Primer Trabajo para entregar

Seminario de Lenguajes opción .NET 1º Semestre - 2025

Importante

Queremos recordarles la importancia de este primer trabajo del curso. Aunque no recibirá una calificación numérica, será objeto de corrección y dará lugar a una devolución que servirá como guía para continuar hacia el próximo trabajo.

La entrega de este primer trabajo es de <u>carácter obligatorio</u> y <u>requisito excluyente</u> para la aprobación del curso.

La entrega debe ser significativa. No se admitirán entregas en blanco ni aquellas que muestren un esfuerzo mínimo, en lugar de un sincero compromiso y dedicación en la realización del trabajo.

Entrega

Fecha límite: 22/05/2025 hasta las 13:00 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.

Se deberá entregar:

- El código de la solución completa
- Un documento explicativo (puede ser .pdf o .md) donde se detalle cómo probar la funcionalidad desarrollada desde Program.cs. Se debe explicitar código de ejemplo junto con la salida por consola producida.
- Subir la solución completa y el documento explicativo, todo comprimido en un único archivo (preferentemente .zip). El nombre del archivo zip debe contener los apellidos de los autores del trabajo, ejemplo: Apellido1 Apellido2 Trabajo1.zip.

Sistema de Gestión del Centro Deportivo Universitario

Objetivo

Se requiere desarrollar un sistema para la gestión de eventos deportivos específicos y las inscripciones (reservas) correspondientes dentro de un centro universitario. Este sistema permitirá:

- Registrar Personas, quienes podrán ser participantes en eventos o responsables de la organización de los mismos.
- Definir Eventos Deportivos concretos, cada uno con su fecha, hora de inicio, duración y cupo máximo de participantes.
- Gestionar las Reservas que las personas realizan para participar en dichos eventos, incluyendo un control básico del estado de su asistencia.

El diseño del sistema se fundamentará en los principios de la **Arquitectura Limpia**, priorizando la separación de responsabilidades y el desacoplamiento de componentes mediante el uso del patrón de **Inyección de Dependencia (DI)**. Cada una de las entidades principales –Persona, EventoDeportivo y Reserva– será identificable de forma única. El sistema deberá soportar las operaciones fundamentales de gestión de datos (altas, bajas, modificaciones y listados) a través de casos de uso bien definidos.

★ Implementación: Estructura del Proyecto

La solución se llamará **CentroEventos** y se dividirá obligatoriamente en **3 proyectos** siguiendo una estructura de Arquitectura Limpia:

- 1. **CentroEventos.Aplicacion** (Biblioteca de Clases .NET 8)
 - Contendrá el núcleo de la lógica de negocio.
 - **NO** debe tener dependencias de otros proyectos de esta solución.
 - Incluirá:
 - Las clases de las **entidades** del sistema.
 - Las **interfaces** de los repositorios.
 - Las interfaces de otros servicios (como autorización).
 - Las clases de Casos de Uso.
 - Las clases de **excepciones** personalizadas.
 - Las clases de Validadores.
 - La implementación concreta de un servicio de autorización provisional.
- 2. | Centro Eventos. Repositorios (Biblioteca de Clases . NET 8)
 - o Contendrá la implementación concreta de la persistencia.
 - DEBE tener una referencia al proyecto CentroEventos.Aplicacion (para conocer las entidades y las interfaces que implementa).

- NO debe tener referencia a CentroEventos.Consola.
- Incluirá:
 - Las implementaciones concretas (clases) de las **interfaces de repositorio** (usando archivos de texto plano para esta entrega).
- 3. **Example 1** Centro Eventos. Consola (Aplicación de Consola .NET 8)
 - o Será el punto de entrada y la interfaz de usuario.
 - DEBE tener referencias a los proyectos CentroEventos.Aplicacion y CentroEventos.Repositorios.
 - o Incluirá:
 - El código para **probar los casos de uso** implementados.

