Trabajo Final para entregar

Seminario de Lenguajes opción .NET

2do. Semestre - 2024

NOTA: Desarrollar con la versión **.NET 8.0**. No deshabilitar **<ImplicitUsings>** ni **<Nullable>** en los archivos de proyecto (**.csproj**). Resolver todos los *warnings* del compilador respecto de las referencias **null**.

Fecha de publicación: 31/10/2024

Fecha límite para la entrega: 18/11/2024 14:00 hs.

El trabajo puede realizarse en grupo de hasta 4 integrantes.

La entrega se realizará por medio de un formulario de Google que se publicará más adelante. Debe subirse la solución completa (habiendo borrado previamente las carpetas /bin y /obj de todos los proyectos) comprimida en un único archivo (preferentemente .zip). El nombre del archivo debe contener los apellidos de los autores del trabajo.

Sistema de Gestión de Inventario (SGI)

Descripción general

El propósito de este proyecto es expandir las capacidades del "**Sistema de Gestión de Inventario (SGI)**" que se desarrolló previamente.

Utilizando el proyecto original como base, se deberán integrar las siguientes funcionalidades adicionales:

Gestión de Usuarios:

- Desarrollar la funcionalidad necesaria para la gestión de usuarios. Cada usuario debe tener nombre, apellido, correo electrónico, contraseña y una lista de permisos.
- Los usuarios pueden tener múltiples permisos. En principio sólo el Administrador tendrá los permisos necesarios para listar, dar de baja y modificar cualquier usuario o sus permisos. Sin embargo, el Administrador también podrá otorgar a otros usuarios los permisos de UsuarioAlta, UsuarioBaja y UsuarioModificación. De esta manera, la gestión de usuarios también podrá ser realizada por aquellos autorizados.
- El primer usuario que se registre en el sistema será el Administrador, quien contará con todos los permisos del sistema.
- Definir los repositorios y casos de uso que se consideren necesarios.

Servicio de Autorización:

- Desarrollar el servicio de autorización **ServicioAutorizacion** que implemente la interfaz **IServicioAutorizacion**, reemplazando al servicio de autorización provisorio de la entrega inicial
- Este servicio debe verificar realmente si el usuario tiene el permiso requerido.

Persistencia de Datos:

- En el proyecto SGI.Repositorios, emplear Entity Framework Core para persistir datos en una base de datos SQLite, siguiendo la metodología "*code first*".

Interfaz de Usuario:

- Descartar el proyecto SGI.Consola de la primera entrega.
- Desarrollar una nueva interfaz de usuario en un proyecto llamado SGI.UI utilizando Blazor. Diseñar libremente esta interfaz de manera que toda la funcionalidad desarrollada en la primera entrega sea accesible desde esta interfaz, agregando la gestión de usuarios requerida.

Flujo de Gestión:

- Al iniciar la aplicación, se presentará una pantalla de bienvenida que permitirá a los usuarios iniciar sesión o registrarse.
- En caso de registro, se proporcionará un formulario para ingresar los datos personales y establecer una contraseña.
- Los usuarios tendrán la libertad de modificar sus datos personales y contraseña en cualquier momento.
- Por simplicidad no se implementará ningún mecanismo de recuperación de contraseña. En caso de olvido de contraseña, el usuario deberá contactar al Administrador o a otro usuario con permiso adecuado para restablecerla.
- Las contraseñas no se almacenarán directamente en la base de datos; en su lugar, se utilizará una función de hash para mayor seguridad.

Almacenamiento del hash

- El hash de la contraseña debe almacenarse en la base de datos, **nunca la contraseña en sí**. Para verificar la identidad del usuario al iniciar sesión, se vuelve a calcular el hash de la contraseña ingresada y se compara con el hash almacenado. Si los hashes coinciden, el usuario ha ingresado la contraseña correcta y se le permite acceder al sistema.

Permisos de Usuario:

- Los usuarios nuevos contarán inicialmente solo con permisos de lectura.
- Solo el Administrador o los usuarios con los permisos de gestión de usuarios podrán asignar permisos adicionales.

Acceso Administrativo:

Una vez iniciada la sesión, el menú de gestión de usuarios será visible sólo si quien ingresó posee alguno de los permisos de gestión de usuarios o es el Administrador.

Esquema de Permisos:

Mantener el mismo esquema de permisos definido en la primera entrega, con algunas modificaciones:

- En esta entrega, se considerará la posibilidad de que un usuario tenga permiso para eliminar productos pero no para eliminar transacciones individuales. Al eliminar un producto, todas las transacciones asociadas se eliminarán automáticamente, incluso si el usuario no tiene el permiso para dar de baja transacciones (permiso TransaccionBaja**)**.

NOTA: Es conveniente establecer la propiedad *journal mode* de la base de datos sqlite en *DELETE.*

Se puede establecer por código de la siguiente manera:

```
context.Database.EnsureCreated();
var connection = context.Database.GetDbConnection();
connection.Open();
using (var command = connection.CreateCommand())
{
    command.CommandText = "PRAGMA journal_mode=DELETE;";
    command.ExecuteNonQuery();
}
```

Explicación de esta recomendación

La propiedad *journal_mode* se utiliza en una base de datos SQLite para especificar el modo de registro de transacciones que se utilizará. Éste determina cómo se guardan y administran los cambios realizados en la base de datos.

Al utilizar el modo *DELETE* en SQLite, los cambios realizados se reflejan inmediatamente en la base de datos principal. Esto significa que, después de confirmar una transacción, los datos modificados se guardan directamente en el archivo de la base de datos y están disponibles para su lectura inmediata.

