



UNIVERSIDAD DON BOSCO

Facultad de Ingeniería

Escuela de Computación

Manual del Programador – Gestión de Inventarios

Desarrollo de Software Empresarial G02T

Presentado por:

Martínez Huezo, Lenin Steven MH180095 DES104

Marroquín Echegoyen, Erika Alondra ME180102 DES104

Diaz Reyes, Jonathan Omar DR160475 DSE104

Ing. Emerson Cartagena

Soyapango 08 de octubre del 2024.

Contenido

Introducción	3
Objetivo.....	3
Requerimiento del sistema	3
Tipo de arquitectura.....	4
Conexión a base de datos	5
Interfaz de usuario	6
Pruebas automatizadas	9
Controlador: Usuarios	9
Controlador: Categorías	19
Controlador: Productos	27
Controlador: Compras.....	36
Controlador: Ventas	45
Controlador: Proveedores.....	57
Controlador: Empleado	65
Pruebas manuales.....	73
Controlador: Usuarios	73
Controlador: Categorías	82
Controlador: Proveedores.....	88
Controlador: Empleados	94
Controlador: Productos	100
Controlador: Compras.....	106
Controlador: Ventas	111
Buenas practicas	113
Seguridad	115
Rendimiento.....	117
Escalabilidad.....	117

Introducción

La gestión de inventarios es crucial para el correcto funcionamiento de un negocio de rubro comercial transaccional, por tal motivo, el presente documento muestra los puntos más importantes sobre las tecnologías y el desarrollo de un proyecto con dicha finalidad; en el presente documento se presentan aspectos fundamentales que servirán como guía básica e introductoria para los nuevos integrantes del grupo de trabajo, así como tener la función de documentación resumen de cara al negocio que lo implementara.

Objetivo

El proyectoDES_Empresa, tiene por objetivo facilitar el seguimiento y control en los productos disponibles dentro de la empresa, registrando compras y ventas las cuales nutren el flujo de datos, permitiendo una actualización en tiempo real de la cantidad disponible de productos, esto se traduce en un mejor rendimiento de trabajo y mayor aceptación de los clientes al recibir una respuesta inmediata.

Requerimiento del sistema

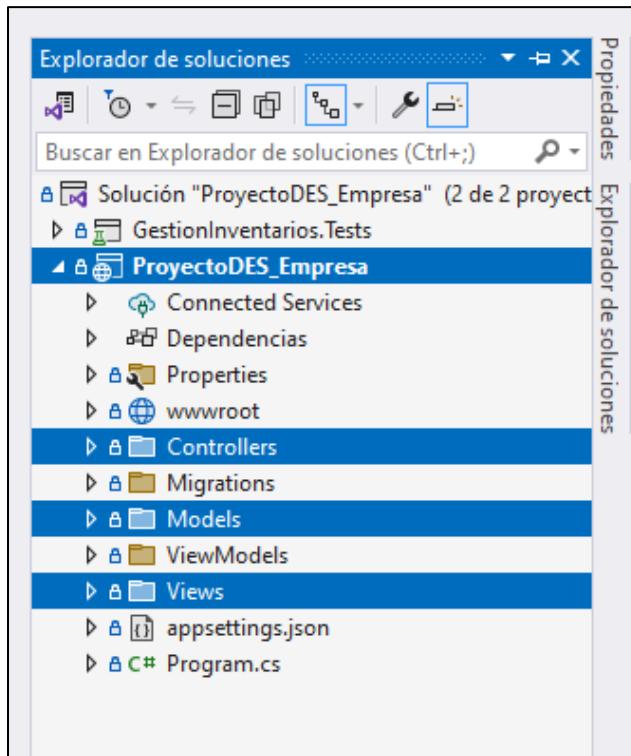
- Requiere un procesador y un sistema operativo de 64 bits
- SO: Windows 10 de 64 bits
- Memoria: 4 GB de RAM
- Almacenamiento: 1 GB de espacio disponible
- Red: Conexión de banda ancha a Internet
- Programas requeridos: SQL server 2022

Tipo de arquitectura

La selección de la arquitectura MVC, radica en la clara separación de las responsabilidades, así tambien, la elevada implementación en el mercado permite una adaptación rápida y eficiente al sistema de trabajo; la división de la aplicación en tres capas (Modelos – Vistas- Controladores) promueve el mantenimiento y la escalabilidad del proyecto.

La separación del proyecto en capas, permite que diferentes equipos de trabajo efectúen cambios en paralelo, reduciendo el tiempo de desarrollo y aumentando la calidad de trabajo para cada capa individual puesto que el enfoque está distribuido, otra ventaja es la detección y corrección de errores, puesto que estos son aislados a una capa única, y estos a su vez a un componente específico teniendo una granularidad de funcionamiento más detallada, por último la reutilización de código es una gran ventaja puesto que facilita la escalabilidad del proyecto, pudiendo incluir nuevas funcionalidades que comparten dichos elementos.

La implementación de la arquitectura MVC, se lleva a cabo bajo un proyecto “Aplicación web de ASP.NET Core (Modelo – vista - Controlador)” proporcionado por Visual Studio.



Conexión a base de datos

Para la conexión a la base de datos, considerando que se utiliza un proyecto “Aplicación web de ASP.NET Core (Modelo – vista - Controlador)” proporcionado por Visual Studio; se realiza mediante los archivos “appsettings.json” y “Program.cs”, ambos proporcionados por la plantilla del proyecto y configurados para esta finalidad; finalmente se cuenta con una clase llamada “EmpresaDBContext.cs” la cual, se encarga de gestionar la conexión con la base de datos y el mapeo de los modelos.

Cabe destacar la instalación de los paquetes NuGet:

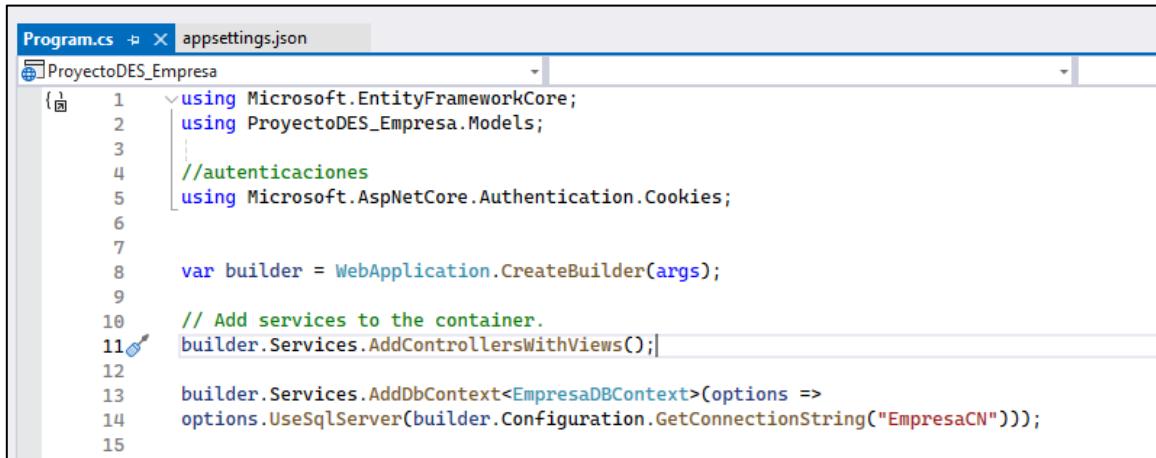
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools



```

Program.cs      appsettings.json ✎ X
Esquema: https://json.schemastore.org/appsettings.json
1  ↴ ↵  {
2    ↴ ↵   "ConnectionStrings": {
3      ↴ ↵     "EmpresaCN": "Server=(localdb)\\MSSQLLocalDB;Database=BD_EMPRESA;Trusted_Connection=True;"
4    },
5    ↴ ↵   "Logging": {
6      ↴ ↵     "LogLevel": {
7        ↴ ↵       "Default": "Information",
8        ↴ ↵       "Microsoft.AspNetCore": "Warning"
9      }
10   },
11   ↴ ↵   "AllowedHosts": "*"
12 }
13

```



```

Program.cs ✎ X appsettings.json
ProyectoDES_Empresa
1  ↴ ↵  using Microsoft.EntityFrameworkCore;
2  ↴ ↵  using ProyectoDES_Empresa.Models;
3  ↴ ↵  |
4  ↴ ↵  //autenticaciones
5  ↴ ↵  using Microsoft.AspNetCore.Authentication.Cookies;
6  ↴ ↵
7  ↴ ↵
8  ↴ ↵  var builder = WebApplication.CreateBuilder(args);
9  ↴ ↵
10 ↴ ↵  // Add services to the container.
11 ↴ ↵  builder.Services.AddControllersWithViews();
12 ↴ ↵
13  ↴ ↵  builder.Services.AddDbContext<EmpresaDBContext>(options =>
14  ↴ ↵    options.UseSqlServer(builder.Configuration.GetConnectionString("EmpresaCN")));
15

```

```

EmpresaDBContext.cs* + X Program.cs appsettings.json
ProyectoDES_Empresa
1 1 using Microsoft.EntityFrameworkCore;
2 2 using ProyectoDES_Empresa.Models.Seeds;
3
4 3 namespace ProyectoDES_Empresa.Models
5 {
6     22 referencias
7     public class EmpresaDBContext : DbContext
8     {
9         1 referencia
10        public EmpresaDBContext(DbContextOptions options) : base(options)
11        {
12        }
13
14        39 referencias
15        public DbSet<Categoria> Categorias { get; set; }
16        20 referencias
17        public DbSet<Empleado> Empleados { get; set; }
18        22 referencias
19        public DbSet<Proveedor> Proveedores { get; set; }
20
21        60 referencias
22        public DbSet<Producto> Productos { get; set; }
23        14 referencias
24        public DbSet<Compra> Compras { get; set; }
25        14 referencias
26        public DbSet<Venta> Ventas { get; set; }
27
28        21 referencias
29        public DbSet<Usuario> Usuarios { get; set; }
30        0 referencias
31        public DbSet<Rol> Roles { get; set; }

```

Interfaz de usuario

La interfaz de usuario presenta un ambiente minimalista con iconografía y botones explícitos fáciles de comprender, la paleta de colores se limita a 4 tonos principales:

- **Blanco:** Siendo el color principal, se utiliza como la base donde se asientan los elementos, proporciona una interfaz limpia y brinda buen contraste para leer textos en el área de trabajo.
- **Gris:** Dado que es un color neutro, se utiliza para las zonas y funciones no críticas, tales como el encabezado y el pie de página, así también en los botones de información, posee un alto contraste con el fondo.
- **Verde:** Se utiliza para los botones de interacción directa con el usuario, tales como “Aregar”, “Buscar” o “Editar”, el tono verde permite una identificación rápida de las acciones más recurrentes para el usuario, así también el progreso adecuado de una acción.
- **Rojo:** Al ser un color llamativo y comúnmente utilizado como alerta, es usado en la acción “Eliminar” así como las alertas por algún error, lo cual indica un progreso inadecuado en el flujo de trabajo o una acción no reversible.



REGISTRO DE COMPRAS							
dd --- aaaa	dd --- aaaa	Buscar categoría...	Buscar producto...	Buscar descripción...	<input type="button" value=""/>	<input type="button" value=""/>	<input type="button" value=""/>
Fecha de compra	Proveedor	Categoría	Nombre Producto	Descripción	Unidades	Costo(\$)	Acciones
1 ago. 2024	Carpinteria Don Mario	Muebles de madera	Closet	1.00 m Cafe	25	124.99	
22 ago. 2024	Distribuidora El Sueño	Camas	Memory Foam	1.40 m	50	154.99	
14 jul. 2024	Almacenes La Moderna	Electrodomésticos	Ventilador	Mini Metalico	100	14.99	
14 sep. 2024	Almacenes La Moderna	Infantil	Coche para bebe	Rosado	20	44.99	
4 ago. 2024	Almacenes La Moderna	Infantil	Cuna	Blanca	10	104.99	
4 mar. 2024	Carpinteria Don Mario	Muebles de madera	Chinero	1.40 m Natural	10	164.99	

MENÚ USUARIOS CONFIGURACIÓN CERRAR SESIÓN

INFORMACIÓN DE LA COMPRA

Ver Lista de Compras / Información de la Compra

COMPRA

Fecha de compra	4 ago. 2024
Proveedor	Almacenes La Moderna
Categoría	Infantil
Nombre de Producto	Cuna
Descripción	Blanca
Unidades compradas	10

EDITAR **REGRESAR**

MENÚ USUARIOS CONFIGURACIÓN CERRAR SESIÓN

ELIMINAR COMPRA

¿Estás seguro que deseas eliminar esta compra?

Fecha de compra	4 ago. 2024
Proveedor	Almacenes La Moderna
Categoría	Infantil
Nombre de Producto	Cuna
Descripción	Blanca
Unidades compradas	10

ELIMINAR **CANCELAR**

MENÚ USUARIOS CONFIGURACIÓN CERRAR SESIÓN

REGISTRAR VENTAS

Ver Ventas Registradas / Ingresar Ventas

Ingrese la información de la venta

No se puede registrar la venta: No hay suficientes unidades del producto. ×

Fecha de venta	06 oct. 2024
Nombre de categoría	Muebles de madera
Nombre de producto	Closest
Descripción de producto	1.00 m Cafe
Unidades vendidas	500

Pruebas automatizadas

Controlador: Usuarios

Nombre del Caso de Prueba

Registrar un usuario válido

Descripción

Este caso de prueba verifica que un usuario puede registrarse correctamente en el sistema cuando proporciona datos válidos.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico válido.
2. La contraseña y la confirmación de la contraseña deben coincidir.
3. El usuario debe ser registrado en la base de datos.
4. La contraseña debe ser almacenada de manera segura (encriptada).
5. El sistema debe mostrar un mensaje de “Registro Exitoso”

```
//Registrar un usuario Válido
[Fact]
0 referencias
public async Task Post_Registrar_Usuario_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);
    var nuevoUsuarioVM = new UsuarioVM
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = "Clave123",
        ConfirmarClaveUsuario = "Clave123"
    };

    var result = await controller.Registrar(nuevoUsuarioVM);

    var viewResult = Assert.IsType<ViewResult>(result);
    Assert.Equal("Registro Exitoso!", controller.ViewBag.RegistroExitoso);

    var usuarioEnDb = context.Usuarios.FirstOrDefault(u => u.CorreoUsuario == "nuevo@correo.com");
    Assert.NotNull(usuarioEnDb);
    Assert.True(BCrypt.Net.BCrypt.Verify("Clave123", usuarioEnDb.ClaveUsuario));
}
```

Ejecución de la prueba:

Ejecutar
 Depurar

Resumen de los detalles de la prueba

- ✔ GestiónInventarios.Tests.UsuariosControllerTests.Post_Registrar_Usuario_CuandoEsValido
- 📄 Origen: [UsuariosControllerTests.cs](#) línea 17
- ⌚ Duración: 347 ms

Nombre del Caso de Prueba

No registrar un usuario cuando las contraseñas no coinciden

Descripción

Este caso de prueba verifica que el sistema no permite registrar un usuario cuando las contraseñas proporcionadas no coinciden.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico válido.
2. La contraseña y la confirmación de la contraseña deben coincidir.
3. Si las contraseñas no coinciden, el usuario no debe ser registrado en la base de datos.
4. El sistema debe mostrar un mensaje de error indicando que las contraseñas no coinciden.

```
//No se registra un usuario cuando las contraseñas no coinciden
[Fact]
0 referencias
public async Task Post_Registrar_Usuario_CuandoClavesNoCoincidan()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);
    var nuevoUsuarioVM = new UsuarioVM
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = "Clave123",
        ConfirmarClaveUsuario = "NoClave123"
    };

    var result = await controller.Registrar(nuevoUsuarioVM);

    var viewResult = Assert.IsType<ViewResult>(result);
    Assert.Equal("Error al registrar usuario: Las contraseñas no coinciden", controller.ViewBag.ErrorClave);
}
```

Ejecución de la prueba:

Ejecutar | Depurar

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.UsuariosControllerTests.Post_Registrar_Usuario_CuandoClavesNoCoincidan
 - ▀ Origen: [UsuariosControllerTests.cs](#) línea 40
 - ⌚ Duración: 1.6 s

Nombre del Caso de Prueba

No registrar un usuario cuando el correo ya está registrado en la base de datos

Descripción

Este caso de prueba verifica que el sistema no permite registrar un usuario cuando el correo electrónico proporcionado ya está registrado en la base de datos.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico válido.
2. Si el correo electrónico ya está registrado, el usuario no debe ser registrado nuevamente.
3. El sistema debe mostrar un mensaje de error indicando que el correo ya está registrado.

```
//No se registra porque el correo ya esta registrado en la BD
[Fact]
● 0 referencias
public async Task Post_Registrar_Usuario_CuandoCorreoYaRegistrado()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);
    var usuarioExistente = new Usuario
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = BCrypt.Net.BCrypt.HashPassword("Clave123"),
        IdRol = 2
    };
    context.Usuarios.Add(usuarioExistente);
    await context.SaveChangesAsync();

    var nuevoUsuarioVM = new UsuarioVM
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = "Clave123",
        ConfirmarClaveUsuario = "Clave123"
    };

    var result = await controller.Registrar(nuevoUsuarioVM);

    var yarResult = Assert.IsType<ViewResult>(result);
    Assert.Equal("Error: El correo brindado ya posee cuenta registrada.", controller.ViewBag.ErrorClave);
}
```

Ejecución de la prueba:

Ejecutar
 Depurar

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.UsuariosControllerTests.Post_Registrar_Usuario_CuandoCorreoYaRegistrado
- 📄 Origen: [UsuariosControllerTests.cs](#) línea 59
- 🕒 Duración: 1.1 s

Nombre del Caso de Prueba

No iniciar sesión si el usuario no existe en la base de datos

Descripción

Este caso de prueba verifica que el sistema no permite iniciar sesión a un usuario que no está registrado en la base de datos.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico y una contraseña.
2. Si el correo electrónico no está registrado en la base de datos, el usuario no debe poder iniciar sesión.
3. El sistema debe mostrar un mensaje de error indicando que las credenciales son incorrectas.

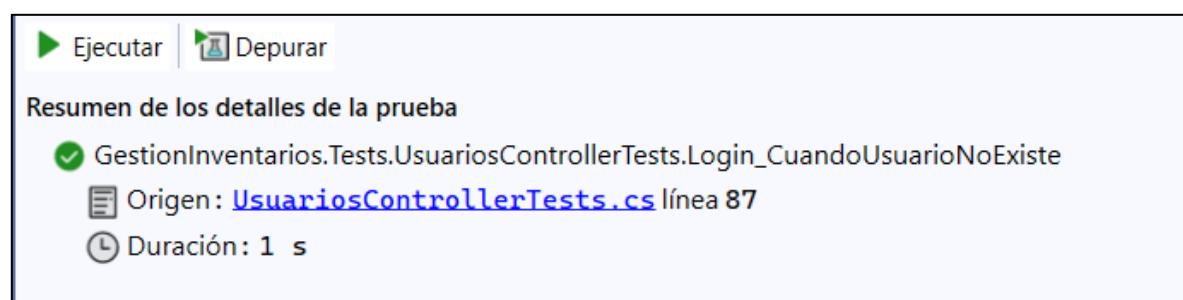
```
//No inicia sesión si el usuario no existe en la base de datos
[Fact]
● | 0 referencias
public async Task Login_CuandoUsuarioNoExiste()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);

    var loginVM = new LoginVM
    {
        CorreoUsuario = "noregistrado@correo.com",
        ClaveUsuario = "Clave123"
    };

    var result = await controller.Login(loginVM);

    var viewResult = Assert.IsType<ViewResult>(result);
    Assert.Equal("Error: Las credenciales son incorrectas", controller.ViewBag.MensajeError);
}
```

Ejecución de la prueba:



Nombre del Caso de Prueba

No iniciar sesión cuando la contraseña es incorrecta

Descripción

Este caso de prueba verifica que el sistema no permite iniciar sesión a un usuario cuando la contraseña proporcionada es incorrecta.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico y una contraseña.
2. Si la contraseña no coincide con la almacenada en la base de datos, el usuario no debe poder iniciar sesión.
3. El sistema debe mostrar un mensaje de error indicando que las credenciales son incorrectas.

```
//No inicia sesión cuando la contraseña es incorrecta
[Fact]
● | 0 referencias
public async Task Login_CuandoClaveEsIncorrecta()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);
    var usuario = new Usuario
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = BCrypt.Net.BCrypt.HashPassword("Clave123"),
        IdRol = 2,
    };
    context.Usuarios.Add(usuario);
    await context.SaveChangesAsync();

    //Verifica credenciales
    var loginVM = new LoginVM
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = "NoClave123"
    };

    var result = await controller.Login(loginVM);

    var viewResult = Assert.IsType<ViewResult>(result);
    Assert.Equal("Error: Las credenciales son incorrectas", controller.ViewBag.MensajeError);
}
```

Ejecución de prueba:

The screenshot shows a test execution interface with the following details:

- Ejecutar | Depurar**: Buttons for running and debugging the test.
- Resumen de los detalles de la prueba**: Summary of test details.
- GestionInventarios.Tests.UsuariosControllerTests.Login_CuandoClaveEsIncorrecta**: The specific test case name, preceded by a green checkmark indicating it passed.
- Origen: UsuariosControllerTests.cs línea 106**: The source code location of the test.
- Duración: 327 ms**: The execution duration of the test.

Nombre del Caso de Prueba

Redirigir correctamente a la vista de edición cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema redirige correctamente a la vista de edición y carga los datos del usuario cuando se proporciona un ID válido.

Criterios de Aceptación

1. El usuario debe existir en la base de datos.
2. El ID proporcionado debe ser válido y corresponder a un usuario existente.
3. El sistema debe redirigir a la vista de edición y cargar los datos del usuario en el modelo.

```
//Redirige correctamente a vista editar cuando el Id es valido
[Fact]
● | 0 referencias
public async Task Editar_RetornaVistaConUsuario_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);

    var usuario = new Usuario
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = BCrypt.Net.BCrypt.HashPassword("Clave123"),
        IdRol = 2,
    };
    context.Usuarios.Add(usuario);
    await context.SaveChangesAsync();

    var result = await controller.Editar(usuario.ID);

    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsType<EditarUsuarioVM>(viewResult.Model);
    Assert.Equal(usuario.CorreoUsuario, model.CorreoUsuario);
}
```

Ejecución de prueba:

Ejecutar | Depurar

Resumen de los detalles de la prueba

GestiónInventarios.Tests.UsuariosControllerTests.Editar_RetornaVistaConUsuario_CuandoldEsValido
🔗 Origen: [UsuariosControllerTests.cs](#) línea 134
🕒 Duración: 130 ms

Nombre del Caso de Prueba

No retornar a la vista de edición cuando el ID es inválido

Descripción

Este caso de prueba verifica que el sistema no redirige a la vista de edición y retorna un resultado de “NotFound” cuando se proporciona un ID inválido.

