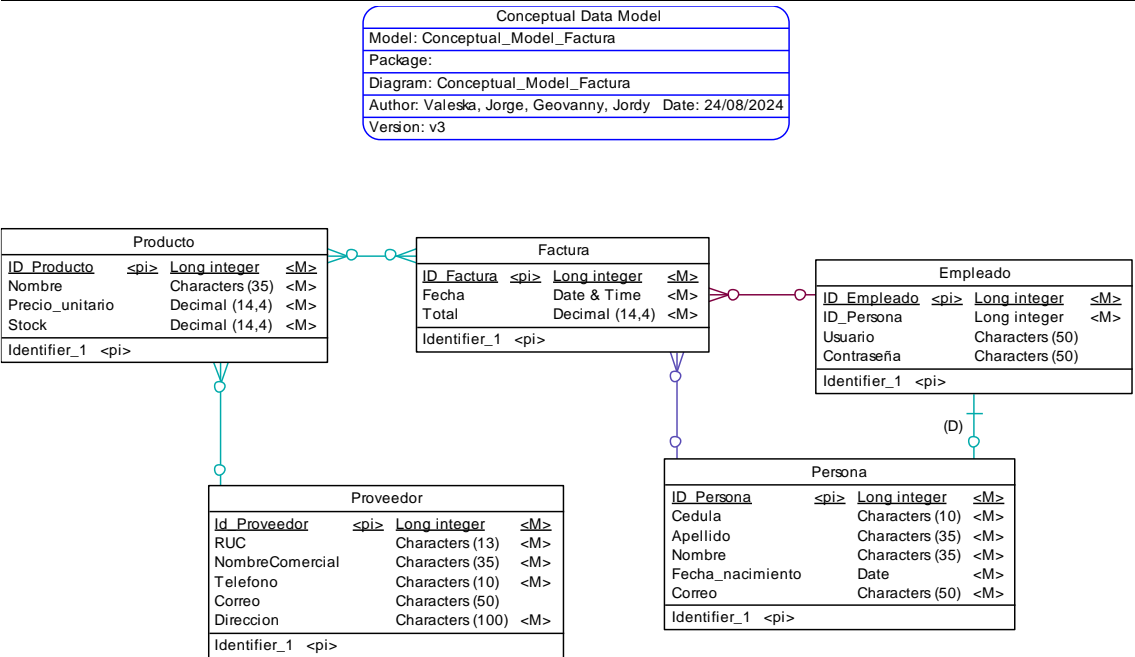


Imagen 2: Modelo Conceptual

1.2. Modelo Lógico

Modelo:	Modelo Lógico
---------	---------------

Autores:	Valeska Chica, Gualpa Steven, Geovanny Ochoa y Jordy Vilcacundo
-----------------	---

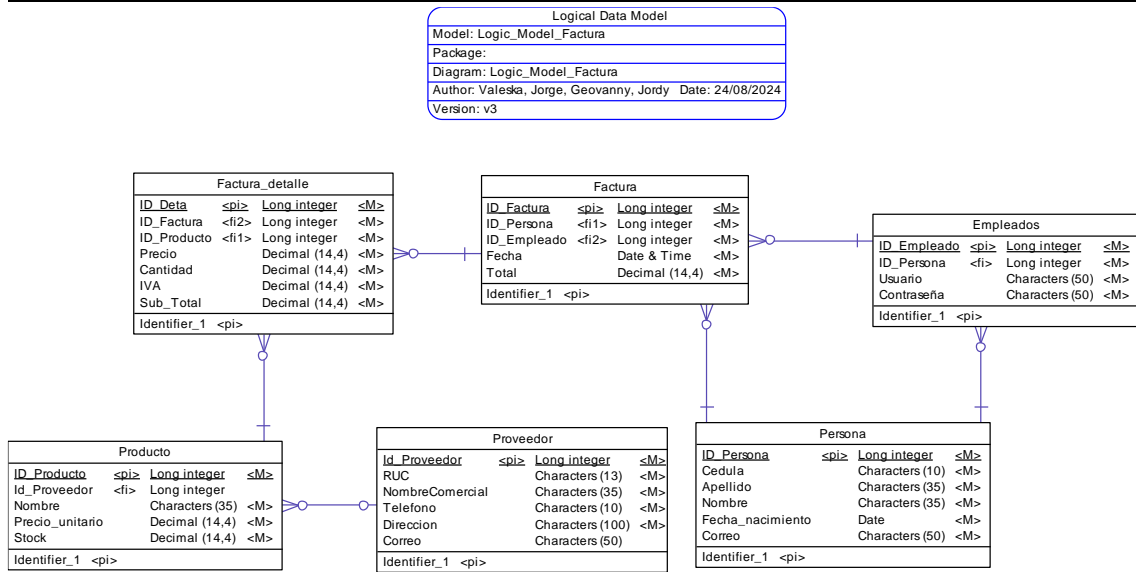


Imagen 3: Modelo Lógico

1.3. Modelo Físico

Modelo:	Modelo Físico
Autores:	Valeska Chica, Gualpa Steven, Geovanny Ochoa y Jordy Vilcacundo

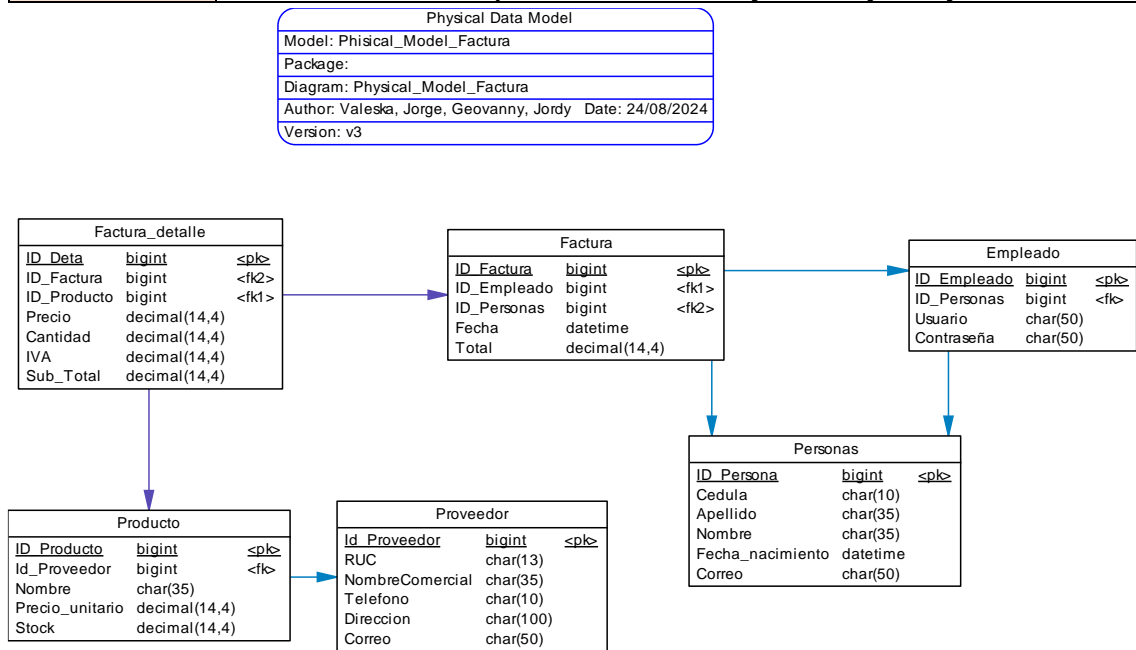


Imagen 4: Modelo Físico

Casos de uso e historias de usuarios

1.4. Diagrama de Caso de uso

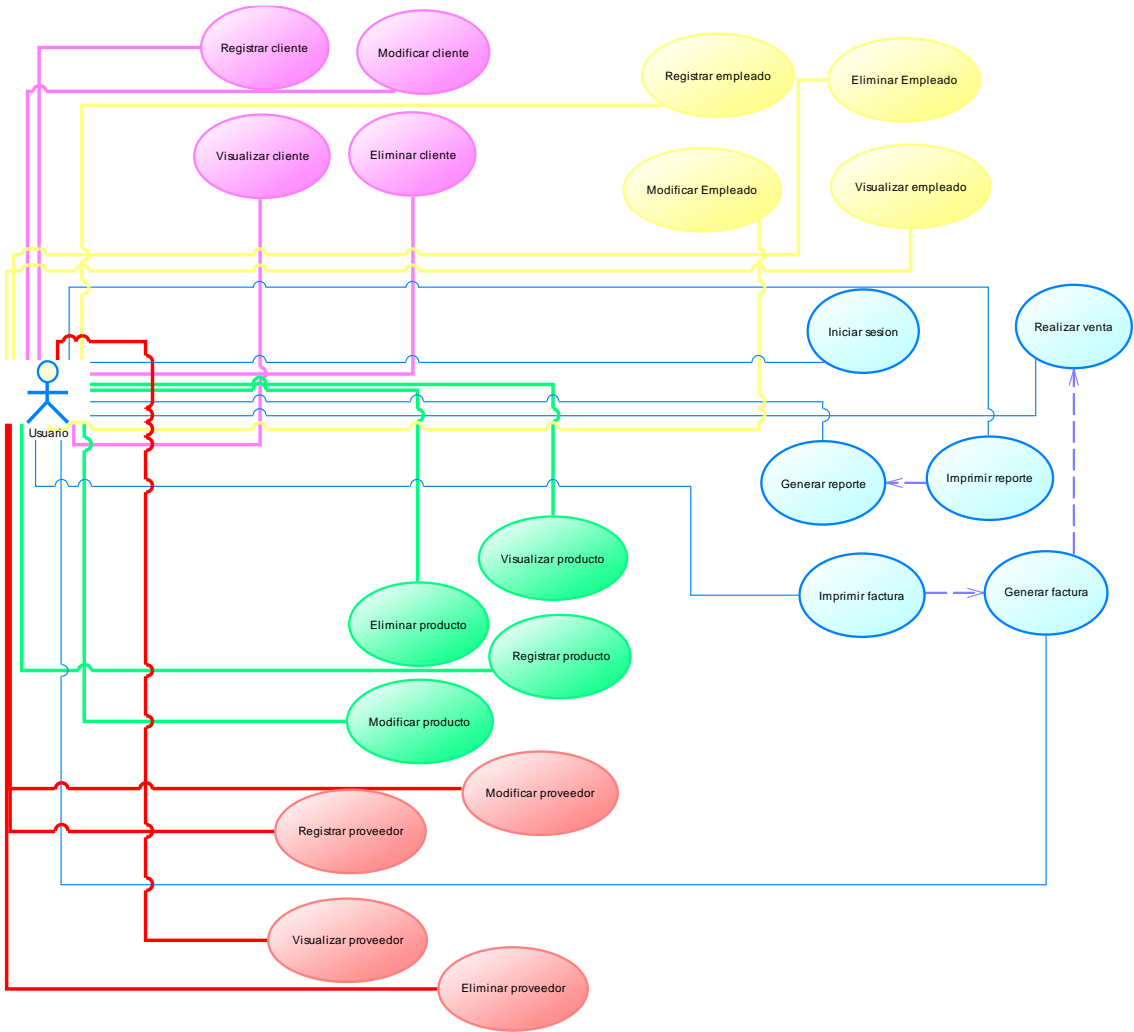


Imagen 5: Diagrama de caso de usos

1.5. Caso de uso #1.- Registrar cliente

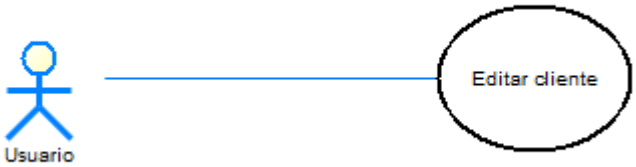
Identificador:	CU-01
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Permitir al usuario registrar nuevos clientes en el sistema.
Precondiciones:	<ul style="list-style-type: none">El sistema de gestión de clientes está operativo y disponible para ingreso de datos
Postcondiciones:	<ul style="list-style-type: none">Un nuevo cliente es registrado en la base de datos.
Disparador:	<ul style="list-style-type: none">El usuario selecciona la opción "Registrar Cliente" desde el menú principal.
Diagrama de caso de uso relacionado.	

Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de registro de clientes	2. El sistema presenta un formulario con los campos para registrar un cliente
3. El usuario ingresa los datos del cliente	
4. El usuario da clic en guardar cliente.	5. El sistema validara todos los campos del formulario.
	6. Guardara todos los datos del cliente en la base de datos.
Cursos alternos (Excepciones)	
5	Falta de información obligatoria (se muestra un mensaje de error y se solicita completar los campos necesarios). Volver al paso 3
5	Conflicto con un cliente existente (se muestra una advertencia y se pide verificar la identidad del nuevo cliente) Volver al paso 3
5	El usuario no ha ingresado todos los datos, el sistema emite un mensaje que indica los campos que faltan por llenar. Volver al paso 3

1.5.1. Historia de usuario #1

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Registrar nuevos clientes en el sistema	Poder gestionar y facturar a los clientes de manera eficiente
Usuario	Listar todos los clientes	Poder revisar si un usuario se encuentra registrado en la base de datos

1.6. Caso de uso #2.- Modificar cliente


Identificador:	CU-02
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Permitir al usuario modificar los datos de un cliente existente
Precondiciones:	<ul style="list-style-type: none"> El cliente debe estar registrado en la base de datos.
Postcondiciones:	<ul style="list-style-type: none"> Los datos del cliente son actualizados en la base de datos.
Disparador:	<ul style="list-style-type: none"> El usuario selecciona la opción "Editar Cliente" desde el menú principal.
Diagrama de caso de uso relacionado.	 <pre> graph LR Usuario((Usuario)) --- EditarCliente((Editar cliente)) </pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de clientes	2. Presenta un formulario con las opciones de gestión

3. El usuario escoge la opción de editar cliente	
4. El usuario selecciona el cliente que quiere modificar	5. El sistema retorna los datos del cliente
6. El usuario ingresa los nuevos datos del cliente	
7. El usuario da clic en guardar información.	8. Validara todos los campos del formulario.
	9. Guardara todos los datos del cliente en la base de datos.
Cursos alternos (Excepciones)	
8	Falta de información obligatoria (se muestra un mensaje de error y se solicita completar los campos necesarios). Volver al paso 6
8	Conflicto con datos únicos de un cliente existente (se muestra una advertencia y se pide verificar la información ingresada) Volver al paso 6
8	El usuario no ha ingresado todos los datos, el sistema emite un mensaje que indica los campos que faltan por llenar. Volver al paso 6

1.6.1. Historia de usuario #2

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Listar todos los clientes	Poder revisar si un usuario se encuentra registrado en la base de datos
Usuario	Modificar clientes registrados	Poder corregir información incorrecta o desactualizada de un cliente.