Entidades a Implementar (en CentroEventos.Aplicacion)

- **Persona:** Representa a cualquier individuo relacionado con el centro deportivo.
 - o Id (int, único, debe ser autoincremental gestionado por el repositorio)
 - DNI (string, único)
 - Nombre (string)
 - Apellido (string)
 - Email (string, único)
 - Telefono (string)
- EventoDeportivo: Representa una instancia específica de un evento deportivo programado en una fecha y hora determinadas.
 - o Id (int, único, debe ser autoincremental gestionado por el repositorio)
 - Nombre (string ej: "Clase de Spinning Avanzado", "Partido final de Vóley")
 - Descripcion (string)
 - FechaHoralnicio (DateTime Fecha y hora exactas de inicio del evento)
 - o DuracionHoras (double Duración del evento en horas, ej: 1.5 para una hora y media)
 - CupoMaximo (int Cantidad máxima de participantes permitidos)
 - Responsableld (int Id de la Persona a cargo del evento)
- Reserva: Representa la inscripción de una persona a un evento deportivo específico.
 - o Id (int, único, debe ser autoincremental gestionado por el repositorio)
 - Personald (int Id de la Persona que hace la reserva)
 - EventoDeportivoId (int Id de la EventoDeportivo reservado)
 - FechaAltaReserva (DateTime Fecha y hora en que se realizó la inscripción)
 - EstadoAsistencia (enum: Pendiente, Presente, Ausente)

Implementar el método ToString() de forma conveniente en las entidades para facilitar la visualización en la consola.

Reglas de Negocio

- Un EventoDeportivo no puede tener más Reservas que su CupoMaximo.
- Una Persona no puede tener más de una Reserva para el mismo EventoDeportivo.
- No puede modificarse un EventoDeportivo cuya FechaHoraInicio haya expirado (es decir, no puede modificarse un evento pasado).
- Al crear o modificar un EventoDeportivo, no puede establecerse la FechaHoralnicio con un valor anterior al actual (es decir que sólo se permite si la fecha que va a registrarse es >= fecha actual).
- No puede eliminarse un EventoDeportivo si existen Reservas asociadas al mismo (independientemente del estado de las reservas).
- No puede eliminarse una Persona si es responsable de algún EventoDeportivo o si existen reservas asociadas a ella (independientemente del estado de las reservas).

☑ Validaciones (Implementar, usando validadores específicos para cada entidad, en el proyecto CentroEventos.Aplicacion)

• EventoDeportivo:

- Nombre y Descripcion no deben estar vacíos.
- o FechaHoralnicio debe ser posterior o igual a la fecha actual.
- CupoMaximo debe ser mayor que cero.
- DuracionHoras debe ser mayor que cero.
- ResponsableId debe corresponder a una Persona existente. (Requiere consulta a IRepositorioPersona)

Persona:

- Nombre, Apellido, DNI, Email no deben estar vacíos.
- o DNI no puede repetirse entre Personas. (Requiere consulta a IRepositorioPersona)
- Email no puede repetirse entre Personas. (Requiere consulta a IRepositorioPersona)

Reserva:

- Personald y EventoDeportivoId deben corresponder a entidades existentes. (Requiere consulta a IRepositorioPersona y IRepositorioEventoDeportivo)
- No permitir que la misma Persona reserve dos veces el mismo EventoDeportivo (Requiere consulta a IRepositorioReserva)
- Verificar cupo disponible en el EventoDeportivo antes de guardar. (Requiere consulta a IRepositorioEventoDeportivo y IRepositorioReserva)

- FalloAutorizacionException: Se lanza cuando se intenta realizar una operación para la cual el usuario del sistema no tiene permiso.
- ValidacionException: Se lanza si algún dato obligatorio está ausente, tiene formato incorrecto, o una regla de validación simple falla (ej. CupoMaximo <= 0, FechaHoraInicio anterior a la fecha y hora actual).
- EntidadNotFoundException: Se lanza si se intenta operar con un ld que no existe (ej. Personald, EventoDeportivold o Reservald no encontrados).
- CupoExcedidoException: Se lanza al intentar registrar una Reserva para un EventoDeportivo sin cupo.
- DuplicadoException: Se lanza al intentar crear una Persona con DNI/Email ya existente, o una Reserva duplicada para la misma persona/actividad.
- OperacionInvalidaException: Se lanza al intentar realizar una operación no permitida por las reglas de negocio (ej. eliminar entidad con dependencias).

Casos de Uso (en CentroEventos.Aplicacion)

Se deben implementar los casos de uso CRUD básicos para realizar Altas, Bajas, Modificaciones y Listado (completo) de **Persona**, **EventoDeportivo** y **Reserva**. Las operaciones de baja/eliminación recibirán el **Id** de la entidad a eliminar.