Criterios de Aceptación

1. El ID proporcionado no debe corresponder a ningún usuario en la base de datos.
2. El sistema debe retornar un resultado de “NotFound”.

```
//No retorna a vista editar puesto que el Id es invalido
[Fact]
✓ | 0 referencias
public async Task Editar_ReturnsNotFound_WhenIdIsInvalid()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);

    var result = await controller.Editar(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:

```
▶ Ejecutar | ⚙ Depurar

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.UsuariosControllerTests.Editar_ReturnsNotFound_WhenIdIsInvalid
    ⌂ Origen: UsuariosControllerTests.cs línea 157
    ⏱ Duración: 1.1 s
```

Nombre del Caso de Prueba

Editar la información del usuario correctamente cuando el modelo es válido

Descripción

Este caso de prueba verifica que el sistema permite editar la información de un usuario correctamente cuando se proporciona un modelo válido.

Criterios de Aceptación

1. El usuario debe existir en la base de datos.
2. El modelo de edición debe ser válido y contener la información correcta.
3. El sistema debe actualizar la información del usuario en la base de datos.
4. El sistema debe redirigir a la acción Index después de la edición exitosa.

```
//Edita la informacion del usuario correctamente cuando el modelo es valido
[Fact]
● 10 referencias
public async Task Editar_Usuario_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);

    //Crea un nuevo usuario
    var usuario = new Usuario
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = BCrypt.Net.BCrypt.HashPassword("Clave123"),
        IdRol = 2,
    };

    context.Usuarios.Add(usuario);
    await context.SaveChangesAsync();

    //Nueva informacion del usuario
    var editarUsuario = new EditarUsuarioVM
    {
        ID = usuario.ID,
        CorreoUsuario = "nuevo@correo.com",
        ClaveAntigua = "Clave123",
        NuevaClaveUsuario = "NuevaClave123",
        ConfirmarNuevaClaveUsuario = "NuevaClave123"
    };

    var result = await controller.Editar(editarUsuario);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var usuarioEnDb = context.Usuarios.FirstOrDefault(p => p.ID == usuario.ID);
    Assert.NotNull(usuarioEnDb);
    Assert.Equal("nuevo@correo.com", usuarioEnDb.CorreoUsuario);
    Assert.True(BCrypt.Net.BCrypt.Verify("NuevaClave123", usuarioEnDb.ClaveUsuario));
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.UsuariosControllerTests.Editar_Usuario_CuandoEsValido
⠀ Origen: UsuariosControllerTests.cs línea 169
⠀ Duración: 518 ms

Nombre del Caso de Prueba

No actualizar el registro cuando la información no es válida

Descripción

Este caso de prueba verifica que el sistema no permite actualizar la información de un usuario cuando el modelo de edición contiene datos inválidos, como contraseñas que no coinciden.

Criterios de Aceptación

1. El usuario debe existir en la base de datos.
2. El modelo de edición debe ser inválido (por ejemplo, las nuevas contraseñas no coinciden, la contraseña antigua no es correcta o el correo ya está registrado).
3. El sistema debe retornar a la vista de edición con un mensaje de error.
4. La información del usuario en la base de datos no debe ser actualizada.

```
//No actualiza el resgitro puesto que la informacion no es valida
[Fact]
➊ 0 referencias
public async Task Editar_Post_Usuario_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);

    //Crea un nuevo usuario
    var usuario = new Usuario
    {
        CorreoUsuario = "nuevo@correo.com",
        ClaveUsuario = BCrypt.Net.BCrypt.HashPassword("Clave123"),
        IdRol = 2,
    };
    context.Usuarios.Add(usuario);
    await context.SaveChangesAsync();

    //Edita el usuario previamente creado pero las claves las nuevas claves no coinciden
    var editarUsuario = new EditarUsuarioVM
    {
        ID = usuario.ID,
        CorreoUsuario = "nuevo@correo.com",
        ClaveAntigua = "Clave123",
        NuevaClaveUsuario = "NuevaClave123",
        ConfirmarNuevaClaveUsuario = "NoNuevaClave123"
    };

    // Se genera un error en el modelo: las claves no coinciden
    controller.ModelState.AddModelError("ConfirmarNuevaClaveUsuario", "Las nuevas contraseñas no coinciden.");
}
```

```
var result = await controller.Editar(editarUsuario);

var viewResult = Assert.IsType<ViewResult>(result);
var returnValue = Assert.IsType<EditarUsuarioVM>(viewResult.Model);
Assert.Equal(editarUsuario.CorreoUsuario, returnValue.CorreoUsuario);

// Verificar que el resgitro no se ha actualizado
var usuarioEnDb = context.Usuarios.FirstOrDefault(p => p.ID == usuario.ID);
Assert.NotNull(usuarioEnDb);
Assert.Equal("nuevo@correo.com", usuarioEnDb.CorreoUsuario);
Assert.True(BCrypt.Net.BCrypt.Verify("Clave123", usuarioEnDb.ClaveUsuario));
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ➊ GestiónInventarios.Tests.UsuariosControllerTests.Editar_Post_Usuario_CuandoNoEsValido
- ➋ Origen: [UsuariosControllerTests.cs](#) línea 208
- ➌ Duración: 273 ms

Nombre del Caso de Prueba

Eliminar un usuario cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite eliminar un usuario correctamente cuando se proporciona un ID válido.

Criterios de Aceptación

1. El usuario debe existir en la base de datos.
2. El ID proporcionado debe ser válido y corresponder a un usuario existente.
3. El sistema debe eliminar el usuario de la base de datos.
4. El sistema debe redirigir a la acción Index después de la eliminación exitosa.

```
[Fact]
➊ | 0 referencias
public async Task Eliminar_Usuario_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new UsuariosController(context);

    // Agregar un usuario
    var usuario = new Usuario
    {
        CorreoUsuario = "user@correo.com",
        ClaveUsuario = BCrypt.Net.BCrypt.HashPassword("Clave123"),
        IdRol = 2,
    };
    context.Usuarios.Add(usuario);
    await context.SaveChangesAsync();

    var result = await controller.EliminarConfirmado(usuario.ID);

    //Verifica que el usuario ya no esta en la base de datos.
    var usuarioEliminado = await context.Usuarios.FindAsync(usuario.ID);
    Assert.Null(usuarioEliminado);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);
}
```

Ejecución de prueba:

Resumen de los detalles de la prueba
➊ GestiónInventarios.Tests.UsuariosControllerTests.Eliminar_Usuario_CuandoEsValido
⠀ Origen: UsuariosControllerTests.cs línea 251
⠀ Duración: 164 ms

Controlador: Categorías

Nombre del Caso de Prueba

Crear una categoría cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva categoría correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. La categoría debe tener un nombre y una descripción válidos.
2. El sistema debe agregar la nueva categoría a la base de datos.
3. El sistema debe redirigir a la acción Index después de la creación exitosa

```
//Ingresa la categoria cuando la informacion es valida
[Fact]
● 0 referencias
public async Task Crear_Categoría_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

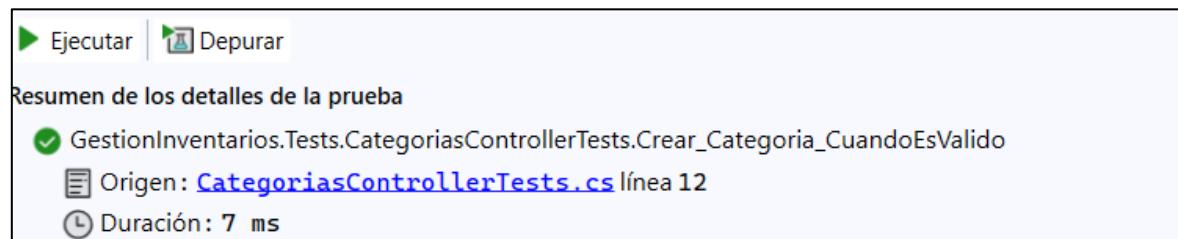
    var categoria = new Categoria
    {
        NombreCategoria = "Juego de Sala",
        DescripcionCategoria = "Juegos de sala de diversos modelos y tamaños"
    };

    var result = await controller.Create(categoria);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    var categoriaCreada = await context.Categorias.FirstOrDefaultAsync(c => c.NombreCategoria == "Juego de Sala");
    Assert.NotNull(categoriaCreada);
    Assert.Equal("Juego de Sala", categoriaCreada.NombreCategoria);
}
```

Ejecución de la prueba:



Nombre del Caso de Prueba

No crear una categoría cuando el modelo no es válido

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva categoría cuando el modelo contiene datos inválidos, como un nombre de categoría o descripción vacío.

Criterios de Aceptación

1. La categoría debe tener un nombre y una descripción válidos.
2. Si el nombre de la categoría está vacío, el sistema debe retornar a la vista de creación con un mensaje de error.
3. La nueva categoría no debe ser añadida a la base de datos.

```
//No ingresa la categoria cuando el modelo no es valido
[Fact]
● 0 referencias
public async Task Crear_Categoría_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

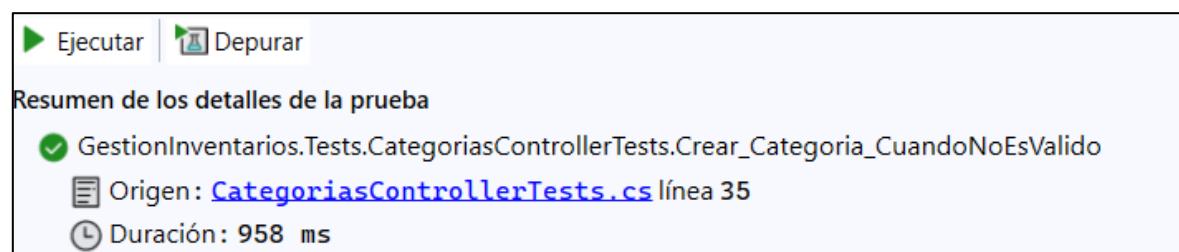
    var categoria = new Categoria
    {
        NombreCategoria = "",
        DescripcionCategoria = "Juegos de sala de diversos modelos y tamaños"
    };

    // Error de validación en el Nombre de Categoria
    controller.ModelState.AddModelError("NombreCategoria", "El nombre es requerido.");

    var result = await controller.Create(categoria);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Categoria>(viewResult.Model);
    Assert.Equal(categoria.DescripcionCategoria, returnValue.DescripcionCategoria);
}
```

Ejecución de la prueba:



Nombre del Caso de Prueba

Editar una categoría cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar una categoría correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. La categoría debe existir en la base de datos.
2. El modelo de edición debe ser válido y contener la información correcta.
3. El sistema debe actualizar la información de la categoría en la base de datos.
4. El sistema debe redirigir a la acción Index después de la edición exitosa.

```
//Edita la categoria cuando es valida la informacion
[Fact]
● 0 referencias
public async Task Editar_Post_Categoría_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

    //Crea una nueva categoria
    var categoria = new Categoria
    {
        NombreCategoria = "Juego de Sala",
        DescripcionCategoria = "Juegos de sala de diversos modelos y tamaños"
    };
    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    //Actualizacion de parametros
    categoria.NombreCategoria = "Producto Metalico";
    categoria.DescripcionCategoria = "Diversos productos metalicos: estantes, comedores, percheros./";

    var result = await controller.Edit(categoria.ID, categoria);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    //Verifica que la informacion ha sido actualizada
    var categoriaActualizada = context.Categorias.FirstOrDefault(p => p.ID == categoria.ID);
    Assert.NotNull(categoriaActualizada);
    Assert.Equal("Producto Metalico", categoriaActualizada.NombreCategoria);
    Assert.Equal("Diversos productos metalicos: estantes, comedores, percheros.", categoriaActualizada.DescripcionCategoria);
}
```

Ejecución de prueba:

Ejecutar | Depurar

Resumen de los detalles de la prueba

✓ GestiónInventarios.Tests.CategoriasControllerTests.Editar_Post_Categoría_CuandoEsValido

📄 Origen: [CategoriasControllerTests.cs](#) línea 58

⌚ Duración: 8 ms

Nombre del Caso de Prueba

No actualizar la categoría cuando la información no es válida

Descripción

Este caso de prueba verifica que el sistema no permite actualizar una categoría cuando el modelo contiene datos inválidos, como un nombre de categoría vacío.

Criterios de Aceptación

1. La categoría debe existir en la base de datos.
2. El modelo de edición debe ser inválido (por ejemplo, el nombre de la categoría está vacío).
3. El sistema debe retornar a la vista de edición con un mensaje de error.
4. La información de la categoría en la base de datos no debe ser actualizada.

```
//No Actualiza la categoria cuando la informacion no es valida
[Fact]
● | 0 referencias
public async Task Editar_Post_Categoría_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

    //Crea una nueva categoria
    var categoria = new Categoria
    {
        NombreCategoria = "Juego de Sala",
        DescripcionCategoria = "Juegos de sala de diversos modelos y tamaños"
    };
    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Error de validación
    controller.ModelState.AddModelError("NombreCategoria", "El nombre es requerido.");

    var result = await controller.Edit(categoria.ID, categoria);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Categoria>(viewResult.Model);
    Assert.Equal("Juego de Sala", returnValue.NombreCategoria);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.CategoríasControllerTests.Editar_Post_Categoría_CuandoNoEsValido
- 📄 Origen: [CategoríasControllerTests.cs](#) línea 90
- ⌚ Duración: 37 ms

Nombre del Caso de Prueba

Obtener detalles de la categoría cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite obtener los detalles de una categoría correctamente cuando se proporciona un ID válido.

Criterios de Aceptación

1. La categoría debe existir en la base de datos.
2. El ID proporcionado debe ser válido y corresponder a una categoría existente.
3. El sistema debe retornar los detalles de la categoría en la vista.

```
//Obtener detalles de la categoria cuando el Id es Valido
[Fact]
➊ | 0 referencias
public async Task Details_Categoría_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

    //Crea categoria
    var categoria = new Categoria
    {
        NombreCategoria = "Juego de Sala",
        DescripcionCategoria = "Juegos de sala de diversos modelos y tamaños"
    };
    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    //Verifica que el ID existe
    var result = await controller.Details(categoria.ID);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Categoria>(viewResult.Model);
    Assert.Equal("Juego de Sala", returnValue.NombreCategoria);
}
```

Ejecución de prueba:

Ejecutar
 Depurar

Resumen de los detalles de la prueba

- ➊ GestiónInventarios.Tests.CategoríasControllerTests.Details_Categoría_CuandoidEsValido
 - ⠁ Origen: [CategoríasControllerTests.cs](#) línea 116
 - ⌚ Duración: 26 ms

Nombre del Caso de Prueba

No obtener los detalles del registro cuando el ID es inválido

Descripción

Este caso de prueba verifica que el sistema no permite obtener los detalles de una categoría cuando se proporciona un ID inválido.

Criterios de Aceptación

1. El ID proporcionado no debe corresponder a ninguna categoría en la base de datos.
2. El sistema debe retornar un resultado de “NotFound”.

```
//No obtiene los detalles del registro cuando el id es invalido
[Fact]
✔ | 0 referencias
public async Task Details_Categoría_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

    var result = await controller.Details(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de pruebas:

```
▶ Ejecutar | ⚙ Depurar

Resumen de los detalles de la prueba
✔ GestiónInventarios.Tests.CategoriasControllerTests.Details_Categoría_CuandoidNoEsValido
    ⏺ Origen: CategoriasControllerTests.cs línea 140
    ⏳ Duración: 56 ms
```

Nombre del Caso de Prueba

Eliminar la categoría cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite eliminar una categoría correctamente cuando se proporciona un ID válido.

Criterios de Aceptación

1. La categoría debe existir en la base de datos.
2. El ID proporcionado debe ser válido y corresponder a una categoría existente.
3. El sistema debe eliminar la categoría de la base de datos.
4. El sistema debe redirigir a la acción Index después de la eliminación exitosa.

```
//Elimina el registro cuando el Id es valido
[Fact]
● | 0 referencias
public async Task Eliminar_Categoría_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

    //Crea nueva categoria
    var categoria = new Categoria
    {
        NombreCategoria = "Juego de Sala",
        DescripcionCategoria = "Juegos de sala de diversos modelos y tamaños"
    };
    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    var result = await controller.DeleteConfirmed(categoria.ID);

    // Verificar que la categoría ha sido eliminada
    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var categoriaEnDb = context.Categorias.FirstOrDefault(p => p.ID == categoria.ID);
    Assert.Null(categoriaEnDb);
}
```

Ejecución de prueba:

The screenshot shows a test execution interface. At the top, there are two buttons: 'Ejecutar' (Execute) and 'Depurar' (Debug). Below these, a section titled 'Resumen de los detalles de la prueba' (Test details summary) displays the following information:

- A green checkmark icon followed by the text 'GestionInventarios.Tests.CategoriasControllerTests.Eliminar_Categoría_CuandoldEsValido'.
- An icon of a document with a list (document icon) followed by the text 'Origen: CategoriasControllerTests.cs línea 152'.
- An icon of a clock followed by the text 'Duración: 4 ms'.

Nombre del Caso de Prueba

No eliminar la categoría cuando el ID es inválido

Descripción

Este caso de prueba verifica que el sistema no permite eliminar una categoría cuando se proporciona un ID inválido.

Criterios de Aceptación

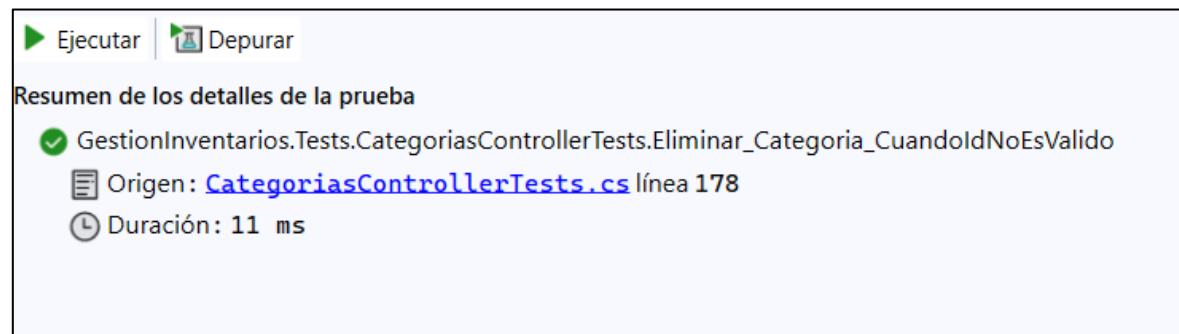
1. El ID proporcionado no debe corresponder a ninguna categoría en la base de datos.
2. El sistema debe retornar un resultado de “NotFound”.

```
//No elimina el registro cuando el Id es invalido
[Fact]
● | 0 referencias
public async Task Eliminar_Categoría_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new CategoriasController(context);

    var result = await controller.DeleteConfirmed(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:



Controlador: Productos

Nombre del Caso de Prueba

Ingresar un producto nuevo con información válida

Descripción

Este caso de prueba verifica que el sistema permite agregar un nuevo producto a la base de datos correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. El producto debe tener un nombre, descripción, unidades y costo válidos.
2. El sistema debe agregar el nuevo producto a la base de datos.
3. El sistema debe redirigir a la acción Index después de la creación exitosa.

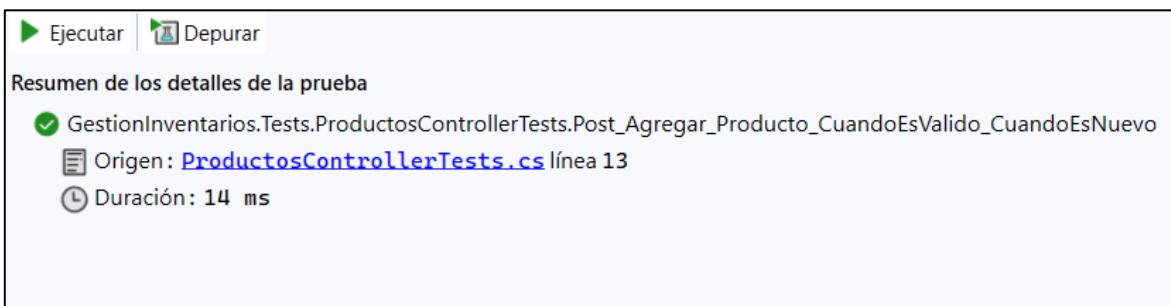
```
//Ingresar a base de datos un producto nuevo con informacion valida.
[Fact]
● 0 referencias
public async Task Post_Agregar_Producto_CuandoEsValido_CuandoEsNuevo()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);
    var nuevoProducto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Gavetero",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 124.99m
    };

    var result = await controller.Create(nuevoProducto);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var productoEnDb = context.Productos.FirstOrDefault(p => p.NombreProducto == "Gavetero" && p.IdCategoria == 1);
    Assert.NotNull(productoEnDb);
    Assert.Equal("Gavetero", productoEnDb.NombreProducto);
    Assert.Equal(10, productoEnDb.UnidadesProducto);
    Assert.Equal(124.99m, productoEnDb.CostoProducto);
}
```

Ejecución de pruebas:



Nombre del Caso de Prueba

No ingresar un producto nuevo con información no válida

Descripción

Este caso de prueba verifica que el sistema no permite agregar un nuevo producto a la base de datos cuando se proporciona información no válida.

Criterios de Aceptación

1. El producto debe tener un nombre, descripción, unidades y costo válidos.
2. Si el nombre del producto está vacío o no cumple con los requisitos, el sistema debe retornar a la vista de creación con un mensaje de error.
3. El nuevo producto no debe ser añadido a la base de datos.

```
//No Ingresar a base de datos un producto nuevo con informacion no valida.
[Fact]
● | 0 referencias
public async Task Post_Agregar_Producto_CuandoNoEsValido_CuandoEsNuevo()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);
    var nuevoProducto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 100,
        CostoProducto = 124.99m
    };

    //Envia el producto de manera invalida
    controller.ModelState.AddModelError("NombreProducto", "Required");

    var result = await controller.Create(nuevoProducto);

    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsType<Producto>(viewResult.Model);
    Assert.Equal("Closet", model.NombreProducto);
    Assert.False(controller.ModelState.IsValid);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba	
✓	GestionInventarios.Tests.ProductosControllerTests.Post_Agregar_Producto_CuandoNoEsValido_CuandoEsNuevo
📄	Origen: ProductosControllerTests.cs línea 40
⌚	Duración: 958 ms

Nombre del Caso de Prueba

Ingresar un producto que ya posee registro

Descripción

Este caso de prueba verifica que el sistema permite agregar un producto a la base de datos correctamente cuando se proporciona información válida, incluso si el producto ya existe, sumando las unidades nuevas a las existentes.

Criterios de Aceptación

1. El producto debe tener un nombre, descripción, unidades y costo válidos.
2. Si el producto ya existe en la base de datos, el sistema debe sumar las unidades nuevas a las existentes.
3. El sistema debe redirigir a la acción Index después de la creación exitosa.

