Facultad de Ingeniería

Bernard Wand Polak

Sports Manager

Diseño de Aplicaciones 2
Obligatorio 1

Estudiantes:
Santiago Díaz - 159593
Santiago López - 183800

Grupo: N6A
Docente: Ignacio Valle Dubé

Índice

1. Descripción general

1.1 Identificación

El sistema llevará el nombre de ***Sports Manager***.

1.2 Alcance del producto

El sistema Sports Manager podrá gestionar y manejar de forma óptima las operaciones de un gestor de encuentros, las cuales estarán divididas en módulos como Usuario, Equipo, Deporte y Evento. Sports Manager permitirá ser extensible para poder facilitar futuras mejoras al producto.

1.3 Descripción general del trabajo

Se busco una solución lo más extensible posible a la hora de agregar nuevos módulos y funcionalidades. La alta cohesión y el bajo acoplamiento de los componentes hacen que, a nuestro entender, esto sea posible. Se usaron patrones de diseño como Singleton, Factory Method y Statergy, así como la división en varios paquetes los componentes en común. Se utilizó la metodología Test-Drive Development (TDD) en el proceso del desarrollo del software para asegurar la buena calidad del código y enfocarse en las necesidades que el cliente necesita.

2. Especificación de Requerimientos

2.1 Requerimientos Funcionales

RF 001 - *Alta de usuario*
Descripción: Los administradores del sistema podrán agregar nuevos usuarios.
Prioridad: Alta

RF 002 - *Modificación de usuario*
Descripción: Los administradores del sistema podrán modificar la información de los usuarios.
Prioridad: Alta

RF 003 - *Baja de usuario*
Descripción: Los administradores del sistema podrán eliminar usuarios.
Prioridad: Media

RF 004 - *Alta de equipo*
Descripción: Los administradores del sistema podrán agregar nuevos equipos.
Prioridad: Alta

RF 005 - *Modificación de equipo*
Descripción: Los administradores del sistema podrán modificar la información de los equipos.
Prioridad: Alta

RF 006 - *Baja de equipo*
Descripción: Los administradores del sistema podrán eliminar equipos.
Prioridad: Media

RF 007 - *Alta de deporte*
Descripción: Los administradores del sistema podrán agregar nuevos deportes.
Prioridad: Alta

RF 008 - *Modificación de deporte*
Descripción: Los administradores del sistema podrán modificar la información de los deportes.
Prioridad: Alta

RF 009 - *Baja de deporte*
Descripción: Los administradores del sistema podrán eliminar deportes.
Prioridad: Media

RF 010 - *Alta de encuentro*
Descripción: Los administradores del sistema podrán agregar nuevos encuentros. No puede haber es más de un encuentro por día para un mismo equipo.

Prioridad: Alta

RF 011 - *Modificación de encuentro*
Descripción: Los administradores del sistema podrán modificar la información de los encuentros.
Prioridad: Alta

RF 012 - *Baja de encuentro*
Descripción: Los administradores del sistema podrán eliminar encuentros.
Prioridad: Media

RF 013 - *Generación de fixture de encuentros*
Descripción: Los administradores del sistema podrán generar fixtures de encuentros, manualmente o mediante algún mecanismo automático.
Prioridad: Media

RF 014 - *Reporte de encuentros para un deporte*
Descripción: Los administradores del sistema podrán obtener un reporte de los encuentros de determinado deporte.
Prioridad: Media

RF 015 - *Reporte de encuentros para un equipo en particular*
Descripción: Los administradores del sistema podrán obtener un reporte de los encuentros para un equipo en particular.
Prioridad: Media

RF 016 - *Lista de comentarios de encuentros de equipos favoritos*
Descripción: Todos los usuarios podrán visualizar la lista de todos los comentarios de los encuentros de todos los equipos que el usuario está siguiendo como favorito.
Prioridad: Media

RF 017 - *Calendario de eventos*
Descripción: Todos los usuarios podrán visualizar un calendario de eventos agrupados por deporte, y al pasar por arriba de uno de estos, se debe mostrar el detalle de los eventos de esa fecha.
Prioridad: Media

RF 018 - *Realizar comentarios*
Descripción: Todos los usuarios podrán realizar comentarios en los encuentros.
Prioridad: Media

RF 019 - *Lista de equipos con filtros*
Descripción: Todos los usuarios podrán visualizar la lista de equipos con sus datos, pudiendo filtrar y ordenar los mismos en forma creciente y decreciente por nombre.
Prioridad: Media

RF 020 - *Agregar a favoritos*
Descripción: Todos los usuarios podrán seleccionar uno o más usuarios de la lista de equipos (RF 019), y agregarlos como favoritos.
Prioridad: Media

RF 021 - *Administrar favoritos*
Descripción: Todos los usuarios podrán visualizar sus favoritos, pudiendo eliminarlos si así lo desea.
Prioridad: Media

RF 022 - *Loguin*
Descripción: Los usuarios contarán con una un inicio de sesión
Prioridad: Alta

RF 023 - *LogOut*
Descripción: Los usuarios contarán con una un deslogueo
Prioridad: Alta

2.2 Requerimientos No Funcionales

RNF 001 - Ambiente de desarrollo
Descripción: Se utilizará Visual Studio 2017 como IDE y .Net con C# como lenguaje de desarrollo y Framework .Net Core 2.1.

RNF 002 - Extensibilidad
Descripción: Deberá ser trivial agregar un nuevo módulo a la plataforma, sin que ello implique alterar el código principal de la misma.

RNF 003 - Persistencia

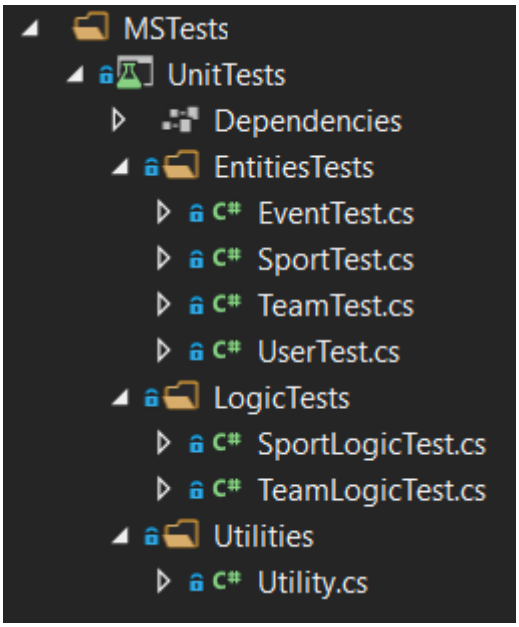
Descripción: Los datos deberán ser persistidos en una base de datos Microsoft SQL Server Express 2012 o mayor, utilizando Enith Framework con la estrategia Code First.

3. Descripción del diseño propuesto

El diseño planteado por nosotros busca dar solución a la letra del obligatorio aplicando los conceptos de Clean Code y principios SOLID vistos en clase. El objetivo es producir una solución bien estructurada y con código de calidad, lo que permitirá la extensibilidad a futuro.