Por otro lado, en otros modos, que implican el uso de registros de transacciones, los cambios no se aplican directamente a la base de datos principal. En cambio, se escriben

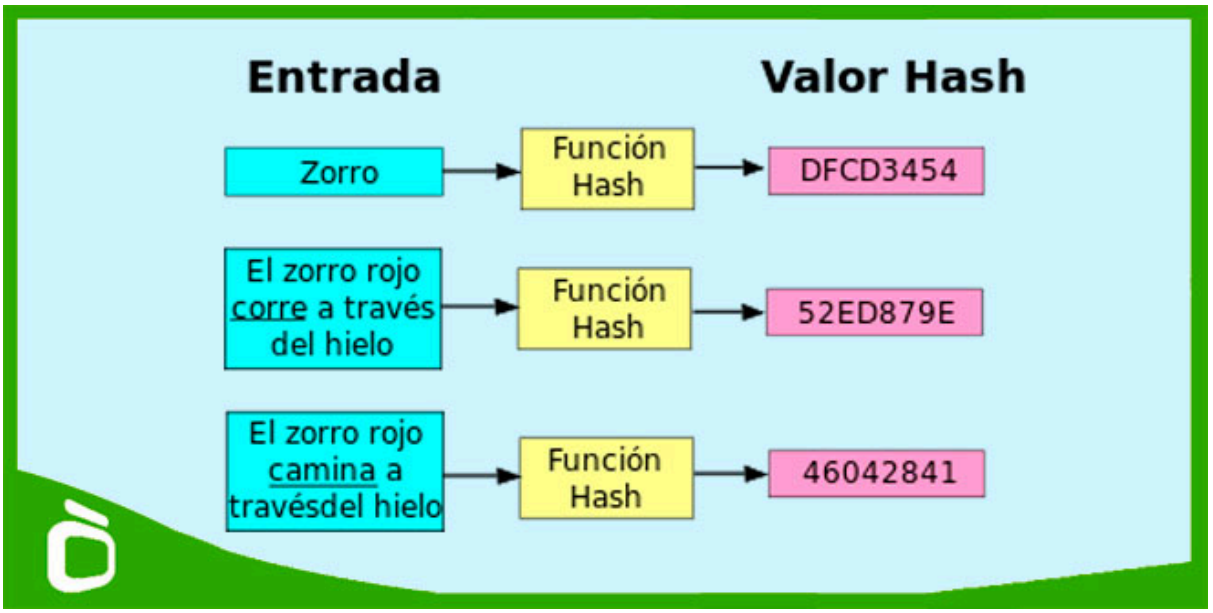
primero en el archivo de registro de transacciones. Luego, en segundo plano, se realizan las operaciones de fusionar y aplicar los cambios al archivo de la base de datos principal.

Como resultado, si se accede a la base de datos con otra herramienta mientras se utiliza un modo de registro de transacciones, es posible que no se vean los cambios reflejados inmediatamente. Es necesario esperar a que se realice la fusión y aplicación de los registros de transacciones antes de que los cambios sean visibles en la base de datos principal. La frecuencia con la que se realiza este proceso de fusión y aplicación puede variar y depende de factores como la configuración y la carga de trabajo del sistema.

Por lo tanto, el modo *DELETE* proporciona una escritura directa y visible en la base de datos, mientras que otros modos de registro pueden introducir una latencia en la propagación de los cambios debido a las operaciones de fusión y aplicación que deben llevarse a cabo.

Apartado sobre función de hash.

Una función hash es un procedimiento que transforma una información determinada (por ejemplo, un texto) en una secuencia alfanumérica “única” de longitud fija, denominada hash (resumen).



Un hash no es un cifrado porque su resultado no puede descifrarse para obtener el original. El proceso es irreversible.

Es útil para comprobar la integridad de la información. Si volvemos a calcular el hash obtenemos el mismo valor significa, en la práctica, que la información no ha sido adulterada.

En este trabajo es necesario utilizar una función criptográfica de hash segura. Estas funciones deben cumplir algunos requisitos:

- Eficiencia computacional: El cálculo del hash debe ser computacionalmente eficiente.

- Difusión o efecto avalancha: un pequeño cambio en el input debe producir un cambio significativo en el output
- Resistencia a la preimagen (Unidireccionalidad): imposible "revertir" la función hash (encontrar el input a partir de un output determinado)
- Resistencia a colisión: La probabilidad de colisionar debe ser tan baja que requeriría millones de años de computaciones
- Resistencia a la segunda preimagen: Dado un hash debe ser inviable encontrar una input que produzca el mismo valor de hash

Son ejemplos de funciones de hash: MD5, SHA1, SHA-256, RIPEMD-160, etc.

SHA-256 produce un hash de 256 bits (32 bytes). Es una de las más usadas por su equilibrio entre seguridad y costo computacional de generación. El algoritmo SHA256 (Secure Hash Algorithm 256 bits) funciona dividiendo la entrada en bloques de 512 bits y sometiendo a cada bloque a una serie de rondas de procesamiento, que incluyen: rotaciones a nivel de bits, operaciones lógicas (AND, OR, XOR), adiciones modulares y mezclas de bits. En este link https://sha256algorithm.com/ se puede observar al algoritmo SHA256 en funcionamiento.

¿Cómo utilizar SHA-256 en .NET?

Afortunadamente, no es necesario implementar SHA-256 desde cero en .NET, ya que la clase SHA256 del espacio de nombres System.Security.Cryptography proporciona la funcionalidad necesaria. Investigar sobre ella y utilizarla para obtener el hash de las contraseñas de los usuarios.