```
//Ingresar a una base de datos un producto que ya posee registro
[Fact]
● 0 referencias
public async Task Post_Agregar_Producto_CuandoEsValido_CuandoYaExiste()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);
    var productoExistente = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Gavetero",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 50,
        CostoProducto = 124.99m
    };
    context.Productos.Add(productoExistente);
    await context.SaveChangesAsync();

    var nuevoProducto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Gavetero",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 100,
        CostoProducto = 124.99m
    };

    var result = await controller.Create(nuevoProducto);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var productoEnDb = context.Productos.FirstOrDefault(
        p => p.NombreProducto == "Gavetero" && p.IdCategoria == 1);
    Assert.NotNull(productoEnDb);
    Assert.Equal("Gavetero", productoEnDb.NombreProducto);
    //Valida la suma de las unidades nuevas con las existentes.
    Assert.Equal(150, productoEnDb.UnidadesProducto);
    Assert.Equal(124.99m, productoEnDb.CostoProducto);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.ProductosControllerTests.Post_Agregar_Producto_CuandoEsValido_CuandoYaExiste
- 📄 Origen: [ProductosControllerTests.cs](#) línea 67
- ⌚ Duración: 3 ms

Nombre del Caso de Prueba

Retornar la información correcta de un producto cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite obtener los detalles de un producto correctamente cuando se proporciona un ID válido.

Criterios de Aceptación

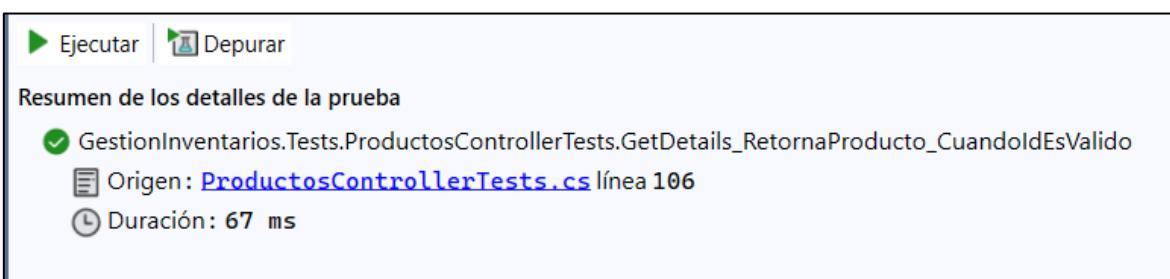
1. El producto debe existir en la base de datos.
2. El ID proporcionado debe ser válido y corresponder a un producto existente.
3. El sistema debe retornar los detalles del producto en la vista.

```
//Retorna la informacion correcta de un producto cuando el ID es valido
[Fact]
➊ | 0 referencias
public async Task GetDetails_RetornaProducto_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);
    var producto = new Producto
    {
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 100,
        CostoProducto = 124.99m,
        IdCategoria = 1
    };
    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    var result = await controller.Details(producto.ID);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Producto>(viewResult.Model);
    Assert.Equal("Closet", returnValue.NombreProducto);
}
```

Ejecución de la prueba:



Nombre del Caso de Prueba

Retornar NotFound cuando el ID no existe

Descripción

Este caso de prueba verifica que el sistema responde con un resultado de “NotFound” cuando se proporciona un ID que no corresponde a ningún producto en la base de datos.

Criterios de Aceptación

1. El ID proporcionado no debe corresponder a ningún producto en la base de datos.
2. El sistema debe retornar un resultado de “NotFound”.

```
//Cuando el ID es no valido, responde con un NotFound
[Fact]
● | 0 referencias
public async Task GetProductoDetails_RetornaNotFound_CuandoIdNoExiste()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);

    var result = await controller.Details(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:



Nombre del Caso de Prueba

Editar un producto cuando la información nueva es válida

Descripción

Este caso de prueba verifica que el sistema permite editar un producto correctamente cuando se proporciona información válida para la actualización.

Criterios de Aceptación

1. El producto debe existir en la base de datos.
2. El modelo de edición debe ser válido y contener la información correcta.
3. El sistema debe actualizar la información del producto en la base de datos.
4. El sistema debe redirigir a la acción Index después de la edición exitosa.

```
//Edita cuando la informacion nueva es valida para la actualizacion
[Fact]
➊ | 0 referencias
public async Task Edit_Post_Producto_CuandoActualizacionEsValida()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);
    var producto = new Producto
    {
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 100,
        CostoProducto = 124.99m,
        IdCategoria = 1
    };
    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    //Actualizacion de parametros
    producto.NombreProducto = "Chinero";
    producto.DescripcionProducto = "1.50 Cafe";

    var result = await controller.Edit(producto.ID, producto);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var productoEnDb = context.Productos.FirstOrDefault(p => p.ID == producto.ID);
    Assert.NotNull(productoEnDb);
    Assert.Equal("Chinero", productoEnDb.NombreProducto);
    Assert.Equal("1.50 Cafe", productoEnDb.DescripcionProducto);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba	
✓	GestionInventarios.Tests.ProductosControllerTests.Edit_Post_Producto_CuandoActualizacionEsValida
☰	Origen: ProductosControllerTests.cs línea 142
⌚	Duración: 94 ms

Nombre del Caso de Prueba

No editar el registro cuando la información es inválida para la actualización

Descripción

Este caso de prueba verifica que el sistema no permite editar un producto cuando el modelo contiene datos inválidos, como un nombre de producto vacío.

Criterios de Aceptación

1. El producto debe existir en la base de datos.
2. El modelo de edición debe ser inválido (por ejemplo, el nombre del producto está vacío).
3. El sistema debe retornar a la vista de edición con un mensaje de error.
4. La información del producto en la base de datos no debe ser actualizada.

```
//No edita el registro cuando la informacion es invalida para la actualizacion
[Fact]
● | 0 referencias
public async Task Edit_Post_Producto_CuandoActualizacionNoEsValida()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);
    var producto = new Producto
    {
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 100,
        CostoProducto = 124.99m,
        IdCategoria = 1
    };
    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    //Se envia el modelo con un error de validacion
    controller.ModelState.AddModelError("NombreProducto", "Required");

    var result = await controller.Edit(producto.ID, producto);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Producto>(viewResult.Model);
    Assert.Equal("Closet", returnValue.NombreProducto);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

<input checked="" type="checkbox"/> GestiónInventarios.Tests.ProductosControllerTests.Edit_Post_Producto_CuandoActualizacionNoEsValida
Origen: ProductosControllerTests.cs línea 174
Duración: 2 ms

Nombre del Caso de Prueba

Eliminar un producto cuando el ID es válido y otras tablas no dependen de este registro

Descripción

Este caso de prueba verifica que el sistema permite eliminar un producto correctamente cuando se proporciona un ID válido y no hay dependencias en otras tablas.

Criterios de Aceptación

1. El producto debe existir en la base de datos.
2. El ID proporcionado debe ser válido y corresponder a un producto existente.
3. No debe haber dependencias en otras tablas que impidan la eliminación.
4. El sistema debe eliminar el producto de la base de datos.
5. El sistema debe redirigir a la acción Index después de la eliminación exitosa.

```
//Elimina cuando el ID es valido y otras tablas no dependen de este registro
[Fact]
● | 0 referencias
public async Task Delete_Producto_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);
    var producto = new Producto
    {
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 100,
        CostoProducto = 124.99m,
        IdCategoria = 1
    };
    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    var result = await controller.DeleteConfirmed(producto.ID);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var productoEnDb = context.Productos.FirstOrDefault(p => p.ID == producto.ID);
    Assert.Null(productoEnDb);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.ProductosControllerTests.Delete_Producto_CuandoIdEsValido
- 📄 Origen: [ProductosControllerTests.cs](#) línea 201
- ⌚ Duración: 4 ms

Nombre del Caso de Prueba

No eliminar el registro porque el ID no es válido, retorna NotFound

Descripción

Este caso de prueba verifica que el sistema responde con un resultado de “NotFound” cuando se proporciona un ID que no corresponde a ningún producto en la base de datos.

Criterios de Aceptación

1. El ID proporcionado no debe corresponder a ningún producto en la base de datos.
2. El sistema debe retornar un resultado de “NotFound”.

```
//No elimina el registro porque el ID no es valido, retorna notFound
[Fact]
✓ | 0 referencias
public async Task Delete_Producto_CuandoIdNoExiste()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProductosController(context);

    var result = await controller.DeleteConfirmed(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:



Controlador: Compras

Nombre del Caso de Prueba

Crear una nueva compra y un nuevo producto cuando el producto no existe

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva compra y un nuevo producto correctamente cuando se proporciona información válida y el producto no existe previamente en la base de datos.

Criterios de Aceptación

1. El producto no debe existir previamente en la base de datos.
2. La información proporcionada para el producto y la compra debe ser válida.
3. El sistema debe crear el nuevo producto y la nueva compra en la base de datos.
4. El sistema debe redirigir a la acción Index después de la creación exitosa.

```
// Crea una nueva compra y un nuevo producto cuando el producto no existe
[Fact]
● | 0 referencias
public async Task Crear_Compra_ProductoNuevo_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    //Crea un nuevo producto
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Gavetero",
        DescripcionProducto = "1.30 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    //Crea la nueva compra
    var compra = new Compra
    {
        FechaCompra = DateTime.Now,
        IdProveedor = 1,
        UnidadesCompra = 10
    };

    //Crea ambos registros en sus respectivos modelos
    var result = await controller.Create(producto, compra);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    var productoCreado = await context.Productos.FirstOrDefaultAsync(p => p.NombreProducto == "Gavetero");
    Assert.NotNull(productoCreado);
    Assert.Equal(10, productoCreado.UnidadesProducto);

    var compraCreada = await context.Compras.FirstOrDefaultAsync(c => c.IdProducto == productoCreado.ID);
    Assert.NotNull(compraCreada);
    Assert.Equal(10, compraCreada.UnidadesCompra);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.ComprasControllerTests.Crear_Compra_ProductoNuevo_CuandoEsValido ☰ Origen: ComprasControllerTests.cs línea 12 ⌚ Duración: 1.1 s

Nombre del Caso de Prueba

Crear una compra para un producto existente y actualizar sus unidades

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva compra y actualizar correctamente las unidades de un producto existente cuando se proporciona información válida.

Criterios de Aceptación

1. El producto debe existir previamente en la base de datos.
2. La información proporcionada para el producto y la compra debe ser válida.
3. El sistema debe actualizar las unidades del producto existente sumando las unidades de la nueva compra.
4. El sistema debe crear la nueva compra en la base de datos.
5. El sistema debe redirigir a la acción Index después de la creación exitosa.

```
// Actualiza las unidades del producto existente y crea una nueva compra
[Fact]
● | 0 referencias
public async Task Crear_Compra_ProductoExistente_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    //Crea un nuevo producto inicial
    var productoExistente = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Gavetero",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 5,
        CostoProducto = 100
    };
    context.Productos.Add(productoExistente);
    await context.SaveChangesAsync();

    //Establece el producto asociado a la compra
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Gavetero",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    //Genera la nueva compra
    var compra = new Compra
    {
        FechaCompra = DateTime.Now,
        IdProveedor = 1,
        UnidadesCompra = 10
    };

    var result = await controller.Create(producto, compra);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    //Verifica la suma de las unidades existentes con las ingresadas
    var productoActualizado = await context.Productos.FirstOrDefaultAsync(p => p.NombreProducto == "Gavetero");
    Assert.NotNull(productoActualizado);
    Assert.Equal(15, productoActualizado.UnidadesProducto);

    var compraCreada = await context.Compras.FirstOrDefaultAsync(c => c.IdProducto == productoActualizado.ID);
    Assert.NotNull(compraCreada);
    Assert.Equal(10, compraCreada.UnidadesCompra);
}
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.ComprasControllerTests.Crear_Compra_ProductoExistente_CuandoEsValido
- ⠀ Origen: [ComprasControllerTests.cs](#) línea 52
- ⠀ Duración: 12 ms

Nombre del Caso de Prueba

No crea la compra cuando el modelo no es válido

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva compra cuando el modelo de datos no es válido, y que retorna la vista correspondiente con el modelo de compra.

Criterios de Aceptación

1. El producto debe tener un error de validación en uno de los campos.
2. La información proporcionada para el producto y la compra debe ser válida, excepto por el error de validación.
3. El sistema no debe crear la nueva compra en la base de datos.
4. El sistema debe retornar la vista correspondiente con el modelo de compra.

```
// No crea la compra cuando el modelo no es válido
[Fact]
➊ | 0 referencias
public async Task Crear_Compra_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    //Crea un nuevo producto
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };
    //Crea una nueva compra
    var compra = new Compra
    {
        FechaCompra = DateTime.Now,
        IdProveedor = 1,
        UnidadesCompra = 10
    };

    //Se envia un error de NombreProducto null
    controller.ModelState.AddModelError("NombreProducto", "El nombre es requerido.");

    var result = await controller.Create(producto, compra);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Compra>(viewResult.Model);
    Assert.Equal(compra, returnValue);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.ComprasControllerTests.Crear_Compra_CuandoNoEsValido
⠀ Origen: ComprasControllerTests.cs línea 104
⠀ Duración: 4 ms

Nombre del Caso de Prueba

Editar una compra cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar una compra y actualizar correctamente las unidades del producto asociado cuando se proporciona información válida.

Criterios de Aceptación

1. El producto debe existir previamente en la base de datos.
2. La información proporcionada para el producto y la compra debe ser válida.
3. El sistema debe actualizar las unidades del producto existente.
4. El sistema debe redirigir a la acción Index después de la edición exitosa.
5. El sistema debe reflejar los cambios en la base de datos.

```
//Edita cuando la informacion es valida
[Fact]
➊ | 0 referencias
public async Task Editar_Compra_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    //Establece el producto asociado a la compra
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 5,
        CostoProducto = 100
    };

    //Genera la nueva compra
    var compra = new Compra
    {
        FechaCompra = DateTime.Now,
        IdProveedor = 1,
        UnidadesCompra = 5
    };

    //Actualizamos el parametro
    producto.UnidadesProducto = 10;

    var result = await controller.Create(producto, compra);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    //Se comprueba que el dato ha sido actualizado
    var compraActualizada = await context.Compras.FirstOrDefaultAsync(c => c.ID == compra.ID);
    Assert.NotNull(compraActualizada);
    Assert.Equal(10, compraActualizada.UnidadesCompra);
}
}
```

Ejecución de pruebas:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.ComprasControllerTests.Editar_Compra_CuandoEsValido
Origen: ComprasControllerTests.cs línea 138
Duración: 7 ms

Nombre del Caso de Prueba

No edita la compra cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite editar una compra cuando el modelo de datos no es válido, y que retorna la vista correspondiente sin realizar cambios en la base de datos.

Criterios de Aceptación

1. El producto debe existir previamente en la base de datos.
2. La información proporcionada para la compra debe contener un error de validación en uno de sus campos.
3. El sistema no debe actualizar la información del producto existente.
4. El sistema debe retornar la vista correspondiente sin realizar cambios en la base de datos.

```
//No Edita cuando la informacion es invalida
[Fact]
● 0 referencias
public async Task Editar_Compra_CuandoEsInvalido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    //Establece el producto asociado a la compra
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 5,
        CostoProducto = 100
    };

    //Genera la nueva compra
    var compra = new Compra
    {
        FechaCompra = DateTime.Now,
        IdProveedor = 1,
        UnidadesCompra = 5
    };

    controller.ModelState.AddModelError("UnidadesCompra", "El campo UnidadesCompra es requerido.");
    var result = await controller.Create(producto, compra);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    //Se comprueba que el dato ha sido actualizado
    var compraActualizada = await context.Compras.FirstOrDefaultAsync(c => c.ID == compra.ID);
    Assert.NotNull(compraActualizada);
    Assert.Equal(5, compraActualizada.UnidadesCompra);
}
```

Ejecución de las pruebas:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.ComprasControllerTests.Editar_Compra_CuandoEsInvalido 📄 Origen: ComprasControllerTests.cs línea 177 ⌚ Duración: 8 ms

Nombre del Caso de Prueba

Obtener detalles de la compra cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite obtener los detalles de una compra existente cuando se proporciona un ID válido.

Criterios de Aceptación

1. El producto y el proveedor deben existir previamente en la base de datos.
2. La compra debe existir previamente en la base de datos.
3. El sistema debe retornar la vista correspondiente con el modelo de compra.
4. El sistema debe mostrar los detalles correctos de la compra, incluyendo el nombre del producto asociado.

```
// Obtener detalles de la compra cuando el ID es válido
[Fact]
➊ | 0 referencias
public async Task Details_Compra_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    // Crea producto y proveedor asociados a la compra
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 5,
        CostoProducto = 100
    };
    context.Productos.Add(producto);

    var proveedor = new Proveedor
    {
        NombreProveedor = "Muebles El Roble"
    };
    context.Proveedores.Add(proveedor);
    await context.SaveChangesAsync();

    // Genera la nueva compra
    var compra = new Compra
    {
        FechaCompra = DateTime.Now,
        IdProveedor = proveedor.ID,
        IdProducto = producto.ID,
        UnidadesCompra = 5
    };
    context.Compras.Add(compra);
    await context.SaveChangesAsync();

    // Verifica que el ID existe
    var result = await controller.Details(compra.ID);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Compra>(viewResult.Model);
    Assert.Equal(compra.ID, returnValue.ID);
    Assert.Equal("Closet", returnValue.Producto.NombreProducto);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
➌ GestiónInventarios.Tests.ComprasControllerTests.Details_Compra_CuandoidEsValido
⠁ Origen: ComprasControllerTests.cs línea 215
⌚ Duración: 42 ms

Nombre del Caso de Prueba

No obtiene los detalles del registro cuando el ID es inválido

Descripción

Este caso de prueba verifica que el sistema no permite obtener los detalles de una compra cuando se proporciona un ID inválido, y que retorna un resultado de “No encontrado”.

Criterios de Aceptación

1. El ID proporcionado para la compra no debe existir en la base de datos.
2. El sistema debe retornar un resultado de “No encontrado” (NotFoundResult).

```
// No obtiene los detalles del registro cuando el ID es inválido
[Fact]
● | 0 referencias
public async Task Details_Compra_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    var result = await controller.Details(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:



Nombre del Caso de Prueba

Elimina el registro cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite eliminar una compra existente cuando se proporciona un ID válido, y que la compra es eliminada de la base de datos.

Criterios de Aceptación

1. El producto y el proveedor deben existir previamente en la base de datos.
2. La compra debe existir previamente en la base de datos.
3. El sistema debe eliminar la compra de la base de datos.
4. El sistema debe redirigir a la acción Index después de la eliminación exitosa.

```
// Elimina el registro cuando el ID es válido
[Fact]
● | 0 referencias
public async Task Eliminar_Compra_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    // Crea producto y proveedor asociados a la compra
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 5,
        CostoProducto = 100
    };
    context.Productos.Add(producto);

    var proveedor = new Proveedor
    {
        NombreProveedor = "Muebles El Roble"
    };
    context.Proveedores.Add(proveedor);
    await context.SaveChangesAsync();

    // Crea compra
    var compra = new Compra
    {
        FechaCompra = DateTime.Now,
        IdProveedor = proveedor.ID,
        IdProducto = producto.ID,
        UnidadesCompra = 5
    };
    context.Compras.Add(compra);
    await context.SaveChangesAsync();

    var result = await controller.DeleteConfirmed(compra.ID);

    // Verificar que la compra ha sido eliminada
    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var compraEnDb = context.Compras.FirstOrDefault(c => c.ID == compra.ID);
    Assert.Null(compraEnDb);
}
```

Ejecución de pruebas:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.ComprasControllerTests.Eliminar_Compra_CuandoidEsValido
█ Origen: ComprasControllerTests.cs línea 273
⌚ Duración: 17 ms

Nombre del Caso de Prueba

No elimina el registro cuando el ID es inválido

Descripción

Este caso de prueba verifica que el sistema no permite eliminar una compra cuando se proporciona un ID inválido, y que retorna un resultado de “No encontrado”.

Criterios de Aceptación

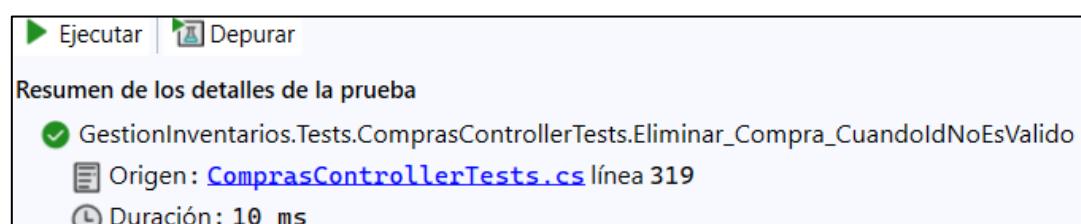
1. El ID proporcionado para la compra no debe existir en la base de datos.
2. El sistema debe retornar un resultado de “No encontrado” (NotFoundResult).

```
// No elimina el registro cuando el ID es inválido
[Fact]
● | 0 referencias
public async Task Eliminar_Compra_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ComprasController(context);

    var result = await controller.DeleteConfirmed(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de las pruebas:



Controlador: Ventas

Nombre del Caso de Prueba

Crear una venta cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva venta y actualizar correctamente las unidades del producto asociado cuando se proporciona información válida.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La información proporcionada para la venta debe ser válida.
3. El sistema debe crear la nueva venta en la base de datos.
4. El sistema debe actualizar las unidades del producto existente.
5. El sistema debe redirigir a la acción Index después de la creación exitosa.