En los puntos siguientes describiremos la solución justificando las decisiones tomadas.

3.1. UnitTests

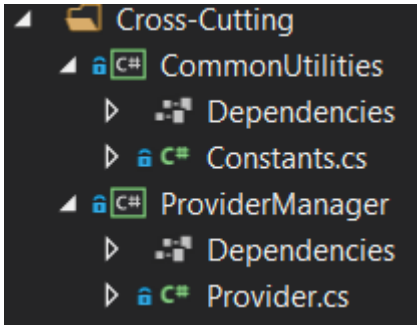


Se automatizan los casos de prueba utilizando **MSTests**. Dentro de la carpeta de solución ***MSTests*** se encuentran el proyecto ***UnitTests***. El mismo es un proyecto de pruebas unitarias, en la cual se creó una clase para cada entidad de negocio. Los test unitarios apuntan al testeo de las entidades del dominio y lógica de negocio, por lo que se utilizó el ***Framework Moq*** para abstraer las pruebas de los métodos de acceso a datos. Se cuenta con clases separadas para probar las entidades del dominio, y otras para probar la lógica de negocio.

En la siguiente imagen se evidencia un test unitario implementado en la clase ***SportLogicTest***, en el cual se intenta validar la creación de un deporte, y se utiliza la herramienta Mocking para los métodos **GetSports** y **AddSport**.

```
[TestMethod]
0 references | santilo, 2 days ago | 1 author, 1 change | 0 exceptions
public void AddSportThatNotExists()
{
    // Creo el objeto mock, en este caso una implementacion mockeada de ISportPersistance.
    var mock = new Mock<ISportPersistance>();
    mock.Setup(up => up.GetSports()).Returns(new List<Sport>());
    mock.Setup(mr => mr.AddSport(It.IsAny<Sport>())).Verifiable();

    // Instancio SportLogic con el mock como parametro.
    SportLogic userLogic = new SportLogic(mock.Object);
    Sport sportToAdd = Utility.GenerateRandomSport();
    userLogic.AddSport(sportToAdd);

    Assert.IsTrue(true);
}
```

3.2. Cross-Cutting



La carpeta de solución ***Cross-Cutting*** contiene el proyecto ***CommonUtilities*** y ***ProviderManager***. Esta carpeta contiene clases que son transversales a la solución.

- ***CommonUtilities***

En este proyecto tenemos la clase *Constants*, contiene constantes estáticas con mensajes a mostrar en UI.

- ***ProviderManager***

A modo de generar un bajo acoplamiento, decidimos exponer interfaces (como las que se encuentran en **3_BusinessLayer/BusinessContracts**) y ocultar implementaciones. Por este motivo necesitamos este proyecto para inicializar y obtener las operaciones de los módulos que queramos.

Implementamos el **patrón singleton** en la clase *Provider* para poder manejarnos con una única instancia, y en su constructor le decimos quienes serán las implementaciones a las interfaces que posee y expone:

```
public class Provider
{
    private IUserLogic UserLogic;
    private ITeamLogic TeamLogic;
    private ISportLogic SportLogic;
    private IEventLogic EventLogic;

    #region Singleton
    // Variable estática para la instancia, se necesita utilizar una función lambda ya que el constructor es privado.
    private static readonly Lazy<Provider> currentInstance = new Lazy<Provider>(() => new Provider());
    1 reference | santilo, 16 days ago | 2 authors, 5 changes | 0 exceptions
    private Provider()
    {
        this.UserLogic = new UserLogic(new UserPersistance());
        this.TeamLogic = new TeamLogic(new TeamPersistance());
        this.SportLogic = new SportLogic(new SportPersistance());
        this.EventLogic = new EventLogic(new EventPersistance());
    }
    2 references | Santiago Díaz, 18 days ago | 1 author, 1 change | 0 exceptions
    public static Provider GetInstance
    {
        get
        {
            return currentInstance.Value;
        }
    }
    #endregion

    0 references | Santiago Díaz, 17 days ago | 1 author, 1 change | 0 exceptions
    public IUserLogic GetUserOperations()
    {
        return this.UserLogic;
    }
```

De esta manera podemos utilizar a **Provider** desde la **WebApi** de la siguiente forma logrando desacoplarse de la implementación.

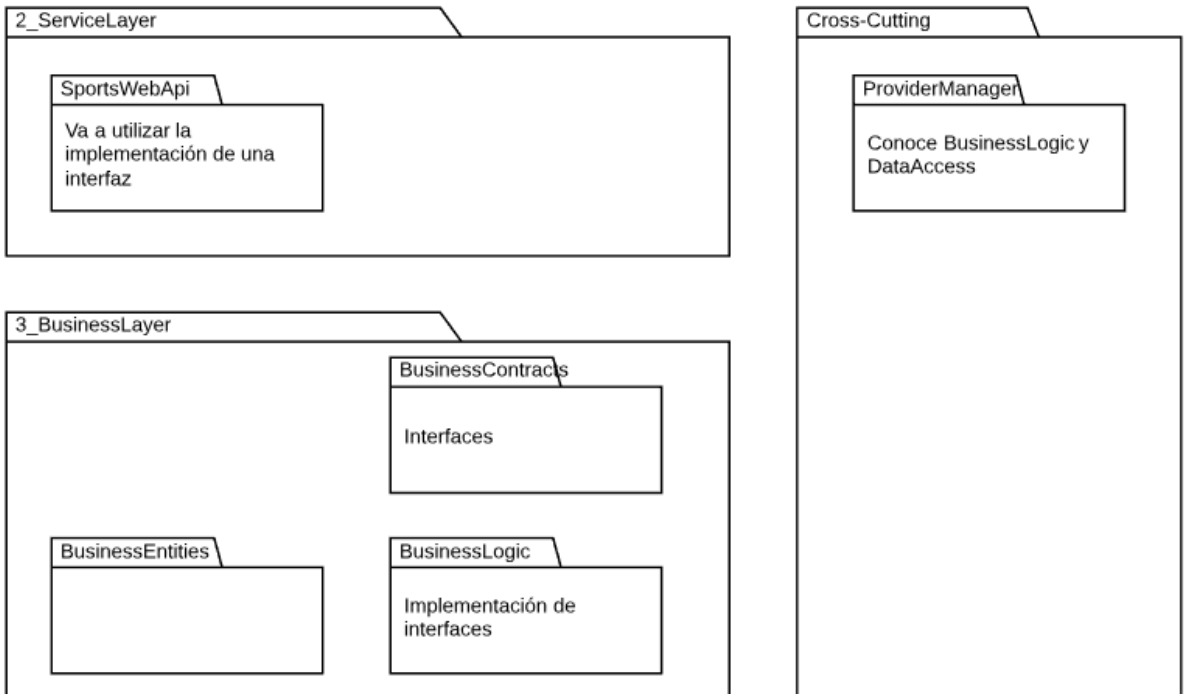
```
[Route("api/[controller]")]
[ApiController]
0 references | santilo, 1 day ago | 1 author, 2 changes
public class SportController : ControllerBase
{
    private ISportLogic sportOperations = Provider.GetInstance.GetSportOperations();

    [HttpPost()]
    0 references | santilo, 1 day ago | 1 author, 1 change | 0 requests | 0 exceptions
    public IActionResult AddSport([FromBody] AddSportInput addSportInput)
    {
        try
        {
            if (addSportInput == null) return BadRequest();

            Sport newSport = new Sport
            {
                Name = addSportInput.Name
            };

            sportOperations.AddSport(newSport);
            return Ok();
        }
        catch (Exception ex)//TODO: Ver como manejar los errores.
        {
            return this.StatusCode(500, ex.Message);
        }
    }
```

