Seminario .NET- Proyecto

Primera Entrega

Integrantes: Bie Leandro, Castañeda Isabella, Dobal Federico, Villca Facundo.

Definición general del proyecto

El proyecto que se nos solicitó busca representar el sistema para un **Centro de Eventos Deportivos**. Su funcionalidad se basa en administrar eventos deportivos, relacionarlos con otras entidades (personas y reservas) mediante casos de uso habituales, para posteriormente persistir toda la información en repositorios de archivos de texto.

Los objetivos principales del proyecto son:

1. Proveer a los usuarios[1] del Centro un sistema básico para la **administración** de sus eventos deportivos, personas y reservas.
2. Implementar un **CRUD** (crear, leer, actualizar y borrar) en los casos de uso del sistema.
3. Incorporar **validaciones** predefinidas al ejecutar un caso de uso.
4. Expresar la falla de las validaciones mediante **excepciones** personalizadas.
5. Respetar la **inyección de dependencias** por constructor.
6. **Persistir** toda la información del CRUD en archivos de texto.

Arquitectura del sistema

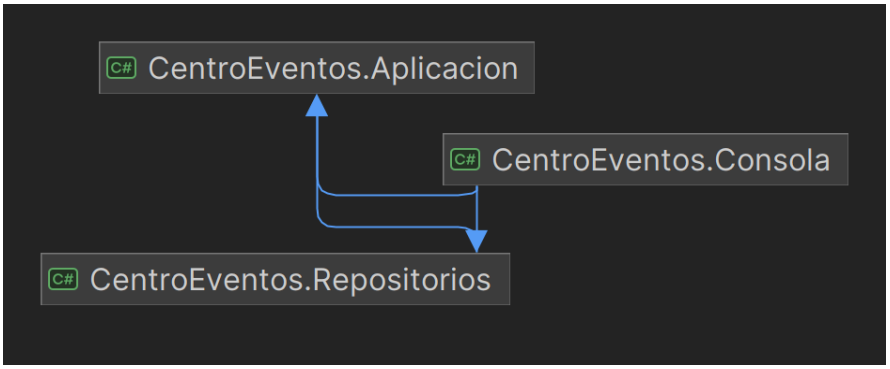
El sistema está organizado de la siguiente manera:

- Declaración de las interfaces Eventos Deportivos, Personas, Reservas y Servicio de Autorización, y los métodos necesarios para cada uno.
- Representaciones de las entidades correspondientes. En este caso, las mismas son Evento Deportivo, Persona y Reserva.
- Declaración de los repositorios e implementación de los métodos de la interfaz de la que derivan.
- Persistencia de datos a través de archivos de texto plano.
- Implementación de casos de uso.
- Uso de validadores y excepciones.

Al mismo tiempo, se hace uso del concepto de Arquitectura Limpia, donde se separan los elementos en “capas” de manera que los elementos externos conozcan a los internos y nunca al revés. Esto permite que el sistema sea sostenible a largo plazo y en caso de alguna modificación que se requiera realizar a futuro, no implique una reconstrucción completa del código, si no, solamente de la interfaz de usuario y los repositorios. A continuación se adjunta una imagen que explica esto de manera gráfica, donde se muestra que la Interfaz de Usuario (Consola) conoce las referencias de

[1] En esta primera entrega del proyecto, no se implementarán los usuarios de forma completa. Llámese *usuarios* a aquellos que pongan a prueba el sistema.

Repositorios y Aplicación; Repositorios de Aplicación, y Aplicación no posee la referencia a ningún módulo pues su único deber es ser el núcleo inmutable del sistema:



CentroEventos.Aplicación

Sus referencias incluyen: -

La Aplicación es el *core* de la solución. Maneja la lógica del sistema y se divide en:

- **Entities:**
  - EventoDeportivo
  - Persona
  - Reserva
- **Enums:**

  Enumeraciones requeridas para tipar las asistencias a los eventos, y el permiso que posee cada usuario.
  - Asistencia:
    - Pendiente
    - Presente
    - Ausente
  - Permiso:
    - EventoAlta
    - EventoModificacion
    - EventoBaja
    - ReservaAlta
    - ReservaModificacion
    - ReservaBaja
    - UsuarioAlta
    - UsuarioModificacion
    - UsuarioBaja
- **Exceptions:**

  Excepciones personalizadas para arrojar en caso de que un validador devuelva falso:
  - CupoExcedidoException
  - DuplicadoException
  - EntidadNotFoundException: esta es la única Exception que se lanza en los repositorios.