1.7. Caso de uso #3.- Eliminar cliente

Identificador:	CU-03
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Eliminar la información de un cliente existente en el sistema.
Precondiciones:	<ul style="list-style-type: none"> El cliente debe estar registrado en la base de datos.
Postcondiciones:	<ul style="list-style-type: none"> El cliente ya no podrá realizar compras
Disparador:	<ul style="list-style-type: none"> El usuario selecciona la opción "Eliminar Cliente" desde el menú principal.
Diagrama de caso de uso relacionado.	 <pre> graph LR Usuario[Usuario] --- Eliminar((Eliminar cliente)) </pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de clientes	2. Presenta un formulario con las opciones de gestión
3. El usuario escoge la opción de eliminar cliente	

4. El usuario ingresa los datos del cliente	5. El sistema retorna los datos del cliente
6. El usuario da clic en eliminar cliente.	7. El sistema muestra un mensaje en donde indica si está seguro de eliminar el cliente.
8. El usuario acepta eliminar el cliente	9. Eliminará todos los datos del cliente en la base de datos.

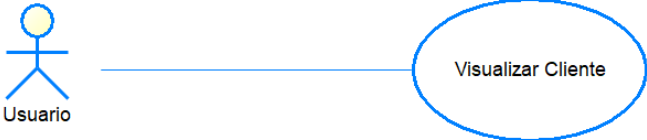
Cursos alternos (Excepciones)

No existen excepciones

1.7.1. Historia de usuario #3

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Eliminar Cliente	Poder eliminar los datos de un cliente que ya está registrado en el sistema.

1.8. Caso de uso #4.- Visualizar cliente

Identificador:	CU-04
iteración:	01, Última modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Visualizar la información de un cliente en el sistema
Precondiciones:	<ul style="list-style-type: none"> El cliente debe estar registrado en la base de datos.
Postcondiciones:	<ul style="list-style-type: none"> Lista de clientes registrados en la base de datos
Disparador:	<ul style="list-style-type: none"> El usuario decide consultar la información de un cliente
Diagrama de caso de uso relacionado.	 <pre> graph LR Usuario((Usuario)) --- VisualizarCliente((Visualizar Cliente)) </pre>

Flujo normal (Escenario)

Actor	Sistema
1. El usuario accede al módulo de clientes	2. Presenta un formulario con las opciones de gestión
3. El usuario escoge la opción de buscar cliente	
4. El usuario ingresa los datos del cliente	5. El sistema retorna los datos del cliente

Cursos alternos (Excepciones)

4	Si el cliente no existe, el sistema muestra un mensaje de error. Volver al paso 4.
---	--

1.8.1. Historia de usuario #4

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Eliminar Cliente	Poder eliminar los datos de un cliente que ya está registrado en el sistema.
Usuario	Listar todos los clientes	Poder revisar si un usuario se encuentra registrado en la base de datos
Usuario	Modificar cliente	Poder corregir información incorrecta o desactualizada de un cliente.

1.9. Caso de uso #5.- Registrar empleado

Identificador:	CU-05
iteración:	01, Última modificación: 14/07/2024
Actores:	Usuario
Objetivo en contexto:	Permitir al administrador registrar nuevos empleados en el sistema
Precondiciones:	<ul style="list-style-type: none"> El sistema de gestión de empleados está operativo y disponible para ingreso de datos. El administrador ha iniciado sesión en el sistema
Postcondiciones:	<ul style="list-style-type: none"> Un nuevo empleado es registrado en la base de datos. Se asigna un identificador único al nuevo empleado
Disparador:	<ul style="list-style-type: none"> El administrador selecciona la opción "Registrar Empleado" desde el menú de gestión de personal.
Diagrama de caso de uso relacionado.	<pre> graph LR Usuario[Usuario] --- RegistrarEmpleado((Registrar empleado)) </pre>

Flujo normal (Escenario)

Actor	Sistema
1. El administrador accede al módulo de registro de empleados	2. El sistema presenta un formulario con los campos para registrar un empleado
3. El administrador ingresa los datos del empleado (nombre, apellido, cédula, correo electrónico, fecha de nacimiento, usuario y contraseña)	
4. El administrador da clic en "Guardar Empleado"	5. El sistema valida todos los campos del formulario
	6. El sistema genera un identificador único para el empleado
	7. El sistema guarda todos los datos del empleado en la base de datos
	8. El sistema muestra un mensaje de confirmación del registro exitoso

Cursos alternos (Excepciones)

5	Falta de información obligatoria (se muestra un mensaje de error y se solicita completar los campos necesarios). Volver al paso 3
5	Conflicto con un empleado existente (se muestra una advertencia y se pide verificar la información del nuevo empleado). Volver al paso 3
5	Formato inválido en algún campo (por ejemplo, correo electrónico mal formado). Se muestra un mensaje de error indicando el campo específico. Volver al paso 3

1.9.1. Historia de usuario #5

Historias de usuarios		
Rol	Deseo	Objetivo
Administrador	Registrar nuevos empleados en el sistema	Poder gestionar eficientemente el personal de la empresa y asignar roles en el sistema

Administrador	Mantener un registro actualizado del personal	Facilitar la gestión de recursos humanos y el control de acceso al sistema
---------------	---	--

1.10. Caso de uso #6.- Modificar empleado

Identificador:	CU-06
iteración:	01, Última modificación: 14/07/2024
Actores:	Usuario
Objetivo en contexto:	Permitir al administrador modificar los datos de un empleado existente en el sistema.
Precondiciones:	<ul style="list-style-type: none">El empleado debe estar registrado en la base de datos.El administrador ha iniciado sesión en el sistema.
Postcondiciones:	<ul style="list-style-type: none">Los datos del empleado son actualizados en la base de datos
Disparador:	<ul style="list-style-type: none">El administrador selecciona la opción "Modificar Empleado" desde el menú de gestión de personal
Diagrama de caso de uso relacionado.	<pre>graph LR; Usuario[Usuario] --- ModificarEmpleado((Modificar Empleado));</pre>

Flujo normal (Escenario)	
Actor	Sistema
1. El administrador accede al módulo de empleados.	2. El sistema presenta un formulario con las opciones de gestión
3. El administrador escoge la opción de editar empleado.	
4. El administrador ingresa el identificador del empleado a modificar.	5. El sistema retorna los datos actuales del empleado
6. El administrador modifica los datos necesarios del empleado.	
7. El administrador da clic en “Guardar cambios”	8. El sistema valida todos los campos del formulario.
	9. El sistema actualiza los datos del empleado en la base de datos.
	10. El sistema muestra un mensaje de confirmación de la actualización exitosa


Cursos alternos (Excepciones)	
4	Si el empleado no existe, el mensaje muestra un mensaje de error. Volver al paso 4
8	Falta información obligatoria (muestra mensaje de error y se solicita completar los campos necesario). Volver al paso 6
8	Formato invalido en algún campo modificado. Se muestra un mensaje de error indicando el campo específico. Volver al paso 6

1.10.1. Historia de usuario #6

Historias de usuarios		
Rol	Deseo	Objetivo
Administrador	Modificar datos de empleados registrados	Poder mantener actualizada la información del personal en el sistema

Administrador	Corregir errores en los registros de empleados	Asegurar la precisión de la información del personal en la base de datos.
---------------	--	---

1.11. Caso de uso #7.- Eliminar empleado

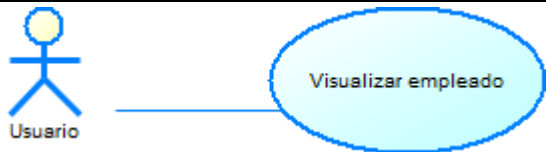
Identificador:	CU-07
iteración:	01, Ultima modificación: 14/07/2024
Actores:	Administrador
Objetivo en contexto:	Permitir al administrador eliminar un empleado del sistema
Precondiciones:	<ul style="list-style-type: none">El empleado debe estar registrado en la base de datosEl administrador ha iniciado sesión en el sistema
Postcondiciones:	<ul style="list-style-type: none">El empleado es marcado como inactivo en la base de datosSe registra la fecha de baja del empleado
Disparador:	<ul style="list-style-type: none">El administrador selecciona la opción “Eliminar empleado” desde el menú de gestión personal.
Diagrama de caso de uso relacionado.	 <pre>graph LR Usuario((Usuario)) --- EliminarEmpleado([Eliminar Empleado])</pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El administrador accede al módulo de empleados	2. El sistema presenta un formulario con las opciones de gestión
3. El administrador escoge la opción de eliminar empleado	
4. El administrador ingresa el identificador del empleado a eliminar	5. El sistema retorna los datos del empleado
6. El administrador confirma la eliminación del empleado	7. El sistema muestra un mensaje de advertencia solicitando confirmación
8. El administrador confirma la acción	9. El sistema marca al empleado como inactivo en la base de datos
	10. El sistema muestra un mensaje de confirmación de la eliminación exitosa
Cursos alternos (Excepciones)	
4	Si el empleado no existe, el sistema muestra un mensaje de error. Volver al paso 4
8	Si el administrador cancela la acción, el sistema vuelve al menú principal de gestión de empleados.

1.11.1. Historia de usuario #7

Historias de usuarios		
Rol	Deseo	Objetivo
Administrador	Eliminar empleados del sistema	Poder mantener actualizada la lista de personal activo en la empresa

1.12. Caso de uso #8.- Visualizar empleado

Identificador:	CU-08
----------------	-------


iteración:	01, Última modificación: 14/07/2024
Actores:	Usuario
Objetivo en contexto:	Permitir a los usuarios autorizados visualizar la información detallada de un empleado en el sistema
Precondiciones:	<ul style="list-style-type: none"> El empleado debe estar registrado en la base de datos El usuario ha iniciado sesión en el sistema y tiene los permisos necesarios
Postcondiciones:	<ul style="list-style-type: none"> Se muestra la información detallada del empleado
Disparador:	<ul style="list-style-type: none"> El usuario selecciona la opción “visualizar empleado” desde el menú de gestión personal
Diagrama de caso de uso relacionado.	
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de empleados	2. El sistema presenta un formulario con las opciones de gestión
3. El usuario escoge la opción de visualizar empleado	
4. El usuario ingresa el identificador o nombre del empleado a visualizar	5. El sistema busca al empleado en la base de datos
	6. El sistema muestra la información detallada del empleado.
Cursos alternos (Excepciones)	
4	Si el empleado no existe se muestra un mensaje de error. Vuelve al paso 4

1.12.1. Historia de usuario #8

Historias de usuarios		
Rol	Deseo	Objetivo
Administrador	Visualizar información detallada de los empleados	Poder acceder rápidamente a los datos del personal necesario

1.13. Caso de uso #9.- Iniciar sesión


Identificador:	CU-09
iteración:	01, Última modificación: 14/07/2024
Actores:	Usuario
Objetivo en contexto:	Permitir a los usuarios autenticarse en el sistema para acceder a sus funcionalidades
Precondiciones:	<ul style="list-style-type: none"> El usuario debe tener una cuenta registrada en el sistema El sistema de autenticación está operativo
Postcondiciones:	<ul style="list-style-type: none"> El usuario inicia sesión y accede a la interfaz principal
Disparador:	<ul style="list-style-type: none"> El usuario accede a la página de inicio de sesión del sistema

Diagrama de caso de uso relacionado.	
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede a la página de inicio de sesión	2. El sistema muestra el formulario de inicio de sesión
3. El usuario ingresa su usuario y contraseña	
4. El usuario da clic en “iniciar sesión”	5. El sistema autentica al usuario
	6. El sistema redirige al usuario a la interfaz principal
Cursos alternos (Excepciones)	
5	Si las credenciales son incorrectas, el sistema muestra un mensaje de error. Volver al paso 3

1.13.1. Historia de usuario #9

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Iniciar sesión de forma segura al sistema	Acceder a las funcionalidades correspondientes a su rol

1.14. Caso de uso #10.- Registrar producto

Identificador:	CU-10
iteración:	01, Última modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Agregar un nuevo producto al sistema para su posterior gestión y seguimiento
Precondiciones:	<ul style="list-style-type: none"> El sistema de gestión de producto está operativo y disponible para ingreso de datos
Postcondiciones:	<ul style="list-style-type: none"> Un nuevo producto es registrado en la base de datos.
Disparador:	<ul style="list-style-type: none"> Un nuevo producto es registrado en la base de datos.
Diagrama de caso de uso relacionado.	
Flujo normal (Escenario)	
Actor	Sistema
9. El usuario accede al módulo de registro de Productos	10. Presenta un formulario con los campos para registrar un producto
11. El usuario ingresa los datos del producto	
12. El usuario da clic en guardar producto.	13. Validara todos los campos del formulario.
	14. Guardara todos los datos del producto en la base de datos.
Cursos alternos (Excepciones)	

5	Falta de información obligatoria (se muestra un mensaje de error y se solicita completar los campos necesarios). Volver al paso 3
5	Conflicto con un producto existente. Volver al paso 3
5	El usuario no ha ingresado todos los datos, el sistema emite un mensaje que indica los campos que faltan por llenar. Volver al paso 3

1.14.1. Historia de usuario #10

Historias de usuarios		
Rol	Deseo	Objetivo
Administrador	Listar todos los productos	Poder revisar si un producto ya está registrado en el sistema.
Administrador	Registrar nuevos productos	Poder gestionar y facturar los productos vendidos de manera eficiente

1.15. Caso de uso #11.- Modificar producto

Identificador:	CU-11
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Actualizar la información de un producto existente en el sistema.
Precondiciones:	<ul style="list-style-type: none"> El producto debe estar registrado en la base de datos.
Postcondiciones:	<ul style="list-style-type: none"> Los datos del producto son actualizados en la base de datos.
Disparador:	<ul style="list-style-type: none"> El usuario selecciona la opción "Modificar Producto" desde el menú principal.
Diagrama de caso de uso relacionado.	<pre> graph LR Usuario[Usuario] --- ModificarProducto((Modificar producto)) </pre>

Flujo normal (Escenario)

Actor	Sistema
1. El usuario accede al módulo de productos	2. Presenta un formulario con las opciones de gestión
3. El usuario escoge la opción de editar producto	
4. El usuario ingresa los datos del producto	5. El sistema retorna los datos del producto
6. El usuario ingresa los nuevos datos del producto	
7. El usuario da clic en guardar producto.	8. Validara todos los campos del formulario.
	9. Guardara todos los datos del producto en la base de datos.

