Ejercicio 8/9

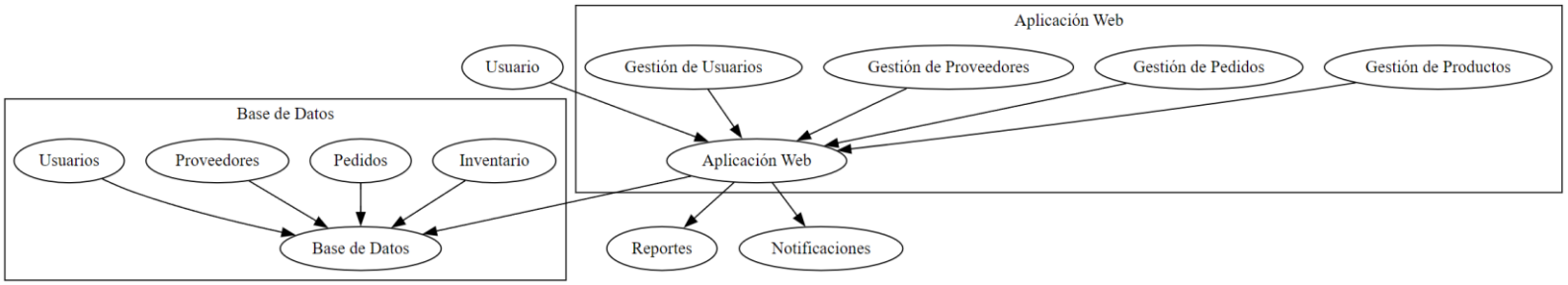
1. **Análisis de Requerimientos:**
   Ejercicio 1: Un cliente te solicita una aplicación web para gestionar su inventario. Define los requisitos funcionales y no funcionales del sistema.
   Requisitos Funcionales:
   - Permitir el registro y actualización de productos en el inventario, incluyendo campos como nombre, descripción, cantidad, precio, proveedor, etc.
   - Implementar funcionalidades de búsqueda y filtrado de productos en el inventario.
   - Permitir la generación de informes y reportes del inventario, como listados de productos, niveles de stock, valor total del inventario, etc.
   - Implementar flujos de pedidos y compras de productos, incluyendo la generación de órdenes de compra.
   - Permitir la gestión de proveedores, incluyendo datos de contacto e historial de compras.
   - Implementar alertas y notificaciones cuando los niveles de stock de un producto lleguen a un mínimo definido.
   - Permitir la asignación de permisos y roles de usuario para acceder a las diferentes funcionalidades.

   Requisitos No Funcionales:
   - La aplicación debe ser accesible a través de un navegador web moderno, con un diseño responsive que se adapte a diferentes dispositivos.
   - El sistema debe ser escalable y soportar un número creciente de usuarios y productos en el inventario.
   - La información del inventario debe estar protegida con mecanismos de seguridad como autenticación de usuarios, encriptación de datos y respaldos periódicos.
   - El tiempo de respuesta de la aplicación debe ser óptimo, con tiempos de carga y procesamiento de datos aceptables para el usuario.
   - La interfaz de usuario debe ser intuitiva y fácil de usar, siguiendo principios de diseño de experiencia de usuario.
   - El sistema debe estar desarrollado utilizando tecnologías y frameworks web modernos, con un mantenimiento y actualización sencillos.
   - La aplicación debe cumplir con los estándares y regulaciones aplicables al manejo de inventarios y datos de clientes.

Ejercicio 2: Redacta un caso de uso para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.