Además de los 12 casos anteriores debe implementarse **ListarEventosConCupoDisponibleUseCase** para obtener un listado de los eventos futuros donde aún existen cupos disponibles y **ListarAsistenciaAEventoUseCase** para obtener la lista de todos los asistentes a un evento pasado.

Ejemplo ilustrativo de casos de uso específicos con detalle de la lógica necesaria:

ReservaAltaUseCase

- Constructor: Recibe IRepositorioReserva, IRepositorioEventoDeportivo, IRepositorioPersona, IServicioAutorizacion.
- Método: void Ejecutar(Reserva datosReserva, int idUsuario)

Lógica:

- Verificar si el usuario está autorizado a realizar la operación utilizando
 IServicioAutorizacion. Lanzar FalloAutorizacionException si el usuario no está autorizado.
- Validar existencia de datosReserva.Personald y datosReserva.EventoDeportivold consultando sus respectivos repositorios. Lanzar EntidadNotFoundException si alguna no existe.
- Validar si hay cupo disponible y lanzar la excepción CupoExcedidoException si no lo hay.

- 4. Validar que la Persona no tenga ya una reserva para ese EventoDeportivo. Lanzar DuplicadoException si ya existe.
- Si todo OK, completar datosReserva (FechaAltaReserva, EstadoAsistencia), y agregarla usando IRepositorioReserva.

ListarEventosConCupoDisponibleUseCase

- o Constructor: Recibe IRepositorio Evento Deportivo, IRepositorio Reserva.
- Método: List<EventoDeportivo> Ejecutar()
- Lógica: Obtener todos los EventoDeportivos cuya FechaHoraInicio sea futura. Para cada uno, contar sus reservas. Devolver solo aquellos eventos donde el conteo de reservas sea menor que su CupoMaximo.

Rermisos de usuario y servicio de autorización (en CentroEventos.Aplicacion)

Implementar el tipo enumerativo Permiso que se utilizará para gestionar la autorización de los usuarios en el sistema. Cada usuario del sistema podrá poseer **uno o varios de los siguientes permisos**, que habilitan acciones específicas sobre las entidades del sistema:

Permiso	Descripción
EventoAlta	Puede crear nuevos eventos deportivos en el centro
EventoModificacion	Puede modificar los detalles de los eventos deportivos
EventoBaja	Puede eliminar eventos deportivos del centro
ReservaAlta	Puede registrar nuevas reservas
ReservaModificacion	Puede modificar las reservas
ReservaBaja	Puede dar de baja reservas
UsuarioAlta	Puede dar de alta nuevos usuarios del sistema
UsuarioModificacion	Puede modificar los datos de los usuarios
UsuarioBaja	Puede dar de baja usuarios del sistema

Se asume que todos los usuarios tendrán derecho de lectura en relación a todas las entidades. Por lo tanto no se requerirá gestionar la autorización para la generación de listados.

Servicio de Autorización

Se requiere desarrollar un servicio que **valide si un usuario tiene los permisos necesarios** para realizar una operación sobre el sistema. Este servicio debe implementar la siguiente interfaz:

```
public interface IServicioAutorizacion
{
```

```
bool PoseeElPermiso(int IdUsuario, Permiso permiso);
```

Servicio Provisional

Para esta primera entrega, se desarrollará una **implementación provisional** del servicio de autorización llamada Servicio Autorización Provisorio.

Este servicio responderá de la siguiente manera:

- Si IdUsuario == 1, devuelve true para cualquier permiso solicitado.
- Si el IdUsuario es distinto de 1, devuelve false para todos los permisos.

→ Nota: Este servicio es provisorio. En la próxima entrega, se implementará un sistema de gestión de usuarios completo, y el servicio provisional será reemplazado por una versión definitiva que validará los permisos basados reales asignados a los usuarios.