```
// Crear una venta cuando la información es válida
[Fact]
● 0 referencias
public async Task Crear_Venta_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoría asociado a la venta
    var categoria = new Categoria
    {
        NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba"
    };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = categoria.ID, NombreProducto = "Closet", DescripcionProducto = "1.00 Blanco", UnidadesProducto = 10, CostoProducto = 100
    };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now, UnidadesVenta = 5, PrecioUnitarioVenta = 200, PrecioTotalVenta = 200, IdEmpleado = 1
    };

    string Categoria = categoria.NombreCategoria;
    string NombreProducto = producto.NombreProducto;
    string DescripcionProducto = producto.DescripcionProducto;

    var result = await controller.Create(Categoria, NombreProducto, DescripcionProducto, venta);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    var ventaCreada = await context.Ventas.FirstOrDefaultAsync(v => v.ID == venta.ID);
    Assert.NotNull(ventaCreada);
    Assert.Equal(5, ventaCreada.UnidadesVenta);

    var productoActualizado = await context.Productos.FirstOrDefaultAsync(p => p.ID == producto.ID);
    Assert.NotNull(productoActualizado);
    Assert.Equal(5, productoActualizado.UnidadesProducto);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.VentasControllerTests.Crear_Venta_CuandoEsValido
- 📄 Origen: [VentasControllerTests.cs](#) línea 12
- ⌚ Duración: 9 ms

Nombre del Caso de Prueba

No crear una venta cuando el modelo no es válido

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva venta cuando el modelo de datos no es válido, y que retorna la vista correspondiente con el modelo de venta.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La información proporcionada para la venta debe contener un error de validación en uno de sus campos.
3. El sistema no debe crear la nueva venta en la base de datos.
4. El sistema debe retornar la vista correspondiente con el modelo de venta.

```
// No crear una venta cuando el modelo no es válido
[Fact]
● | 0 referencias
public async Task Crear_Venta_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoria asociado a la venta
    var categoria = new Categoria
    { NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba" };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    { IdCategoria = categoria.ID, NombreProducto = "Closet", DescripcionProducto = "1.00 Blanco",
      UnidadesProducto = 10, CostoProducto = 100 };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    { FechaVenta = DateTime.Now, UnidadesVenta = 5, PrecioUnitarioVenta = 200,
      PrecioTotalVenta = 200, IdEmpleado = 1 };

    string Categoria = categoria.NombreCategoria;
    string NombreProducto = producto.NombreProducto;
    string DescripcionProducto = producto.DescripcionProducto;

    // Error de validación en el PrecioUnitarioVenta
    controller.ModelState.AddModelError("PrecioUnitarioVenta", "El precio unitario es requerido.");

    var result = await controller.Create(Categoria, NombreProducto, DescripcionProducto, venta);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Venta>(viewResult.Model);
    Assert.Equal(venta.PrecioUnitarioVenta, returnValue.PrecioUnitarioVenta);
}
```

Ejecución de pruebas:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.VentasControllerTests.Crear_Venta_CuandoNoEsValido
└ Origen: VentasControllerTests.cs línea 61
└ Duración: 5 ms

Nombre del Caso de Prueba

No crear una venta cuando no hay suficientes unidades

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva venta cuando no hay suficientes unidades del producto disponible, y que retorna la vista correspondiente con el modelo de venta.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La información proporcionada para la venta debe contener un error de validación en el campo UnidadesVenta indicando que no hay suficientes unidades del producto.
3. El sistema no debe crear la nueva venta en la base de datos.
4. El sistema debe retornar la vista correspondiente con el modelo de venta.

```
public async Task Crear_Venta_CuandoNoHaySuficientesUnidades()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoria asociado a la venta
    var categoria = new Categoria
    {
        NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba"
    };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = categoria.ID,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        UnidadesVenta = 5,
        PrecioUnitarioVenta = 200,
        PrecioTotalVenta = 200,
        IdEmpleado = 1
    };
}

string Categoria = categoria.NombreCategoria;
string NombreProducto = producto.NombreProducto;
string DescripcionProducto = producto.DescripcionProducto;

// Error de validación en el PrecioUnitarioVenta
controller.ModelState.AddModelError("UnidadesVenta", "No se puede registrar la venta: No hay suficientes unidades del producto.");
var result = await controller.Create(Categoria, NombreProducto, DescripcionProducto, venta);

var viewResult = Assert.IsType<ViewResult>(result);
var returnValue = Assert.IsType<Venta>(viewResult.Model);
Assert.Equal(venta.UnidadesVenta, returnValue.UnidadesVenta);
}
```

Ejecución de pruebas:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.VentasControllerTests.Crear_Venta_CuandoNoHaySuficientesUnidades
└ Origen: VentasControllerTests.cs línea 111
└ Duración: 3 ms

Nombre del Caso de Prueba

No crear una venta cuando el precio de venta es menor que el costo del producto

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva venta cuando el precio de venta es menor que el costo del producto, y que retorna la vista correspondiente con un mensaje de error.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La información proporcionada para la venta debe contener un precio de venta menor que el costo del producto.
3. El sistema no debe crear la nueva venta en la base de datos.
4. El sistema debe retornar la vista correspondiente con un mensaje de error indicando que el precio de venta es menor que el costo del producto.

```
public async Task Crear_Venta_CuandoPrecioVentaEsMenorQueCosto()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoria asociado a la venta
    var categoria = new Categoria
    {
        NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba"
    };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = categoria.ID,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        UnidadesVenta = 5,
        PrecioUnitarioVenta = 1,
        PrecioTotalVenta = 5,
        IdEmpleado = 1
    };
}

string Categoria = categoria.NombreCategoria;
string NombreProducto = producto.NombreProducto;
string DescripcionProducto = producto.DescripcionProducto;

var result = await controller.Create(Categoria, NombreProducto, DescripcionProducto, venta);

var viewResult = Assert.IsType<ViewResult>(result);
var returnValue = Assert.IsType<Venta>(viewResult.Model);
Assert.Equal(venta.PrecioUnitarioVenta, returnValue.PrecioUnitarioVenta);
Assert.Equal("No se puede registrar la venta: El precio de venta es menor que el costo del producto", controller.ViewBag.ErrorMessage);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.VentasControllerTests.Crear_Venta_CuandoPrecioVentaEsMenorQueCosto
█ Origen: VentasControllerTests.cs línea 164
⌚ Duración: 6 ms

Nombre del Caso de Prueba

Editar una venta cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar una venta existente y actualizar correctamente los detalles de la venta cuando se proporciona información válida.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La venta debe existir previamente en la base de datos.
3. La información proporcionada para la venta debe ser válida.
4. El sistema debe actualizar los detalles de la venta en la base de datos.
5. El sistema debe redirigir a la acción Index después de la edición exitosa.

```
public async Task Editar_Venta_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoria asociado a la venta
    var categoria = new Categoria
    {
        NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba"
    };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = categoria.ID,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        UnidadesVenta = 5,
        PrecioUnitarioVenta = 200,
        PrecioTotalVenta = 1000,
        IdEmpleado = 1
    };

    context.Ventas.Add(venta);
    await context.SaveChangesAsync();

    // Actualiza la venta
    venta.UnidadesVenta = 3;
    venta.PrecioTotalVenta = 180;

    string Categoria = categoria.NombreCategoria;
    string NombreProducto = producto.NombreProducto;
    string DescripcionProducto = producto.DescripcionProducto;

    var result = await controller.Edit(venta.ID, Categoria, NombreProducto, DescripcionProducto, venta);
    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    var ventaActualizada = await context.Ventas.FirstOrDefaultAsync(v => v.ID == venta.ID);
    Assert.NotNull(ventaActualizada);
    Assert.Equal(3, ventaActualizada.UnidadesVenta);
    Assert.Equal(180, ventaActualizada.PrecioTotalVenta);
}
```

Ejecución de pruebas:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.VentasControllerTests.Editar_Venta_CuandoEsValido
- ⠀ Origen: [VentasControllerTests.cs](#) línea 215
- ⠀ Duración: 17 ms

Nombre del Caso de Prueba

No editar una venta cuando el ID no es válido

Descripción

Este caso de prueba verifica que el sistema no permite editar una venta cuando se proporciona un ID de venta no válido.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La venta no debe existir en la base de datos con el ID proporcionado.
3. El sistema debe devolver un resultado de “NotFound” cuando se intenta editar una venta con un ID no válido.

```
public async Task Editar_Venta_CuandoIdNoValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoria asociado a la venta
    var categoria = new Categoria
    {
        NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba"
    };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = categoria.ID,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        UnidadesVenta = 5,
        PrecioUnitarioVenta = 200,
        PrecioTotalVenta = 1000,
        IdEmpleado = 1
    };
}

string Categoria = categoria.NombreCategoria;
string NombreProducto = producto.NombreProducto;
string DescripcionProducto = producto.DescripcionProducto;

var result = await controller.Edit(2,Categoria,NombreProducto, DescripcionProducto,venta);

Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
● GestiónInventarios.Tests.VentasControllerTests.Editar_Venta_CuandoidNoValido
⠀ Origen: VentasControllerTests.cs línea 275
⠀ Duración: 2 ms

Nombre del Caso de Prueba

No editar una venta cuando el modelo no es válido

Descripción

Este caso de prueba verifica que el sistema no permite editar una venta cuando el modelo de datos no es válido, y que devuelve la vista con los datos originales y los mensajes de error correspondientes.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La venta debe existir previamente en la base de datos.
3. El modelo de datos proporcionado para la venta debe contener errores de validación.
4. El sistema debe devolver la vista de edición con los datos originales y los mensajes de error.

```
public async Task Editar_Venta_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoria asociado a la venta
    var categoria = new Categoria
    {
        NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba"
    };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = categoria.ID,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        UnidadesVenta = 5,
        PrecioUnitarioVenta = 200,
        PrecioTotalVenta = 1000,
        IdEmpleado = 1
    };
}
```

```
string Categoria = categoria.NombreCategoria;
string NombreProducto = producto.NombreProducto;
string DescripcionProducto = producto.DescripcionProducto;

// Error de validación en el PrecioUnitarioVenta
controller.ModelState.AddModelError("PrecioUnitarioVenta", "El precio unitario es requerido.");

var result = await controller.Edit(venta.ID, Categoria, NombreProducto, DescripcionProducto, venta);

var viewResult = Assert.IsType<ViewResult>(result);
var returnValue = Assert.IsType<Venta>(viewResult.Model);
Assert.Equal(venta.PrecioUnitarioVenta, returnValue.PrecioUnitarioVenta);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.VentasControllerTests.Editar_Venta_CuandoNoEsValido
▀ Origen: VentasControllerTests.cs línea 324
⌚ Duración: 985 ms

Nombre del Caso de Prueba

No editar una venta cuando el precio de venta es menor que el costo del producto

Descripción

Este caso de prueba verifica que el sistema no permite editar una venta cuando el precio de venta es menor que el costo del producto, y que devuelve la vista con un mensaje de error adecuado.

Criterios de Aceptación

1. La categoría y el producto deben existir previamente en la base de datos.
2. La venta debe existir previamente en la base de datos.
3. El precio de venta proporcionado debe ser menor que el costo del producto.
4. El sistema debe devolver la vista de edición con un mensaje de error indicando que el precio de venta no puede ser menor que el costo del producto.

```
public async Task Editar_Venta_CuandoPrecioVentaEsMenorQueCosto()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea categoria asociado a la venta
    var categoria = new Categoria
    {
        NombreCategoria = "Mueble", DescripcionCategoria = "Muebles de prueba"
    };

    context.Categorias.Add(categoria);
    await context.SaveChangesAsync();

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = categoria.ID,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };

    context.Productos.Add(producto);
    await context.SaveChangesAsync();

    // Genera la nueva venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        UnidadesVenta = 5,
        PrecioUnitarioVenta = 200,
        PrecioTotalVenta = 1000,
        IdEmpleado = 1
    };

    context.Ventas.Add(venta);
    await context.SaveChangesAsync();

    context.Ventas.Add(venta);
    await context.SaveChangesAsync();

    // Actualiza la venta
    venta.PrecioUnitarioVenta = 40;

    string Categoria = categoria.NombreCategoria;
    string NombreProducto = producto.NombreProducto;
    string DescripcionProducto = producto.DescripcionProducto;

    var result = await controller.Edit(venta.ID, Categoria, NombreProducto, DescripcionProducto, venta);
    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Venta>(viewResult.Model);
    Assert.Equal(venta.PrecioUnitarioVenta, returnValue.PrecioUnitarioVenta);
    Assert.Equal("No se puede actualizar la venta: El precio de venta es menor que el costo del producto", controller.ViewBag.ErrorMessage);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.VentasControllerTests.Editar_Venta_CuandoPrecioVentaEsMenorQueCosto
- ⠀ Origen: [VentasControllerTests.cs](#) línea 378
- ⠀ Duración: 179 ms

Nombre del Caso de Prueba

Obtener detalles de la venta cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite obtener los detalles de una venta existente cuando se proporciona un ID de venta válido.

Criterios de Aceptación

1. La categoría, el producto y el empleado deben existir previamente en la base de datos.
2. La venta debe existir previamente en la base de datos.
3. El sistema debe devolver la vista de detalles con la información correcta de la venta, producto y empleado.

```
public async Task Details_Venta_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Closet",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };
    context.Productos.Add(producto);

    var empleado = new Empleado
    {
        NombreEmpleado = "Juan",
        ApellidoEmpleado = "Perez",
        ComisionVentaEmpleado = 2
    };
    context.Empleados.Add(empleado);
    await context.SaveChangesAsync();

    // Crea venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        IdProducto = producto.ID,
        UnidadesVenta = 2,
        PrecioUnitarioVenta = 150,
        PrecioTotalVenta = 300,
        IdEmpleado = 1
    };
    context.Ventas.Add(venta);
    await context.SaveChangesAsync();

    // Verifica que el ID existe
    var result = await controller.Details(venta.ID);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Venta>(viewResult.Model);
    Assert.Equal(venta.ID, returnValue.ID);
    Assert.Equal("Closet", returnValue.Producto.NombreProducto);
    Assert.Equal("Juan", returnValue.Empleado.NombreEmpleado);
}
```

Ejecución de pruebas:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.VentasControllerTests.Details_Venta_CuandoidEsValido
└ Origen: VentasControllerTests.cs línea 434
└ Duración: 35 ms

Nombre del Caso de Prueba

No obtiene los detalles del registro cuando el ID es inválido

Descripción

Este caso de prueba verifica que el sistema no permite obtener los detalles de una venta cuando se proporciona un ID de venta no válido, devolviendo un resultado de “NotFound”.

Criterios de Aceptación

1. El ID de venta proporcionado no debe existir en la base de datos.
2. El sistema debe devolver un resultado de “NotFound” cuando se intenta obtener los detalles de una venta con un ID no válido.

```
// No obtiene los detalles del registro cuando el ID es inválido
[Fact]
✓ | 0 referencias
public async Task Details_Venta_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    var result = await controller.Details(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:

```
▶ Ejecutar | ⚙ Depurar

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.VentasControllerTests.Details_Venta_CuandoidNoEsValido
    ⚡ Origen: VentasControllerTests.cs línea 484
    ⏱ Duración: 2 ms
```

Nombre del Caso de Prueba

Eliminar venta cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite eliminar un registro de venta existente cuando se proporciona un ID de venta válido.

Criterios de Aceptación

1. La categoría, el producto y el empleado deben existir previamente en la base de datos.
2. La venta debe existir previamente en la base de datos.
3. El sistema debe eliminar la venta y redirigir a la vista de índice.
4. La venta eliminada no debe existir en la base de datos después de la operación.

```
public async Task Eliminar_Venta_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    // Crea producto asociado a la venta
    var producto = new Producto
    {
        IdCategoria = 1,
        NombreProducto = "Gavetero",
        DescripcionProducto = "1.00 Blanco",
        UnidadesProducto = 10,
        CostoProducto = 100
    };
    context.Productos.Add(producto);

    var empleado = new Empleado
    {
        NombreEmpleado = "Manuel",
        ApellidoEmpleado = "Perez",
        ComisionVentaEmpleado = 2
    };
    context.Empleados.Add(empleado);
    await context.SaveChangesAsync();

    // Crea venta
    var venta = new Venta
    {
        FechaVenta = DateTime.Now,
        IdProducto = producto.ID,
        UnidadesVenta = 2,
        PrecioUnitarioVenta = 150,
        PrecioTotalVenta = 300,
        IdEmpleado = empleado.ID
    };
    context.Ventas.Add(venta);
    await context.SaveChangesAsync();

    var result = await controller.DeleteConfirmed(venta.ID);

    // Verificar que la venta ha sido eliminada
    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var ventaEnDb = context.Ventas.FirstOrDefault(v => v.ID == venta.ID);
    Assert.Null(ventaEnDb);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestionInventarios.Tests.VentasControllerTests.Eliminar_Venta_CuandoidEsValido
└ Origen: VentasControllerTests.cs línea 496
└ Duración: 11 ms

Nombre del Caso de Prueba

No eliminar venta cuando el ID no es válido

Descripción

Este caso de prueba verifica que el sistema no permite eliminar un registro de venta cuando se proporciona un ID de venta no válido.

Criterios de Aceptación

1. El ID de venta proporcionado no debe existir en la base de datos.
2. El sistema debe devolver un resultado de “No encontrado” (NotFoundResult).
3. Ningún registro de venta debe ser eliminado de la base de datos.

```
// No elimina el registro cuando el ID es inválido
[Fact]
✓ | 0 referencias
public async Task Eliminar_Venta_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new VentasController(context);

    var result = await controller.DeleteConfirmed(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:

The screenshot shows a test execution interface with the following details:

- Ejecutar** (Run) button.
- Depurar** (Debug) button.
- Resumen de los detalles de la prueba** (Test details summary):
 - GestionInventarios.Tests.VentasControllerTests.Eliminar_Venta_CuandoidNoEsValido** (Test name)
 - Origen: VentasControllerTests.cs línea 546** (Origin: VentasControllerTests.cs line 546)
 - Duración: 7 ms** (Duration: 7 ms)

Controlador: Proveedores

Nombre del Caso de Prueba

Crear proveedor cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear un nuevo proveedor cuando se proporciona información válida.

Criterios de Aceptación

1. La información del proveedor debe ser válida.
2. El sistema debe crear el proveedor y redirigir a la vista de índice.
3. El proveedor creado debe existir en la base de datos con la información proporcionada.

```
// Ingresar el proveedor cuando la información es válida
[Fact]
✓ | 0 referencias
public async Task Crear_Proveedor_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    var proveedor = new Proveedor
    {
        NombreProveedor = "Duraban",
    };

    var result = await controller.Create(proveedor);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    var proveedorCreado = await context.Proveedores.FirstOrDefaultAsync(
        c => c.NombreProveedor == "Duraban");

    Assert.NotNull(proveedorCreado);
    Assert.Equal("Duraban", proveedorCreado.NombreProveedor);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.ProveedoresControllerTests.Crear_Proveedor_CuandoEsValido
- 🔗 Origen: [ProveedoresControllerTests.cs](#) línea 12
- ⌚ Duración: 13 ms

Nombre del Caso de Prueba

No crear proveedor cuando el modelo no es válido

Descripción

Este caso de prueba verifica que el sistema no permite crear un nuevo proveedor cuando la información proporcionada no es válida.

Criterios de Aceptación

1. La información del proveedor debe ser inválida o incompleta.
2. El sistema debe detectar el error de validación y no crear el proveedor.
3. El sistema debe devolver la vista de creación con el modelo de proveedor y los mensajes de error correspondientes.

```
// No ingresa el proveedor cuando el modelo no es válido
[Fact]
✓ | 0 referencias
public async Task Crear_Proveedor_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    var proveedor = new Proveedor
    {
        NombreProveedor = "",
    };

    // Error de validación en el Nombre de Proveedor
    controller.ModelState.AddModelError("NombreProveedor", "El nombre es requerido.");

    var result = await controller.Create(proveedor);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Proveedor>(viewResult.Model);
    Assert.Equal(proveedor.NombreProveedor, returnValue.NombreProveedor);
}
```

Ejecución de la prueba:

Ejecutar
 Depurar

Resumen de los detalles de la prueba

✓ GestiónInventarios.Tests.ProveedoresControllerTests.Crear_Proveedor_CuandoNoEsValido

 Origen: [ProveedoresControllerTests.cs](#) línea 36

 Duración: 3 ms

Nombre del Caso de Prueba

Editar proveedor cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar un proveedor existente cuando se proporciona información válida.

Criterios de Aceptación

1. El proveedor debe existir previamente en la base de datos.
2. La información actualizada del proveedor debe ser válida y completa.
3. El sistema debe actualizar el proveedor y redirigir a la vista de índice.
4. La información del proveedor debe reflejar los cambios realizados en la base de datos.

```
//Edita el proveedor cuando es valida la informacion
[Fact]
➊ | 0 referencias
public async Task Editar_Post_Proveedor_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    //Crea un nuevo proveedor
    var proveedor = new Proveedor
    {
        NombreProveedor = "Duraban",
    };
    context.Proveedores.Add(proveedor);
    await context.SaveChangesAsync();

    //Actualizacion de parametros
    proveedor.NombreProveedor = "LG";

    var result = await controller.Edit(proveedor.ID, proveedor);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    //Verifica que la informacion ha sido actualizada
    var proveedorActualizado = context.Proveedores.FirstOrDefault(p => p.ID == proveedor.ID);
    Assert.NotNull(proveedorActualizado);
    Assert.Equal("LG", proveedorActualizado.NombreProveedor);
}
```

Ejecución de la prueba:

Ejecutar | Depurar

Resumen de los detalles de la prueba

➌ GestiónInventarios.Tests.ProveedoresControllerTests.Editar_Post_Proveedor_CuandoEsValido

📄 Origen: [ProveedoresControllerTests.cs](#) línea 58

⌚ Duración: 34 ms

Nombre del Caso de Prueba

No actualizar proveedor cuando la información no es válida

Descripción

Este caso de prueba verifica que el sistema no permite actualizar un proveedor existente cuando se proporciona información no válida.

Criterios de Aceptación

1. El proveedor debe existir previamente en la base de datos.
2. La información actualizada del proveedor debe ser inválida o incompleta.
3. El sistema debe detectar el error de validación y no actualizar el proveedor.
4. El sistema debe devolver la vista de edición con el modelo de proveedor y los mensajes de error correspondientes.

```
//No Actualiza el proveedor cuando la informacion no es valida
[Fact]
✓ | 0 referencias
public async Task Editar_Post_Proveedor_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    //Crea un nuevo proveedor
    var proveedor = new Proveedor
    {
        NombreProveedor = "Duraban",
    };
    context.Proveedores.Add(proveedor);
    await context.SaveChangesAsync();

    // Error de validación
    controller.ModelState.AddModelError("NombreProveedor", "El nombre es requerido.");

    var result = await controller.Edit(proveedor.ID, proveedor);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Proveedor>(viewResult.Model);
    Assert.Equal("Duraban", returnValue.NombreProveedor);
}
```

Ejecución de la prueba:

 Ejecutar
 Depurar

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.ProveedoresControllerTests.Editar_Post_Proveedor_CuandoNoEsValido
- 🔗 Origen: [ProveedoresControllerTests.cs](#) línea 87
- 🕒 Duración: 10 ms

Nombre del Caso de Prueba

Obtener detalles del proveedor cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite obtener los detalles de un proveedor existente cuando se proporciona un ID de proveedor válido.

Criterios de Aceptación

1. El proveedor debe existir previamente en la base de datos.
2. El sistema debe devolver la vista de detalles con la información correcta del proveedor.
3. La información del proveedor devuelta debe coincidir con la información almacenada en la base de datos.

```
//Obtener detalles del proveedor cuando el Id es Valido
[Fact]
✓ | 0 referencias
public async Task Details_Proveedor_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    //Crea proveedor
    var proveedor = new Proveedor
    {
        NombreProveedor = "Duraban",
    };
    context.Proveedores.Add(proveedor);
    await context.SaveChangesAsync();

    //Verifica que el ID existe
    var result = await controller.Details(proveedor.ID);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Proveedor>(viewResult.Model);
    Assert.Equal("Duraban", returnValue.NombreProveedor);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.ProveedoresControllerTests.Details_Proveedor_CuandoIdEsValido
Origen: ProveedoresControllerTests.cs línea 112
Duración: 2 ms

Nombre del Caso de Prueba

No obtener detalles del proveedor cuando el ID no es válido

Descripción

Este caso de prueba verifica que el sistema no permite obtener los detalles de un proveedor cuando se proporciona un ID de proveedor no válido.

Criterios de Aceptación

1. El ID de proveedor proporcionado no debe existir en la base de datos.
2. El sistema debe devolver un resultado de “No encontrado” (NotFoundResult).
3. Ninguna información del proveedor debe ser devuelta.