Y desde **BusinessLogic** podemos desacoplar la **Persistencia**, de la siguiente forma:

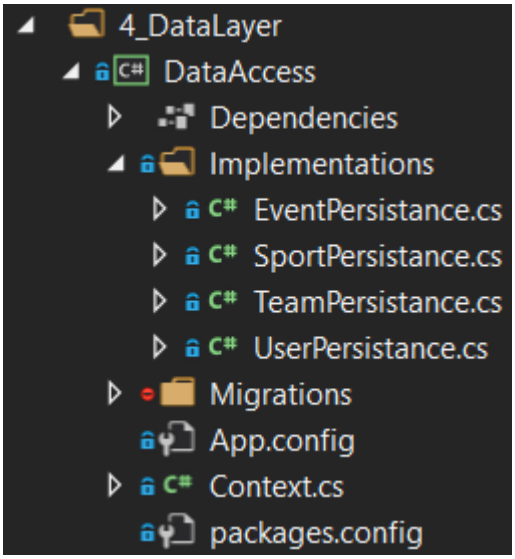
```
public class SportLogic : BusinessContracts.ISportLogic
{
    private ISportPersistance persistanceProvider;

    5 references | 0/4 passing | santilo, 2 days ago | 2 authors, 2 changes | 0 exceptions
    public SportLogic(ISportPersistance provider)
    {
        this.persistanceProvider = provider;
    }

    4 references | 0/2 passing | santilo, 2 days ago | 1 author, 1 change | 0 exceptions
    public void AddSport(Sport sportToAdd)
    {
        if (this.IsSportInSystem(sportToAdd))
            throw new Exception(Constants.SportErrors.ERROR_SPORT_ALREADY_EXISTS);
        else
            this.persistanceProvider.AddSport(sportToAdd);
    }
```

El constructor, al recibir como parámetro una implementación de la interfaz ***ISportPersistance***, evitamos el acoplamiento sobre cómo se van a persistir los datos, y nos evitamos problemas mayores si el día de mañana cambia la forma en la que se guardan los mismos.

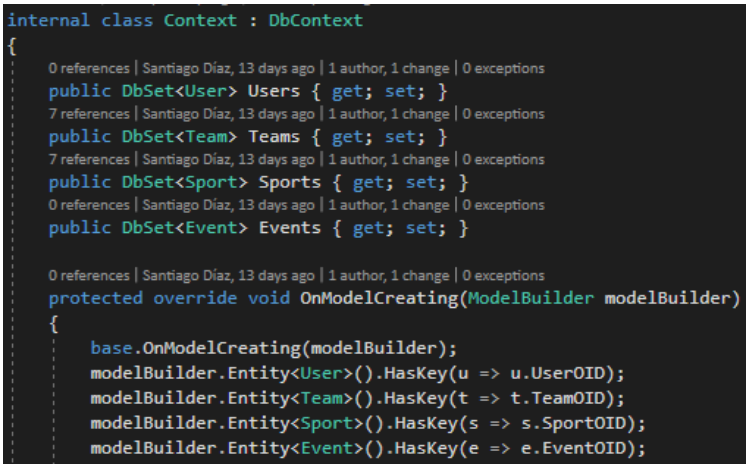
A nivel de paquetes, este comportamiento se podría visualizar de la siguiente manera:



3.3. DataLayer

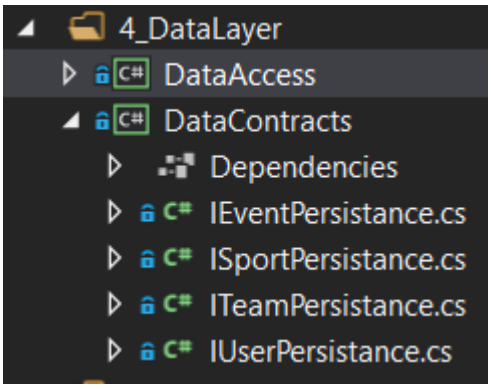


La carpeta de solución ***4_DataLayer*** contiene el proyecto de ***DataAcces*** donde se tienen las implementaciones a las Interfaces que utiliza cada clase de la lógica de negocios.



```
internal class Context : DbContext
{
    0 references | Santiago Díaz, 13 days ago | 1 author, 1 change | 0 exceptions
    public DbSet<User> Users { get; set; }
    7 references | Santiago Díaz, 13 days ago | 1 author, 1 change | 0 exceptions
    public DbSet<Team> Teams { get; set; }
    7 references | Santiago Díaz, 13 days ago | 1 author, 1 change | 0 exceptions
    public DbSet<Sport> Sports { get; set; }
    0 references | Santiago Díaz, 13 days ago | 1 author, 1 change | 0 exceptions
    public DbSet<Event> Events { get; set; }

    0 references | Santiago Díaz, 13 days ago | 1 author, 1 change | 0 exceptions
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Entity<User>().HasKey(u => u.UserOID);
        modelBuilder.Entity<Team>().HasKey(t => t.TeamOID);
        modelBuilder.Entity<Sport>().HasKey(s => s.SportOID);
        modelBuilder.Entity<Event>().HasKey(e => e.EventOID);
```

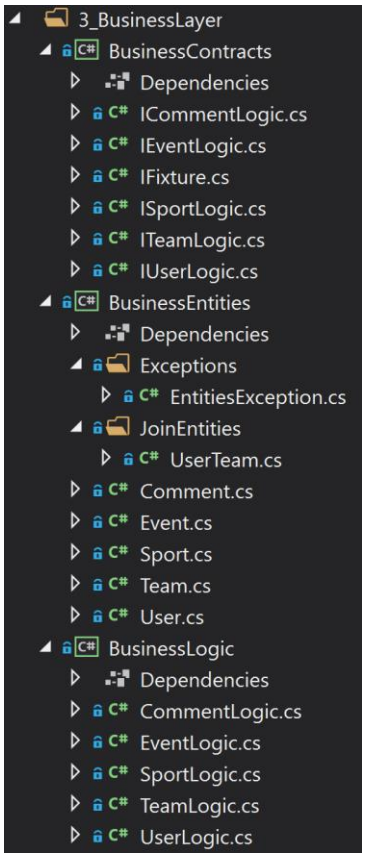
También tenemos la clase **Context**, quién hereda de **DBContext** para poder utilizar **EntityFramework**.
En esta clase **Context** se tienen los **DataSets** que son manejados desde las implementaciones que se encuentran en la carpeta Implementations.