Repositorios (en CentroEventos.Repositorios)

Implementar las clases para las interfaces IRepositorioPersona, IRepositorioEventoDeportivo, IRepositorioReserva utilizando **archivos de texto plano** (formato simple a elección: CSV, un atributo por línea, etc.) para la persistencia de datos. Deben manejar las operaciones CRUD básicas necesarias para los casos de uso y asegurar que los datos persistan entre ejecuciones del programa. Implementar un mecanismo simple para la generación de IDs únicos para las nuevas entidades (ej. leyendo el último ID del archivo correspondiente y sumando 1, o manteniendo un archivo separado para el último IDs de cada entidad). Los IDs deben ser únicos y no reutilizables, incluso si la entidad se elimina, ejemplo: IDs 1, 2, 3 \rightarrow se elimina 2 \rightarrow nuevo ID sería 4, no 2 (inclusive si se hubiese eliminado el 3, último elemento, también el nuevo ID sería 4).

Importante: Desarrollar un repositorio por cada entidad aporta alta cohesión, menor acoplamiento, mayor testabilidad y mayor escalabilidad.

Nota: Para actualizar los datos en los archivos de texto no se deben tener en cuenta aspectos de rendimiento. Por ejemplo, se podría sobrescribir el archivo completo sólo para modificar un registro en el mismo. En la próxima entrega vamos a utilizar un manejador de bases de datos para gestionar la persistencia de manera más eficiente.

Proyecto de consola donde se pueden probar los casos de uso implementados. En el documento explicativo debe figurar cómo probar funcionalidades desde Program.cs.

Requisitos Técnicos

- Desarrollar en .NET 8.0.
- No deshabilitar <ImplicitUsings> ni <Nullable> en los archivos .csproj.
- Resolver todos los warnings del compilador, especialmente los relacionados con referencias null.
- Utilizar Inyección de Dependencias por Constructor donde sea aplicable (principalmente en Casos de Uso).
- Utilizar interfaces para definir los contratos de los repositorios y servicios.
- Mantener la estructura limpia de 3 proyectos con las dependencias correctas.

Trabajo Final para entregar

Seminario de Lenguajes opción .NET

1er. Semestre - 2025

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: 5/6/2025

Fecha límite para la entrega: 19/06/2025 23:59 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 del Centro Deportivo Universitario

Descripción general

El propósito de este proyecto es expandir las capacidades del "Sistema de Gestión del Centro Deportivo Universitario" 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 del sistema. 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. También definir un validador para la entidad Usuario (nombre, apellido y contraseñas requeridos).

Servicio de Autorización:

- Desarrollar el servicio de autorización ServicioAutorización que implemente la interfaz IServicioAutorización, 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 CentroEventos.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 CentroEventos.Consola de la primera entrega.
- Desarrollar una nueva interfaz de usuario en un proyecto llamado CentroEventos.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. El contacto es de manera directa con la persona, no se debe implementar en el sistema.
- 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. Es decir pueden ver las reservas, socios y eventos.
- 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.

Apartado sobre base de datos.

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.

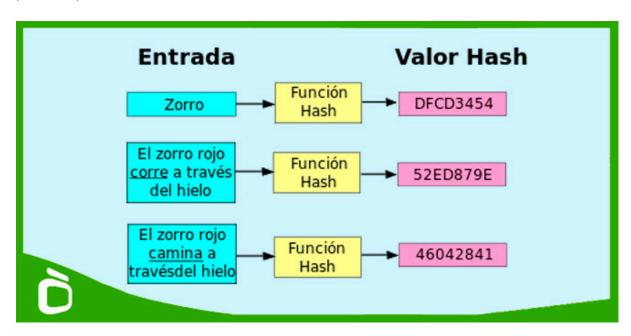
Por otro lado, en otros modos, que implican el uso de registros de transacciones, los cambios no se aplican directamente a la base de datos principal. En cambio, se escriben primero en el archivo de registro de transacciones. Luego, en segundo plano, se realizan las operaciones de fusionar y aplicar 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 y 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 computaciona**l: 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.

Apartado para la sesión del usuario

Para gestionar la sesión del usuario autenticado dentro de la aplicación, se recomienda implementar un servicio que almacene el usuario actual al momento del inicio de sesión. Este servicio debe registrarse con un alcance **Scoped**, lo que asegura que la información del usuario se mantenga disponible durante toda la vida útil de la conexión en Blazor Server.

Cabe destacar que no es necesario persistir la sesión del usuario entre múltiples pestañas del navegador ni ante una recarga de la página, dado que dichas acciones provocan el restablecimiento de la conexión, generando una nueva instancia del servicio con alcance scoped.