Caso de Uso	Agregar un nuevo producto
Actores	Usuario-Administrador
Descripción	Este caso de uso describe el proceso para que un usuario administrador pueda agregar un nuevo producto al inventario de la aplicación web.
Pre-Condición	El Usuario-Administrador necesita haber iniciado sesión y tener permisos para poder agregar un nuevo producto.
Escenario Principal	1. El usuario administrador selecciona la opción "Agregar Producto" en el menú principal de la aplicación. 2. El sistema muestra un formulario para capturar los datos del nuevo producto, incluyendo campos como: • Nombre del producto • Descripción • Categoría • Cantidad en stock • Precio unitario • Proveedor • Datos de contacto del proveedor 3. El usuario administrador completa los campos requeridos en el formulario y hace clic en "Guardar". 4. El sistema valida que los datos ingresados sean correctos y cumplan con los requisitos establecidos. 5. Si la validación es exitosa, el sistema guarda el nuevo producto en el inventario y muestra un mensaje de confirmación al usuario.
Flujos Alternativos	2ª. Si el usuario administrador decide cancelar la operación, puede hacer clic en "Cancelar" y el sistema cierra el formulario sin guardar el producto. 4ª. Si alguno de los datos ingresados no cumple con los requisitos, el sistema muestra un mensaje de error indicando los campos que deben ser corregidos. 5ª. Si la validación falla, el sistema muestra un mensaje de error indicando los campos que deben ser corregidos.
Requisitos	• El sistema debe validar que el nombre del producto no se repita en el inventario. • El sistema debe permitir la carga de imágenes o archivos adjuntos relacionados con el producto. • El sistema debe generar automáticamente un código único de identificación para cada producto.
Post-Condición	El Usuario-Administrador ha agregado un nuevo producto con éxito.

2. **Diseño del Sistema:**
   Ejercicio 3: Elabora un diagrama de flujo de datos para la aplicación web del ejercicio 1.
   - El Usuario interactúa con la Aplicación Web para realizar diversas funcionalidades.
   - La Aplicación Web gestiona los diferentes módulos:
     - Gestión de Productos: Permite administrar los productos del inventario.
     - Gestión de Pedidos: Permite administrar los pedidos y órdenes de compra.
     - Gestión de Proveedores: Permite administrar la información de los proveedores.
     - Gestión de Usuarios: Permite administrar los perfiles y permisos de los usuarios.
   - Cada módulo de la Aplicación Web almacena sus respectivos datos en la Base de Datos, la cual contiene las entidades de Inventario, Pedidos, Proveedores y Usuarios.
   - La Aplicación Web también genera Reportes y Notificaciones a partir de la información almacenada en la Base de Datos.



Ejercicio 4: Diseña la interfaz de usuario para la pantalla de "Inicio" de la aplicación web del ejercicio 1.

- Barra de Navegación: Proporciona acceso rápido a las funciones principales de la aplicación. El usuario puede navegar fácilmente entre las diferentes secciones, como agregar, modificar, eliminar productos y ver el inventario.

- Encabezado de la Página: Título y subtítulo que proporciona una visión general amigable y motivadora de la aplicación.

- Sección Principal: Contiene tres paneles:
    - Panel de Resumen Rápido: Ofrece una vista general de la situación actual del inventario con indicadores clave.
    - Panel de Acciones Rápidas: Proporciona accesos directos a las acciones más comunes que el usuario necesita realizar.
    - Panel de Inventario Reciente: Muestra una tabla con los productos más recientes, facilitando la revisión rápida de las últimas adiciones o cambios.

- Pie de Página: Contiene enlaces útiles y la información de derechos de autor, proporcionando recursos adicionales y legales al usuario.

3. **Diseño del Programa:**

Ejercicio 5: Elige una arquitectura adecuada para la aplicación web del ejercicio 1 y justifica tu elección.

Una arquitectura adecuada para esta aplicación web sería una arquitectura de n-capas. Esta arquitectura separa la aplicación en diferentes capas lógicas, como la capa de presentación (interfaz de usuario), la capa de lógica de negocio y la capa de acceso a datos. Esta separación de responsabilidades facilita el mantenimiento, la escalabilidad y la seguridad de la aplicación. Algunas ventajas de utilizar una arquitectura de n-capas para esta aplicación web son:

- Escalabilidad: Cada capa puede escalarse de manera independiente según las necesidades de la aplicación, lo que permite un crecimiento gradual y eficiente.
- Modularidad: La separación en capas permite un desarrollo y despliegue más modular, lo que facilita la actualización y el mantenimiento de la aplicación.
- Seguridad: Cada capa puede tener sus propios mecanismos de seguridad, como la autenticación y autorización de usuarios, el cifrado de datos, etc.
- Rendimiento: La separación de responsabilidades en capas permite optimizar el rendimiento de cada parte de la aplicación de manera independiente.