En esta carpeta de solución se encuentra el proyecto **DataContracts**. El mismo contiene las interfaces que son implementadas por **BusinessLogic** y podría utilizarse para los test unitarios también, aunque en esta oportunidad para los mismos usamos Framework Mocking.

3.4. BusinessLayer

La carpeta de solución ***3_BusinessLayer*** contiene los proyectos principales del sistema. Las características principales de los mismos son:



- ***BusinessContracts*:**

En este proyecto se encuentran las interfaces que serán implementadas por cada módulo de negocio, esto nos permite el día de mañana, modificar o cambiar íntegramente, por ejemplo, la lógica de negocio ***SportLogic***, sin tener mayor impacto en otras secciones de la solución.

- ***BusinessEntities*:**

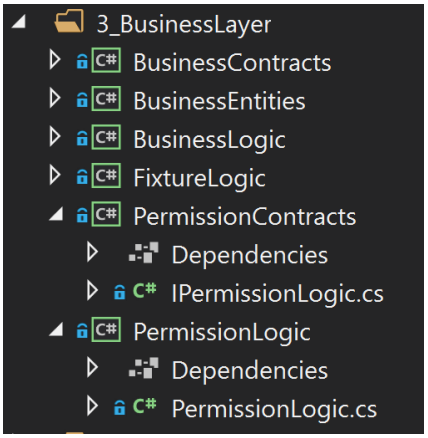
En este proyecto se encuentran las clases del dominio del problema.

- ***BusinessLogic*:**

Aquí se encuentra la lógica de negocio de cada módulo, como por ejemplo el *GetSports* o el *AddSport*. Este proyecto contiene la lógica de negocio a ser ejecutada antes de pasar por la capa de Persistencia e impactar los datos en la base de datos.
Cada una de estas clases, implementan una interfaz con las operaciones particulares para cada lógica.

- ***PermissionLogic*:**

En este componente se encuentra la lógica asociada a la autenticación y autorización (LogIn, HasPermission, LogOut, etc) de la aplicación. Se optó por separar esto en un componente aparte para poder desacoplarlo de los componentes de negocio propiamente dichos y así minimizar el impacto de cambio sobre esto a futuro.



- ***PermissionContacts*:**

Aquí se encuentran los contratos que deberán implementar las clases en *PermissionLogic*. De igual manera que lo mencionado anteriormente, los contratos se colocaron en este componente aparte con el fin de bajar el acoplamiento y aumentar su cohesión.

- ***FixtureLogic*:**

En este componente se encuentran las implementaciones de la interfaz *IFixture* la cual pertenece al componente de *BusinessContracts*.

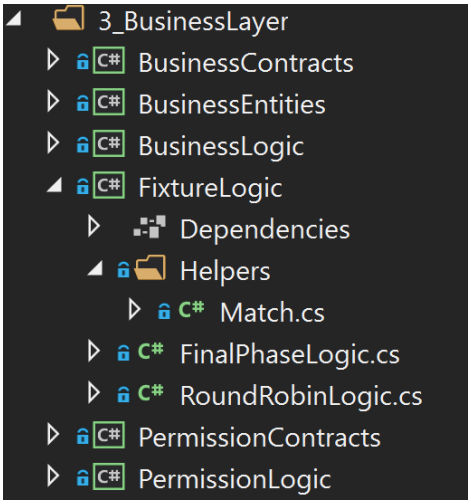
Para poder cumplir con el **RF 013** implementamos el patrón strategy utilizando la Interfaz *IFixture* como abstracción. Dicha interfaz es devuelta por el método *GetFixtureGenerator* de la clase Provider con la instanciación de la implementación según el tipo de fixture que se quiera generar.

```
public IFixture GetFixtureGenerator(FixtureType fixtureType)
{
    IFixture fixtureGenerationAlgorithm;
    switch (fixtureType)
    {
        case FixtureType.FinalPhase:
            fixtureGenerationAlgorithm = new FinalPhaseLogic();
            break;
        case FixtureType.RoundTrip:
            fixtureGenerationAlgorithm = new RoundRobinLogic();
            break;
        default:
            fixtureGenerationAlgorithm = null;
            break;
    }
    return fixtureGenerationAlgorithm;
}
```

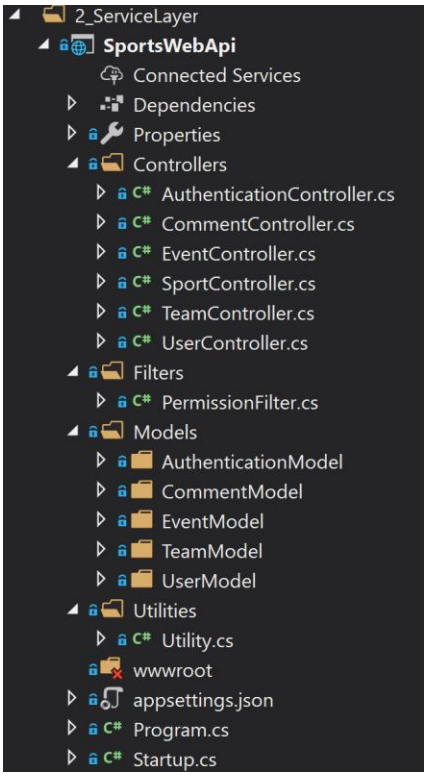
Se implementaron 2 formas de generación de fixtures, las cuales ambas devuelven una lista de eventos a la Web API (sin impactar en la BD) y esta los devuelve en formato json.

• El algoritmo ***RoundRobin***, se encarga de generar eventos ida y vuelta según la cantidad de equipos en el deporte. Valida que no pueda jugar un mismo equipo contra sí mismo y que tampoco pueda jugar ese mismo equipo el mismo día. La periodicidad entre encuentros para un mismo equipo es de 1 día, siendo la fecha del primer evento a generar la que se da por parámetro.

• El algoritmo ***FinalPhase*** genera la llave correspondiente a la fase final de un campeonato, por ejemplo, el campeonato FIFA del mundo. Se valida que la cantidad de equipos en el deporte sea potencia de 2 para poder realizarse (2, 4, 8, 16, etc).