Cursos alternos (Excepciones)


5	El producto ingresado no se encuentra registrado (muestra mensaje de error). Volver al paso 4
---	---

8	Falta de información obligatoria (se muestra un mensaje de error y se solicita completar los campos necesarios). Volver al paso 6
8	Conflicto con datos únicos de un producto existente. Volver al paso 6
8	El usuario no ha ingresado todos los datos, el sistema emite un mensaje que indica los campos que faltan por llenar. Volver al paso 6

1.15.1. Historia de usuario #11

Historias de usuarios		
Rol	Deseo	Objetivo
Administrador	Listar todos los productos	Poder revisar si un producto ya está registrado en el sistema.
Administrador	Modificar productos registrados	Poder corregir información o actualizar información del producto.

1.16. Caso de uso #12.- Eliminar producto

Identificador:	CU-12
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Eliminar la información de un producto existente en el sistema.
Precondiciones:	<ul style="list-style-type: none"> El producto debe estar registrado en la base de datos.
Postcondiciones:	<ul style="list-style-type: none"> El producto es eliminado de la base de datos.
Disparador:	<ul style="list-style-type: none"> El usuario selecciona la opción "Eliminar Producto" desde el menú principal.
Diagrama de caso de uso relacionado.	 <pre> graph LR Usuario((Usuario)) --- EliminarProducto((Eliminar producto)) </pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de producto	2. Presenta un formulario con las opciones de gestión
3. El usuario escoge la opción de eliminar producto	
4. El usuario ingresa los datos del producto	5. El sistema retorna los datos del producto
6. El usuario da clic en eliminar producto.	7. El sistema muestra un mensaje en donde indica si está seguro de eliminar el producto.
8. El usuario acepta eliminar el producto	9. Eliminará todos los datos del producto en la base de datos.
Cursos alternos (Excepciones)	
No existen excepciones	

1.16.1. Historia de usuario #12

Historias de usuarios		
Rol	Deseo	Objetivo

Administrador	Eliminar Producto	Poder eliminar los datos de un producto que ya está registrado en el sistema.
---------------	-------------------	---

1.17. Caso de uso #13.- Visualizar producto

Identificador:	CU-13
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Visualizar la información de un producto en el sistema
Precondiciones:	<ul style="list-style-type: none">El producto debe estar registrado en la base de datos.
Postcondiciones:	<ul style="list-style-type: none">El producto estará disponible para su venta.
Disparador:	<ul style="list-style-type: none">El usuario decide consultar la información de un producto
Diagrama de caso de uso relacionado.	<pre>graph LR Usuario[Usuario] --- Visualizar((Visualizar producto))</pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de producto	2. Presenta un formulario con las opciones de gestión
3. El usuario escoge la opción de buscar producto	
4. El usuario ingresa los datos del producto	5. El sistema retorna los datos del producto
Cursos alternos (Excepciones)	
4	Si el producto no existe, el sistema muestra un mensaje de error. Volver al paso 4.

1.17.1. Historia de usuario #14

Historias de usuarios		
Rol	Deseo	Objetivo
Administrador	Eliminar Producto	Poder eliminar los datos de un producto que ya está registrado en el sistema.
Administrador	Listar todos los productos	Poder revisar si un producto ya está registrado en el sistema.
Administrador	Modificar productos registrados	Poder corregir información o actualizar información del producto.

1.18. Caso de uso #14.- Generar Factura

Identificador:	CU-14
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Permitir al usuario generar nuevas facturas en el sistema
Precondiciones:	<ul style="list-style-type: none">El sistema de facturación deberá estar operativo y disponible

Postcondiciones:	<ul style="list-style-type: none">Una nueva factura es registrada en la base de datos.
Disparador:	<ul style="list-style-type: none">El usuario selecciona la opción "Generar Factura".
Diagrama de caso de uso relacionado.	<pre>graph LR Usuario((Usuario)) --- GenerarFactura((Generar Factura))</pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario selecciona la opción "Generar Factura"	2. El sistema muestra una visualización de la factura generada
3. El usuario verifica que todo sea correcto	
4. El usuario presiona "Confirmar factura"	5. El sistema genera la factura
	6. El sistema guarda la factura en la base de datos
Cursos alternos (Excepciones)	
3	Los datos de la factura no son correctos (Cancela el proceso)

1.18.1. Historia de usuario #14

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Desea generar una factura para registrar una venta realizada a un cliente.	Poder documentar y registrar las transacciones de venta realizadas en el sistema

1.19. Caso de uso #15.- Imprimir Factura

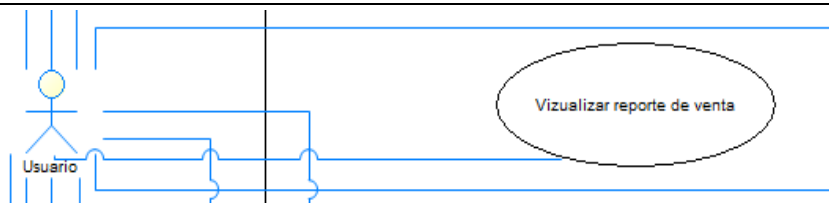
Identificador:	CU-15
iteración:	01, Última modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Permitir al usuario imprimir una factura generada en el sistema
Precondiciones:	<ul style="list-style-type: none">El usuario debe haber generado una factura en el sistema.La factura debe estar disponible para impresión.
Postcondiciones:	<ul style="list-style-type: none">Se imprime la factura seleccionada en el formato deseado.
Disparador:	<ul style="list-style-type: none">El usuario selecciona la opción "Imprimir Factura" desde el menú de opciones de la factura.
Diagrama de caso de uso relacionado.	<pre>graph LR Usuario((Usuario)) --- ImprimirFactura((Imprimir Factura))</pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario selecciona la opción "Imprimir Factura" desde el menú de opciones de la factura.	2. El sistema muestra la factura en el formato deseado para impresión.

3. El usuario envía la factura a la impresora seleccionada.	
Cursos alternos (Excepciones)	
1	Si no hay factura seleccionada para imprimir, el sistema muestra un mensaje de error indicando que no hay factura disponible.

1.19.1. Historia de usuario #15

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Desea poder imprimir una factura generada en el sistema.	Poder obtener una copia física de la factura para propósitos de archivo o entrega al cliente.

1.20. Caso de uso #16.- Visualizar Reporte

Identificador:	CU-16
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Consultar y visualizar informes o reportes generados por el sistema.
Precondiciones:	<ul style="list-style-type: none">El producto debe estar registrado en la base de datos.El cliente debe estar registrado en la base de datos.Debe existir ventas registradas en la Base de Datos
Postcondiciones:	<ul style="list-style-type: none">La Factura estará disponible para su visualización en la Base de datos.
Disparador:	<ul style="list-style-type: none">El usuario desea visualizar un reporte.
Diagrama de caso de uso relacionado.	
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de reportes	2. Presenta un formulario con las opciones de reportes
3. El usuario escoge la opción de reporte deseado.	4. El sistema retorna los datos del cliente
5. El usuario escoge la opción de buscar producto, cliente o venta	
6. El usuario ingresa los datos del producto, cliente o venta	7. El sistema retorna los datos solicitados
8. El usuario ingresa escoge la opción de visualizar reporte.	9. El sistema muestra el informe generado
Cursos alternos (Excepciones)	
7	Si el cliente no existe, el sistema muestra un mensaje de error. Volver al paso 4.
7	Si el producto no existe, el sistema muestra un mensaje de error. Volver al paso 4.
7	Si la venta no existe, el sistema muestra un mensaje de error. Volver al paso 4.

1.20.1. Historia de usuario #16

Historias de usuarios		
Rol	Deseo	Objetivo
Usuario	Listar todas las ventas.	Poder revisar las ventas registradas en la base de datos.
Usuario	Listar todos los productos.	Poder revisar si un producto ya está registrado en el sistema.
Usuario	Listar todos los clientes.	Poder revisar si un usuario se encuentra registrado en la base de datos

1.21. Caso de uso #17.- Imprimir Reporte

Identificador:	CU-17
iteración:	01, Ultima modificación: 26/05/2024
Actores:	Usuario
Objetivo en contexto:	Imprimir informes o reportes generados por el sistema.
Precondiciones:	<ul style="list-style-type: none"> El usuario debe haber generado un reporte en el sistema. El reporte debe estar disponible para impresión.
Postcondiciones:	<ul style="list-style-type: none"> Se imprime el reporte seleccionado en el formato deseado.
Disparador:	<ul style="list-style-type: none"> El usuario selecciona la opción "Imprimir Reporte" desde el menú de opciones del reporte.
Diagrama de caso de uso relacionado.	<pre> graph LR Usuario((Usuario)) --- ImprimirReporte((Imprimir reporte)) </pre>

Flujo normal (Escenario)

Actor	Sistema
1. El usuario accede al módulo de reportes	2. Presenta un formulario con las opciones de reportes
3. El usuario escoge la opción de reporte deseado.	4. El sistema retorna los datos del cliente
5. El usuario escoge la opción de buscar producto, cliente o venta	
6. El usuario ingresa los datos del producto, cliente o venta	7. El sistema retorna los datos solicitados
8. El usuario ingresa escoge la opción de visualizar reporte.	9. El sistema muestra el informe generado
10. El usuario escoge la opción de imprimir reporte	11. El sistema imprime el reporte generado.

Cursos alternos (Excepciones)

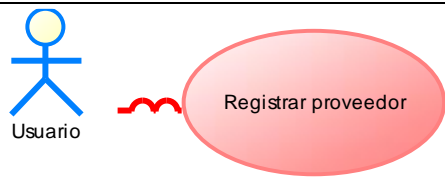
7	Si el cliente no existe, el sistema muestra un mensaje de error. Volver al paso 4.
7	Si el producto no existe, el sistema muestra un mensaje de error. Volver al paso 4.
7	Si la venta no existe, el sistema muestra un mensaje de error. Volver al paso 4.

1.21.1. Historia de usuario #17

Historias de usuarios

Rol	Deseo	Objetivo
Usuario	Listar todas las ventas.	Poder revisar las ventas registradas en la base de datos.
Usuario	Listar todos los productos.	Poder revisar si un producto ya está registrado en el sistema.
Usuario	Listar todos los clientes.	Poder revisar si un usuario se encuentra registrado en la base de datos

1.22. Caso de uso #18.- Registrar proveedor

Identificador:	CU-18
iteración:	01, Ultima modificación: 24/08/2024
Actores:	Usuario
Objetivo en contexto:	Registrar un nuevo proveedor en el sistema.
Precondiciones:	<ul style="list-style-type: none">El usuario debe haber iniciado sesión con permisos de administrador.El proveedor por registrar no debe existir en el sistema
Postcondiciones:	<ul style="list-style-type: none">Se crea un nuevo registro de proveedor en el sistema.Se asigna un identificador único al nuevo proveedor.
Disparador:	<ul style="list-style-type: none">El usuario selecciona la opción "Registrar Nuevo Proveedor" en el módulo de gestión de proveedores
Diagrama de caso de uso relacionado.	

Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de gestión de proveedores	2. Presenta un formulario para registrar un nuevo proveedor.
3. El usuario completa los campos requeridos (nombre, RUC, dirección, teléfono, correo electrónico, etc.).	
4. El usuario selecciona la opción "Guardar"	5. Valida los datos ingresados.
	6. Verifica que el proveedor no exista en el sistema
	7. Crea un nuevo registro de proveedor
	8. Asigna un identificador único al proveedor.
	9. Muestra un mensaje de confirmación del registro exitoso.

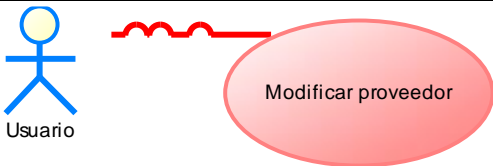
Cursos alternos (Excepciones)	
5	Si los datos ingresados no son válidos el sistema muestra mensajes de error específicos para cada campo inválido. (Volver paso 3)
6	Si el proveedor ya existe en el sistema el sistema muestra un mensaje indicando que el proveedor ya está registrado. (Volver al paso 2)

1.22.1. Historia de usuario #18

Historias de usuarios		
Rol	Deseo	Objetivo

Como administrador del sistema	Poder registrar nuevos proveedores en el sistema	Para mantener una base de datos actualizada de nuestros proveedores y facilitar la gestión de compras e inventario
--------------------------------	--	--

1.23. Caso de uso #19.- Modificar proveedor

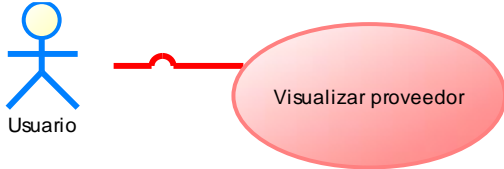
Identificador:	CU-19		
iteración:	01, Ultima modificación: 24/08/2024		
Actores:	Usuario		
Objetivo en contexto:	Modificar la información de un proveedor existente en el sistema.		
Precondiciones:	<ul style="list-style-type: none">El usuario debe haber iniciado sesión con permisos de administrador.El proveedor por editar debe existir en el sistema.		
Postcondiciones:	<ul style="list-style-type: none">La información del proveedor se actualiza en el sistema.		
Disparador:	<ul style="list-style-type: none">El usuario selecciona la opción "Editar Proveedor" en la ficha del proveedor.		
Diagrama de caso de uso relacionado.			
Flujo normal (Escenario)			
Actor		Sistema	
1. El usuario busca y selecciona el proveedor a editar.		2. Muestra la información actual del proveedor en un formulario editable.	
3. El usuario modifica los campos necesarios.			
4. El usuario selecciona la opción "Guardar cambios".		5. Valida los datos modificados	
		6. Actualiza la información del proveedor en el sistema.	
		7. Muestra un mensaje de confirmación de la actualización exitosa.	
Cursos alternos (Excepciones)			
5	Si los datos modificados no son válidos el sistema muestra mensajes de error específicos para cada campo inválido (vuelve al paso 3)		