    - FalloAutorizacionException: Exception arrojada cuando un usuario no posee el permiso para ejecutar un caso de uso.
    - OperacionInvalidaException
    - ValidacionException: la tomamos como la Exception más genérica. La lanzamos mayormente en validaciones no tan específicas, por ejemplo cuando una entidad tiene números en su nombre, o cuando un string ingresa en blanco.
- **Interfaces:**
    - IRepositorioEventoDeportivo
    - IRepositorioPersona
    - IRepositorioReserva
    - IServicioAutorizacion
- **Services:**

  Implementa a IServicioAutorizacion y decide si un usuario, dado su ID, posee el permiso para ejecutar. En esta primera entrega, un usuario es poseedor del permiso solamente si su ID == 1.
- **UseCases:**

  Se implementan los UseCases de manera que antes de ejecutar el método correspondiente al mismo (Alta,Baja,Modificación,etc) se realicen las validaciones necesarias y verificaciones de que se cumpla con las reglas de negocio adecuadas a cada caso. Por ejemplo, que al querer dar de alta una persona, la misma posea un nombre válido, o que al querer modificar un Evento Deportivo con una fecha, la misma no sea inferior a la actual.
- **Validators:**

  Se hace uso de los validadores de manera que se pueda asegurar de que se cumplen las restricciones necesarias para cada UseCase. Por ejemplo, que el formato de los datos recibidos sean correctos antes de ser utilizados. En este caso, se creó una clase correspondiente a cada validador, de manera que al no cumplirse la misma, se pueda lanzar la excepción adecuada.

CentroEventos.Consola

Sus referencias incluyen: CentroEventos.Aplicacion, CentroEventos.Repositorios

La UI de la solución es la consola.

Programa principal

En el programa principal creamos las instancias para el Servicio de Autorización, los Validadores y Casos de Uso de las 3 entidades, los Repositorios a inyectar, y codificamos el cuerpo principal dentro de un try-catch. Gracias al Selector, el usuario mismo controla el flujo del programa con menús y selectores.

Se proporciona al usuario el poder de elegir en primer lugar, con cuál entidad trabajar, y qué hacer con la misma (Alta, Baja, Modificar, Listar).

Selector

La clase Selector es una adición propia con varios usos. Busca abstraer la acción de seleccionar algo en la consola, y así hacer el código del programa principal más legible.

A la hora de implementar, por ejemplo, la entidad Reserva, nos enfrentamos con la problemática de que el usuario tenía que conocer el ID de la Persona a cargo de la reserva y del Evento Deportivo a reservar. Lo mismo para dar de alta un Evento Deportivo, el usuario solo identificaba al responsable a cargo por su ID.

Por esa razón, consideramos que sus métodos más importantes son los que seleccionan entidades (`Selector.Personas`, `Selector.EventosDeportivos`, `Selector.Reservas`). Para deshacernos de este problema de seguridad, dónde el ID lo manejaba el usuario, la Clase selector proporciona al exterior un listado de las entidades a seleccionar, enumeradas desde el 1, así el usuario solo se encarga de elegir la entidad que desea de esa lista (mediante su número en la lista), e internamente se recupera el ID correspondiente a la entidad.

CentroEventos.Repositorios

Sus referencias incluyen: CentroEventos.Aplicacion

Los Repositorios son las clases que implementan los métodos correspondientes de la interfaz de la que derivan y al mismo tiempo controlan los archivos de texto que contienen los datos de las entidades correspondientes (Evento Deportivo, Reserva o Persona). Asimismo, los repositorios se encargan de asignar y auto-incrementar el ID a cada entidad que se da de alta. Los IDs se almacenan en su propio archivo de texto y no son visibles para el usuario, si no que se gestionan internamente para poder diferenciar a dichas entidades y realizar los métodos correspondientes bajo el criterio de los mismos.

Proceso de trabajo

El desarrollo de este proyecto fue realizado en equipo compuesto por cuatro integrantes. En una primera instancia, intentamos trabajar de forma simultánea utilizando una herramienta llamada *Live Server*, sin embargo, esta opción resultó poco viable debido a los constantes retrasos en la sincronización, lo cual dificultaba el trabajo fluido y coordinado entre los miembros.

Ante esta dificultad, optamos por una nueva estrategia de trabajo: dividir las responsabilidades según las entidades del sistema y gestionar el proyecto a través de la plataforma GitHub. Esta decisión resultó ser la más cómoda y efectiva para todos los integrantes, ya que nos permitió llevar un control claro de los cambios realizados en el proyecto, sin la necesidad de que todos estuviéramos conectados de manera simultánea. De este modo, cada miembro pudo avanzar de forma autónoma y subir sus aportes al repositorio común.

Es importante destacar que esta división de tareas no implicó una falta de colaboración activa. Por el contrario, la asignación de entidades se realizó con el objetivo de

optimizar el ritmo de trabajo, pero asegurando siempre una comprensión general del código desarrollado por los demás miembros. Esto permitió mantener una visión integral del proyecto y facilitó la integración de los diferentes componentes.

Conclusión

Como primer acercamiento a un trabajo en equipo dentro del ámbito del desarrollo de software con .NET y C#, consideramos que fue una experiencia sumamente enriquecedora. Nos enfrentamos a diversos desafíos que supimos afrontar de forma colaborativa, tanto en lo relativo a la dinámica grupal —como la unificación de criterios, la comunicación efectiva y la elección de herramientas adecuadas— como en aspectos técnicos propios del proyecto, tales como la gestión interna del ID de los usuarios y la implementación de una arquitectura limpia, entre otros. Creemos que el proyecto respeta firmemente el principio de inversión de dependencia, las referencias de los proyectos, el concepto de arquitectura cebolla. Además, se implementó todo lo requerido por las consignas. A continuación se adjunta un vistazo general del diagrama de dependencias de la solución completa:

Diagrama de dependencias - .NET primera entrega.svg