3.5. 2_ServiceLayer

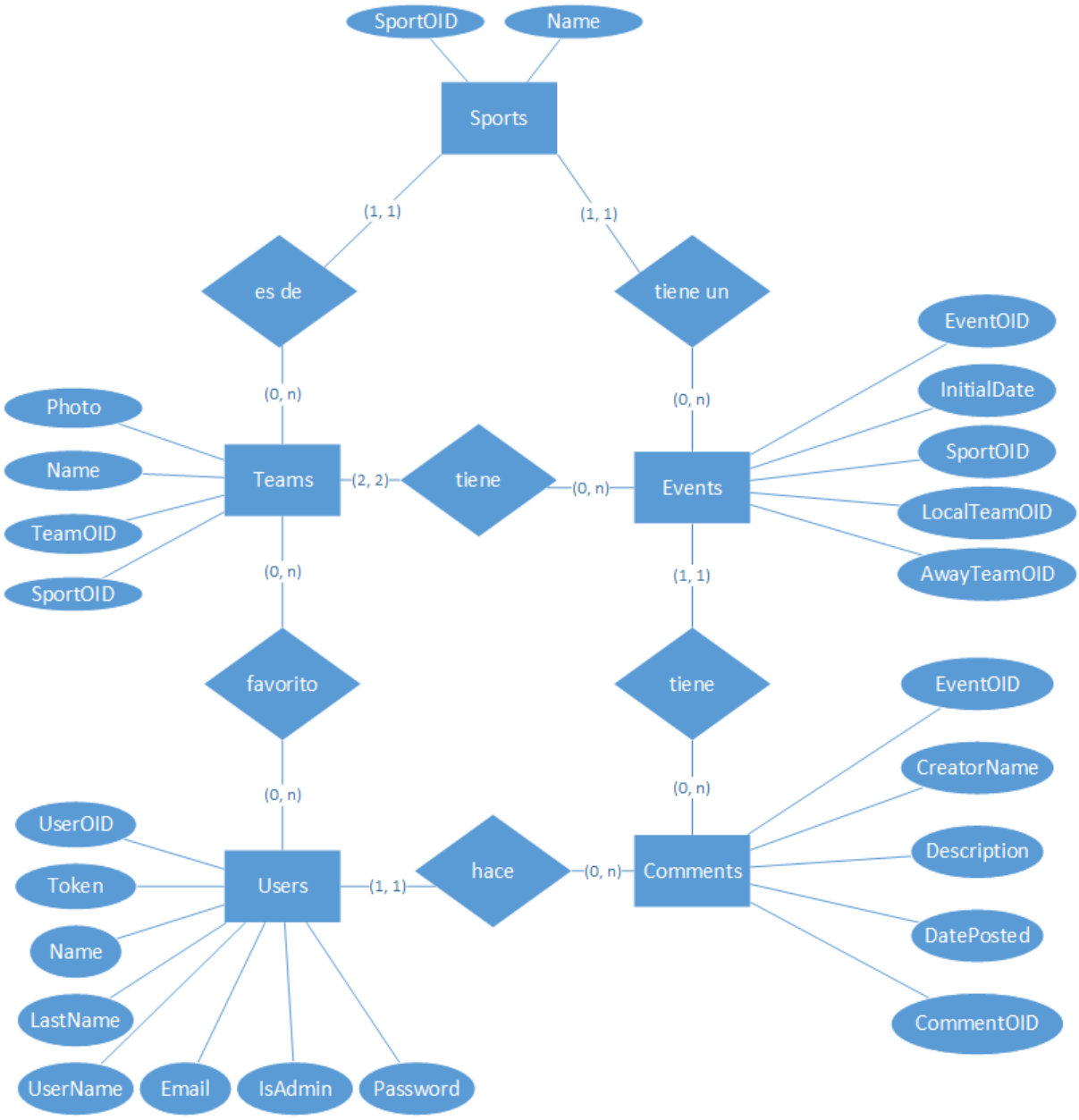


En esta capa se encuentra nuestra API Rest, la cual ofrece todas las operaciones necesarias para obtener la completitud de los requerimientos funcionales.
Como ya se especificó en el punto ***3.2. Cross-Cutting***, el mismo consumirá el ***Provider*** para obtener las operaciones de la lógica de negocio.
También encontraremos un ***Controller*** para cada módulo desarrollado, para independizar las implementaciones por módulos, así como encontraremos en la carpeta ***Models***, modelos de entidades para la transferencia de entidades a través de la capa Rest hacia el exterior o viceversa.

La clase ***PermissionFilter*** se encarga de manejar la autorizaciones a los llamados de los endpoints. Se usa como decorado en todos los métodos de la WebApi a modo de validar si el usuario que quiere realizar determinado request tiene los privilegios adecuados para hacerlo.

4. Modelado de Base de Datos

En la siguiente sección, se adjunta el modelo de entidad relación final de la base de datos.

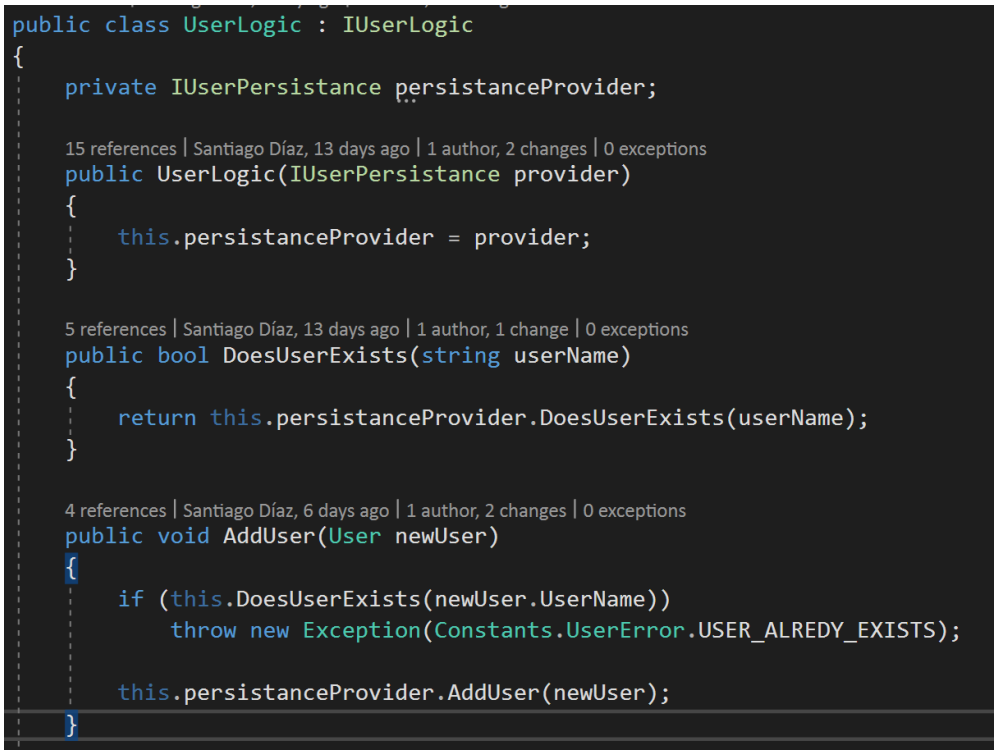


5. Evidencias de Clean Code

Esta sección del documento pretende mostrar evidencia mediante screenshots del código, que seguimos un proceso de desarrollo utilizando las prácticas mencionadas en el libro Clean Code de Robert C. Martin.

