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 carácter obligatorio y requisito excluyente 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. 💾 **CentroEventos.Repositorios** (Biblioteca de Clases - .NET 8)
    - 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. 🖥️ **CentroEventos.Consola** (Aplicación de Consola - .NET 8)
    - Será el punto de entrada y la interfaz de usuario.
    - **DEBE** tener referencias a los proyectos CentroEventos.Aplicacion y CentroEventos.Repositorios.
    - 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.
    - 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.
    - Id (int, único, debe ser autoincremental gestionado por el repositorio)
    - Nombre (string - ej: "Clase de Spinning Avanzado", "Partido final de Vóley")
    - Descripcion (string)
    - FechaHoraInicio (DateTime - Fecha y hora exactas de inicio del evento)
    - DuracionHoras (double - Duración del evento en horas, ej: 1.5 para una hora y media)
    - CupoMaximo (int - Cantidad máxima de participantes permitidos)
    - ResponsableId (int - Id de la Persona a cargo del evento)
- **Reserva:** Representa la inscripción de una persona a un evento deportivo específico.
    - Id (int, único, debe ser autoincremental gestionado por el repositorio)
    - PersonaId (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 FechaHoraInicio 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.
  - FechaHoraInicio 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.
  - DNI no puede repetirse entre Personas. *(Requiere consulta a IRepositorioPersona)*
  - Email no puede repetirse entre Personas. *(Requiere consulta a IRepositorioPersona)*
- **Reserva:**
  - PersonaId 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)*

🚨 Excepciones (Definir Clases en CentroEventos.Aplicacion)

- 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 Id que no existe (ej. PersonaId, EventoDeportivoId o ReservaId 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:**
        1. Verificar si el usuario está autorizado a realizar la operación utilizando IServicioAutorizacion. Lanzar FalloAutorizacionException si el usuario no está autorizado.
        2. Validar existencia de datosReserva.PersonaId y datosReserva.EventoDeportivoId consultando sus respectivos repositorios. Lanzar EntidadNotFoundException si alguna no existe.
        3. 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.
5. Si todo OK, completar datosReserva (FechaAltaReserva, EstadoAsistencia), y agregarla usando IRepositorioReserva.

- **`ListarEventosConCupoDisponibleUseCase`**
  - Constructor: Recibe `IRepositorioEventoDeportivo`, `IRepositorioReserva`.
  - Método: `List<EventoDeportivo> Ejecutar()`
  - **Lógica:** Obtener todos los `EventoDeportivo`s cuya `FechaHoraInicio` sea futura. Para cada uno, contar sus reservas. Devolver solo aquellos eventos donde el conteo de reservas sea menor que su `CupoMaximo`.

🔐 Permisos 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 `ServicioAutorizacionProvisorio`.

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 → se elimina 2 → 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.

🖥️ CentroEventos.Consola

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.