1.23.1. Historia de usuario #19

Historias de usuarios		
Rol	Deseo	Objetivo
Como administrador del sistema	poder editar la información de los proveedores existentes	Para mantener actualizados los datos de nuestros proveedores y asegurar la precisión de la información en el sistema

1.24. Caso de uso #20.- visualizar proveedor

Identificador:	CU-20
iteración:	01, Última modificación: 24/08/2024

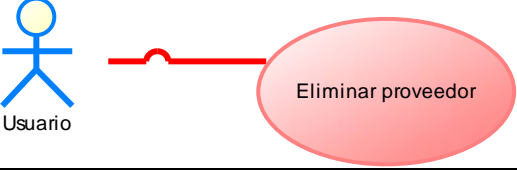
Actores:	Usuario
Objetivo en contexto:	Visualizar la información detallada de un proveedor específico.
Precondiciones:	<ul style="list-style-type: none"> El usuario debe haber iniciado sesión en el sistema. El proveedor a visualizar debe existir en el sistema
Postcondiciones:	<ul style="list-style-type: none"> Se muestra la información completa del proveedor seleccionado.
Disparador:	<ul style="list-style-type: none"> El usuario selecciona un proveedor de la lista o busca un proveedor específico.
Diagrama de caso de uso relacionado.	 <pre> graph LR Usuario((Usuario)) --- VisualizarProveedor(Visualizar proveedor) </pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario accede al módulo de gestión de proveedores	2. Muestra una lista de proveedores o un campo de búsqueda.
3. El usuario selecciona un proveedor o ingresa criterios de búsqueda.	4. Busca y recupera la información del proveedor
	5. Muestra la información detallada del proveedor.
Cursos alternos (Excepciones)	
3	Si el proveedor no se encuentra el sistema muestra un mensaje indicando que el proveedor no existe (volver al paso 2)

1.24.1. Historia de usuario #20

Historias de usuarios		
Rol	Deseo	Objetivo
Como usuario del sistema	poder ver la información detallada de un proveedor	Para consultar sus datos y tomar decisiones informadas en relación a compras e inventario

1.25. Caso de uso #21.- Eliminar proveedor

Identificador:	CU-21
iteración:	01, Última modificación: 24/08/2024
Actores:	Usuario
Objetivo en contexto:	Eliminar un proveedor del sistema.
Precondiciones:	<ul style="list-style-type: none"> El usuario debe haber iniciado sesión con permisos de administrador. El proveedor a eliminar debe existir en el sistema. El proveedor no debe tener transacciones o relaciones activas en el sistema.
Postcondiciones:	<ul style="list-style-type: none"> El proveedor se elimina del sistema.
Disparador:	<ul style="list-style-type: none"> El usuario selecciona la opción "Eliminar Proveedor" en la ficha del proveedor.

Diagrama de caso de uso relacionado.	 <pre> graph LR Usuario[Usuario] --- EliminarProveedor(Eliminar proveedor) </pre>
Flujo normal (Escenario)	
Actor	Sistema
1. El usuario busca y selecciona el proveedor a eliminar.	2. Muestra la información del proveedor y solicita confirmación para eliminar.
3. El usuario confirma la eliminación.	4. Verifica que no existan transacciones o relaciones activas con el proveedor
	5. Elimina el registro del proveedor del sistema.
	6. Muestra un mensaje de confirmación de la eliminación exitosa
Cursos alternos (Excepciones)	
4	Si existen transacciones o relaciones activas con el proveedor el sistema muestra un mensaje indicando que no se puede eliminar el proveedor debido a relaciones existentes. (volver al paso 1)

1.25.1. Historia de usuario #21

Historias de usuarios		
Rol	Deseo	Objetivo
Como administrador del sistema	poder eliminar proveedores del sistema	Para mantener la base de datos actualizada y eliminar registros de proveedores con los que ya no se trabaja

Diagramas

1.26. Diagrama de Clases

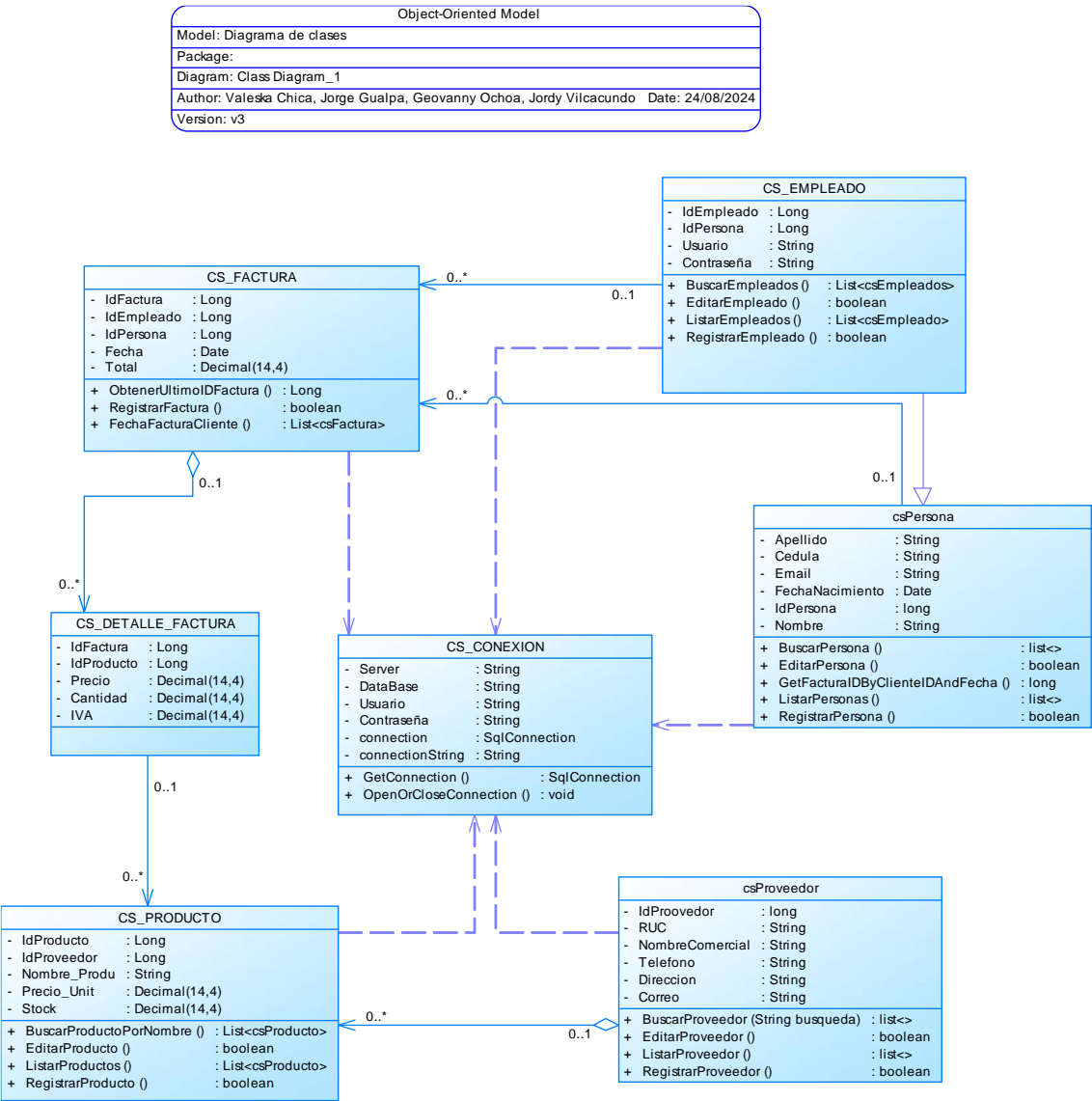


Imagen 6: Diagrama de clases

Diagrama de la base de datos

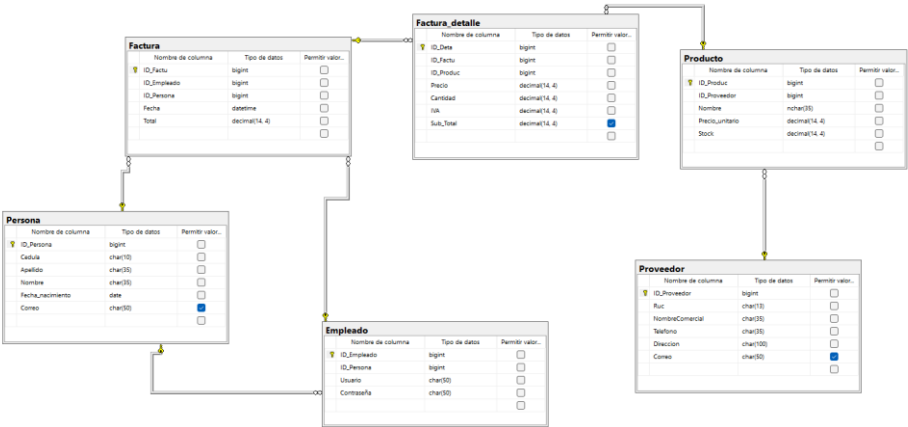


Imagen 7: Diagrama de la base de datos

Interfaces

1. Registro de clientes

The screenshot shows the 'Registrar clientes' (Register clients) form within the SGFacturacion application. On the left is a green sidebar with a menu containing icons and labels for 'Cliente', 'Producto', 'Facturar', 'Reportes', and 'Empleado'. The main area has a green header with the title 'Registrar clientes' and a close button. The form fields include: 'Cedula:' (text input), 'Nombres:' (text input), 'Apellidos:' (text input), 'Email:' (text input), and 'Fecha de nacimiento:' (date picker set to 'sábado, 1 de enero de 2000'). A green 'Registrar' button is at the bottom right.

Imagen 8: Interfaz de registro de cliente

2. Registro de productos

The screenshot shows the 'Registrar productos' (Register products) form. The layout is similar to the client registration form, with a green sidebar menu and a main area with a green header titled 'Registrar productos'. The form fields are: 'Nombre:' (text input), 'Precio unitario:' (text input), and 'Stock:' (text input). A green 'Registrar' button is located at the bottom right.

Imagen 9: Interfaz de registro de producto

3. Registro de empleado

The screenshot shows the 'Registrar empleados' (Register employees) form. The sidebar menu on the left includes an additional option, 'Empleado', which is currently selected. The main area has a green header titled 'Registrar empleados'. The form fields include: 'Cedula:' and 'Email:' (text inputs), 'Nombres:' and 'Apellidos:' (text inputs), 'Usuario:' and 'Contraseña:' (text inputs), and 'Fecha de nacimiento:' (date picker set to 'sábado, 1 de enero de 2000'). A green 'Registrar' button is at the bottom right.

Imagen 10: Interfaz de registro de empleado

4. Registro de Proveedor

Registrar Proveedor

RUC: Telefono:

Correo: Direccion:

Razon Comercial:

Registrar

Imagen 11: Interfaz de registro de Proveedor

5. Registro de factura

Registrar Factura

Buscar cliente:

Cliente: Pedro José García Torres

Buscar producto: Cantidad:

Producto: Precio: Stock:

Lista de productos

ID_Producto	Nombre	Precio	Cantidad
6	Helado de Salce...	2,5000	15

Realizar factura

Imagen 12: Interfaz de registro de factura

6. Generación de factura

SGFacturación

Tipo de reporte: Nº Factura:

10/03/2022 0:00:00

Factura Nº: 5

Cliente: Espinoza Rodríguez Carlos Alberto

ID Produc	Producto	Precio	Cantidad	Sub Total
21	Fruta deshidratada	3,7500	10	43,1250
22	Quinoa	2,5000	20	57,5000
23	Arroz de Daule	1,0000	15	17,2500
24	Camarones enlatados	8,0000	7	64,4000
25	Moras de los Andes	4,0000	12	55,2000

TOTAL: 237,4750

Imagen 13: Interfaz de generación de factura:

Revisión Técnica

Para el desarrollo de la revisión técnica se procederá a realizar una tarea en cada uno de los módulos que contiene el software, para así poder dar las observaciones en caso de que exista alguna inconsistencia:

- **Módulo de clientes**

○ Registrar Cliente

En el módulo de clientes, la función de registrar cliente se detalla cada campo que se debe ingresar, de tal forma que se encuentra validado para permitir ingresar el tipo de dato que se puede registrar, de la misma manera que no permita enviar campos vacíos, cada aspecto se tiene validado de tal forma que no se encontraron fallos, y su funcionalidad es correcta y completa.

Imagen 14: Registrar Cliente

○ Editar Cliente

En el módulo de clientes, la función de modificar cliente permite buscar y seleccionar el cliente que se desea actualizar. Una vez seleccionado, los datos del cliente se muestran automáticamente en los campos correspondientes, listos para ser editados. Cada campo está debidamente validado para asegurar que solo se ingresen los tipos de datos correctos y que no se dejen espacios vacíos. El sistema verifica minuciosamente cada aspecto de la información ingresada, garantizando que no haya errores en el proceso de modificación. Se ha comprobado que la funcionalidad de esta característica es completa y opera correctamente, cumpliendo con todos los requisitos establecidos.

IdPersona	Cedula	Nombre	Apellido	FechaNacimiento	Email
1	0506078901	Carlos Alberto	Espinoza Rodr...	05/05/1985	carlos.espinoz...
2	0607089012	Lucia Andrea	Flores Castillo	10/06/1990	lucia.flores@g...
3	0708090123	Pedro Jos	Garcia Torres	15/07/1987	pedro.garcia...
4	0809101234	Sofia Elena	Hernandez Riv...	20/08/1992	sofia.hernand...
5	0910112345	Jos Miguel	Jimenez Moreno	25/09/1985	josejimenez@...
6	1011123456	Elena Patricia	Lopez Rojas	30/10/1990	elena.lopez@...
7	1112134567	Luis Enrique	Martinez Salazar	15/11/1987	luis.martinez...
8	1213145678	Carmen Teresa	Navarrete Ro...	20/12/1992	carmen.navarr...

Imagen 15: Editar Clientes

○ Buscar y mostrar cliente

En el módulo de clientes, la función de lista de clientes ofrece una interfaz eficiente para localizar y visualizar la información de los clientes registrados. El sistema permite realizar búsquedas utilizando diversos criterios, como

nombre, número de identificación o cualquier otro dato relevante. Una vez efectuada la búsqueda, los resultados se presentan de manera clara y organizada.