```
//No obtiene los detalles del registro cuando el id es invalido
[Fact]
✔ | 0 referencias
public async Task Details_Proveedor_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    var result = await controller.Details(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:



Nombre del Caso de Prueba

Eliminar proveedor cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite eliminar un registro de proveedor existente cuando se proporciona un ID de proveedor válido.

Criterios de Aceptación

1. El proveedor debe existir previamente en la base de datos.
2. El sistema debe eliminar el proveedor y redirigir a la vista de índice.
3. El proveedor eliminado no debe existir en la base de datos.

```
//Elimina el registro cuando el Id es valido
[Fact]
➊ | 0 referencias
public async Task Eliminar_Proveedor_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    //Crea nueva categoria
    var proveedor = new Proveedor
    {
        NombreProveedor = "Duraban",
    };
    context.Proveedores.Add(proveedor);
    await context.SaveChangesAsync();

    var result = await controller.DeleteConfirmed(proveedor.ID);

    // Verificar que el proveedor ha sido eliminado
    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var proveedorEnDb = context.Proveedores.FirstOrDefault(p => p.ID == proveedor.ID);
    Assert.Null(proveedorEnDb);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ➊ GestionInventarios.Tests.ProveedoresControllerTests.Eliminar_Proveedor_CuandoidEsValido
 - ⠁ Origen: [ProveedoresControllerTests.cs](#) línea 147
 - ⌚ Duración: 6 ms

Nombre del Caso de Prueba

No eliminar proveedor cuando el ID no es válido

Descripción

Este caso de prueba verifica que el sistema no permite eliminar un registro de proveedor cuando se proporciona un ID de proveedor no válido.

Criterios de Aceptación

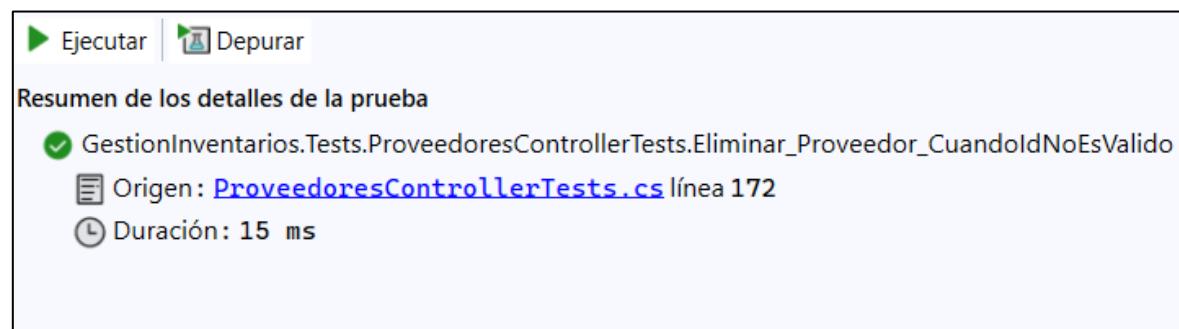
1. El ID de proveedor proporcionado no debe existir en la base de datos.
2. El sistema debe devolver un resultado de “No encontrado” (NotFoundResult).
3. Ningún registro de proveedor debe ser eliminado de la base de datos.

```
//No elimina el registro cuando el Id es invalido
[Fact]
✓ | 0 referencias
public async Task Eliminar_Proveedor_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new ProveedoresController(context);

    var result = await controller.DeleteConfirmed(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:



Controlador: Empleado

Nombre del Caso de Prueba

Crear empleado cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear un nuevo empleado cuando se proporciona información válida.

Criterios de Aceptación

1. La información del empleado debe ser válida y completa.
2. El sistema debe crear el empleado y redirigir a la vista de índice.
3. El empleado creado debe existir en la base de datos con la información proporcionada.

```
//Ingresa el empleado cuando la informacion es valida
[Fact]
✔ | 0 referencias
public async Task Crear_Emppleado_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new EmpleadosController(context);

    var empleado = new Empleado
    {
        NombreEmpleado = "Adonay",
        ApellidoEmpleado = "Lopez",
        ComisionVentaEmpleado = 5
    };

    var result = await controller.Create(empleado);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    var empleadoCreada = await context.Empleados.FirstOrDefaultAsync(
        c => c.NombreEmpleado == "Adonay");
    Assert.NotNull(empleadoCreada);
    Assert.Equal("Adonay", empleadoCreada.NombreEmpleado);
    Assert.Equal("Lopez", empleadoCreada.ApellidoEmpleado);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba

- ✔ GestiónInventarios.Tests.EmpleadosControllerTests.Crear_Emppleado_CuandoEsValido
- 📄 Origen: [EmpleadosControllerTests.cs](#) línea 12
- ⌚ Duración: 12 ms

Nombre del Caso de Prueba

No crear empleado cuando el modelo no es válido

Descripción

Este caso de prueba verifica que el sistema no permite crear un nuevo empleado cuando la información proporcionada no es válida.

Criterios de Aceptación

1. La información del empleado debe ser inválida o incompleta.
2. El sistema debe detectar el error de validación y no crear el empleado.
3. El sistema debe devolver la vista de creación con el modelo de empleado y los mensajes de error correspondientes.

```
//No ingresa el empleado cuando el modelo no es valido
[Fact]
● | 0 referencias
public async Task Crear_Emppleado_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new EmpleadosController(context);

    var empleado = new Empleado
    {
        NombreEmpleado = "",
        ApellidoEmpleado = "Lopez",
        ComisionVentaEmpleado = 5
    };

    // Error de validación en el Nombre de Empleado
    controller.ModelState.AddModelError("NombreEmpleado", "El nombre es requerido.");

    var result = await controller.Create(empleado);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Empleado>(viewResult.Model);
    Assert.Equal(empleado.NombreEmpleado, returnValue.NombreEmpleado);
}
```

Ejecución de la prueba:

Ejecutar Depurar

Resumen de los detalles de la prueba

✓ GestiónInventarios.Tests.EmpleadosControllerTests.Crear_Emppleado_CuandoNoEsValido

⠀ Origen: [EmpleadosControllerTests.cs](#) línea 38

⠀ Duración: 3 ms

Nombre del Caso de Prueba

Editar empleado cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar un empleado existente cuando se proporciona información válida.

Criterios de Aceptación

1. El empleado debe existir previamente en la base de datos.
2. La información actualizada del empleado debe ser válida y completa.
3. El sistema debe actualizar el empleado y redirigir a la vista de índice.
4. La información del empleado debe reflejar los cambios realizados en la base de datos

```
//Edita el empleado cuando la informacion es valida
[Fact]
➊ | 0 referencias
public async Task Editar_Post_Emppleado_CuandoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new EmpleadosController(context);

    //Crea un nuevo empleado
    var empleado = new Empleado
    {
        NombreEmpleado = "Adonay",
        ApellidoEmpleado = "Lopez",
        ComisionVentaEmpleado = 5
    };
    context.Empleados.Add(empleado);
    await context.SaveChangesAsync();

    //Actualizacion de parametros
    empleado.NombreEmpleado = "Carlos";
    empleado.ApellidoEmpleado = "Torres";

    var result = await controller.Edit(empleado.ID, empleado);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);

    //Verifica que la informacion ha sido actualizada
    var empleadoActualizado = context.Empleados.FirstOrDefault(p => p.ID == empleado.ID);
    Assert.NotNull(empleadoActualizado);
    Assert.Equal("Carlos", empleadoActualizado.NombreEmpleado);
    Assert.Equal("Torres", empleadoActualizado.ApellidoEmpleado);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba	
✓	GestionInventarios.Tests.EmpleadosControllerTests.Editar_Post_Emppleado_CuandoEsValido
⃡	Origen: EmpleadosControllerTests.cs línea 62
⌚	Duración: 1.1 s

Nombre del Caso de Prueba

No actualizar empleado cuando la información no es válida

Descripción

Este caso de prueba verifica que el sistema no permite actualizar un empleado existente cuando se proporciona información no válida.

Criterios de Aceptación

1. El empleado debe existir previamente en la base de datos.
2. La información actualizada del empleado debe ser inválida o incompleta.
3. El sistema debe detectar el error de validación y no actualizar el empleado.
4. El sistema debe devolver la vista de edición con el modelo de empleado y los mensajes de error correspondientes.

```
//No Actualiza el empleado cuando la informacion no es valida
[Fact]
✓ | 0 referencias
public async Task Editar_Post_Emppleado_CuandoNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new EmpleadosController(context);

    //Crea una nuevo empleado
    var empleado = new Empleado
    {
        NombreEmpleado = "Adonay",
        ApellidoEmpleado = "Lopez",
        ComisionVentaEmpleado = 5
    };
    context.Empleados.Add(empleado);
    await context.SaveChangesAsync();

    // Error de validación
    controller.ModelState.AddModelError("NombreEmpleado", "El nombre es requerido.");

    var result = await controller.Edit(empleado.ID, empleado);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Empleado>(viewResult.Model);
    Assert.Equal("Adonay", returnValue.NombreEmpleado);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.EmpleadosControllerTests.Editar_Post_Emppleado_CuandoNoEsValido
Origen: EmpleadosControllerTests.cs línea 95
Duración: 2 ms

Nombre del Caso de Prueba

Obtener detalles del empleado cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite obtener los detalles de un empleado existente cuando se proporciona un ID de empleado válido.

Criterios de Aceptación

1. El empleado debe existir previamente en la base de datos.
2. El sistema debe devolver la vista de detalles con la información correcta del empleado.
3. La información del empleado devuelta debe coincidir con la información almacenada en la base de datos.

```
//Obtener detalles del empleado cuando el Id es Valido
[Fact]
✓ | 0 referencias
public async Task Details_Emppleado_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new EmpleadosController(context);

    //Crea empleado
    var empleado = new Empleado
    {
        NombreEmpleado = "Adonay",
        ApellidoEmpleado = "Lopez",
        ComisionVentaEmpleado = 5
    };
    context.Empleados.Add(empleado);
    await context.SaveChangesAsync();

    //Verifica que el ID existe
    var result = await controller.Details(empleado.ID);

    var viewResult = Assert.IsType<ViewResult>(result);
    var returnValue = Assert.IsType<Empleado>(viewResult.Model);
    Assert.Equal("Adonay", returnValue.NombreEmpleado);
    Assert.Equal("Lopez", returnValue.ApellidoEmpleado);
}
```

Ejecución de la prueba:

Resumen de los detalles de la prueba
✓ GestiónInventarios.Tests.EmpleadosControllerTests.Details_Emppleado_CuandoldEsValido
█ Origen: EmpleadosControllerTests.cs línea 122
⌚ Duración: 5 ms

Nombre del Caso de Prueba

No obtener detalles del empleado cuando el ID no es válido

Descripción

Este caso de prueba verifica que el sistema no permite obtener los detalles de un empleado cuando se proporciona un ID de empleado no válido.

Criterios de Aceptación

1. El ID de empleado proporcionado no debe existir en la base de datos.
2. El sistema debe devolver un resultado de “No encontrado” (NotFoundResult).
3. Ninguna información del empleado debe ser devuelta.

```
//No obtiene los detalles del registro cuando el id es invalido  
[Fact]  
✔ | 0 referencias  
public async Task Details_Emppleado_CuandoIdNoEsValido()  
{  
    var context = Setup.GetInMemoryDatabaseContext();  
    var controller = new EmpleadosController(context);  
  
    var result = await controller.Details(999);  
  
    Assert.IsType<NotFoundResult>(result);  
}
```

Ejecución de la prueba:

▶ Ejecutar | ⚙ Depurar

Resumen de los detalles de la prueba

- ✔ GestiónInventarios.Tests.EmpleadosControllerTests.Details_Emppleado_CuandoIdNoEsValido
📄 Origen: [EmpleadosControllerTests.cs](#) línea 148
⌚ Duración: 11 ms

Nombre del Caso de Prueba

Eliminar empleado cuando el ID es válido

Descripción

Este caso de prueba verifica que el sistema permite eliminar un registro de empleado existente cuando se proporciona un ID de empleado válido.

Criterios de Aceptación

1. El empleado debe existir previamente en la base de datos.
2. El sistema debe eliminar el empleado y redirigir a la vista de índice.
3. El empleado eliminado no debe existir en la base de datos después de la operación.

```
//Elimina el registro cuando el Id es valido
[Fact]
● 0 referencias
public async Task Eliminar_Emppleado_CuandoIdEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new EmpleadosController(context);

    //Crea empleado
    var empleado = new Empleado
    {
        NombreEmpleado = "Adonay",
        ApellidoEmpleado = "Lopez",
        ComisionVentaEmpleado = 5
    };
    context.Empleados.Add(empleado);
    await context.SaveChangesAsync();

    var result = await controller.DeleteConfirmed(empleado.ID);

    // Verificar que el empleado ha sido eliminado
    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal(nameof(controller.Index), redirectResult.ActionName);

    var empleadoEnDb = context.Empleados.FirstOrDefault(p => p.ID == empleado.ID);
    Assert.Null(empleadoEnDb);
}
```

Ejecución de la prueba:

Ejecutar | Depurar

Resumen de los detalles de la prueba

- ✓ GestiónInventarios.Tests.EmpleadosControllerTests.Eliminar_Emppleado_CuandoidEsValido
- ▀ Origen: [EmpleadosControllerTests.cs](#) línea 160
- ⌚ Duración: 5 ms

Nombre del Caso de Prueba

No eliminar empleado cuando el ID no es válido

Descripción

Este caso de prueba verifica que el sistema no permite eliminar un registro de empleado cuando se proporciona un ID de empleado no válido.

Criterios de Aceptación

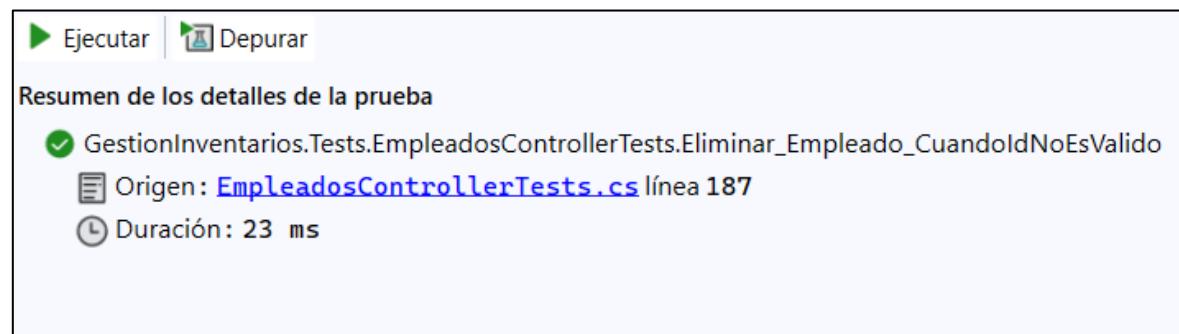
1. El ID de empleado proporcionado no debe existir en la base de datos.
2. El sistema debe devolver un resultado de “No encontrado” (NotFoundResult).
3. Ningún registro de empleado debe ser eliminado de la base de datos.

```
/No elimina el registro cuando el Id es invalido
[Fact]
✔ | 0 referencias
public async Task Eliminar_Emppleado_CuandoIdNoEsValido()
{
    var context = Setup.GetInMemoryDatabaseContext();
    var controller = new EmpleadosController(context);

    var result = await controller.DeleteConfirmed(999);