Capítulo 2 - Nombres con sentido

Respecto a los nombre, utilizamos palabras nemotecnias, fáciles de pronunciar y no son verbos (excepto en los métodos). Los métodos indican claramente lo que hacen con solo leer su firma. Ejemplo:



```
public class UserLogic : IUserLogic
{
    private IUserPersistance persistanceProvider;

    15 references | Santiago Díaz, 13 days ago | 1 author, 2 changes | 0 exceptions
    public UserLogic(IUserPersistance provider)
    {
        this.persistanceProvider = provider;
    }

    5 references | Santiago Díaz, 13 days ago | 1 author, 1 change | 0 exceptions
    public bool DoesUserExists(string userName)
    {
        return this.persistanceProvider.DoesUserExists(userName);
    }

    4 references | Santiago Díaz, 6 days ago | 1 author, 2 changes | 0 exceptions
    public void AddUser(User newUser)
    {
        if (this.DoesUserExists(newUser.UserName))
            throw new Exception(Constants.UserError.USER_ALREDY_EXISTS);

        this.persistanceProvider.AddUser(newUser);
    }
```

Capítulo 3 - Funciones

Las funciones y métodos fueron desarrollados teniendo en cuenta las siguientes pautas: máximo 3 parámetros de entrada, tener la menor cantidad de líneas posible, firmas con nombres descriptivos y que realicen una única tarea. Se buscó siempre partir cada método en dos o más métodos, siempre que esto ayudará a mejorar la calidad del código. Tuvimos presente el lineamiento de definir a cada función en el orden en que iban apareciendo en el código.

```
namespace BusinessLogic
{
    10 references | santilo, 7 days ago | 2 authors, 4 changes
    public class TeamLogic : BusinessContracts.ITeamLogic
    {
        private ITeamPersistance persistanceProvider;

        5 references | santilo, 12 days ago | 2 authors, 2 changes | 0 exceptions
        public TeamLogic(ITeamPersistance provider)
        {
            this.persistanceProvider = provider;
        }

        4 references | santilo, 12 days ago | 1 author, 2 changes | 0 exceptions
        public void AddTeam(Team newTeam)
        {
            if (this.IsTeamInSystem(newTeam))
                throw new Exception(Constants.TeamErrors.ERROR_TEAM_ALREADY_EXISTS);
            else
                this.persistanceProvider.AddTeam(newTeam);
        }

        1 reference | santilo, 12 days ago | 1 author, 2 changes | 0 exceptions
        private bool IsTeamInSystem(Team team)
        {
            bool result = false;
            List<Team> systemTeams = this.persistanceProvider.GetTeams();
            foreach (var teamAux in systemTeams)
            {
                if (teamAux.Equals(team)) { result = true; };

            }
            return result;
        }
```

Capítulo 4 - Comentarios

Clean Code menciona que se debe limitar el uso de comentario a lugares donde realmente sea necesario. Nos propusimos escribir código lo suficientemente claro como para poder evitarlos, pero en ciertos lugares fue necesario para explicar algunas particularidades del lenguaje como por ejemplo:

```
#region Singleton
// Variable estática para la instancia, se necesita utilizar una función lambda ya que el constructor es privado.
private static readonly Lazy<Provider> currentInstance = new Lazy<Provider>(() => new Provider());
1 reference | Santiago Díaz, 13 days ago | 2 authors, 6 changes | 0 exceptions
private Provider()
{
```

Capítulo 5 - Formato

Tratamos de que los archivos en general no sea muy grandes en lo que respecta al número de líneas, además de tener una estructura que permita la fácil lectura de la clase y sus métodos. Para esto, dejamos una línea libre entre cada método, cada variable fue declara lo más cercano posible a su uso y en algunos casos se dejaron líneas en blanco dentro del cuerpo de algunos métodos con el fin de que su lectura sea lo más sencilla posible.

```
public bool ModifyUser(User userWithModifications)
{
    try
    {
        bool changesWereMade = false;
        User userToModify = this.GetUserByUserName(userWithModifications.UserName);

        if (userToModify == null)
            throw new EntitiesException(Constants.UserError.USER_NOT_FOUND, ExceptionStatusCode.NotFound);

        if (this.CheckForModifications(userToModify, userWithModifications))
        {
            this.persistanceProvider.ModifyUser(userToModify);
            changesWereMade = true;
        }

        return changesWereMade;
    }
    catch (Exception ex)
    {
        throw new Exception(Constants.Errors.UNEXPECTED, ex);
    }
}
```

Capítulo 6 - Objetos y estructuras de datos

Intentamos esconder todo lo que fue posible y aumentar la visibilidad a medida que fuese necesario. Cumplimos con la Ley de Demeter y así evitar el conocido “choque de trenes”. En los casos en pudimos haber caído en este error, partimos en más de una línea lo que podríamos hacer en dos.

Capítulo 7 - Procesar errores

No utilizamos códigos de error, sino que manejamos los errores mediante el uso de excepciones. Nos generamos una clase *EntitiesExeption* que hereda de *Exceptions* con el fin de manejar nuestros propios errores.

```
namespace BusinessEntities.Exceptions
{
    11 references | Santiago Díaz, 6 days ago | 1 author, 2 changes
    public class EntitiesException : Exception
    {

        4 references | Santiago Díaz, 6 days ago | 1 author, 1 change | 0 exceptions
        public ExceptionStatusCode StatusCode { get; private set; }

        0 references | Santiago Díaz, 6 days ago | 1 author, 2 changes | 0 exceptions
        public EntitiesException()
        {
            this.StatusCode = ExceptionStatusCode.Undefined;
        }

        3 references | Santiago Díaz, 6 days ago | 1 author, 2 changes | 0 exceptions
        public EntitiesException(string message, ExceptionStatusCode statusCode)
        : base(message)
        {
            this.StatusCode = statusCode;
        }

        0 references | Santiago Díaz, 6 days ago | 1 author, 1 change | 0 exceptions
        public EntitiesException(string message, ExceptionStatusCode statusCode, Exception inner)
        : base(message, inner)
        {
            this.StatusCode = statusCode;
        }
    }
    10 references | Santiago Díaz, 6 days ago | 1 author, 1 change
    public enum ExceptionStatusCode
    {
        Undefined,
        NotFound,
        InvalidData
    }
}
```

```
public void AddEvent(Event newEvent)
{
    if (this.DoesTeamsEventExists(newEvent))
        throw new EntitiesException(Constants.EventError.ALREADY_EXISTS, ExceptionStatusCode.InvalidData);

    this.PersistanceProvider.AddEvent(newEvent);
}
```

Capítulo 8 - Límites

Este capítulo refiere principalmente a la utilización de código de herramientas o de implementaciones de código de terceros. Dado que no es nuestro caso, no incluimos evidencia.