Ejercicio 6: Diseña la base de datos para la aplicación web del ejercicio 1.

```
CREATE TABLE Proveedores (
 id_proveedor INT AUTO_INCREMENT PRIMARY KEY,
 nombre_proveedor VARCHAR(100),
 telefono VARCHAR(20),
 email VARCHAR(50),
 direccion VARCHAR(200)
);

CREATE TABLE Productos (
 id_producto INT AUTO_INCREMENT PRIMARY KEY,
 nombre_producto VARCHAR(100),
 descripcion TEXT,
 categoria VARCHAR(50),
 cantidad_stock INT,
 precio_unitario DECIMAL(10,2),
 id_proveedor INT,
 FOREIGN KEY (id_proveedor) REFERENCES Proveedores(id_proveedor)
);
```

4. **Diseño:**

Utilizando los siguientes diagramas resuelva los casos de usos de los ejercicios 7 y 8:

- **Diagrama de Dominio: Identifica las entidades, atributos y relaciones del sistema.**
- **Diagrama de Robustez: Analiza cómo el sistema responde a diferentes escenarios de uso.**
- **Prototipo: Crea una versión simplificada del sistema para probar la usabilidad y funcionalidad.**
- **Diagrama de Secuencia: Describe la interacción entre los diferentes objetos del sistema.**
- **Diagrama de Clases: Define las clases, sus atributos, métodos y relaciones.**

Ejercicio 7: Implementa la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1 utilizando el lenguaje de programación de tu preferencia.

Código:

```
//MÉTODO PARA AGREGAR UN PRODUCTO
public void AgregarProducto {
    GuardarProducto(nombreProducto, descripcion, categoria, cantidadStock, precioUnitario,
    idProveedor, contactoProveedor);
}

//GUARDAR EL PRODUCTO EN EL SQL
public void GuardarProducto(string nombreProducto, string descripcion, string categoria, int
cantidadStock, decimal precioUnitario, int idProveedor, string contactoProveedor)
{
  // Consultar el ID del proveedor en la base de datos
  int proveedorId = ObtenerIdProveedor(idProveedor);

  // Insertar el nuevo producto en la base de datos
  string query = "INSERT INTO Productos (nombre_producto, descripcion, categoria,
cantidad_stock, precio_unitario, id_proveedor, datos_contacto_proveedor) " +
          "VALUES (@nombreProducto, @descripcion, @categoria, @cantidadStock,
@precioUnitario, @idProveedor, @contactoProveedor)";

  using (SqlConnection connection = new SqlConnection(connectionString))
  {
    connection.Open();
    using (SqlCommand command = new SqlCommand(query, connection))
    {
      command.Parameters.AddWithValue("@nombreProducto", nombreProducto);
      command.Parameters.AddWithValue("@descripcion", descripcion);
      command.Parameters.AddWithValue("@categoria", categoria);
      command.Parameters.AddWithValue("@cantidadStock", cantidadStock);
      command.Parameters.AddWithValue("@precioUnitario", precioUnitario);
      command.Parameters.AddWithValue("@idProveedor", proveedorId);
      command.Parameters.AddWithValue("@contactoProveedor", contactoProveedor);
      command.ExecuteNonQuery();
    }
  }
}
```

Pseudocódigo:

// MÉTODO PARA AGREGAR UN PRODUCTO

Metodo AgregarProducto

Llamar a GuardarProducto con (nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, idProveedor, contactoProveedor)

// GUARDAR EL PRODUCTO EN LA BASE DE DATOS SQL

Metodo GuardarProducto(nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, idProveedor, contactoProveedor)

// Consultar el ID del proveedor en la base de datos

proveedorId = ObtenerIdProveedor(idProveedor)

// Preparar la consulta para insertar el nuevo producto

query = "INSERTAR EN Productos (nombre_producto, descripcion, categoria, cantidad_stock, precio_unitario, id_proveedor, datos_contacto_proveedor) " +

"VALORES (nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, proveedorId, contactoProveedor)"

// Abrir conexión a la base de datos

AbrirConexion(connectionString)

// Ejecutar la consulta con los parámetros proporcionados

EjecutarComando(query, nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, proveedorId, contactoProveedor)

// Cerrar conexión a la base de datos

CerrarConexion()

Fin Metodo

Ejercicio 8: Implementa la lógica de negocio para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.