    Assert.IsType<NotFoundResult>(result);
}
```

Ejecución de la prueba:



Pruebas manuales

Controlador: Usuarios

Nombre del Caso de Prueba

Registrar un usuario válido

Descripción

Este caso de prueba verifica que un usuario puede registrarse correctamente en el sistema cuando proporciona datos válidos.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico válido.
2. La contraseña y la confirmación de la contraseña deben coincidir.
3. El usuario debe ser registrado en la base de datos.
4. La contraseña debe ser almacenada de manera segura (encriptada).
5. El sistema debe mostrar un mensaje de “Registro Exitoso”

Pasos de Ejecución:

1. Ir a pestaña Usuarios
2. Dar clic en botón “Agregar Usuario”
3. Se ingresa los datos de correo (usuario1@hotmail.com)
4. Se ingresa contraseña: 1234
5. Se ingresa confirmación de contraseña: 1234
6. Obtenemos mensaje “Registro Exitoso”.

REGISTRAR USUARIO NUEVO

[Lista de Usuarios](#) / Registrar Usuario

Ingrese la información del nuevo usuario

Correo de Usuario

Contraseña

Confirmar contraseña
 

REGISTRAR **REGRESAR**

REGISTRAR USUARIO NUEVO

Lista de Usuarios / Registrar Usuario

Ingrese la información del nuevo usuario

Registro Exitoso!

Correo de Usuario
usuario1@hotmail.com

Contraseña
Contraseña

Confirmar contraseña
Confirmar Contraseña

REGISTRAR **REGRESAR**

7. Confirmamos en el Listado de Usuarios

REGISTRO DE USUARIOS

AGREGAR USUARIO

Filtrar por correo...

Correo de usuario	Acciones
lenin@hotmail.com	
erika@hotmail.com	
usuario1@hotmail.com	

8. Confirmamos en la base de datos la creación del nuevo usuario.

SELECT * FROM Usuarios

100 %

Resultados Mensajes

ID	CorreoUsuario	ClaveUsuario	IdRol
1	admin@correo.com	\$2a\$11\$OFAxbiJ2RhLlxZECrVT0hOSWxK.btsic9Lhpop5hoOc...	1
2	lenin@hotmail.com	\$2a\$11\$X2AzetKZoRVNIRvrQ0R0AOMjZXdxJL1LVlbm5zHrPE...	2
3	erika@hotmail.com	\$2a\$11\$e6l4fUhCeoRYpyWQvMIY/uiJJHCDQRz6x0HWE50ih1...	2
4	usuario1@hotmail.com	\$2a\$11\$u6omQ8PmDNOAg7jk.c1huvzvsuSrahw8tcUqsEM7/...	2

Nombre del Caso de Prueba

No registrar un usuario cuando las contraseñas no coinciden

Descripción

Este caso de prueba verifica que el sistema no permite registrar un usuario cuando las contraseñas proporcionadas no coinciden.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico válido.
2. La contraseña y la confirmación de la contraseña deben coincidir.
3. Si las contraseñas no coinciden, el usuario no debe ser registrado en la base de datos.
4. El sistema debe mostrar un mensaje de error indicando que las contraseñas no coinciden.

Pasos de Ejecución:

1. Ir a pestaña Usuarios
2. Dar clic en botón “Agregar Usuario”
3. Se ingresa los datos de correo (usuario2@hotmail.com)
4. Se ingresa contraseña: 1234
5. Se ingresa confirmación de contraseña: 5678
6. Obtenemos mensaje “Error al registrar usuario: Las contraseñas no coinciden”.

REGISTRAR USUARIO NUEVO

[Lista de Usuarios](#) / Registrar Usuario

Ingrese la información del nuevo usuario

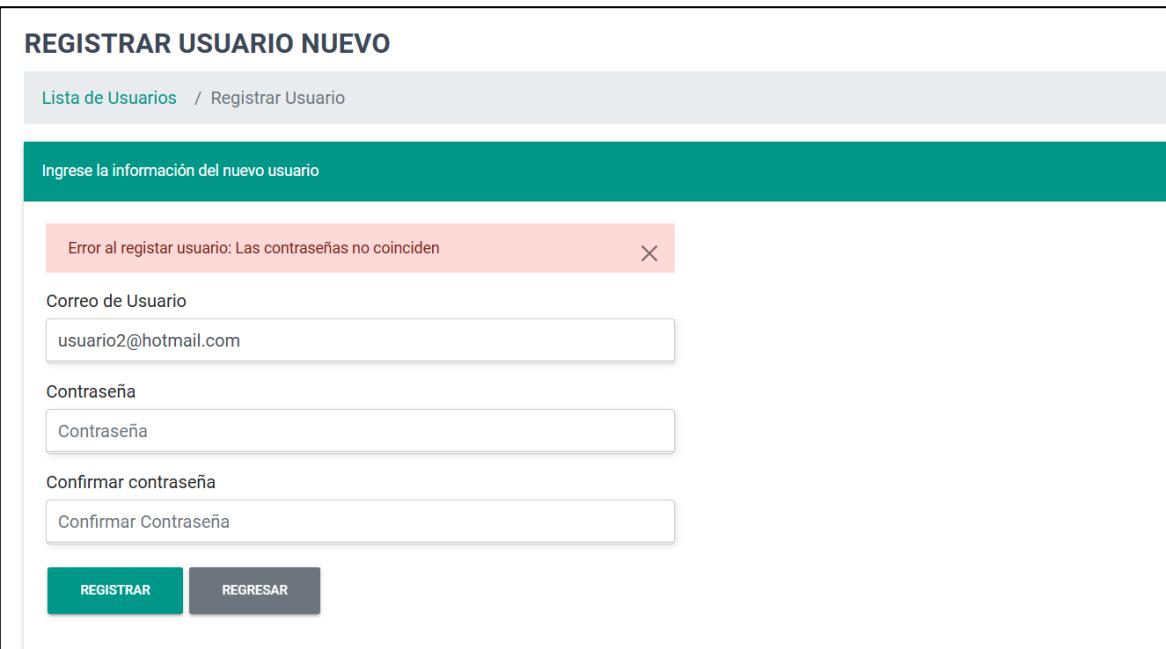
Error al registrar usuario: Las contraseñas no coinciden ×

Correo de Usuario

Contraseña

Confirmar contraseña

REGISTRAR **REGRESAR**



Nombre del Caso de Prueba

No registrar un usuario cuando el correo ya está registrado en la base de datos

Descripción

Este caso de prueba verifica que el sistema no permite registrar un usuario cuando el correo electrónico proporcionado ya está registrado en la base de datos.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico válido.
2. Si el correo electrónico ya está registrado, el usuario no debe ser registrado nuevamente.
3. El sistema debe mostrar un mensaje de error indicando que el correo ya está registrado.

Pasos de Ejecución:

1. Ir a pestaña Usuarios
2. Dar clic en botón “Agregar Usuario”
3. Se ingresa los datos de correo (usuario1@hotmail.com)
4. Se ingresa contraseña: 1234
5. Se ingresa confirmación de contraseña: 1234
6. Obtenemos mensaje “Error: El correo brindado ya posee cuenta registrada”.

REGISTRAR USUARIO NUEVO

[Lista de Usuarios](#) / [Registrar Usuario](#)

Ingrese la información del nuevo usuario

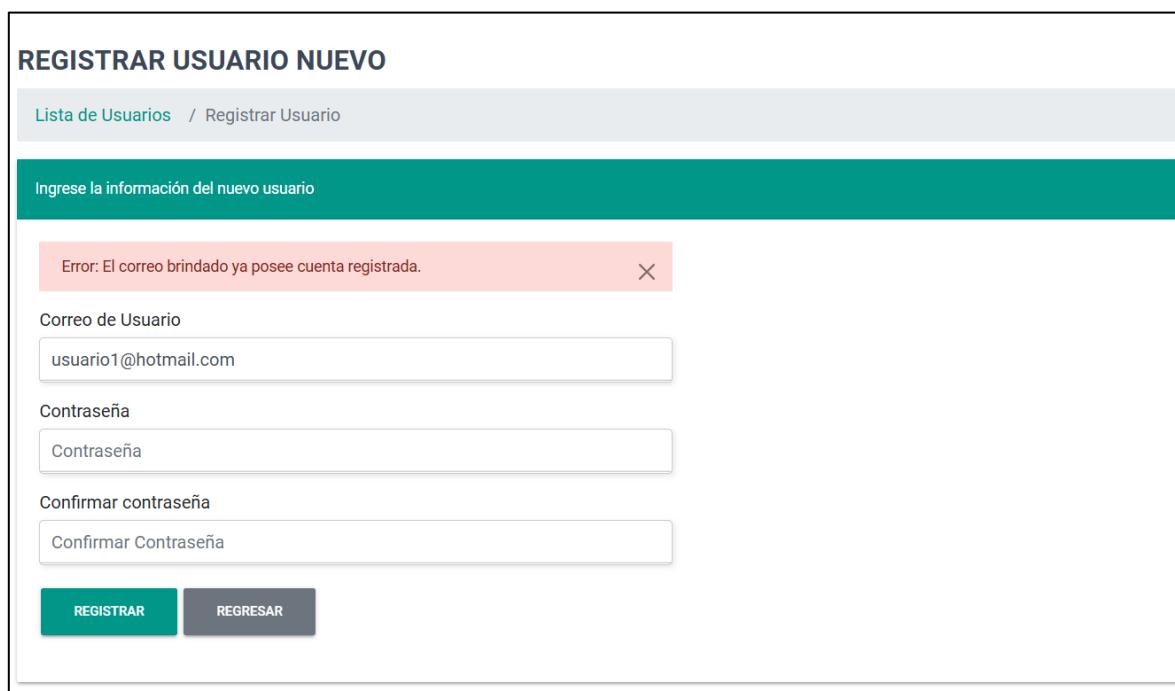
Error: El correo brindado ya posee cuenta registrada. ×

Correo de Usuario

Contraseña

Confirmar contraseña

REGISTRAR **REGRESAR**

A screenshot of a web-based user registration form titled "REGISTRAR USUARIO NUEVO". The form has a header with links for "Lista de Usuarios" and "Registrar Usuario". Below the header is a teal bar with the placeholder "Ingrese la información del nuevo usuario". A red error message box contains the text "Error: El correo brindado ya posee cuenta registrada." with a close button. The main form area has fields for "Correo de Usuario" (with value "usuario1@hotmail.com"), "Contraseña", and "Confirmar contraseña". At the bottom are two buttons: a teal "REGISTRAR" button and a grey "REGRESAR" button.

Nombre del Caso de Prueba

No iniciar sesión si el usuario no existe en la base de datos

Descripción

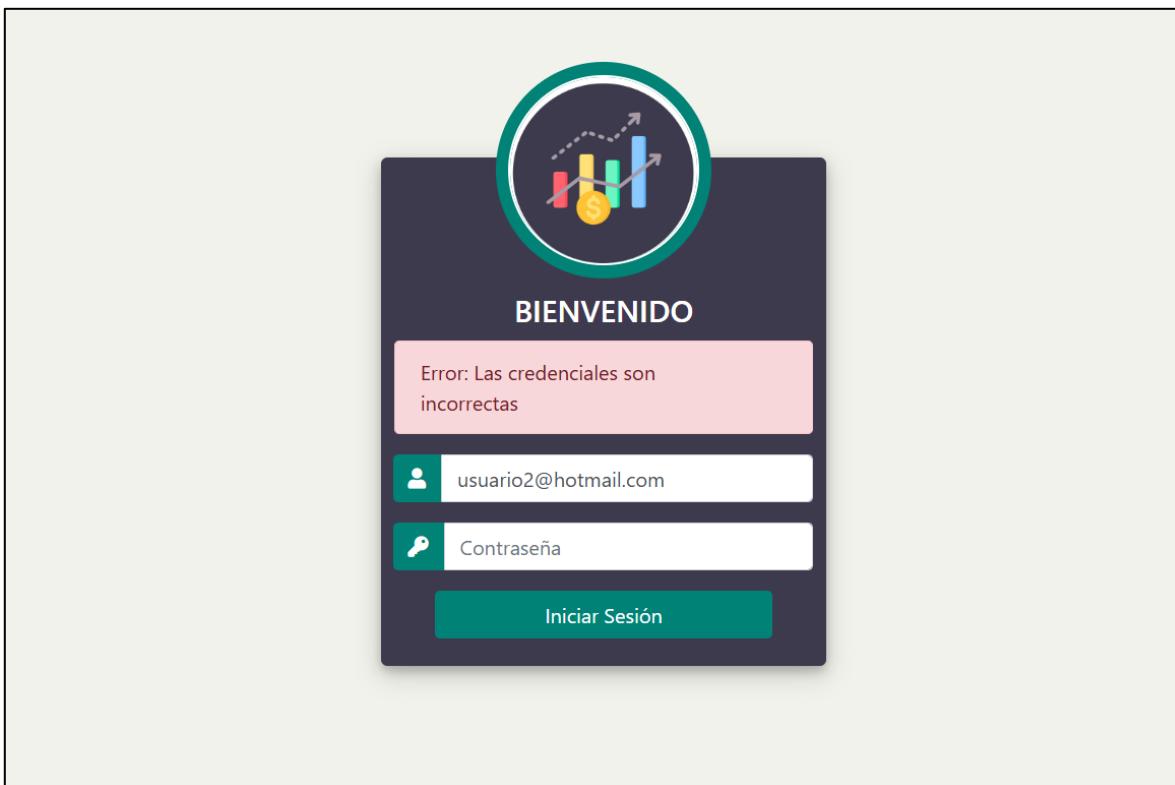
Este caso de prueba verifica que el sistema no permite iniciar sesión a un usuario que no está registrado en la base de datos.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico y una contraseña.
2. Si el correo electrónico no está registrado en la base de datos, el usuario no debe poder iniciar sesión.
3. El sistema debe mostrar un mensaje de error indicando que las credenciales son incorrectas.

Pasos de Ejecución:

1. Iniciar el programa
2. Se ingresa los datos de correo (usuario2@hotmail.com)
3. Se ingresa contraseña: 1234
4. Obtenemos mensaje “Error: Las credenciales son incorrectas”.



Nombre del Caso de Prueba

No iniciar sesión cuando la contraseña es incorrecta

Descripción

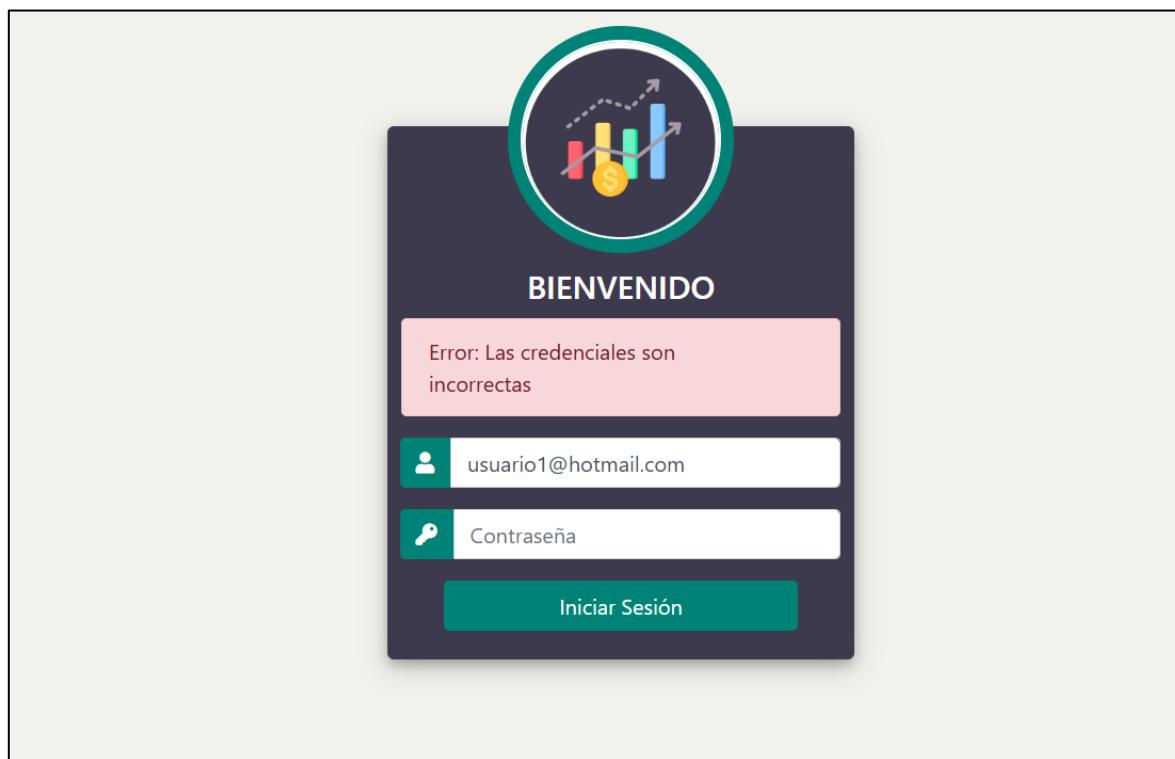
Este caso de prueba verifica que el sistema no permite iniciar sesión a un usuario cuando la contraseña proporcionada es incorrecta.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico y una contraseña.
2. Si la contraseña no coincide con la almacenada en la base de datos, el usuario no debe poder iniciar sesión.
3. El sistema debe mostrar un mensaje de error indicando que las credenciales son incorrectas.

Pasos de Ejecución:

1. Iniciar el programa
2. Se ingresa los datos de correo (usuario1@hotmail.com)
3. Se ingresa contraseña: 5678
4. Obtenemos mensaje “Error: Las credenciales son incorrectas”.



Nombre del Caso de Prueba

No iniciar sesión cuando hay datos faltantes

Descripción

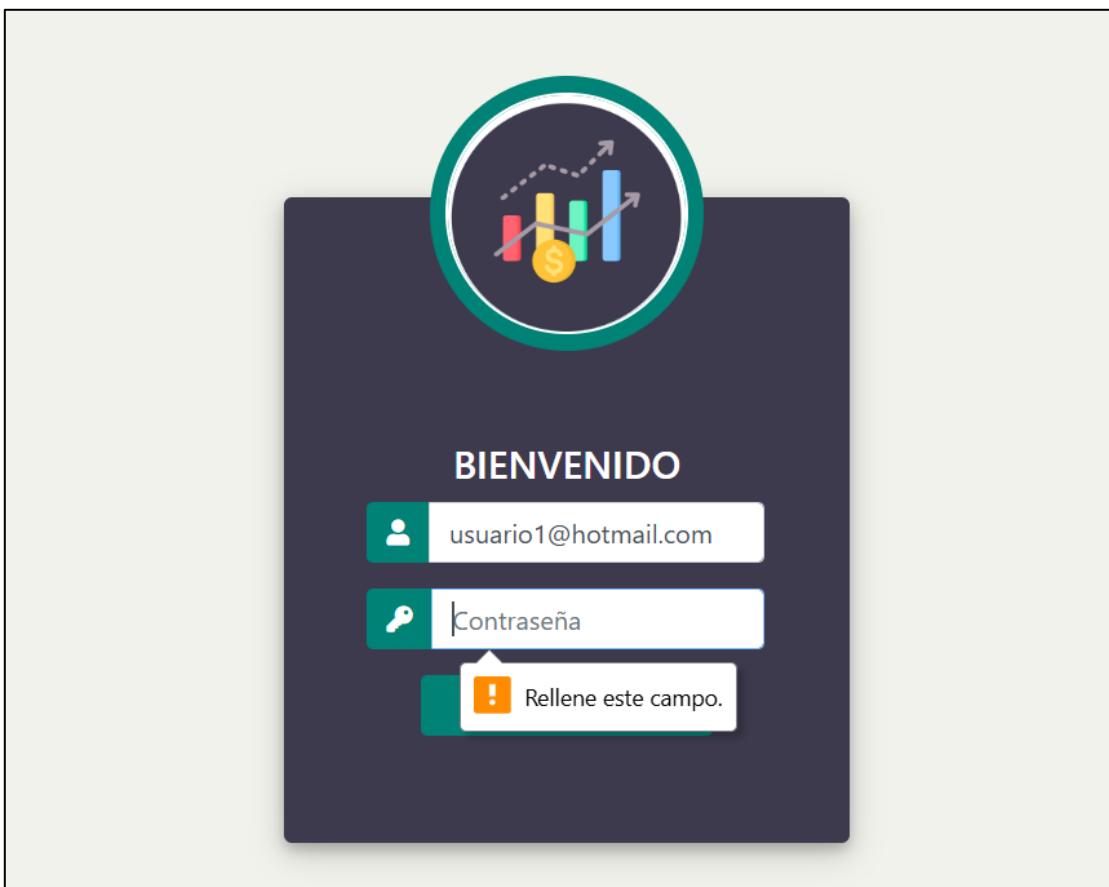
Este caso de prueba verifica que el sistema no permite iniciar sesión a un usuario cuando no proporciona todos los datos.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico y una contraseña.
2. Si no ingresa la contraseña, el usuario no debe poder iniciar sesión.
3. El sistema debe mostrar un mensaje de error indicando que debe llenar los campos.

Pasos de Ejecución:

1. Iniciar el programa
2. Se ingresa los datos de correo (usuario1@hotmail.com) y clic en iniciar sesión.
3. Obtenemos mensaje “Rellene este campo”.



Nombre del Caso de Prueba

No posee acceso cuando el usuario no es el usuario Administrador.

Descripción

Este caso de prueba verifica que el sistema no permite el acceso al CRUD de usuarios cuando el usuario no es tipo “Admin”.

Criterios de Aceptación

1. El usuario debe proporcionar un correo electrónico y una contraseña.
2. Si el correo no pertenece a usuario Admin no muestra pestaña “Usuarios” y “Configuración”.

Pasos de Ejecución:

1. Iniciar el programa
2. Se ingresa los datos de correo (usuario1@hotmail.com).
3. Se ingresa la contraseña “1234”
4. No se puede visualizar la pestaña “Usuarios” y “Configuración”.



5. Se ingresa los datos de correo (admin@correo.com).
6. Se ingresa la contraseña “1234”
7. Se puede visualizar la pestaña “Usuarios” y “Configuración”.



Nombre del Caso de Prueba

No crea un nuevo usuario cuando el modelo es invalido

Descripción

Este caso de prueba verifica que el sistema no permite crear un nuevo usuario cuando hay datos faltantes o el modelo es invalido.

Criterios de Aceptación

1. La información del usuario debe ser inválida o incompleta.
2. El sistema debe detectar los errores de validación y no crear el usuario.
3. El sistema debe devolver la vista de creación con el modelo de usuario y los mensajes de error correspondientes.
4. Ningún registro de usuario debe ser creado en la base de datos.

Pasos de Ejecución:

1. Ir a pestaña “Usuarios”
2. Dar clic en botón “Agregar Usuario”
3. Ingresamos el correo (usuario2@hotmail.com) y dar clic en registrar
4. Obtenemos mensaje “Rellene este campo”.

The screenshot shows a user registration form titled 'REGISTRAR USUARIO NUEVO'. The URL in the address bar is 'Lista de Usuarios / Registrar Usuario'. The form has a teal header bar with the text 'Ingrese la información del nuevo usuario'. Below it, there are input fields for 'Correo de Usuario' containing 'usuario2@hotmail.com' and 'Contraseña' containing 'Contraseña'. The 'Contraseña' field is highlighted with a red border, indicating an error. A tooltip 'Rellene este campo.' with an exclamation mark icon points to this field. Below these are fields for 'Confirmar contraseña' and 'Confirmar Contraseña', both currently empty. At the bottom are two buttons: a teal 'REGISTRAR' button and a grey 'REGRESAR' button.

Controlador: Categorías

Nombre del Caso de Prueba

Crear una categoría cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva categoría correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. La categoría debe tener un nombre y una descripción válidos.
2. El sistema debe agregar la nueva categoría a la base de datos.
3. El sistema debe redirigir a la acción Index después de la creación exitosa

Pasos de Ejecución:

1. Posicionarse en “Categorías”
2. Dar clic en “Registrar nueva categoría”
3. Ingresamos nombre y descripción de una categoría no existente. Nombre: “Muebles de Metal” y descripción “Diversos muebles metálicos: percheros, mesas, zapateras, etc.”
4. Clic en registrar y verificamos en el listado que se ha creado correctamente.

REGISTRAR CATEGORIA

Lista de Categorias / Registrar Categoria

Ingrese el nombre de la categoría.

Nombre de categoria

Descripcion de categoria

REGISTRAR
REGRESAR

Nombre de categoria	Descripcion de categoria	Acciones
Muebles de madera	Muebles artesanales de carpintería	
Camas	Camas de proveedores registrados	
Electrodomesticos	Aparatos electricos para el hogar	
Infantil	Muebles y accesorios para niños y bebes	
Muebles de Metal	Diversos muebles metálicos: percheros, mesas, zapateras, etc.	

Nombre del Caso de Prueba

No crea una categoría cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva categoría correctamente cuando se proporciona un nombre de categoría existente.

Criterios de Aceptación

1. La categoría posee un nombre igual a un registro de la base de datos.
2. El sistema no debe agregar la nueva categoría a la base de datos.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Categorías”
2. Dar clic en “Registrar nueva categoría”
3. Ingresamos nombre y descripción de una categoría existente. Nombre: “Camas” y descripción “Diversos estilos de camas: madera, metal y de marcas registradas”.
4. Clic en Registrar.
5. El sistema nos indica Error: La categoría ya está registrada.

The screenshot shows a user interface for 'REGISTRAR CATEGORIA'. At the top, there's a navigation bar with 'Lista de Categorías' and 'Registrar Categoría'. Below it is a teal header bar with the placeholder 'Ingrese el nombre de la categoría.' In the main area, there's an error message in a pink box: 'Error: La categoría ya está registrada.' with a close button 'X'. Below the message are two input fields: 'Nombre de categoría' containing 'Camas' and 'Descripción de categoría' containing 'Diversos estilos de camas: madera, metal y de marcas registradas'. At the bottom are two buttons: 'REGISTRAR' in a teal box and 'REGRESAR' in a grey box.

Nombre del Caso de Prueba

No edita una categoría cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva categoría correctamente cuando se proporciona un nombre de categoría existente.

Criterios de Aceptación

1. La categoría posee un nombre igual a un registro de la base de datos.
2. El sistema no debe agregar la nueva categoría a la base de datos.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Categorías”
2. Dar clic en “Ver Listado”
3. En Acciones seleccionamos editar una categoría
4. Ingresamos nombre y descripción de una categoría existente. Nombre: “Camas” y descripción “Diversos estilos de camas: madera, metal y de marcas registradas”.
5. Clic en Guardar Cambios.
6. El sistema nos indica Error: La categoría ya está registrada.

EDITAR INFORMACIÓN DE CATEGORIA

Lista de Categorías / Editar Categoría

Ingrese la nueva información de la categoría

Error: La categoría ya está registrada. ×

Nombre de categoría

Camas

Descripción de categoría

Camas de proveedores registrados

GUARDAR CAMBIOSREGRESAR

Nombre del Caso de Prueba

Edita una categoría cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva categoría correctamente cuando se proporciona información valida.

Criterios de Aceptación

1. La categoría debe tener un nombre y una descripción válidos.
2. El sistema debe guardar los cambios de la categoría.
3. El sistema debe redirigir a la acción Index después de la edición exitosa

Pasos de Ejecución:

1. Posicionarse en “Categorías”
2. Dar clic en “Ver Listado”
3. En Acciones seleccionamos editar una categoría
4. Ingresamos nombre y descripción de una categoría existente. Nombre: “Producto Plástico” y descripción “Mesa, Sillas, Cubetas, etc.”.
5. Clic en Guardar Cambios.
6. El sistema nos redirige al Listado y podemos observar la categoría actualizada.

EDITAR INFORMACIÓN DE CATEGORIA

Lista de Categorias / Editar Categoria

Ingrese la nueva información de la categoria

Nombre de categoria
 Producto Plástico

Descripcion de categoria
 Mesa, Sillas, Cubetas, etc.

GUARDAR CAMBIOS
REGRESAR

Nombre de categoria	Descripcion de categoria	Acciones
Muebles de madera	Muebles artesanales de carpinteria	
Camas	Camas de proveedores registrados	
Electrodomesticos	Aparatos electricos para el hogar	
Infantil	Muebles y accesorios para niños y bebes	
Producto Plástico	Mesa, Sillas, Cubetas, etc.	

Nombre del Caso de Prueba

Eliminar la categoría cuando no hay registros de producto dependientes.

Descripción

Este caso de prueba verifica que el sistema permite eliminar una categoría cuando no existen productos asociados a ella.

Criterios de Aceptación

1. La categoría debe existir previamente en la base de datos.
2. No debe haber productos asociados a la categoría.
3. El sistema debe eliminar la categoría y redirigir a la vista de índice.
4. La categoría eliminada no debe existir en la base de datos después de la operación.

Pasos de Ejecución:

1. Posicionarse en “Categorías”
2. Dar clic en “Ver Listado”
3. En Acciones seleccionamos eliminar una categoría, en este caso; nombre: “Producto Plástico” y descripción “Mesa, Sillas, Cubetas, etc.”.
4. Clic en Eliminar.
5. El sistema nos redirige al Listado y podemos observar la categoría ha sido eliminada.

ELIMINAR CATEGORIA

¿Estás seguro que deseas eliminar esta categoría?

Nombre de categoria	Producto Plástico
Descripcion de categoria	Mesa, Sillas, Cubetas, etc.

ELIMINAR
Cancelar

Buscar por nombre categoria...		
Nombre de categoria	Descripcion de categoria	Acciones
Muebles de madera	Muebles artesanales de carpinteria	
Camas	Camas de proveedores registrados	
Electrodomesticos	Aparatos electricos para el hogar	
Infantil	Muebles y accesorios para niños y bebes	

Nombre del Caso de Prueba

No eliminar la categoría cuando hay registros de producto dependientes

Descripción

Este caso de prueba verifica que el sistema no permite eliminar una categoría cuando existen productos asociados a ella.

Criterios de Aceptación

1. La categoría debe existir previamente en la base de datos.
2. Debe haber productos asociados a la categoría.
3. El sistema debe detectar la dependencia y no eliminar la categoría.
4. El sistema debe devolver un mensaje de error indicando que la categoría no puede ser eliminada debido a productos dependientes.

Pasos de Ejecución:

1. Posicionarse en “Categorías”
2. Dar clic en “Ver Listado”
3. En Acciones seleccionamos eliminar una categoría, en este caso seleccionamos la categoría “Camas”, puesto que posee productos asociados.
4. Clic en Eliminar.
5. El sistema nos muestra el mensaje de error: Un registro de productos utiliza este registro.

ELIMINAR CATEGORIA

¿Estás seguro que deseas eliminar esta categoría?

Nombre de categoria	Camas
Descripción de categoria	Camas de proveedores registrados

ELIMINAR **CANCELAR**

ELIMINAR CATEGORIA

Error al eliminar: Un registro de productos utiliza este registro

¿Estás seguro que deseas eliminar esta categoría?

Nombre de categoria	
Descripción de categoria	

ELIMINAR **CANCELAR**

Controlador: Proveedores

Nombre del Caso de Prueba

Crear un proveedor cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear un nuevo proveedor correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. El proveedor debe tener un nombre válido.
2. El sistema debe agregar el nuevo proveedor a la base de datos.
3. El sistema debe redirigir a la acción Index después de la creación exitosa.

Pasos de Ejecución:

1. Posicionarse en “Proveedores”.
2. Dar clic en “Registrar nuevo proveedor”.
3. Ingresar nombre de un proveedor no existente. Nombre: “Duraban”.
4. Clic en registrar y verificar en el listado que se ha creado correctamente.

REGISTRAR PROVEEDORES

Ingrese el nombre del proveedor.

Nombre de proveedor

REGISTRAR **REGRESAR**

REGISTRO DE PROVEEDORES

Acciones		AGREGAR PROVEEDOR
Buscar por nombre de proveedor... <input type="text"/> 🔍		
Nombre de proveedor	Acciones	
Duraban	✍ 🕒 ✖	

Nombre del Caso de Prueba

No crear un proveedor cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite crear un nuevo proveedor correctamente cuando se proporciona un nombre de proveedor existente.

Criterios de Aceptación

1. El proveedor posee un nombre igual a un registro de la base de datos.
2. El sistema no debe agregar el nuevo proveedor a la base de datos.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Proveedores”.
2. Dar clic en “Registrar nuevo proveedor”.
3. Ingresar el nombre de un proveedor existente. Nombre: “Duraban”.
4. Clic en Registrar.
5. El sistema nos indica Error: El proveedor ya está registrado

REGISTRAR PROVEEDORES

Listado de Proveedores / Registrar Proveedor

Ingrese el nombre del proveedor.

Nombre de proveedor
Duraban

REGISTRAR **REGRESAR**

Ingrese el nombre del proveedor.

El proveedor ya está registrado. ×

Nombre de proveedor
Duraban

REGISTRAR **REGRESAR**

Nombre del Caso de Prueba

No editar un proveedor cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite editar un proveedor correctamente cuando se proporciona un nombre de proveedor existente.

Criterios de Aceptación

1. El proveedor posee un nombre igual a un registro de la base de datos.
2. El sistema no debe actualizar el proveedor en la base de datos.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Proveedores”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar editar el proveedor “Mabe”.
4. Ingresar el nombre de un proveedor existente. Nombre: “Duraban”.
5. Clic en Guardar Cambios.
6. El sistema nos indica Error: El proveedor ya está registrado.

The image displays two screenshots of a software application interface for modifying supplier information. Both screenshots show a teal header bar with the text "MODIFICAR PROVEEDORES". Below the header, there is a navigation bar with the text "Lista de Proveedores / Modificar Proveedor".

Screenshot 1 (Top): This screenshot shows the input field for "Nombre de proveedor" containing the value "Duraban". Below the input field are two buttons: "GUARDAR CAMBIOS" (in green) and "REGRESAR" (in grey).

Screenshot 2 (Bottom): This screenshot shows the same input field and button layout. A red error message box is displayed above the input field, containing the text "El proveedor ya está registrado." (The supplier already exists). There is a small "X" icon in the top right corner of the error message box.

Nombre del Caso de Prueba

Editar un proveedor cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar un proveedor correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. El proveedor debe tener un nombre válido.
2. El sistema debe guardar los cambios del proveedor.
3. El sistema debe redirigir a la acción Index después de la edición exitosa.

Pasos de Ejecución:

1. Posicionarse en “Proveedores”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar editar proveedor “Duraban”.
4. Ingresar el nombre de un proveedor. Nombre: “DalePlus”.
5. Clic en Guardar Cambios.
6. El sistema nos redirige al Listado y podemos observar el proveedor actualizado.

MODIFICAR PROVEEDORES

[Lista de Proveedores](#) / Modificar Proveedor

Ingrese la nueva información del proveedor.

Nombre de proveedor

GUARDAR CAMBIOS **REGRESAR**

REGISTRO DE PROVEEDORES

AGREGAR PROVEEDOR

Buscar por nombre de proveedor...

Nombre de proveedor	Acciones
DalePlus	

Nombre del Caso de Prueba

Eliminar el proveedor cuando no hay registros de compras dependientes

Descripción

Este caso de prueba verifica que el sistema permite eliminar un proveedor cuando no existen compras asociadas a él.

Criterios de Aceptación

1. El proveedor debe existir previamente en la base de datos.
2. No debe haber compras asociadas al proveedor.
3. El sistema debe eliminar el proveedor y redirigir a la vista de índice.
4. El proveedor eliminado no debe existir en la base de datos después de la operación.

Pasos de Ejecución:

1. Posicionarse en “Proveedores”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar eliminar un proveedor, en este caso; nombre: “DalePlus”.
4. Clic en Eliminar.
5. El sistema nos redirige al Listado y podemos observar que el proveedor ha sido eliminado.

ELIMINAR PROVEEDOR

¿Estás seguro que deseas eliminar este proveedor?

Nombre de proveedor	DalePlus
<input style="background-color: red; color: white; border: 1px solid red; padding: 5px 10px; border-radius: 5px; font-weight: bold; margin-right: 10px;" type="button" value="ELIMINAR"/> <input style="background-color: #ccc; color: black; border: 1px solid #ccc; padding: 5px 10px; border-radius: 5px; font-weight: bold;" type="button" value="CANCELAR"/>	

Nombre de proveedor	Acciones
Samsung	
Distribuidora El Sueño	
Almacenes La Moderna	
Carpintería Don Mario	
Mabe	

Nombre del Caso de Prueba

No eliminar el proveedor cuando hay registros de compras dependientes

Descripción

Este caso de prueba verifica que el sistema no permite eliminar un proveedor cuando existen compras asociadas a él.

Criterios de Aceptación

1. El proveedor debe existir previamente en la base de datos.
2. Debe haber compras asociadas al proveedor.
3. El sistema debe detectar la dependencia y no eliminar el proveedor.
4. El sistema debe devolver un mensaje de error indicando que el proveedor no puede ser eliminado debido a compras dependientes.

Pasos de Ejecución:

1. Posicionarse en “Proveedores”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar eliminar un proveedor, en este caso seleccionamos el proveedor “Carpintería Don Mario”, puesto que posee compras asociadas.
4. Clic en Eliminar.
5. El sistema nos muestra el mensaje de error: Un registro de compras utiliza este registro.

ELIMINAR PROVEEDOR

¿Estás seguro que deseas eliminar este proveedor?

Nombre de proveedor	Carpintería Don Mario
<input style="background-color: red; color: white; border: 1px solid black; padding: 5px; margin-right: 10px;" type="button" value="ELIMINAR"/> <input style="background-color: #f0f0f0; border: 1px solid black; padding: 5px;" type="button" value="CANCELAR"/>	

ELIMINAR PROVEEDOR

Error al eliminar: Un registro de compras utiliza este registro ×

¿Estás seguro que deseas eliminar este proveedor?

Nombre de proveedor	
<input style="background-color: red; color: white; border: 1px solid black; padding: 5px; margin-right: 10px;" type="button" value="ELIMINAR"/> <input style="background-color: #f0f0f0; border: 1px solid black; padding: 5px;" type="button" value="CANCELAR"/>	

Controlador: Empleados

Nombre del Caso de Prueba

Crear un empleado cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite crear un nuevo empleado correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. El empleado debe tener un nombre, apellido y porcentaje válidos.
2. El sistema debe agregar el nuevo empleado a la base de datos.
3. El sistema debe redirigir a la acción Index después de la creación exitosa.

Pasos de Ejecución:

1. Posicionarse en “Empleados”.
2. Dar clic en “Registrar nuevo empleado”.
3. Ingresar nombre, apellido y porcentaje valido. Nombre: “Jorge”, Apellido: “Pérez”, Porcentaje: “2”.
4. Clic en registrar y verificar en el listado que se ha creado correctamente.

REGISTRAR EMPLEADOS

Listado de Empleados / Registrar Empleados

Ingrese los datos solicitados del empleado.

Nombre de empleado

Apellido de empleado

Porcentaje de ganancia (%)

REGISTRAR **REGRESAR**

REGISTRO DE EMPLEADOS

AGREGAR EMPLEADO

Filtrar por nombre...	Filtrar por apellido...		
Nombre de empleado	Apellido de empleado	Porcentaje de ganancia (%)	Acciones
Jorge	Perez	2.00	

Nombre del Caso de Prueba

No crear un empleado cuando el porcentaje es inválido

Descripción

Este caso de prueba verifica que el sistema no permite crear un nuevo empleado correctamente cuando se proporciona un porcentaje inválido.

Criterios de Aceptación

1. El porcentaje sobrepasa el 5% de comisión.
2. El sistema no debe agregar el nuevo empleado a la base de datos.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Empleados”.
2. Dar clic en “Registrar nuevo empleado”.
3. Ingresar nombre, apellido y porcentaje valido. Nombre: “Jorge”, Apellido: “Pérez”, Porcentaje: “7”.
4. Verificar que el sistema muestra el mensaje de error: “El porcentaje no puede ser menor a 0% ni superar el 5%.”

REGISTRAR EMPLEADOS

[Lista de Empleados](#) / Registrar Empleados

Ingrese los datos solicitados del empleado.

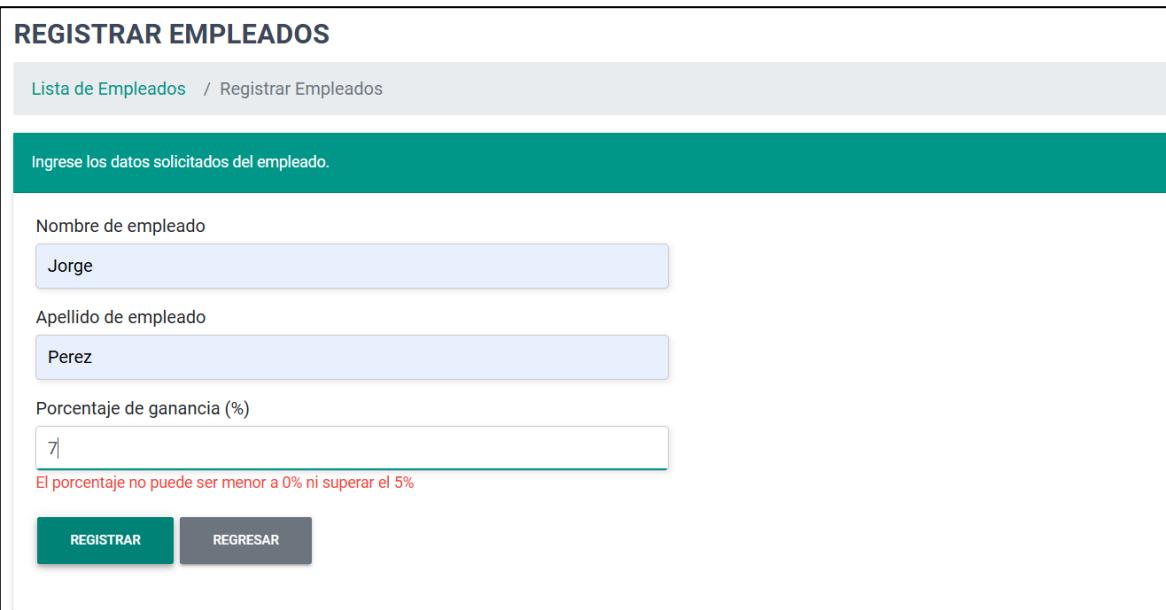
Nombre de empleado

Apellido de empleado

Porcentaje de ganancia (%)