Cada campo de búsqueda está validado para aceptar solo los tipos de datos pertinentes, evitando errores de entrada. El sistema no permite realizar búsquedas con campos vacíos, asegurando que siempre se proporcione al menos un criterio de búsqueda.



IdPersona	Cedula	Nombre	Apellido	FechaNacimiento	Email
1	0506078901	Carlos Alberto	Espinoza Rodr...	05/05/1985	carlos.espinoz...
2	0607089012	Luc a Andrea	Flores Castillo	10/06/1990	lucia.flores@g...
3	0708090123	Pedro Jos	Garc a Torres	15/07/1987	pedro.garcia@g...
4	0809101234	Sof a Elena	Hern ndez Riv...	20/08/1992	sofia.hernand...
5	0910112345	Jos Miguel	Jim nez Moreno	25/09/1985	jose.jimenez@g...
6	1011123456	Elena Patricia	L pez Rojas	30/10/1990	elena.lopez@g...
7	1112134567	Luis Enrique	Mart nez Salazar	15/11/1987	luis.martinez...
8	1213145678	Carmen Teresa	Navarrete Ro...	20/12/1992	carmen.navarr...
9	1314156789	Jorge Arturo	Ortiz S nchez	05/01/1985	jorge.ortiz@g...
10	1415167890	Andrea Gabriela	P rez Su rez	15/02/1990	andrea.perez...
11	1516178901	Diego Alejandro	Quintero Vargas	20/03/1987	diego.quinter...
12	1617189012	Fernanda Luc a	Ram rez Vera	25/04/1992	fernanda.rami...
13	1718190123	Mario Esteban	S nchez Aguirre	30/05/1985	mario.sanchez...
14	1819201234	Isabel Cristina	Toledo Carrillo	05/06/1990	isabel.toledo...

Imagen 16: Lista de clientes registrados

- **Módulo de Empleados**
 - **Registrar empleado**

En el módulo de empleados, la función de registrar empleado detalla cada campo necesario, incluyendo la información personal y las credenciales para inicio de sesión. Cada campo está validado para permitir solo el tipo de dato correcto, sin admitir campos vacíos. Todos los aspectos están minuciosamente verificados, asegurando que no haya fallos en el proceso de registro, tanto de la información personal como de las credenciales de acceso. Se ha comprobado que la funcionalidad es correcta y completa, garantizando un registro seguro y eficiente de nuevos empleados en el sistema.



Imagen 17: Registro de empleados

- **Editar empleado**

En el módulo de empleados, la función de modificar empleado permite buscar y seleccionar el empleado que se desea actualizar. Al seleccionar un

empleado, sus datos personales y credenciales de inicio de sesión se muestran automáticamente en los campos correspondientes, listos para ser editados. Cada campo está debidamente validado, incluyendo las opciones para actualizar el nombre de usuario y la contraseña, asegurando que se mantengan los estándares de seguridad. El sistema verifica cuidadosamente cada aspecto de la información ingresada, tanto para los datos personales como para las credenciales, garantizando la integridad y seguridad de la información actualizada. Se ha confirmado que esta funcionalidad opera de manera correcta y completa, permitiendo modificaciones seguras en todos los aspectos del perfil del empleado.



IdEmpleado	Usuario	IdPersona	Cedula	Nombre	Apellido	FechaNacimiento	Email
1	jp	0	0102034567	Luis Eduar...	Andrade P...	15/01/1985	stevengua...
2	mariagom...	0	0203045678	Mar a Fer...	Barrera G ...	20/02/1990	maria.barr...
3	carloslopez	0	0304056789	Juan Carlos	C rdenas L...	25/03/1987	juan.carde...
4	anamartin...	0	0405067890	Ana Isabel	D az Mart ...	30/04/1992	ana.diaz@...

Imagen 18: Editar un empleado

○ Buscar y mostrar empleado

En el módulo de empleados, la función de listar empleados muestra todos los registros en un DataGridView. Este componente presenta de forma clara y organizada la información básica de cada empleado, como nombre, identificación, cargo y nombre de usuario. La función está optimizada para cargar y mostrar eficientemente grandes cantidades de registros. Se excluyen datos sensibles como contraseñas por seguridad. El DataGridView permite ordenar las columnas para facilitar la búsqueda visual. Se ha verificado que esta funcionalidad opera correctamente, ofreciendo una visión general rápida y completa de todos los empleados en el sistema.



IdEmpleado	Usuario	IdPersona	Cedula	Nombre	Apellido	FechaNacimiento	Email
1	jp	0	0102034567	Luis Eduar...	Andrade P...	15/01/1985	stevengua...
2	mariagom...	0	0203045678	Mar a Fern...	Barrera G ...	20/02/1990	maria.barr...
3	carloslopez	0	0304056789	Juan Carlos	C rdenas L...	25/03/1987	juan.carde...
4	anamartinez	0	0405067890	Ana Isabel	D az Mart ...	30/04/1992	ana.diaz@...

Imagen 19: Lista de empleados registrados

- **Módulo de Proveedor**

- **Registrar Proveedor**

En el módulo de proveedores, la función de registro permite ingresar los datos esenciales del proveedor: RUC, teléfono, correo, dirección y Nombre comercial. Cada campo está validado para aceptar solo el tipo de dato correcto y no se permiten campos vacíos, asegurando la integridad de la información. Se ha comprobado que esta funcionalidad cumple con todos los requisitos y opera de manera efectiva.

SGFacturacion

Registrar Proveedor

RUC: Telefono:

Correo: Direccion:

Razon Comercial:

Registrar

Imagen 20: Registro de proveedor

- **Editar Proveedor**

La función de modificar proveedor permite buscar y seleccionar el proveedor a actualizar. Los datos actuales del proveedor se muestran automáticamente en los campos correspondientes para ser editados. Cada entrada está validada para asegurar la consistencia de los datos. La funcionalidad ha sido verificada, garantizando que las modificaciones se procesen correctamente.

SGFacturacion

Editar Proveedores

RUC: 1790012345678 Telefono: 02-2999999

Correo: info@cocacola.com Direccion: Av. Principal 123, Quito

Razon Comercial: CocaCola

Editar

Buscar:

ID_Proveedor	Ruc	RazonComercial	Telefono	Direccion	Email
1	1790012345677	CocaCola	02-2999999	Av. Principal 1...	info@cocacol...
2	1798765432109	Pepsi	02-2888888	Av. Secundaria...	contacto@pep...
3	1791234567890	Tony	07-2777777	Calle Falsa 789...	ventas@tony.c...
4	1790987654321	Danec	04-2666666	Parque Industr...	soporte@danec...
5	1791357924680	Maggi	03-2555555	Zona Industrial...	help@maggi.c...
10002	1111111111111	pepito	11111111111	aaaaaaaaaaaaa...	cocacola@gm...

Imagen 21: Editar proveedor

- **Buscar y mostrar Proveedor**

La función de mostrar proveedores despliega todos los registros en un DataGridView (DGV), permitiendo una visualización clara y organizada de los datos. Esta herramienta facilita la revisión de la información sin necesidad de filtros complejos, presentando los detalles de cada proveedor de manera accesible y ordenada.

ID_Proveedor	Ruc	RazonComercial	Telefono	Direccion	Email
1	1790012345677	CocaCola	02-2999999	Av. Principal 1...	info@cocacola...
2	1798765432109	Pepsi	02-2888888	Av. Secundaria...	contacto@pep...
3	1791234567890	Tony	07-2777777	Calle Falsa 789...	ventas@tony.c...
4	1790987654321	Danec	04-2666666	Parque Industr...	soporte@danec...
5	1791357924680	Maggi	03-2555555	Zona Industrial...	help@maggi.c...
10002	1111111111111	pepito	1111111111	aaaaaaaaaaaaa...	cocacola@gm...

Imagen 22: Lista de proveedores

- **Módulo de Producto**

- **Registrar Producto**

En el módulo de productos, la función de registro permite ingresar los datos esenciales del producto: nombre, precio unitario, stock y razón comercial (proveedor). El proveedor se selecciona desde un combo box que muestra los proveedores registrados. Cada campo está validado para asegurar la entrada de datos correcta y completa, evitando errores en el proceso de registro.

Imagen 23: Registro de productos

- **Editar Producto**

La función de modificar producto permite seleccionar un producto existente para actualizar sus datos. Al elegir un producto, sus detalles se cargan automáticamente en los campos correspondientes, listos para ser editados. Los campos están validados para garantizar la integridad de la información, asegurando que los cambios se apliquen correctamente.

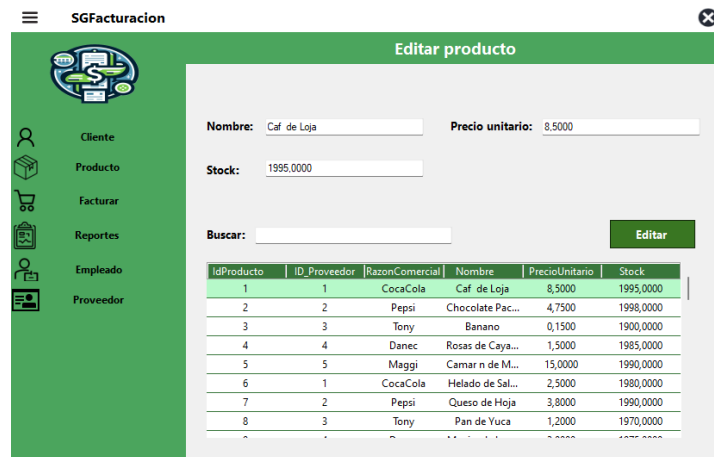


Imagen 24: Editar productos

- **Buscar y Mostrar producto**

La función de mostrar productos despliega todos los registros de productos en un DataGridView (DGV), permitiendo visualizar de manera ordenada los datos como nombre, precio, stock y proveedor. Esta vista facilita la revisión y gestión de los productos sin necesidad de filtros adicionales, mostrando la información de manera clara y accesible.



Imagen 25: Lista de productos registrados

- **Módulo de factura**

- **Generar Factura**

El modulo de factura, la función de generar una nueva factura funciona de forma correcta, el cargado de los datos así como la generación de la factura esta totalmente funcional, una vez que se realiza las factura se muestra un reporte donde se encuentra el detalle de la factura.

Imagen 26: Generar factura

ID Produc	Producto	Precio	Cantidad	Sub Total
5	Camar n de Manab	15,0000	20,0000	345,0000
17	Caf de Gal pagos	10,0000	100,0000	1150,0000

TOTAL: 1495,0000

Imagen 27: Factura generada en forma de reporte

○ Generar reportes

La función de generar reportes funciona de forma correcta, permite la generación de distintos reportes, así como poder generar las facturas según las fechas en las que fueron generadas.

ID Produc	Producto	Precio	Cantidad	Sub Total
5	Camar n de Manab	15,0000	20,0000	345,0000
17	Caf de Gal pagos	10,0000	100,0000	1150,0000

Imagen 28: Reporte de factura según la fecha generada

Pruebas unitarias

RegistrarCliente_ShouldReturnTrue_WhenRegistrationIsSuccessful

Descripción:

Esta prueba unitaria verifica que el método RegistrarPersona() de la clase csPersona funcione correctamente al registrar un nuevo cliente en el sistema. La prueba crea un nuevo objeto cliente con datos válidos y espera que el método de registro retorne true, indicando un registro exitoso.

Precondiciones:

- La clase csPersona está implementada y accesible.
- El método RegistrarCliente() está implementado en la clase csCliente.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- No existe un cliente con la cédula "2100562384" en el sistema.

Pasos:

- Arrange (Preparación):

Crear un nuevo objeto csPersona con los siguientes datos:

ID: 1

Cédula: "2100562321"

Nombre: "Maria"

Apellido: "Elizabe"

Fecha: Fecha y hora actual

Email: "asss22@example.com"

- Act (Acción):

Llamar al método RegistrarPersona () del objeto cliente creado.

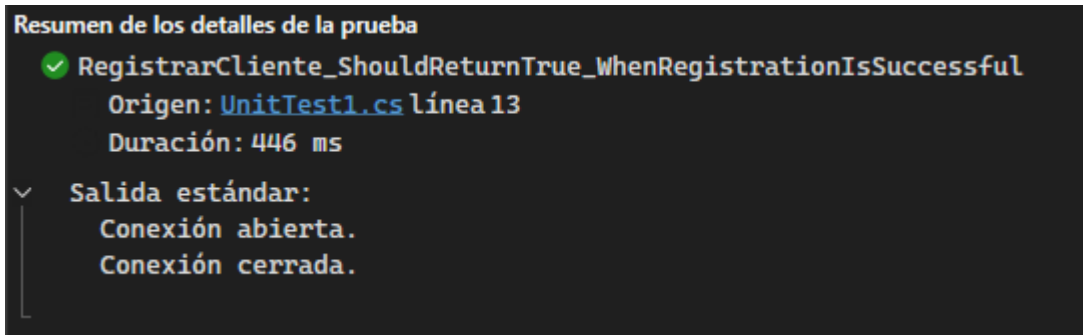
Almacenar el resultado booleano en la variable 'registrado'.

- Assert (Afirmación):

Verificar que el valor de 'registrado' sea true utilizando Assert.IsTrue().

Resultado esperado:

El método RegistrarCliente() debe retornar true, indicando que el cliente se registró exitosamente en el sistema.



Resumen de los detalles de la prueba

✓ RegistrarCliente_ShouldReturnTrue_WhenRegistrationIsSuccessful

Origen: UnitTest1.cs línea 13

Duración: 446 ms

Salida estándar:

Conexión abierta.

Conexión cerrada.

Código:



```
[TestMethod]
public void RegistrarCliente_ShouldReturnTrue_WhenRegistrationIsSuccessful()
{
    // Arrange
    var cliente = new csPersona(1, cedula: "2100562321", nombre: "Maria", apellido: "Elizabe", DateTime.Now, email: "asss22@example.com");

    // Act
    bool registrado = cliente.RegistrarPersona();

    // Assert
    Assert.IsTrue(registrado);
}
```

RegistrarCliente_ShouldReturnFalse_OnDuplicateCedula

Descripción:

Esta prueba unitaria verifica que el método RegistrarPersona () de la clase csPersona retorne false cuando se intenta registrar un cliente con una cédula que ya existe en el sistema.

La prueba simula un intento de registro de un cliente con una cédula duplicada y espera que el método rechace el registro.

Precondiciones:

- La clase csPersona está implementada y accesible.
- El método RegistrarPersona() está implementado en la clase csPersona.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- Ya existe un cliente en el sistema con la cédula " 0304056789".