Capítulo 9 - Pruebas de unidad

Pusimos en práctica el proceso de desarrollo TDD y escribimos pruebas unitarias antes de comenzar el desarrollo propiamente dicho. De todas formas, luego de implementar las funcionalidad agregamos pruebas a efectos de mejorar la cobertura.
Seguimos los lineamientos de test unitario FIRST qué dicen que las pruebas deben ser rápidas, independientes, repetibles, auto evaluables y exhaustivas y oportunas (deben escribirse antes de la implementación).
En las pruebas aplicamos los mismos estándares de codificación que en el resto del proyecto, incluso creamos una clase estática *Utility* con el fin de agilizar el desarrollo de las pruebas.

```
[TestMethod]
0 references | Santiago Díaz, 12 days ago | 1 author, 2 changes | 0 exceptions
public void TryGetUserByUserName()
{
    try
    {
        var mock = new Mock<IUserPersistance>();
        User mockedUser = Utility.GenerateRandomUser("santidiaz");
        mock.Setup(up => up.GetUserByUserName("santidiaz")).Returns(mockedUser);

        UserLogic userLogic = new UserLogic(mock.Object);
        string userToBeSearch = "santidiaz";
        User foundUser = userLogic.GetUserByUserName(userToBeSearch);

        Assert.AreEqual(foundUser.UserName, userToBeSearch);
    }
    catch (Exception ex)
    {
        Assert.Fail(ex.Message);
    }
}
```

Capítulo 10 - Clases

El tamaño de las clases son bastante reducidos y cumplimos con el principio de responsabilidad única. Intentamos que nuestras clases tengan la menor cantidad de líneas posibles e hicimos un énfasis en seguir los principios de responsabilidad única (SRP) y abierto cerrado (OCP), de forma de aumentar la cohesión y bajar el acoplamiento como se menciona a más detalle en la sección de justificación de diseño.

Capítulo 11 - Sistemas

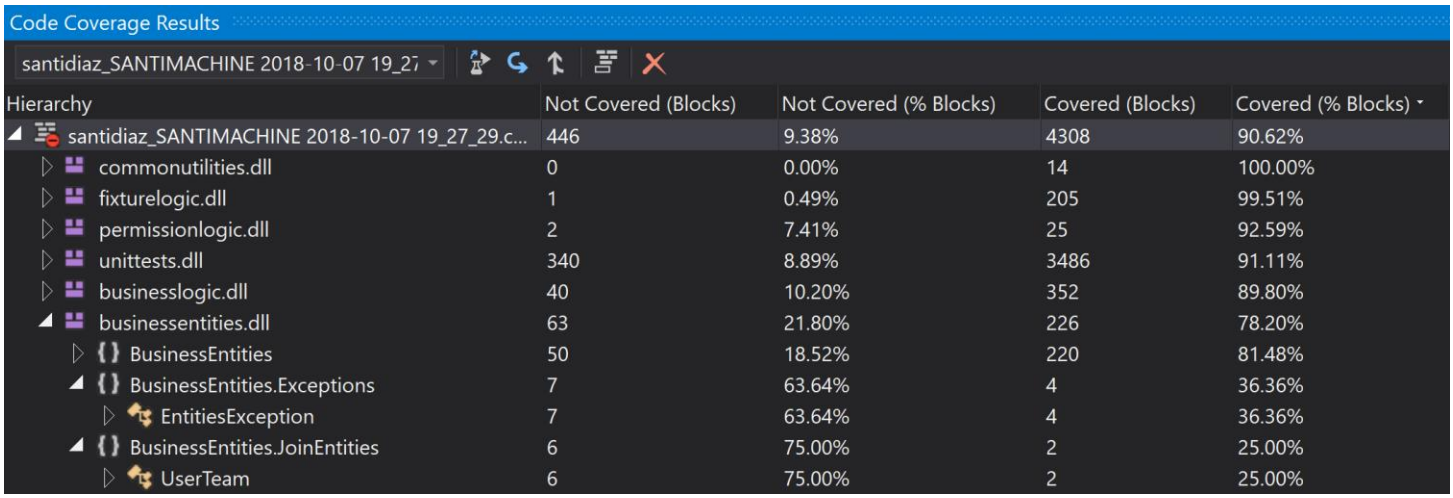
Al igual que en el punto anterior, la mejor forma de mostrar el seguimiento de los lineamientos de este capítulo es ir a la sección Justificación del diseño. Como podrá verse ahí, separamos el problema de construir el sistema en construir varios subsistemas.

Capítulo 12 - Emergencias

Buscamos que nuestra solución fuese sencilla. Al utilizar TDD logramos cumplir con esto sin mayores inconvenientes, dado que hacíamos pruebas, implementamos el código necesario para pasar esas pruebas y refactorizábamos, iterando hasta tener la funcionalidad implementada.

6. Resultado de Pruebas Unitarias

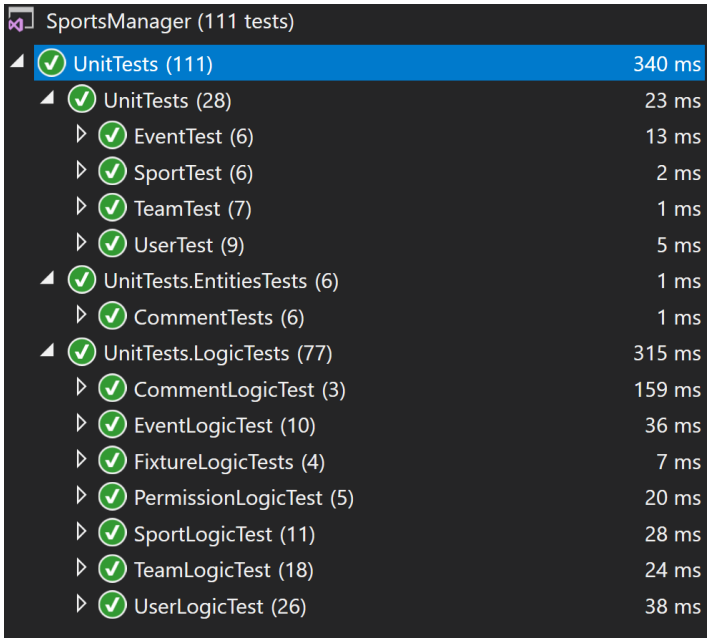
El desarrollo de la solución fue siguiendo las prácticas del proceso TDD. A continuación se muestra el análisis de cobertura realizado con Visual Studio. Algunos puntos donde la cobertura parecería ser baja, como *JoinEntities* o *Exceptions* se debe a que entendimos que no era necesario aplicar tests sobre estas entidades o métodos particulares de las mismas (constructor sin parámetros de *EntitiesException*) ya que no suma al objetivo del obligatorio.



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
santidiaz_SANTIMACHINE 2018-10-07 19_27_29.c...	446	9.38%	4308	90.62%
commonutilities.dll	0	0.00%	14	100.00%
fixturelogic.dll	1	0.49%	205	99.51%
permissionlogic.dll	2	7.41%	25	92.59%
unittests.dll	340	8.89%	3486	91.11%
businesslogic.dll	40	10.20%	352	89.80%
businessentities.dll	63	21.80%	226	78.20%
BusinessEntities	50	18.52%	220	81.48%
BusinessEntities.Exceptions	7	63.64%	4	36.36%
EntitiesException	7	63.64%	4	36.36%
BusinessEntities.JoinEntities	6	75.00%	2	25.00%
UserTeam	6	75.00%	2	25.00%

Como se puede apreciar logramos una cobertura del dominio del sistema de un 90.62%.

Todas las pruebas unitarias realizadas obtuvieron resultados satisfactorios, tal como se muestra en la siguiente imagen.