El porcentaje no puede ser menor a 0% ni superar el 5%

REGISTRAR **REGRESAR**



Nombre del Caso de Prueba

No editar un empleado cuando el porcentaje es inválido

Descripción

Este caso de prueba verifica que el sistema no permite editar un empleado correctamente cuando se proporciona un porcentaje inválido.

Criterios de Aceptación

1. El porcentaje sobrepasa el 5% de comisión.
2. El sistema no debe actualizar el empleado en la base de datos.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Empleados”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar editar el empleado “Jorge Pérez”.
4. Ingresar un porcentaje inválido. Porcentaje: “7”.
5. Clic en Guardar Cambios.
6. Verificar que el sistema muestra el mensaje de error: “El porcentaje no puede ser menor a 0% ni superar el 5%.”

MODIFICAR INFORMACIÓN DE EMPLEADO

Lista de Empleados / Modificar Información de Empleado

Ingrese los datos solicitados del empleado.

Nombre de empleado

Apellido de empleado

Porcentaje de ganancia (%)

El porcentaje no puede ser menor a 0% ni superar el 5%

GUARDAR CAMBIOS **REGRESAR**

Nombre del Caso de Prueba

Editar un empleado cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar un empleado correctamente cuando se proporciona un porcentaje válido.

Criterios de Aceptación

1. El porcentaje debe estar entre 0% y 5%.
2. El sistema debe actualizar el empleado en la base de datos.
3. El sistema debe redirigir a la acción Index después de la edición exitosa.

Pasos de Ejecución:

1. Posicionarse en “Empleados”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar editar el empleado “Jorge Pérez”.
4. Ingresar un porcentaje válido. Porcentaje: “3”.
5. Clic en Guardar Cambios.
6. Verificar que el sistema redirige a la acción Index y que el empleado ha sido actualizado correctamente en la base de datos.

Ingrese los datos solicitados del empleado.

Nombre de empleado	<input type="text" value="Jorge"/>
Apellido de empleado	<input type="text" value="Perez"/>
Porcentaje de ganancia (%)	<input type="text" value="3"/>
<input type="button" value="GUARDAR CAMBIOS"/> <input type="button" value="REGRESAR"/>	

Filtrar por nombre... Filtrar por apellido...

Nombre de empleado	Apellido de empleado	Porcentaje de ganancia (%)	Acciones
Jorge	Perez	3.00	

Nombre del Caso de Prueba

Eliminar un empleado cuando no hay registros de ventas dependientes

Descripción

Este caso de prueba verifica que el sistema permite eliminar un empleado cuando no existen ventas asociadas a él.

Criterios de Aceptación

1. El empleado debe existir previamente en la base de datos.
2. No debe haber ventas asociadas al empleado.
3. El sistema debe eliminar el empleado y redirigir a la vista de índice.
4. El empleado eliminado no debe existir en la base de datos después de la operación.

Pasos de Ejecución:

1. Posicionarse en “Empleados”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar eliminar un empleado, en este caso; nombre: “Jorge Pérez”.
4. Clic en Eliminar.
5. Verificar que el sistema redirige al Listado y que el empleado ha sido eliminado correctamente.

ELIMINAR PROVEEDOR

¿Estás seguro que deseas eliminar este empleado?

Nombre de empleado	Jorge
Apellido de empleado	Perez
Porcentaje de ganancia (%)	3.00

ELIMINAR
Cancelar

Nombre de empleado	Apellido de empleado	Porcentaje de ganancia (%)	Acciones
Juan	Perez	1.50	
Pamela	Torres	1.00	
Patricia	Gomez	2.00	
Jonathan	Reyes	1.50	
Mardoqueo	Torres	3.00	

Nombre del Caso de Prueba

No eliminar el empleado cuando hay registros de ventas dependientes

Descripción

Este caso de prueba verifica que el sistema no permite eliminar un empleado cuando existen ventas asociadas a él.

Criterios de Aceptación

1. El empleado debe existir previamente en la base de datos.
2. Debe haber ventas asociadas al empleado.
3. El sistema debe detectar la dependencia y no eliminar el empleado.
4. El sistema debe devolver un mensaje de error indicando que el empleado no puede ser eliminado debido a ventas dependientes.

Pasos de Ejecución:

1. Posicionarse en “Empleados”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar eliminar un empleado, en este caso seleccionamos el empleado “Juan Pérez”, puesto que posee ventas asociadas.
4. Clic en Eliminar.
5. Verificar que el sistema muestra el mensaje de error: “Un registro de ventas utiliza este registro.”

ELIMINAR PROVEEDOR

¿Estás seguro que deseas eliminar este empleado?

Nombre de empleado	Juan
Apellido de empleado	Perez
Porcentaje de ganancia (%)	1.50

ELIMINAR **CANCELAR**

ELIMINAR PROVEEDOR

Error al eliminar: Un registro de ventas utiliza este registro X

¿Estás seguro que deseas eliminar este empleado?

Nombre de empleado	
Apellido de empleado	
Porcentaje de ganancia (%)	

ELIMINAR **CANCELAR**

Controlador: Productos

Nombre del Caso de Prueba

Ingresar un producto nuevo con información válida

Descripción

Este caso de prueba verifica que el sistema permite agregar un nuevo producto a la base de datos correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. El producto debe tener un nombre de categoría, nombre de producto, descripción, unidades disponibles y costo válidos.
2. El sistema debe agregar el nuevo producto a la base de datos.
3. El sistema debe redirigir a la acción Index después de la creación exitosa.

Pasos de Ejecución:

1. Posicionarse en “Productos”.
2. Dar clic en “Registrar nuevo producto”.
3. Ingresar nombre de categoría, nombre de producto, descripción, unidades disponibles y costo de un producto no existente. Ejemplo: Nombre de Categoría: “Muebles de Madera”, Nombre de Producto: “Mesa de Centro”, Descripción de Producto: “Madera procesada”, Unidades Disponibles: “10” y Costo de Producto: “40.00”
4. Clic en Registrar.
5. Verificar en el listado que el producto ha sido creado correctamente.

Ingrese la información del producto que desea agregar

Nombre de categoria

Nombre de producto

Descripcion de producto

Unidades disponibles

Costo del producto (\$)

INGRESAR
REGRESAR

Categoría	Nombre de producto	Descripción de producto	Unidades disponibles	Costo del producto (\$)	Acciones
Muebles de madera	Mesa de centro	Madera procesada	10	40.00	

Nombre del Caso de Prueba

Ingresar un producto que ya posee registro

Descripción

Este caso de prueba verifica que el sistema permite agregar un producto a la base de datos correctamente cuando se proporciona información válida, incluso si el producto ya existe, sumando las unidades nuevas a las existentes.

Criterios de Aceptación

1. El producto debe tener un nombre, descripción, unidades y costo válidos.
2. Si el producto ya existe en la base de datos, el sistema debe sumar las unidades nuevas a las existentes.
3. El sistema debe redirigir a la acción Index después de la creación exitosa.

Pasos de Ejecución:

1. Posicionarse en “Productos”.
2. Dar clic en “Registrar nuevo producto”.
3. Ingresar nombre de categoría, nombre de producto, descripción, unidades disponibles y costo de un producto existente. Ejemplo: Nombre de Categoría: “Muebles de Madera”, Nombre de Producto: “Mesa de Centro”, Descripción de Producto: “Madera procesada”, Unidades Disponibles: “5” y Costo de Producto: “40.00”.
4. Clic en Registrar.
5. Verificar en el listado que las unidades del producto han sido actualizadas correctamente (deberían ser 15 unidades en total).

Ingrese la información del producto que desea agregar

Nombre de categoria	Muebles de madera
Nombre de producto	Mesa de centro
Descripcion de producto	Madera procesada
Unidades disponibles	5
Costo del producto (\$)	40
INGRESAR	REGRESAR

Categoría	Nombre de producto	Descripción de producto	Unidades disponibles	Costo del producto (\$)	Acciones
Muebles de madera	Mesa de centro	Madera procesada	15	40.00	

Nombre del Caso de Prueba

No crear un producto cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite agregar un nuevo producto a la base de datos cuando se proporciona información inválida.

Criterios de Aceptación

1. El producto debe tener un nombre de categoría, nombre de producto, descripción, unidades disponibles y costo válidos.
2. El sistema no debe agregar el producto a la base de datos si la información es inválida.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Productos”.
2. Dar clic en “Registrar nuevo producto”.
3. Ingresar nombre de categoría, nombre de producto, descripción, unidades disponibles y costo de un producto con información inválida. Ejemplo: Nombre de Categoría: “Muebles de Madera”, Nombre de Producto: “”, Descripción de Producto: “”, Unidades Disponibles: “-20” y Costo de Producto: “-40.00”
4. Clic en Registrar.
5. Verificar que el sistema muestra el mensaje de error correspondiente, indicando que la información proporcionada es inválida.

The screenshot shows a web-based application for managing inventories. The top navigation bar includes links for 'Inventarios' and 'Iniciar Sesión'. The main page title is 'Inventarios / Ingresar Producto'. Below the title, a teal header bar contains the text 'Ingrese la información del producto que desea agregar'. The form fields are as follows:

- Nombre de categoría:** A text input field containing 'Muebles de madera'.
- Nombre de producto:** An empty text input field with a red error message below it: 'Este campo es requerido'.
- Descripción de producto:** An empty text input field with a red error message below it: 'Este campo es requerido'.
- Unidades disponibles:** A dropdown menu showing '-20' with a red error message below it: 'Las unidades deben ser mayores a 0'.
- Costo del producto (\$):** A text input field showing '-40' with a red error message below it: 'El costo unitario es invalido'.

At the bottom of the form are two buttons: a teal 'INGRESAR' button and a grey 'REGRESAR' button.

Nombre del Caso de Prueba

No editar un producto cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite editar un producto correctamente cuando se proporciona información inválida.

Criterios de Aceptación

1. El producto debe tener un nombre de categoría, nombre de producto, descripción, unidades disponibles y costo válidos.
2. El sistema no debe actualizar el producto en la base de datos si la información es inválida.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Productos”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar editar un producto existente, por ejemplo, “Mesa de Centro”.
4. Ingresar un producto con información inválida. Ejemplo: Nombre de Categoría: “Muebles de Madera”, Nombre de Producto: “Mesa de Centro”, Descripción de Producto: “”, Unidades Disponibles: “-20” y Costo de Producto: “-40.00”
5. Clic en Guardar Cambios.
6. Verificar que el sistema muestra el mensaje de error correspondiente, indicando que la información proporcionada es inválida.

MODIFICAR INVENTARIOS

Inventarios / Modificar Inventario

Ingrese la nueva información del producto

Nombre de categoria	Muebles de madera
Nombre de producto	Mesa de centro
Descripcion de producto	 Este campo es requerido
Unidades disponibles	-20 Las unidades deben ser mayores a 0
Costo del producto (\$)	-40 El costo unitario es invalido

GUARDAR CAMBIOS **REGRESAR**

Nombre del Caso de Prueba

Eliminar un producto cuando no hay registros de compras o ventas dependientes

Descripción

Este caso de prueba verifica que el sistema permite eliminar un producto cuando no existen compras o ventas asociadas a él.

Criterios de Aceptación

1. El producto debe existir previamente en la base de datos.
2. No debe haber compras o ventas asociadas al producto.
3. El sistema debe eliminar el producto y redirigir a la vista de índice.
4. El producto eliminado no debe existir en la base de datos después de la operación.

Pasos de Ejecución:

1. Posicionarse en “Productos”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar eliminar un producto, en este caso; nombre: “Mesa de Centro”.
4. Clic en Eliminar.
5. Verificar que el sistema redirige al Listado y que el producto ha sido eliminado correctamente.

ELIMINAR PRODUCTO

¿Estás seguro que deseas eliminar este producto?

Categoría	Muebles de madera
Nombre de producto	Mesa de centro
Descripción de producto	Madera procesada
Unidades disponibles	15
Costo del producto (\$)	40.00

ELIMINAR **CANCELAR**

INVENTARIOS

AGREGAR PRODUCTO

Filtrar por categoría... Filtrar por producto... Filtrar por descripción...

Categoría	Nombre de producto	Descripción de producto	Unidades disponibles	Costo del producto (\$)	Acciones

Nombre del Caso de Prueba

No eliminar un producto cuando hay registros de compras y ventas dependientes

Descripción

Este caso de prueba verifica que el sistema no permite eliminar un producto cuando existen compras y ventas asociadas a él.

Criterios de Aceptación

1. El producto debe existir previamente en la base de datos.
2. Debe haber compras y/o ventas asociadas al producto.
3. El sistema debe detectar la dependencia y no eliminar el producto.
4. El sistema debe devolver un mensaje de error indicando que el producto no puede ser eliminado debido a compras y ventas dependientes.

Pasos de Ejecución:

1. Posicionarse en “Productos”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar eliminar un producto, en este caso seleccionamos el producto “Closet 1.00m Café”, puesto que posee compras y ventas asociadas.
4. Clic en Eliminar.
5. Verificar que el sistema muestra el mensaje de error: “Un registro de compras o ventas utiliza este registro.”

ELIMINAR PRODUCTO

¿Estás seguro que deseas eliminar este producto?

Categoría	Muebles de madera
Nombre de producto	Closet
Descripción de producto	1.00 m Café
Unidades disponibles	23
Costo del producto (\$)	124.99

ELIMINAR **CANCELAR**

ELIMINAR PRODUCTO

Error al eliminar: Un registro de compras o ventas utiliza este registro X

¿Estás seguro que deseas eliminar este producto?

Categoría	
Nombre de producto	
Descripción de producto	
Unidades disponibles	
Costo del producto (\$)	

ELIMINAR **CANCELAR**

Controlador: Compras

Nombre del Caso de Prueba

Crear una nueva compra cuando el producto no existe

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva compra y un nuevo producto correctamente cuando se proporciona información válida y el producto no existe previamente en la base de datos.

Criterios de Aceptación

1. El producto no debe existir previamente en la base de datos.
2. La información proporcionada para el producto y la compra debe ser válida.
3. El sistema debe crear el nuevo producto y la nueva compra en la base de datos.
4. El sistema debe redirigir a la acción Index después de la creación exitosa.

Pasos de Ejecución:

1. Posicionarse en “Compras”.
2. Dar clic en “Registrar nueva compra”.
3. Ingresar la información de la compra y del producto no existente. Ejemplo: Fecha de compra: “07/10/2024”, Nombre de proveedor: “Samsung”, Nombre de categoría: “Electrodomésticos”, Nombre de producto: “Lavadora”, Descripción de producto: “16KG”, Unidades disponibles: “5” y Costo del producto (\$): “400.00”.
4. Clic en Registrar.
5. Verificar en el listado que la compra y el producto han sido creados correctamente.

INFORMACIÓN DE COMPRA							
Fecha de compra	07/10/2024						
Nombre de proveedor	Samsung						
INFORMACIÓN DEL PRODUCTO							
Nombre de categoría	Electrodomésticos						
Nombre de producto	Lavadora						
Descripción de producto	16KG						
Unidades disponibles	5						
Costo del producto (\$)	400						
7/10/2024	Samsung	Electrodomésticos	Lavadora	16KG	5	400.00	  

Nombre del Caso de Prueba

Crear una nueva compra cuando el producto ya existe

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva compra correctamente cuando se proporciona información válida y el producto ya existe previamente en la base de datos.

Criterios de Aceptación

1. El producto debe existir previamente en la base de datos.
2. La información proporcionada para la compra debe ser válida.
3. El sistema debe actualizar las unidades disponibles del producto existente y crear la nueva compra en la base de datos.
4. El sistema debe redirigir a la acción Index después de la creación exitosa.

Pasos de Ejecución:

1. Posicionarse en “Compras”.
2. Dar clic en “Registrar nueva compra”.
3. Ingresar la información de la compra y del producto existente. Ejemplo: Fecha de compra: “07/10/2024”, Nombre de proveedor: “Samsung”, Nombre de categoría: “Electrodomésticos”, Nombre de producto: “Lavadora”, Descripción de producto: “16KG”, Unidades disponibles: “10” y Costo del producto (\$): “400.00”.
4. Clic en Registrar.
5. Verificar en el listado que la compra ha sido creada correctamente y que las unidades del producto han sido actualizadas (sumando las nuevas unidades a las existentes).

REGISTRO DE COMPRAS							
						AGREGAR COMPRA	
dd/mm/aaaa	<input type="button" value=""/>	dd/mm/aaaa	<input type="button" value=""/>	Buscar categoría...	Buscar producto...	Buscar descripción...	<input type="button" value=""/>
Fecha de compra	Proveedor	Categoría	Nombre Producto	Descripción	Unidades	Costo(\$)	Acciones
7/10/2024	Samsung	Electrodomésticos	Lavadora	16KG	5	400.00	
8/10/2024	Samsung	Electrodomésticos	Lavadora	16KG	10	400.00	

INVENTARIOS					
				AGREGAR PRODUCTO	
Filtrar por categoría...		Filtrar por producto...		Filtrar por descripción...	
Categoría	Nombre de producto	Descripción de producto	Unidades disponibles	Costo del producto (\$)	Acciones
Electrodomésticos	Lavadora	16KG	15	400.00	

Nombre del Caso de Prueba

No crear una nueva compra cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva compra cuando se proporciona información inválida.

Criterios de Aceptación

1. La información proporcionada para el producto y la compra debe ser válida.
2. El sistema no debe crear el producto ni la compra en la base de datos si la información es inválida.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Compras”.
2. Dar clic en “Registrar nueva compra”.
3. Ingresar la información de la compra y del producto existente no válidos. Ejemplo: Fecha de compra: “7/10/2024”, Nombre de proveedor: “Samsung”, Nombre de categoría: “Electrodomésticos”, Nombre de producto: “”, Descripción de producto: “”, Unidades disponibles: “-10” y Costo del producto (\$): “-400.00”.
4. Clic en Registrar.
5. Verificar que el sistema muestra el mensaje de error correspondiente, indicando que la información proporcionada es inválida.

INFORMACIÓN DE COMPRA	
Fecha de compra	<input type="text" value="07/10/2024"/> 
Nombre de proveedor	<input type="text" value="Samsung"/>
INFORMACIÓN DEL PRODUCTO	
Nombre de categoría	<input type="text" value="Electrodomesticos"/>
Nombre de producto	<input type="text"/>
Este campo es requerido	
Descripción de producto	<input type="text"/>
Este campo es requerido	
Unidades disponibles	<input type="text" value="-5"/>
Las unidades deben ser mayores a 0	
Costo del producto (\$)	<input type="text" value="-400"/>
El costo unitario es invalido	

Nombre del Caso de Prueba

Editar una compra cuando la información es válida

Descripción

Este caso de prueba verifica que el sistema permite editar una compra correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. La información proporcionada para el producto y la compra debe ser válida.
2. El sistema debe actualizar la compra y el producto en la base de datos.
3. El sistema debe redirigir a la acción Index después de la edición exitosa.

Pasos de Ejecución:

1. Posicionarse en “Compras”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar editar una compra existente, por ejemplo, la compra con el producto “Lavadora”.
4. Ingresar la información válida para la compra y el producto. Ejemplo: Fecha de compra: “08/10/2024”, Nombre de proveedor: “Samsung”, Nombre de categoría: “Electrodomésticos”, Nombre de producto: “Lavadora”, Descripción de producto: “16KG”, Unidades compradas: “10” (actualizando las unidades) y Costo del producto (\$): “400.00”
5. Clic en Guardar Cambios.
6. Verificar que el sistema redirige a la acción Index y que la compra y el producto han sido actualizados correctamente en la base de datos.

INFORMACIÓN DE LA COMPRAS	
Fecha de compra	<input type="text" value="08/10/2024"/> <input type="button" value=""/>
Nombre de proveedor	<input type="text" value="Samsung"/>
Unidades compradas	<input type="text" value="10"/>
INFORMACIÓN DEL PRODUCTO	
Nombre de categoría	<input type="text" value="Electrodomésticos"/>
Nombre de producto	<input type="text" value="Lavadora"/>
Descripción de producto	<input type="text" value="16KG"/>
Costo del producto (\$)	<input type="text" value="400"/>

Nombre del Caso de Prueba

No editar una compra cuando las unidades son negativas

Descripción

Este caso de prueba verifica que el sistema no permite editar una compra correctamente cuando se proporciona un número negativo de unidades.

Criterios de Aceptación

1. La información proporcionada para el producto y la compra debe ser válida.
2. El sistema no debe actualizar la compra ni el producto en la base de datos si las unidades son negativas.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Compras”.
2. Dar clic en “Ver Listado”.
3. En Acciones, seleccionar editar una compra existente, por ejemplo, la compra con el producto “Lavadora”.
4. Ingresar la información válida para la compra y el producto. Ejemplo: Fecha de compra: “08/10/2024”, Nombre de proveedor: “Samsung”, Nombre de categoría: “Electrodomésticos”, Nombre de producto: “Lavadora”, Descripción de producto: “16KG”, Unidades compradas: “-10” (actualizando las unidades) y Costo del producto (\$): “400.00”
5. Clic en Guardar Cambios.
6. Verificar que el sistema muestra el mensaje de error correspondiente, indicando que las unidades no pueden ser negativas

INFORMACIÓN DE LA COMPRA	
Fecha de compra	<input type="text" value="08/10/2024"/> 
Nombre de proveedor	<input type="text" value="Samsung"/>
Unidades compradas	<input type="text" value="-10"/> <small>Las unidades deben ser mayores a 0</small>
INFORMACIÓN DEL PRODUCTO	
Nombre de categoría	<input type="text" value="Electrodomesticos"/>
Nombre de producto	<input type="text" value="Lavadora"/>
Descripción de producto	<input type="text" value="16KG"/>
Costo del producto (\$)	<input type="text" value="400"/>

Controlador: Ventas

Nombre del Caso de Prueba

Crear una nueva venta con información válida

Descripción

Este caso de prueba verifica que el sistema permite crear una nueva venta correctamente cuando se proporciona información válida.

Criterios de Aceptación

1. La información proporcionada para la venta debe ser válida.
2. El sistema debe agregar la nueva venta a la base de datos.
3. El sistema debe redirigir a la acción Index después de la creación exitosa.

Pasos de Ejecución:

1. Posicionarse en “Ventas”.
2. Dar clic en “Registrar nueva venta”.
3. Ingresar la información de la venta con datos válidos. Ejemplo: Fecha de venta: “08/10/2024”, Nombre de categoría: “Electrodomésticos”, Nombre de producto: “Lavadora”, Descripción de producto: “16KG”, Unidades vendidas: “5”, Precio venta unitario (\$): “500.00”, Valor venta total (\$): “2500.00” y Nombre de empleado: “Juan”.
4. Clic en Registrar.
5. Verificar en el listado que la venta ha sido creada correctamente.

Fecha de venta	<input type="text" value="08/10/2024"/>	<input type="button" value=""/>
Nombre de categoría	<input type="text" value="Electrodomésticos"/>	
Nombre de producto	<input type="text" value="Lavadora"/>	
Descripción de producto	<input type="text" value="16KG"/>	
Unidades vendidas	<input type="text" value="5"/>	
Precio venta unitario (\$)	<input type="text" value="500"/>	
Valor venta total (\$)	<input type="text" value="2500.00"/>	
Nombre de empleado	<input type="text" value="Juan"/>	

Fecha de venta	Categoría	Producto	Descripción	Unidades vendidas	Total (\$)	Empleado	Comisión (\$)	Acciones
7/10/2024	Electrodomésticos	Lavadora	16KG	5	2500.00	Juan	37.5	

Nombre del Caso de Prueba

No crear una nueva venta cuando la información es inválida

Descripción

Este caso de prueba verifica que el sistema no permite crear una nueva venta cuando se proporciona información inválida.

Criterios de Aceptación

1. La información proporcionada para la venta debe ser válida.
2. El sistema no debe agregar la venta a la base de datos si la información es inválida.
3. El sistema debe mostrar un mensaje de error respectivo.

Pasos de Ejecución:

1. Posicionarse en “Ventas”.
2. Dar clic en “Registrar nueva venta”.
3. Ingresar la información de la venta con datos inválidos. Ejemplo: Fecha de venta: “05/10/2024”, Nombre de categoría: “Electrodomésticos”, Nombre de producto: “Lavadora”, Descripción de producto: “16KG”, Unidades vendidas: “-5”, Precio venta unitario (\$): “-500.00”, Valor venta total (\$): “-2500.00” y Nombre de empleado: “Juan”.
4. Clic en Registrar.
5. Verificar que el sistema muestra el mensaje de error correspondiente, indicando que la información proporcionada es inválida.

Ingrese la nueva información de la venta

Fecha de venta	<input type="text" value="07/10/2024"/>
Nombre de categoría	<input type="text" value="Muebles de madera"/>
Nombre de producto	<input type="text" value="Closet"/>
Descripción de producto	<input type="text" value="1.00 m Cafe"/>
Unidades vendidas	<input type="text" value="-5"/>
Las unidades vendidas deben ser mayores a 0	
Precio venta unitario (\$)	<input type="text" value="-500.00"/>
El costo unitario es invalido	
Valor venta total (\$)	<input type="text" value="2500.00"/>
Nombre de empleado	<input type="text" value="Juan"/>
<input type="button" value="GUARDAR CAMBIOS"/> <input type="button" value="REGRESAR"/>	

Buenas prácticas

El desarrollo del proyecto cumple con una serie de buenas prácticas con la intención de alcanzar un alto nivel de calidad, estas buenas prácticas son:

- **Comprendión en el código:** cuenta con nombres de variables descriptivas, es ordenado y fácilmente entendible.
- **Pruebas unitarias:** Se efectúan una cantidad considerables de pruebas unitarias por cada modelo del proyecto, esto con la finalidad de garantizar un funcionamiento adecuado.
- **Revisión de código:** El equipo de trabajo realiza revisiones del código para garantizar su adecuado funcionamiento y se realizan pruebas manuales arbitrarias con la misma finalidad.
- **Seguridad:** Se cuenta con validación en los campos de entrada y las opciones disponibles para el usuario, así también, se cuenta con encriptación para las contraseñas de los usuarios disponibles en el sistema.