Pasos:

- Arrange (Preparación):

Crear un nuevo objeto csCliente con los siguientes datos:

ID: 1

Cédula: " 0304056789" (una cédula que se asume ya existe en el sistema)

Nombre: "Jona"

Apellido: "Martin"

Fecha: Fecha y hora actual

Email: "jasasahn@example.com"

- Act (Acción):

Llamar al método RegistrarCliente() del objeto cliente creado.

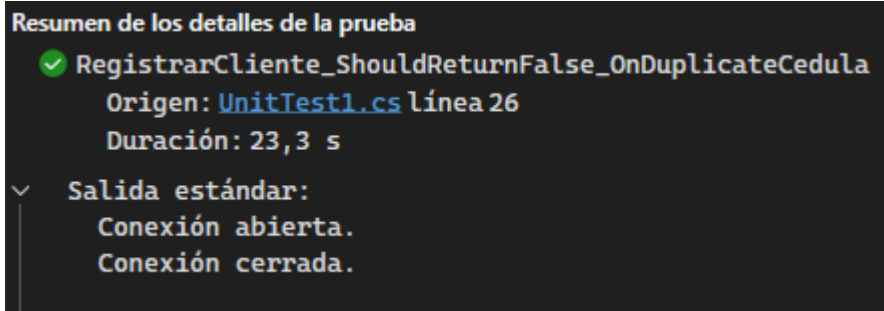
Almacenar el resultado booleano en la variable 'registrado'.

- Assert (Afirmación):

Verificar que el valor de 'registrado' sea false utilizando Assert.IsFalse().

Resultado esperado:

El método RegistrarPersona() debe retornar false, indicando que el cliente no se pudo registrar debido a que la cédula ya existe en el sistema.



Código:

```
[TestMethod]
public void RegistrarCliente_ShouldReturnFalse_OnDuplicateCedula()
{
    // Arrange
    var cliente = new csPersona(1, cedula: "0304056789", nombre: "Jona", apellido: "Martin", DateTime.Now, email: "jasasahn@example.com");

    // Act
    bool registrado = cliente.RegistrarPersona();

    // Assert
    Assert.IsFalse(registrado);
}
```

EditarCliente_ShouldReturnTrue_WhenEditIsSuccessful

Descripción:

Esta prueba unitaria verifica que el método EditarPersona () de la clase csPersona funcione correctamente al editar un cliente existente en el sistema. La prueba crea un objeto cliente con datos actualizados y espera que el método de edición retorne true, indicando una actualización exitosa.

Precondiciones:

- La clase csPersona está implementada y accesible.
- El método EditarPersona () está implementado en la clase csPersona.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- Existe un cliente en el sistema con ID 1.

Pasos:

- Arrange (Preparación):

Crear un objeto csPersona con los siguientes datos:

ID: 1 (asumiendo que este cliente ya existe en el sistema)

Cédula: "1234567890"

Nombre: "John"

Apellido: "Doe"

Fecha: Fecha y hora actual

Email: "john@example.com"

- Act (Acción):

Llamar al método EditarPersona () del objeto cliente creado.

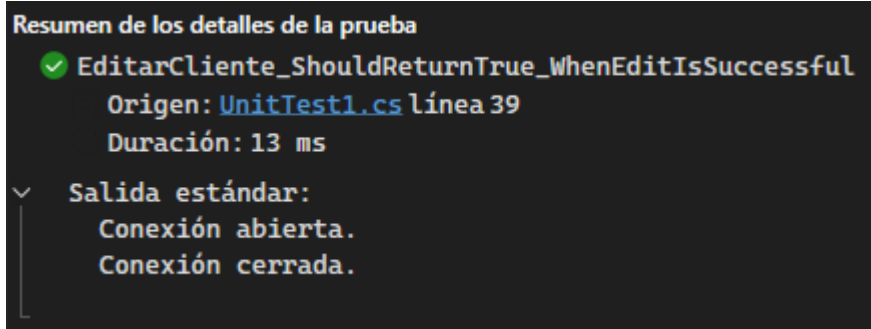
Almacenar el resultado booleano en la variable 'editado'.

- Assert (Afirmación):

Verificar que el valor de 'editado' sea true utilizando Assert.IsTrue().

Resultado esperado:

El método EditarCliente() debe retornar true, indicando que la información del cliente se actualizó exitosamente en el sistema.



Código:

```
[TestMethod]
0 referencias
public void EditarCliente_ShouldReturnTrue_WhenEditIsSuccessful()
{
    // Arrange
    var cliente = new csPersona(1, cedula: "1234567890", nombre: "John", apellido: "Doe", DateTime.Now, email: "john@example.com");

    // Act
    bool editado = cliente.EditarPersona();

    // Assert
    Assert.IsTrue(editado);
}
```

ListarClientes_ShouldReturnListOfClients

Descripción:

Esta prueba unitaria verifica que el método estático ListarPersona() de la clase csPersona funcione correctamente al recuperar una lista de clientes del sistema. La prueba espera que el método devuelva una lista no nula y no vacía de objetos csPersona.

Precondiciones:

- La clase csPersona está implementada y accesible.
- El método estático ListarPersona() está implementado en la clase csPersona.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- Existen uno o más clientes registrados en el sistema.

Pasos:

- Act (Acción):

Llamar al método estático ListarPersona() de la clase csPersona.

Almacenar el resultado (una lista de objetos csPersona) en la variable 'clientes'.

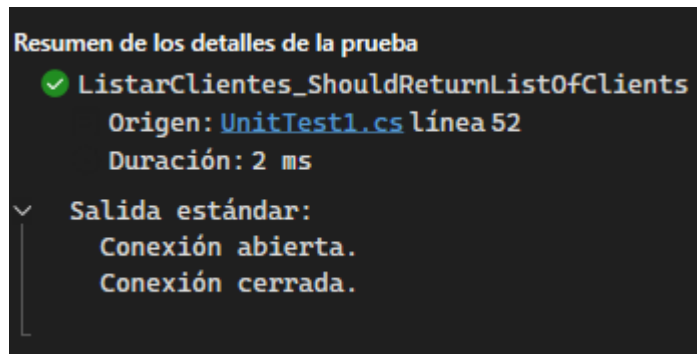
- Assert (Afirmación):

Verificar que la lista 'clientes' no sea nula utilizando Assert.IsNotNull().

Verificar que la lista 'clientes' contenga al menos un elemento utilizando Assert.IsTrue(clientes.Count > 0).

Resultado esperado:

El método ListarPersona() debe retornar una lista no nula que contenga al menos un objeto csPersona.



Código:

```
[TestMethod]
0 | 0 referencias
public void ListarClientes_ShouldReturnListOfClients()
{
    // Act
    List<csPersona> clientes = csPersona.ListarPersona();

    // Assert
    Assert.IsNotNull(clientes);
    Assert.IsTrue(clientes.Count > 0);
}
```

BuscarClientes_ShouldReturnMatchingClients

Descripción:

Esta prueba unitaria verifica que el método estático BuscarPersona () de la clase csPersona funcione correctamente al buscar clientes en el sistema basándose en un término de búsqueda. La prueba espera que el método devuelva una lista no nula y no vacía de objetos csPersona que coincidan con el criterio de búsqueda.

Precondiciones:

- La clase csPersona está implementada y accesible.
- El método estático BuscarPersona() está implementado en la clase csPersona.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- Existen uno o más clientes registrados en el sistema que coinciden con el término de búsqueda "John".

Pasos:

- Arrange (Preparación):
Definir un término de búsqueda. En este caso, se usa "John".
- Act (Acción):

Llamar al método estático BuscarPersona() de la clase csPersona, pasando el término de búsqueda "John".

Almacenar el resultado (una lista de objetos csPersona) en la variable 'clientes'.

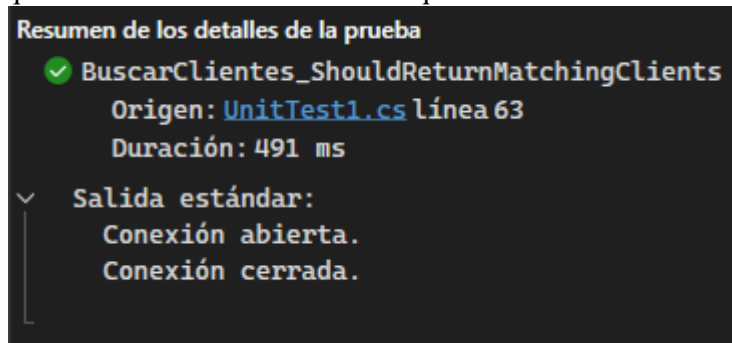
- Assert (Afirmación):

Verificar que la lista 'clientes' no sea nula utilizando Assert.IsNotNull().

Verificar que la lista 'clientes' contenga al menos un elemento utilizando Assert.IsTrue(clientes.Count > 0).

Resultado esperado:

El método BuscarPersona() debe retornar una lista no nula que contenga al menos un objeto csPersona que coincida con el término de búsqueda "John".



Código:

```
[TestMethod]
0 referencias
public void BuscarClientes_ShouldReturnMatchingClients()
{
    // Arrange
    string busqueda = "John"; // Ejemplo de término de búsqueda

    // Act
    List<csPersona> clientes = csPersona.BuscarPersona(busqueda);

    // Assert
    Assert.IsNotNull(clientes);
    Assert.IsTrue(clientes.Count > 0);
}
```

GetFacturaIDByClienteIDAndFecha_ShouldReturnValidID

Descripción:

Esta prueba unitaria verifica que el método estático `GetFacturaIDByClienteIDAndFecha()` de la clase `csPersona` funcione correctamente al recuperar el ID de una factura basándose en el ID de un cliente y una fecha específica. La prueba espera que el método devuelva un ID de factura válido (mayor que cero) para los parámetros proporcionados.

Precondiciones:

- La clase `csPersona` está implementada y accesible.
- El método estático `GetFacturaIDByClienteIDAndFecha()` está implementado en la clase `csPersona`.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- Existe al menos una factura en el sistema para el cliente con ID 1 en la fecha actual.

Pasos:

- Arrange (Preparación):

Definir un ID de cliente válido (en este caso, 1).

Obtener la fecha actual (sin la parte de tiempo) usando `DateTime.Now.Date`.

- Act (Acción):

Llamar al método estático `GetFacturaIDByClienteIDAndFecha()` de la clase `csPersona`, pasando el ID del cliente y la fecha.

Almacenar el resultado (un long que representa el ID de la factura) en la variable 'idFactura'.

- Assert (Afirmación):

Verificar que el valor de 'idFactura' sea mayor que cero utilizando `Assert.IsTrue(idFactura > 0)`.

Resultado esperado:

El método `GetFacturaIDByClienteIDAndFecha()` debe retornar un ID de factura válido (un número entero mayor que cero) para el cliente y la fecha especificados.

```
Resumen de los detalles de la prueba
✅ GetFacturaIDByClienteIDAndFecha_ShouldReturnValidID
  Origen: UnitTest1.cs línea 77
  Duración: 463 ms
  Salida estándar:
    Conexión abierta.
    Conexión cerrada.
```

Código:

```
[TestMethod]
0 | 0 referencias
public void GetFacturaIDByClienteIDAndFecha_ShouldReturnValidID()
{
    // Arrange
    long idCliente = 1; // ID de cliente válido
    DateTime fecha = Convert.ToDateTime("2024-07-18 00:53:52.380"); // Fecha válida

    // Act
    long idFactura = csPersona.GetFacturaIDByClienteIDAndFecha(idCliente, fecha);

    // Assert
    Assert.IsFalse(idFactura >= 1); // Se espera que el ID de factura sea válido
}
```

RegistrarEmpleado_ShouldReturnTrue_WhenRegistrationIsSuccessful

Descripción:

Esta prueba unitaria verifica que el método RegistrarEmpleado() de la clase csEmpleados funcione correctamente al registrar un nuevo empleado en el sistema, incluyendo sus credenciales. La prueba espera que el método retorne true, indicando un registro exitoso.

Precondiciones:

- La clase csEmpleados y su clase interna csCredenciales están implementadas y accesibles.
- El método RegistrarEmpleado() está implementado en la clase csEmpleados.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- No existe un empleado con la cédula "1234567890" en el sistema.
- No existe un usuario con el nombre de usuario "johndoe" en el sistema.

Pasos:

- Arrange (Preparación):

Crear un objeto csCredenciales con nombre de usuario "johndoe" y contraseña "password".

Crear un objeto csEmpleados con los siguientes datos:

ID: 1

Cédula: "1234567890"

Nombre: "John"

Apellido: "Doe"

Fecha: Fecha y hora actual

Email: "john@example.com"

- Act (Acción):

Llamar al método RegistrarEmpleado() del objeto empleado creado, pasando el objeto credenciales como parámetro.

Almacenar el resultado booleano en la variable 'registrado'.

- Assert (Afirmación):

Verificar que el valor de 'registrado' sea true utilizando Assert.IsTrue().

Resultado esperado:

El método RegistrarEmpleado() debe retornar true, indicando que el empleado y sus credenciales se registraron exitosamente en el sistema.

```
Resumen de los detalles de la prueba
✓ RegistrarEmpleado_ShouldReturnTrue_WhenRegistrationIsSuccessful
  Origen: UnitTest1.cs línea 93
  Duración: 4 ms
  Salida estándar:
    Conexión abierta.
    Conexión cerrada.
```

Código:

```

[TestMethod]
// 0 referencias
public void RegistrarEmpleado_ShouldReturnTrue_WhenRegistrationIsSuccessful()
{
    // Arrange
    var credenciales = new csEmpleados.csCredenciales(usuario: "johndoe", contraseña: "password");
    var empleado = new csEmpleados(idEmpleado: 1, cedula: "1234567890", nombre: "John", apellido: "Doe", DateTime.Now, email: "john@example.com");

    // Act
    bool registrado = empleado.RegistrarEmpleado(credenciales);

    // Assert
    Assert.IsTrue(registrado);
}

```

EditarEmpleado_ShouldReturnTrue_WhenEditIsSuccessful

Descripción:

Esta prueba unitaria verifica que el método EditarEmpleado() de la clase csEmpleados funcione correctamente al editar la información de un empleado existente en el sistema, incluyendo la actualización de sus credenciales. La prueba espera que el método retorne true, indicando una edición exitosa.