Código:

```
public class ProductManager
{
  private readonly IProductRepository _productRepository;
  private readonly IProviderRepository _providerRepository;

  public ProductManager(IProductRepository productRepository, IProviderRepository providerRepository)
  {
    _productRepository = productRepository;
    _providerRepository = providerRepository;
  }

  public bool AddProduct(string name, string description, string category, int stock, decimal price, int providerId, string providerContact)
  {
    // Validar los datos del producto
    if (string.IsNullOrEmpty(name) || string.IsNullOrEmpty(description) || string.IsNullOrEmpty(category) || stock < 0 || price <= 0 || providerId <= 0 || string.IsNullOrEmpty(providerContact))
    {
      return false;
    }
    // Verificar que el proveedor existe
    Provider provider = _providerRepository.GetProviderById(providerId);
    if (provider == null)
    { return false;}

    // Crear el nuevo producto
    Product newProduct = new Product
    {
      Name = name,
      Description = description,
      Category = category,
      Stock = stock,
      Price = price,
      ProviderId = providerId,
      ProviderContact = providerContact
    };

    // Guardar el producto en la base de datos
    _productRepository.AddProduct(newProduct);
    return true;
  }
}
```

Pseudocódigo:

```
Clase ProductManager
  Atributos:
    _productRepository
    _providerRepository

  Constructor ProductManager(productRepository, providerRepository)
    _productRepository = productRepository
    _providerRepository = providerRepository

  Método Booleano AddProduct(nombre, descripcion, categoria, stock, precio, idProveedor, contactoProveedor)
    // Validar los datos del producto
    Si nombre está vacío O descripcion está vacía O categoria está vacía O stock < 0 O precio <= 0 O idProveedor <= 0 O contactoProveedor está vacío Entonces
      retornar Falso

    // Verificar que el proveedor existe
    proveedor = _providerRepository.GetProviderById(idProveedor)
    Si proveedor es nulo Entonces
      retornar Falso

    // Crear el nuevo producto
    nuevoProducto = CrearObjeto Product
      Name = nombre
      Description = descripcion
      Category = categoria
      Stock = stock
      Price = precio
      ProviderId = idProveedor
      ProviderContact = contactoProveedor

    // Guardar el producto en la base de datos
    _productRepository.AddProduct(nuevoProducto)
    retornar Verdadero
Fin Clase
```

5. **Pruebas:**

Ejercicio 9: Define un conjunto de pruebas unitarias para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.

Código:

```
//MÉTODO PARA AGREGAR UN PRODUCTO
public void AgregarProducto {
    if(ValidarDatosProducto(nombreProducto, descripcion, categoria, cantidadStock,
    precioUnitario, proveedor, contactoProveedor)) {
            GuardarProducto(nombreProducto, descripcion, categoria, cantidadStock,
            precioUnitario, idProveedor, contactoProveedor);
    }
}

//PRUEBAS
public bool ValidarDatosProducto(string nombreProducto, string descripcion, string categoria,
int cantidadStock, decimal precioUnitario, string proveedor, string contactoProveedor)
  {
    return ValidarNombreProducto(nombreProducto) &&
        ValidarDescripcion(descripcion) &&
        ValidarCategoria(categoria) &&
        ValidarCantidadStock(cantidadStock) &&
        ValidarPrecioUnitario(precioUnitario) &&
        ValidarProveedor(proveedor) &&
        ValidarContactoProveedor(contactoProveedor);
  }

// Valida que el nombre del producto no esté vacío.
  private bool ValidarNombreProducto(string nombreProducto)
  { return !string.IsNullOrEmpty(nombreProducto); }

// Valida que la descripción no esté vacía.
  private bool ValidarDescripcion(string descripcion)
  { return !string.IsNullOrEmpty(descripcion); }

// Valida que la categoría no esté vacía
  private bool ValidarCategoria(string categoria)
  { return !string.IsNullOrEmpty(categoria); }

// Valida que la cantidad en stock sea mayor o igual a 0.
  private bool ValidarCantidadStock(int cantidadStock)
  { return cantidadStock >= 0; }

//Valida que el precio unitario sea mayor a 0.
  private bool ValidarPrecioUnitario(decimal precioUnitario)
  { return precioUnitario > 0; }
```

```
//Valida que el proveedor no esté vacío.
   private bool ValidarProveedor(string proveedor)
   { return !string.IsNullOrEmpty(proveedor); }

//Valida que el contacto del proveedor no esté vacío.
   private bool ValidarContactoProveedor(string contactoProveedor)
   { return !string.IsNullOrEmpty(contactoProveedor); }

//GUARDAR EL PRODUCTO EN EL SQL
public void GuardarProducto(string nombreProducto, string descripcion, string categoria, int cantidadStock, decimal precioUnitario, int idProveedor, string contactoProveedor)
{
   // Consultar el ID del proveedor en la base de datos
   int proveedorId = ObtenerIdProveedor(idProveedor);

   // Insertar el nuevo producto en la base de datos
   string query = "INSERT INTO Productos (nombre_producto, descripcion, categoria, cantidad_stock, precio_unitario, id_proveedor, datos_contacto_proveedor) " +
                  "VALUES (@nombreProducto, @descripcion, @categoria, @cantidadStock, @precioUnitario, @idProveedor, @contactoProveedor)";

   using (SqlConnection connection = new SqlConnection(connectionString))
   {
      connection.Open();
      using (SqlCommand command = new SqlCommand(query, connection))
      {
         command.Parameters.AddWithValue("@nombreProducto", nombreProducto);
         command.Parameters.AddWithValue("@descripcion", descripcion);
         command.Parameters.AddWithValue("@categoria", categoria);
         command.Parameters.AddWithValue("@cantidadStock", cantidadStock);
         command.Parameters.AddWithValue("@precioUnitario", precioUnitario);
         command.Parameters.AddWithValue("@idProveedor", proveedorId);
         command.Parameters.AddWithValue("@contactoProveedor", contactoProveedor);
         command.ExecuteNonQuery();
      }
   }
}
```