```
// POST: Productos/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("ID,IdCategoria,NombreProducto,DescripcionProducto,UnidadesProducto,CostoProducto")] Producto producto)
{
    ModelState.Remove("Categoria");

    if (ModelState.IsValid)
    {
        // Verificar si el producto ya existe con todos los atributos
        var existingProduct = _context.Productos
            .FirstOrDefault(p => p.NombreProducto == producto.NombreProducto &&
                               p.IdCategoria == producto.IdCategoria &&
                               p.DescripcionProducto == producto.DescripcionProducto &&
                               p.CostoProducto == producto.CostoProducto);

        if (existingProduct != null)
        {
            // Si el producto ya existe, suma las unidades
            existingProduct.UnidadesProducto += producto.UnidadesProducto;
            _context.Update(existingProduct);
        }
        else
        {
            // Si el producto no existe, lo agrega como nuevo
            _context.Add(producto);
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
}

ViewData["IdCategoria"] = new SelectList(_context.Categorias, "ID", "NombreCategoria", producto.IdCategoria);
return View(producto);
```

```

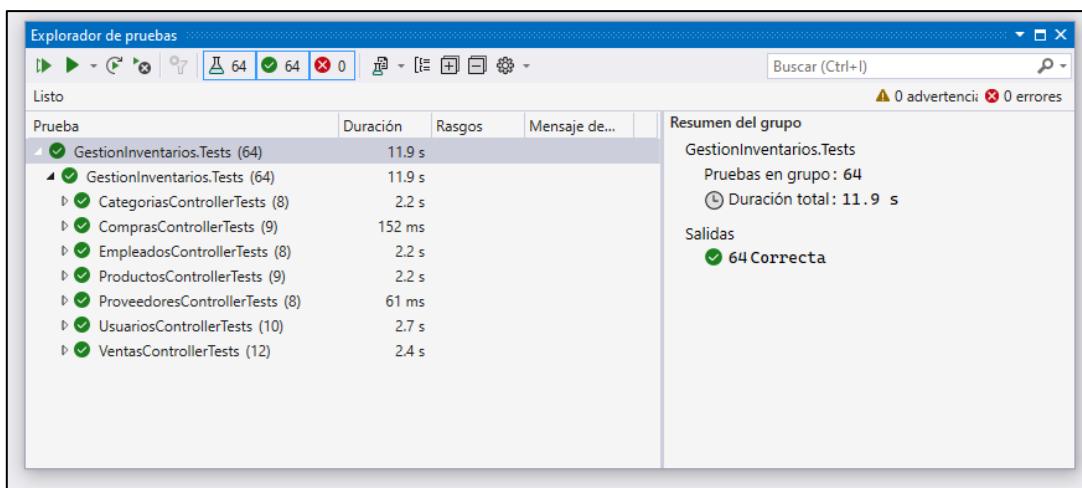
var usuario = await _context.Usuarios.FindAsync(model.ID);
if (usuario == null)
{
    return NotFound();
}

// Verifica si el correo ya está registrado por otro usuario no permite ingresarlo
var usuarioConMismoCorreo = await _context.Usuarios
    .Where(u => u.CorreoUsuario == model.CorreoUsuario && u.ID != model.ID)
    .FirstOrDefaultAsync();

if (usuarioConMismoCorreo != null)
{
    ViewBag.Error = "Error: El correo brindado ya posee cuenta registrada.";
    return View(model);
}

if (!string.IsNullOrEmpty(model.NuevaClaveUsuario) && model.NuevaClaveUsuario == model.ConfirmarNuevaClaveUsuario)
{
    // Verificar la contraseña antigua
    if (BCrypt.Net.BCrypt.Verify(model.ClaveAntigua, usuario.ClaveUsuario))
    {
        // Hashear la nueva contraseña
        usuario.ClaveUsuario = BCrypt.Net.BCrypt.HashPassword(model.NuevaClaveUsuario);
    }
    else
    {
        ViewBag.Error = "Error: La contraseña antigua es incorrecta.";
        return View(model);
    }
}
else if (!string.IsNullOrEmpty(model.NuevaClaveUsuario))
{
    ViewBag.Error = "Error: Las nuevas contraseñas no coinciden.";
    return View(model);
}

```



Seguridad

Se toman medidas de seguridad con el objetivo de reducir al máximo las posibles vulnerabilidades; estas medidas son:

- **Validación de entradas de datos:** Se utiliza “Data Annotations” en los modelos para validar el ingreso de datos por parte del usuario y prevenir ataques mediante inyección SQL.
- **Cifrado de contraseñas:** Al contar con un registro de usuarios, el almacenamiento de contraseñas se realiza posteriormente al cifrado de las mismas, de tal modo que se transportan seguras hacia la base de datos.
- **Administración de roles:** El proyecto cuenta con 2 tipos de usuarios; usuario “Administrador” el cual tiene nivel de acceso 1, y usuario “común” el cual tiene un nivel de acceso 2, esto permite la separación de vistas y control de acceso a funciones críticas, además permite una escalabilidad controlada del programa.

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ProyectoDES_Empresa.Models
{
    20 referencias
    public class Usuario
    {
        20 referencias | 0/4/4 pasando
        public int ID { get; set; }

        [Required(ErrorMessage = "Este campo es requerido")]
        [RegularExpression(@"^([a-zA-Z0-9_.%]+@[a-zA-Z0-9_.%]+\.[a-zA-Z]{2,}?$", ErrorMessage = "El formato del correo electrónico no es válido.")]
        [Display(Name ="Correo de usuario")]
        23 referencias | 0/7/7 pasando
        public string CorreoUsuario { get; set; }

        [Required(ErrorMessage = "Este campo es requerido")]
        14 referencias | 0/7/7 pasando
        public string ClaveUsuario { get; set; }

        [ForeignKey("Rol")]
        11 referencias | 0/6/6 pasando
        public int? IdRol { get; set; }

        2 referencias
        public Rol Rol { get; set; }
    }
}

```



Rendimiento

En tema de rendimiento, el proyecto utiliza una combinación de JavaScript y AJAX, lo cual proporciona una experiencia de usuario dinámica y más interactiva, puesto que no es necesario recargar la página para actualizar algunos datos, por tal motivo se reducen las interrupciones.

Otro punto importante, es que la combinación de JavaScript y AJAX optimiza la transferencia de datos, puesto que se puede seleccionar y enviar únicamente los datos que son necesarios sin cargar de más el tráfico de red, esto es beneficioso puesto que es un proyecto que ejecuta la solicitud de datos en tiempo real de cara al cliente.

Escalabilidad

En tema de escalabilidad, al tratarse de una arquitectura MVC, el proyecto permite la incorporación de vistas que brinden un mejor flujo de trabajo y sea más llamativo para el usuario, así también agregar nuevos modelos, lo cual, amplia la lógica del negocio y se adapta al crecimiento del mismo, en este sentido permite ideas a futuro tales como:

- **Calculadora de cobros y pagos pendientes**
- **Control de planilla**
- **Análisis de datos**
- **Reportes selectivos**