Precondiciones:

- La clase csEmpleados y su clase interna csCredenciales están implementadas y accesibles.
- El método EditarEmpleado() está implementado en la clase csEmpleados.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- Existe un empleado en el sistema con ID 1.
- Existe un usuario con el nombre de usuario "johndoe" en el sistema.

Pasos:

- Arrange (Preparación):

Crear un objeto csCredenciales con nombre de usuario "johndoe" y nueva contraseña "newpassword".

Crear un objeto csEmpleados con los siguientes datos actualizados:

ID: 1 (asumiendo que este empleado ya existe en el sistema)

Cédula: "1234567890"

Nombre: "John"

Apellido: "Doe"

Fecha: Fecha y hora actual

Email: "john@example.com"

- Act (Acción):

Llamar al método EditarEmpleado() del objeto empleado creado, pasando el objeto credenciales como parámetro.

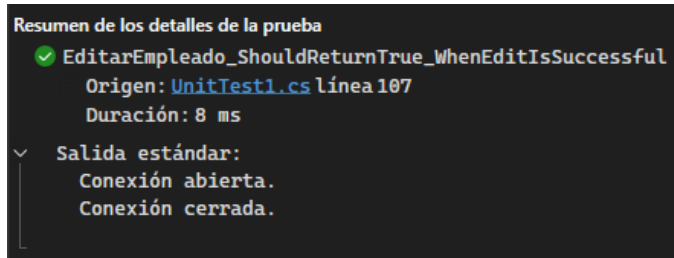
Almacenar el resultado booleano en la variable 'editado'.

- Assert (Afirmación):

Verificar que el valor de 'editado' sea true utilizando Assert.IsTrue().

Resultado esperado:

El método EditarEmpleado() debe retornar true, indicando que la información del empleado y sus credenciales se actualizaron exitosamente en el sistema.



Código:

```
[TestMethod]
[0 referencias]
public void EditarEmpleado_ShouldReturnTrue_WhenEditIsSuccessful()
{
    // Arrange
    var credenciales = new csEmpleados.csCredenciales(usuario: "jp", contraseña: "root");
    var empleado = new csEmpleados(idEmpleado: 1, cedula: "1234567890", nombre: "John", apellido: "Doe", DateTime.Now, email: "john@example.com");

    // Act
    bool editado = empleado.EditarEmpleado(credenciales);

    // Assert
    Assert.IsTrue(editado);
}
```

IniciarSesion_ShouldReturnTrue_WhenCredentialsAreCorrect

Descripción:

Esta prueba unitaria verifica que el método IniciarSesion() de la clase csCredenciales (una clase interna de csEmpleados) funcione correctamente cuando se proporcionan credenciales válidas. La prueba espera que el método retorne true, indicando un inicio de sesión exitoso.

Precondiciones:

- La clase csEmpleados y su clase interna csCredenciales están implementadas y accesibles.
- El método IniciarSesion() está implementado en la clase csCredenciales.
- El sistema de autenticación está disponible y accesible.
- Existe un usuario en el sistema con nombre de usuario "johndoe" y contraseña "newpassword".

Pasos:

- Arrange (Preparación):
Crear un objeto csCredenciales con nombre de usuario "johndoe" y contraseña "newpassword".
- Act (Acción):
Llamar al método IniciarSesion() del objeto credenciales creado.
Almacenar el resultado booleano en la variable 'sesionIniciada'.
- Assert (Afirmación):
Verificar que el valor de 'sesionIniciada' sea true utilizando Assert.IsTrue().

Resultado esperado:

El método IniciarSesion() debe retornar true, indicando que las credenciales proporcionadas son correctas y el inicio de sesión fue exitoso.

```
Resumen de los detalles de la prueba
✓ IniciarSesion_ShouldReturnTrue_WhenCredentialsAreCorrect
  Origen: UnitTest1.cs línea 121
  Duración: 1 ms
  Salida estándar:
    Conexión abierta.
    Conexión cerrada.
```

Código:

```
[TestMethod]
0 referencias
public void IniciarSesion_ShouldReturnTrue_WhenCredentialsAreCorrect()
{
    // Arrange
    var credenciales = new csEmpleados.csCredenciales(usuario: "jp", contraseña: "root");

    // Act
    bool sesionIniciada = credenciales.IniciarSesion();

    // Assert
    Assert.IsTrue(sesionIniciada);
}
```

IniciarSesion_ShouldReturnFalse_WhenIncorrectCredentials

Descripción:

Esta prueba unitaria verifica que el método IniciarSesion() de la clase csCredenciales (una clase interna de csEmpleados) funcione correctamente cuando se proporcionan credenciales incorrectas. La prueba espera que el método retorne false, indicando un intento de inicio de sesión fallido.

Precondiciones:

- La clase csEmpleados y su clase interna csCredenciales están implementadas y accesibles.
- El método IniciarSesion() está implementado en la clase csCredenciales.
- El sistema de autenticación está disponible y accesible.
- Existe un usuario en el sistema con nombre de usuario "ad1", pero su contraseña no es "rootaa".

Pasos:

- Arrange (Preparación):

Crear un objeto csCredenciales con nombre de usuario "ad1" y contraseña incorrecta "rootaa".

- Act (Acción):

Llamar al método IniciarSesion() del objeto credenciales creado.

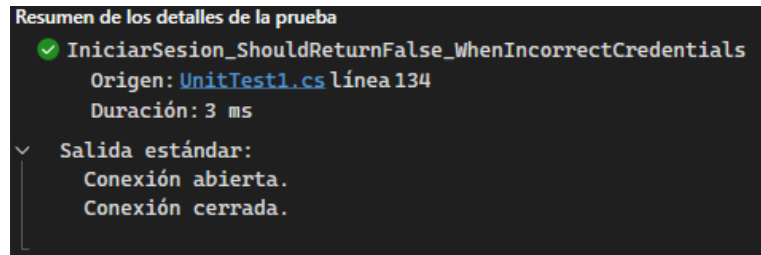
Almacenar el resultado booleano en la variable 'sesionIniciada'.

- Assert (Afirmación):

Verificar que el valor de 'sesionIniciada' sea false utilizando Assert.IsFalse().

Resultado esperado:

El método IniciarSesion() debe retornar false, indicando que las credenciales proporcionadas son incorrectas y el intento de inicio de sesión ha fallado.



Código:

```
[TestMethod]
0 referencias
public void IniciarSesion_ShouldReturnFalse_WhenIncorrectCredentials()
{
    // Arrange
    var credenciales = new csEmpleados.csCredenciales(usuario: "ad1", contraseña: "rootaa");

    // Act
    bool sesionIniciada = credenciales.IniciarSesion();

    // Assert
    Assert.IsFalse(sesionIniciada);
}
```

ListarEmpleados_ShouldReturnListOfEmployees

Descripción:

Esta prueba unitaria verifica que el método estático ListarEmpleados() de la clase csEmpleados funcione correctamente al recuperar una lista de empleados del sistema. La prueba espera que el método devuelva una lista no nula y no vacía de objetos csEmpleados.

Precondiciones:

- La clase csEmpleados está implementada y accesible.
- El método estático ListarEmpleados() está implementado en la clase csEmpleados.
- La base de datos o el sistema de almacenamiento está disponible y accesible.
- Existen uno o más empleados registrados en el sistema.

Pasos:

- Act (Acción):

Llamar al método estático ListarEmpleados() de la clase csEmpleados.

Almacenar el resultado (una lista de objetos csEmpleados) en la variable 'empleados'.

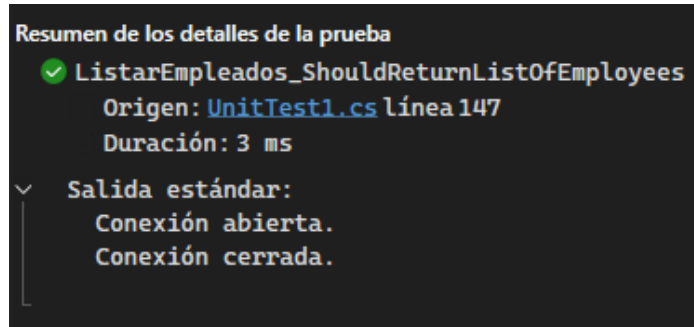
- Assert (Afirmación):

Verificar que la lista 'empleados' no sea nula utilizando Assert.IsNotNull().

Verificar que la lista 'empleados' contenga al menos un elemento utilizando Assert.IsTrue(empleados.Count > 0).

Resultado esperado:

El método ListarEmpleados() debe retornar una lista no nula que contenga al menos un objeto csEmpleados, representando los empleados registrados en el sistema.



Código:

```
public void ListarEmpleados_ShouldReturnListOfEmployees()
{
    // Act
    List<csEmpleados> empleados = csEmpleados.ListarEmpleados();

    // Assert
    Assert.IsNotNull(empleados);
    Assert.IsTrue(empleados.Count > 0);
}
```

BuscarEmpleados_ShouldReturnMatchingEmployees

Descripción:

Esta prueba unitaria verifica que el método estático BuscarEmpleados() de la clase csEmpleados funcione correctamente al buscar empleados que coincidan con el término de búsqueda "John". La prueba espera que el método devuelva una lista no nula y no vacía de objetos csEmpleados que contengan empleados cuyos nombres coincidan con "John".

Precondiciones:

- La clase csEmpleados está implementada y accesible.
- El método estático BuscarEmpleados(string busqueda) está implementado en la clase csEmpleados.
- Existen empleados registrados en el sistema cuyos nombres coincidan con "John".

Pasos:

- Arrange (Preparación)

Asignar el valor "John" a la variable 'busqueda'

- Act (Acción):

Llamar al método estático BuscarEmpleados(busqueda) de la clase csEmpleados.

Almacenar el resultado (una lista de objetos csEmpleados) en la variable empleados.

- Assert (Afirmación):

Verificar que la lista empleados no sea nula utilizando
Assert.IsNotNull(empleados).

Verificar que la lista empleados contenga al menos un elemento utilizando
Assert.IsTrue(empleados.Count > 0).

Resultado esperado:

El método BuscarEmpleados("John") debe retornar una lista no nula que contenga al menos un objeto csEmpleados cuyo nombre coincida con "John", representando los empleados encontrados en el sistema que coinciden con el término de búsqueda.


```
Resumen de los detalles de la prueba
✓ BuscarEmpleados_ShouldReturnMatchingEmployees
  Origen: UnitTest1.cs línea 158
  Duración: 11 ms
  Salida estándar:
    Conexión abierta.
    Conexión cerrada.
```

Código:

```
public void BuscarEmpleados_ShouldReturnMatchingEmployees()
{
    // Arrange
    string busqueda = "John"; // Ejemplo de término de búsqueda

    // Act
    List<csEmpleados> empleados = csEmpleados.BuscarEmpleados(busqueda);

    // Assert
    Assert.IsNotNull(empleados);
    Assert.IsTrue(empleados.Count > 0);
}
```

RegistrarProducto_ShouldReturnTrue_WhenRegistrationIsSuccessful

Descripción:

Esta prueba unitaria verifica que el método RegistrarProducto() de la clase csProducto funcione correctamente al intentar registrar un nuevo producto con los parámetros especificados. La prueba espera que el método retorne true cuando el registro sea exitoso.

Precondiciones:

- La clase csProducto está implementada y accesible.
- Se ha creado una instancia de csProducto con nombre "Producto de Prueba", precio de 10.5 y cantidad inicial de 100.

Pasos:

- Arrange (Preparación):

Crear una nueva instancia de csProducto llamada producto con los valores especificados: nombre "Producto de Prueba", precio 10.5 y cantidad 100.

- Act (Acción):

Llamar al método RegistrarProducto() en la instancia producto.

- Assert (Afirmación):

Verificar que la variable registrada sea true utilizando Assert.IsTrue(registrado).

Resultado esperado:

Se espera que el método RegistrarProducto() retorne true, indicando que el registro del producto fue exitoso.

```
Resumen de los detalles de la prueba
✓ RegistrarProducto_ShouldReturnTrue_WhenRegistrationIsSuccessful
  Origen: UnitTest1.cs línea 173
  Duración: 3 ms
  Salida estándar:
    Conexión abierta.
    Conexión cerrada.
```

Código:

```
public void RegistrarProducto_ShouldReturnTrue_WhenRegistrationIsSuccessful()
{
    // Arrange
    var producto = new csProducto(nombre: "Producto de Prueba", precioUnitario: 10.5m, stock: 100);

    // Act
    bool registrado = producto.RegistrarProducto();

    // Assert
    Assert.IsTrue(registrado);
}
```

EditarProducto_ShouldReturnTrue_WhenEditIsSuccessful

Descripción:

Esta prueba unitaria verifica que el método EditarProducto() de la clase csProducto funcione correctamente al intentar editar un producto existente con los parámetros especificados. La prueba espera que el método retorne true cuando la edición sea exitosa.

Precondiciones:

- La clase csProducto está implementada y accesible.
- Se ha creado una instancia de csProducto con el ID 1, nombre "Producto Actualizado", precio de 15.75 y cantidad de 200, representando un producto existente que se desea editar.

Pasos:

- Arrange (Preparación):

Crear una nueva instancia de csProducto llamada producto con los valores especificados: ID 1, nombre "Producto Actualizado", precio 15.75 y cantidad 200.

- Act (Acción):

Llamar al método EditarProducto() en la instancia producto.

- Assert (Afirmación):

Verificar que la variable editada sea true utilizando Assert.IsTrue(editado).

Resultado esperado:

Se espera que el método EditarProducto() retorne true, indicando que la edición del producto fue exitosa.

```
Resumen de los detalles de la prueba
✓ EditarProducto_ShouldReturnTrue_WhenEditIsSuccessful
  Origen: UnitTest1.cs línea 186
  Duración: 11 ms
  Salida estándar:
    Conexión abierta.
    Conexión cerrada.
```

Código:

```
public void EditarProducto_ShouldReturnTrue_WhenEditIsSuccessful()
{
    // Arrange
    var producto = new csProducto(idProveedor: 1, nombre: "Producto Actualizado", precioUnitario: 15.75m, stock: 200);

    // Act
    bool editado = producto.EditarProducto();

    // Assert
    Assert.IsTrue(editado);
}
```

ListarProductos_ShouldReturnListOfProducts

Descripción:

Esta prueba unitaria verifica que el método estático ListarProductos() de la clase csProducto funcione correctamente al listar los productos existentes en el sistema. La prueba espera que el método devuelva una lista no nula y que contenga al menos un producto.