Pseudocódigo:

```
// MÉTODO PARA AGREGAR UN PRODUCTO
Metodo AgregarProducto
  Si ValidarDatosProducto(nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, proveedor, contactoProveedor) Entonces
    Llamar a GuardarProducto(nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, idProveedor, contactoProveedor)
  Fin Si
Fin Metodo

// VALIDACIÓN DE DATOS
Metodo Booleano ValidarDatosProducto(nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, proveedor, contactoProveedor)
  retornar ValidarNombreProducto(nombreProducto) Y
      ValidarDescripcion(descripcion) Y
      ValidarCategoria(categoria) Y
      ValidarCantidadStock(cantidadStock) Y
      ValidarPrecioUnitario(precioUnitario) Y
      ValidarProveedor(proveedor) Y
      ValidarContactoProveedor(contactoProveedor)
Fin Metodo

// Valida que el nombre del producto no esté vacío.
Metodo Booleano ValidarNombreProducto(nombreProducto)
  retornar nombreProducto no está vacío
Fin Metodo

// Valida que la descripción no esté vacía.
Metodo Booleano ValidarDescripcion(descripcion)
  retornar descripcion no está vacía
Fin Metodo

// Valida que la categoría no esté vacía.
Metodo Booleano ValidarCategoria(categoria)
  retornar categoria no está vacía
Fin Metodo

// Valida que la cantidad en stock sea mayor o igual a 0.
Metodo Booleano ValidarCantidadStock(cantidadStock)
  retornar cantidadStock >= 0
Fin Metodo

// Valida que el precio unitario sea mayor a 0.
Metodo Booleano ValidarPrecioUnitario(precioUnitario)
  retornar precioUnitario > 0
Fin Metodo
```

```
// Valida que el proveedor no esté vacío.
Metodo Booleano ValidarProveedor(proveedor)
  retornar proveedor no está vacío
Fin Metodo

// Valida que el contacto del proveedor no esté vacío.
Metodo Booleano ValidarContactoProveedor(contactoProveedor)
  retornar contactoProveedor no está vacío
Fin Metodo

// GUARDAR EL PRODUCTO EN LA BASE DE DATOS SQL
Metodo GuardarProducto(nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, idProveedor, contactoProveedor)
  // Consultar el ID del proveedor en la base de datos
  proveedorId = ObtenerIdProveedor(idProveedor)

  // Preparar la consulta para insertar el nuevo producto
  query = "INSERTAR EN Productos (nombre_producto, descripcion, categoria, cantidad_stock, precio_unitario, id_proveedor, datos_contacto_proveedor) " +
      "VALORES (nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, proveedorId, contactoProveedor)"

  // Abrir conexión a la base de datos
  AbrirConexion(connectionString)

  // Ejecutar la consulta con los parámetros proporcionados
  EjecutarComando(query, nombreProducto, descripcion, categoria, cantidadStock, precioUnitario, proveedorId, contactoProveedor)

  // Cerrar conexión a la base de datos
  CerrarConexion()
Fin Metodo
```

Ejercicio 10: Ejecuta pruebas de integración para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.
Todo correcto.