Precondiciones:

- La clase csProducto está implementada y accesible.
- Existen productos registrados en el sistema que puedan ser listados por el método ListarProductos().

Pasos:

- Act (Acción):

Llamar al método estático ListarProductos() de la clase csProducto para obtener la lista de productos.

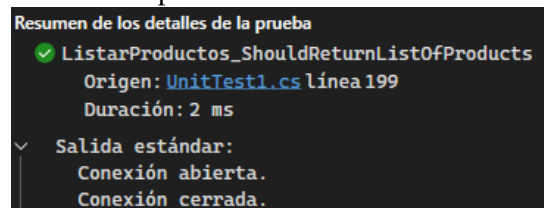
- Assert (Afirmación):

Verificar que la variable productos no sea null utilizando Assert.IsNotNull(productos).

Verificar que la lista productos contenga al menos un elemento utilizando Assert.IsTrue(productos.Count > 0).

Resultado esperado:

Se espera que el método ListarProductos() retorne una lista no nula que contenga al menos un producto, representando los productos existentes en el sistema.



Resumen de los detalles de la prueba

- ✓ ListarProductos_ShouldReturnListOfProducts
- Origen: `UnitTest1.cs` línea 199
- Duración: 2 ms
- Salida estándar:
 - Conexión abierta.
 - Conexión cerrada.

Código:

```
public void ListarProductos_ShouldReturnListOfProducts()
{
    // Act
    List<csProducto> productos = csProducto.ListarProductos();
    ....

    // Assert
    Assert.IsNotNull(productos);
    Assert.IsTrue(productos.Count > 0);
}
```

BuscarProductoPorNombre_ShouldReturnMatchingProducts

Descripción:

Esta prueba unitaria verifica que el método estático BuscarProductoPorNombre() de la clase csProducto funcione correctamente al buscar productos cuyo nombre coincida con el patrón especificado. La prueba espera que el método devuelva una lista no nula y que contenga al menos un producto cuyo nombre coincida con el patrón de búsqueda.

Precondiciones:

- La clase csProducto está implementada y accesible.
- Existen productos registrados en el sistema cuyos nombres coincidan con el patrón de búsqueda especificado ("Banano" en este caso).

Pasos:

- Arrange (Preparación):

Asignar el valor "Banano" a la variable patronNombre como ejemplo de término de búsqueda.

- Act (Acción):

Llamar al método estático BuscarProductoPorNombre(patronNombre) de la clase csProducto para buscar productos que coincidan con el nombre especificado por patronNombre.

- Assert (Afirmación):

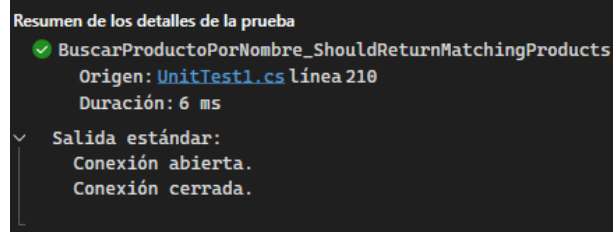
Verificar que la variable productos no sea null utilizando Assert.IsNotNull(productos).

Verificar que la lista productos contenga al menos un elemento utilizando

Assert.IsTrue(productos.Count > 0).

Resultado esperado:

Se espera que el método BuscarProductoPorNombre("Banano") retorne una lista no nula que contenga al menos un producto cuyo nombre coincida con el patrón de búsqueda "Banano", representando los productos encontrados en el sistema que coinciden con el término de búsqueda.

**Código:**

```
public void BuscarProductoPorNombre_ShouldReturnMatchingProducts()
{
    // Arrange
    string patronNombre = "Banano"; // Ejemplo de término de búsqueda

    // Act
    List<csProducto> productos = csProducto.BuscarProductoPorNombre(patronNombre);

    // Assert
    Assert.IsNotNull(productos);
    Assert.IsTrue(productos.Count > 0);
}
```

RegistrarFactura_ShouldReturnTrue_WhenRegistrationIsSuccessful**Descripción:**

Esta prueba unitaria verifica que el método RegistrarFactura() de la clase csFactura funcione correctamente al intentar registrar una nueva factura con los detalles y parámetros especificados. La prueba espera que el método retorne true cuando el registro de la factura sea exitoso.

Precondiciones:

- La clase csFactura está implementada y accesible.
- Se han creado instancias de csFactura.FacturaDetalle para representar los detalles de la factura.
- Se ha creado una lista de detalles de factura (detalles) que contiene al menos dos elementos.
- Se ha creado una instancia de csFactura llamada factura.

- Se ha definido una fecha actual (fecha).
- Se han especificado IDs válidos de cliente (idCliente) y empleado (idEmpleado) en la base de datos.

Pasos:

- Arrange (Preparación):
 - Se han creado dos instancias de csFactura.FacturaDetalle (detalle1 y detalle2) con diferentes IDs de producto, precios, cantidades y valores de IVA.
 - Se ha inicializado una lista detalles que contiene los detalles de factura creados anteriormente.
 - Se ha creado una instancia de csFactura llamada factura.
 - Se ha definido la fecha actual en la variable fecha.
 - Se han especificado IDs válidos de cliente (idCliente) y empleado (idEmpleado).
- Act (Acción):

Llamar al método RegistrarFactura(idCliente, fecha, detalles, idEmpleado) en la instancia factura para intentar registrar la factura con los detalles proporcionados.

- Assert (Afirmación):

Verificar que la variable registrada sea true utilizando Assert.IsTrue(registrado).

Resultado esperado:

Se espera que el método RegistrarFactura() retorne true, indicando que el registro de la factura fue exitoso con los parámetros y detalles proporcionados.

```

Resumen de los detalles de la prueba
✓ RegistrarFactura_ShouldReturnTrue_WhenRegistrationIsSuccessful
  Origen: UnitTest1.cs línea 224
  Duración: 126 ms
  Salida estándar:
    Conexión abierta.
    Conexión cerrada.
  
```

Código:

```

public void RegistrarFactura_ShouldReturnTrue_WhenRegistrationIsSuccessful()
{
    // Arrange
    csFactura.FacturaDetalle detalle1 = new csFactura.FacturaDetalle
    {
        ID_Produc = 1,
        Precio = 10.5m,
        Cantidad = 2,
        IVA = 0.12m
    };

    csFactura.FacturaDetalle detalle2 = new csFactura.FacturaDetalle
    {
        ID_Produc = 2,
        Precio = 15.75m,
        Cantidad = 1,
        IVA = 0.12m
    };

    List<csFactura.FacturaDetalle> detalles = new List<csFactura.FacturaDetalle>();
    detalles.Add(detalle1);
    detalles.Add(detalle2);

    csFactura factura = new csFactura();
    DateTime fecha = DateTime.Now;
    long idCliente = 1; // ID de cliente válido en tu base de datos
    long idEmpleado = 1; // ID de empleado válido en tu base de datos

    // Act
    bool registrado = factura.RegistrarFactura(idPersona: idCliente, fecha, detalles, idEmpleado);

    // Assert
    Assert.IsTrue(registrado);
}

```

ObtenerUltimoIDFactura_ShouldReturnValidID

Descripción:

Esta prueba unitaria verifica que el método ObtenerUltimoIDFactura() de la clase csFactura funcione correctamente al obtener el último ID válido de factura registrado en el sistema. La prueba espera que el método retorne un ID mayor que 0, indicando que se ha obtenido correctamente el último ID de factura.

Precondiciones:

- La clase csFactura está implementada y accesible.
- Se ha creado una instancia de csFactura llamada factura.

Pasos:

- Arrange (Preparación):
Se ha creado una instancia de csFactura llamada factura.
- Act (Acción):

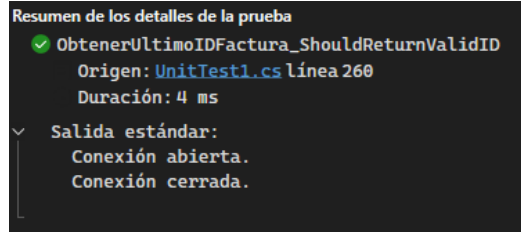
Llamar al método ObtenerUltimoIDFactura() en la instancia factura para obtener el último ID de factura registrado en el sistema.

- Assert (Afirmación):

Verificar que el valor de ultimoID sea mayor que 0 utilizando
Assert.IsTrue(ultimoID > 0).

Resultado esperado:

Se espera que el método ObtenerUltimoIDFactura() retorne un ID válido (mayor que 0), indicando que se ha obtenido correctamente el último ID de factura registrado en el sistema.



Código:

```
public void ObtenerUltimoIDFactura_ShouldReturnValidID()
{
    // Arrange
    csFactura factura = new csFactura();

    // Act
    long ultimoID = factura.ObtenerUltimoIDFactura();

    // Assert
    Assert.IsTrue(ultimoID > 0);
}
```

FechaFacturaCliente_ShouldReturnListOfDates

Descripción:

Esta prueba unitaria verifica que el método estático FechaFacturaCliente() de la clase csFactura funcione correctamente al obtener las fechas de factura asociadas a un cliente específico en el sistema. La prueba espera que el método devuelva una lista no nula y que contenga al menos una fecha de factura para el cliente especificado.

Precondiciones:

- La clase csFactura está implementada y accesible.
- Existe al menos una factura asociada al cliente con el ID especificado (idCliente).

Pasos:

- Arrange (Preparación):

Se ha definido el ID de cliente válido (idCliente) en la base de datos.

- Act (Acción):

Llamar al método estático FechaFacturaCliente(idCliente) de la clase csFactura para obtener las fechas de factura asociadas al cliente especificado.

- Assert (Afirmación):

Verificar que la variable fechasFacturas no sea null utilizando
Assert.IsNotNull(fechasFacturas).

Verificar que la lista fechasFacturas contenga al menos un elemento utilizando
Assert.IsTrue(fechasFacturas.Count > 0).

Resultado esperado:

Se espera que el método FechaFacturaCliente(idCliente) retorne una lista no nula que contenga al menos una fecha de factura asociada al cliente con el ID especificado, representando las fechas de factura encontradas en el sistema para ese cliente.

```
Resumen de los detalles de la prueba
✓ FechaFacturaCliente_ShouldReturnListOfDates
  Origen: UnitTest1.cs línea 273
  Duración: 6 ms
✓ Salida estándar:
  Conexión abierta.
  Conexión cerrada.
```

Código:

```
public void FechaFacturaCliente_ShouldReturnListOfDates()
{
    // Arrange
    long idCliente = 1; // ID de cliente válido en tu base de datos

    // Act
    List<csFactura> fechasFacturas = csFactura.FechaFacturaCliente(idCliente);

    // Assert
    Assert.IsNotNull(fechasFacturas);
    Assert.IsTrue(fechasFacturas.Count > 0);
}
```

Métricas del software v3

Análisis e interpretación de las métricas del software (Visual Studio 2022)

Resultados de métricas del código							
Filtro: Ninguno							
Jerarquía	Índice de mantenimiento	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable	
SGFactuacion (Debug)	78	4,177	7	312	33,329	12,842	
SGFactuacion.Formularios	57	186	7	86	2,931	1,502	
SGFactuacion.BDFacturaDataSetTableAdapters	64	1,278	3	89	7,690	3,044	
SGFactuacion.Clases	65	125	2	25	761	384	
SGFactuacion.DataSets	84	2,346	7	245	19,302	7,224	
SGFactuacion.Properties	86	333	4	70	2,527	758	
	86	8	3	16	118	10	

1. Índice de Mantenimiento (78)

Un índice de mantenimiento alto sugiere que el código en general es bien estructurado y documentado. Sin embargo, es esencial revisar cada clase y formulario por separado:

- **Clases como csConexion.cs y csProducto.cs:** Si estas clases tienen un alto índice de mantenimiento, indica que la lógica de conexión a la base de datos y la gestión de productos están bien encapsuladas y documentadas.
- **Formularios como FormEditorC.cs y FormListarC.cs:** Un alto índice aquí sugeriría que los formularios para editar y listar clientes son relativamente sencillos de modificar o extender.

2. Complejidad Ciclomática (4177)

Esta métrica indica que hay muchos caminos de ejecución posibles, lo cual es típico en aplicaciones con múltiples funcionalidades y flujos de usuario complejos.

- **FormFacturar.cs:** Podría tener múltiples decisiones y ciclos, gestionando diferentes casos de facturación y cálculos.
- **FrmRegistrar.cs y FrmReporte.cs:** Estos formularios también podrían contribuir a la complejidad, especialmente si manejan múltiples tipos de entrada y formatos de reporte.

3. Profundidad de Herencia (7)

Contexto en SGFactuacion: Indica una jerarquía de clases profunda, que puede ser tanto una fuente de reutilización eficiente como un potencial problema de complejidad.

- **Base de Formularios:** Si existe una clase base de formulario de la que heredan muchos otros formularios (como **FormBase.cs**), esto podría ser un indicativo de por qué la profundidad de herencia es alta.

4. Acoplamiento de Clases (312)

Un alto acoplamiento sugiere que muchas clases dependen unas de otras, lo que puede complicar las pruebas y el mantenimiento.

- **Interdependencias:** Clases como **csPersona** y **csFactura** pueden estar altamente interrelacionadas con otras clases que gestionan la lógica de negocio, lo cual podría complicar cambios en las políticas de negocio o en la estructura de datos.

5. Líneas de Código Fuente (33,239) y Líneas de Código Ejecutable (12,842)

El volumen de código en ambos casos es considerable, indicando un proyecto extenso con muchas funcionalidades.

- **Formularios y Clases con Muchas Líneas:** Formularios como **FrmReporteProducto.cs** y **FrmReporteCliente.cs** pueden contener mucha lógica para generar reportes, lo cual podría estar contribuyendo al gran número de líneas de código.

Recomendaciones Específicas

- **Revisión y Refactorización de Código:** Dado el alto número en complejidad ciclomática y acoplamiento, considera revisar y potencialmente refactorizar los formularios y clases con alta complejidad y dependencias.
- **Optimización de Herencia:** Revisa la jerarquía de herencia para simplificarla si es posible, eliminando niveles innecesarios que no añaden valor significativo al diseño.
- **Documentación y Pruebas:** Asegúrate de que todas las clases y formularios estén adecuadamente documentados y cubiertos con pruebas unitarias y de integración para manejar la complejidad inherente y el alto acoplamiento.