6. **Despliegue del Programa:**
   Ejercicio 11: Definir un plan de despliegue para la aplicación web del ejercicio 1.
   - Asegúrate de tener todas las credenciales y permisos necesarios.
   - Despliega la aplicación web en un entorno de pre-producción que sea lo más similar posible al entorno de producción. Realiza pruebas exhaustivas para validar el correcto funcionamiento de la aplicación, incluyendo pruebas de integración, pruebas de usuario y pruebas de carga. Corrige cualquier problema o bug encontrado durante las pruebas.
   - Prepara un anuncio o comunicado para informar a los usuarios o clientes sobre el lanzamiento de la nueva versión de la aplicación web.
   - Coordina con el equipo de marketing o ventas para que estén preparados para atender cualquier inquietud o solicitud de los usuarios.

Ejercicio 12: Despliega la aplicación web del ejercicio 1 en un servidor de producción.

7. **Mantenimiento:**

Ejercicio 13: Definir un plan de mantenimiento para la aplicación web del ejercicio 1.

Para la aplicación web, un buen plan de mantenimiento será aquél que tome esté constituido por 3 acciones fundamentales:

- Programar tareas periódicas de mantenimiento, como:
  - Realizar actualizaciones de seguridad del sistema operativo, frameworks, librerías y otros componentes.
  - Optimizar y mantener la base de datos, incluyendo respaldos, compactación y limpieza de datos obsoletos.
  - Revisar y actualizar la configuración de la infraestructura, como servidores, balanceadores de carga y redes.
- Monitoriza constantemente los comentarios y solicitudes de los usuarios:
  - Monitorear constantemente los comentarios y solicitudes de los usuarios para identificar oportunidades de mejora.
  - Planificar y priorizar las mejoras y actualizaciones en función de su impacto y beneficio para los usuarios.
- Establecer un calendario de lanzamiento de actualizaciones y comunicarlo a los usuarios de manera oportuna.

Ejercicio 14: Implementa una corrección de errores para un problema detectado en la aplicación web del ejercicio 1.

En caso de que se presente un problema dentro de la aplicación web, entonces lo que se deberá hacer son los siguientes pasos:

1. Investigación y análisis del problema:
   - Revisar los registros de errores (logs) de la aplicación para identificar el momento exacto en el que se produce el cierre.
   - Analizar el código fuente relacionado con la funcionalidad de agregar productos para detectar posibles causas, tales como:
     - Excepciones no controladas
     - Problemas de acceso a la base de datos
     - Errores en la lógica de negocio

2. Reproducción y depuración del problema:
   - Intentar reproducir el problema en un entorno de desarrollo o pruebas.
   - Identificar el punto exacto donde ocurre el cierre de la aplicación.

3. Corrección del error:
   - Una vez identificada la causa raíz del problema, implementar los cambios necesarios en el código fuente para corregir el error. Algunos pasos comunes pueden ser:
     - Agregar bloques try-catch para capturar y manejar las excepciones.
     - Verificar la integridad y disponibilidad de los recursos (como la conexión a la base de datos) antes de realizar operaciones.
     - Mejorar la lógica de negocio para evitar condiciones que puedan provocar el cierre de la aplicación.

4. Pruebas y validación:
    - Realizar pruebas exhaustivas en un entorno de pre-producción para verificar que la corrección del error funciona correctamente.
    - Asegurarse de que no se hayan introducido nuevos problemas o regresiones.

5. Despliegue de la corrección:
    - Desplegar la corrección en el entorno de producción siguiendo el plan de despliegue establecido.

6. Comunicación y seguimiento:
    - Informar a los usuarios y al equipo de soporte sobre la corrección del error y el despliegue realizado.
    - Realizar un seguimiento del funcionamiento de la aplicación después del despliegue de la corrección.

8. Nos preparamos para nuevos retos

Ejercicio 15: Arme un equipo de trabajo y defina los roles para realizar los ejercicios anteriores para un futuro dominio de aplicación relacionado con inteligencia artificial generativa.