7. Manual de instalación

1. Abrir el SQL Server Managment
2. Ejecutar el script o levantar el .bak de creación de la base de datos. El mismo se encuentra en la carpeta Documentos de la entrega.
3. Crear un usuario para inicio de sesión, se le debe asignar el nombre IIS APPPOOL/{nombreDelSitioEniis} : en nuestro caso lo nombraremos SportManager.
4. Asignarle al usuario creado la BD de SportManager como db_owner de la misma.
5. Abrir el iis Manager
6. Crear un nuevo sitio con la siguiente información :
    a. Nombre del sitio “SportManager”
    b. Ruta de acceso física : Apuntar a la ruta de acceso física la cual contenta el publich de la solución, preferentemente alojar la carpeta publish en “c:\inetpub”.
    c. Puerto : *Preferentemente el 5000*
7. Validar que el sitio haya levantado.
8. Ajustar el archivo alojado en la carpeta de la publicación publish/appsettings.json, modificar la sección de ConnectionStrings, la key SportManagerCS, el valor Server, por la instancia correspondiente donde se haya instalado restaurado la base de datos de SportManager.
9. Abrir Postman e importar la colección de request alojada en la carpeta Proyecto/SportsManager/PostmanRequests/Obligatorio1_DA2.postman_collection.json
10. Importar el Enviroment a Postman desde el archivo alojado en Proyecto\SportsManager\PostmanRequests\SMEnvironment.postman_environment.json.
11. Configurar las variables del Enviroment importado de las siguiente manera
    IpValue : Se debe asignar la ip del servidor donde se haya montado el sitio.
    portValue : Se debe asignar el puerto por el cual este escuchando el sitio.
    tokenValue : Por defecto está vacío pero cuando se realice un login el mismo se cargará dinámicamente.
12. En caso de haber restaurado la base vacía, ejecutar la siguiente sentencia SQL para contar con un superadmin inicial, si la restauración fue desde la base de datos con información, el mismo ya lo incluye:

```
INSERT [dbo].[Users] ([UserOID], [Token], [Name], [LastName], [UserName], [Email], [IsAdmin], [Password]) VALUES (3, N'c9eabb01-b2ea-492c-bad8-9d6937d63b26', N'superadmin', N'superadmin', N'superadmin', N'a@a.com', 1, N'7C4A8D09CA3762AF61E59520943DC26494F8941B')
GO
```

13. Comenzar a ejecutar las operaciones con el usuario por defecto:
    a. userName : superadmin
    b. password : 123456

8. Anexos

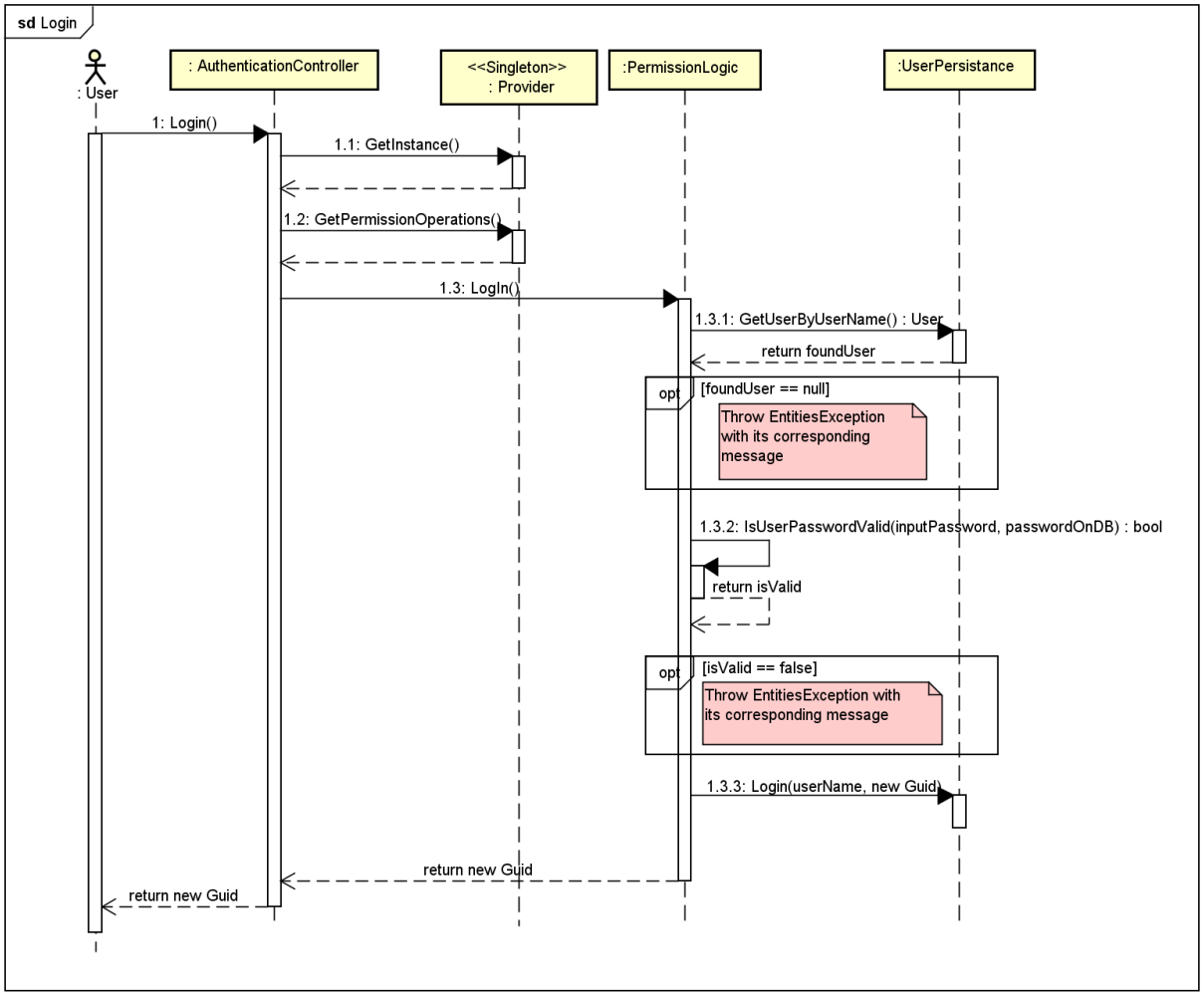
En esta sección se presentarán distintos diagramas de la solución, los mismos también se pueden encontrar en la carpeta ***Documentación/Diagramas*** del repositorio entregado. Respecto a los diagramas de secuencia, entendemos que la mayoría son análogos unos a otros y optamos por mostrar únicamente los que ayudan al entendimiento de la solución.
Nota : Para que los links de referencia a los diagramas funcionen correctamente, se debe contar con permisos hacia el repositorio y estar autenticado en GitHub.

8.1. Diagramas de Secuencia

Se adjuntan distintos diagramas de secuencia para visualizar la interacción de los objetos del sistema y la comunicación entre estos.

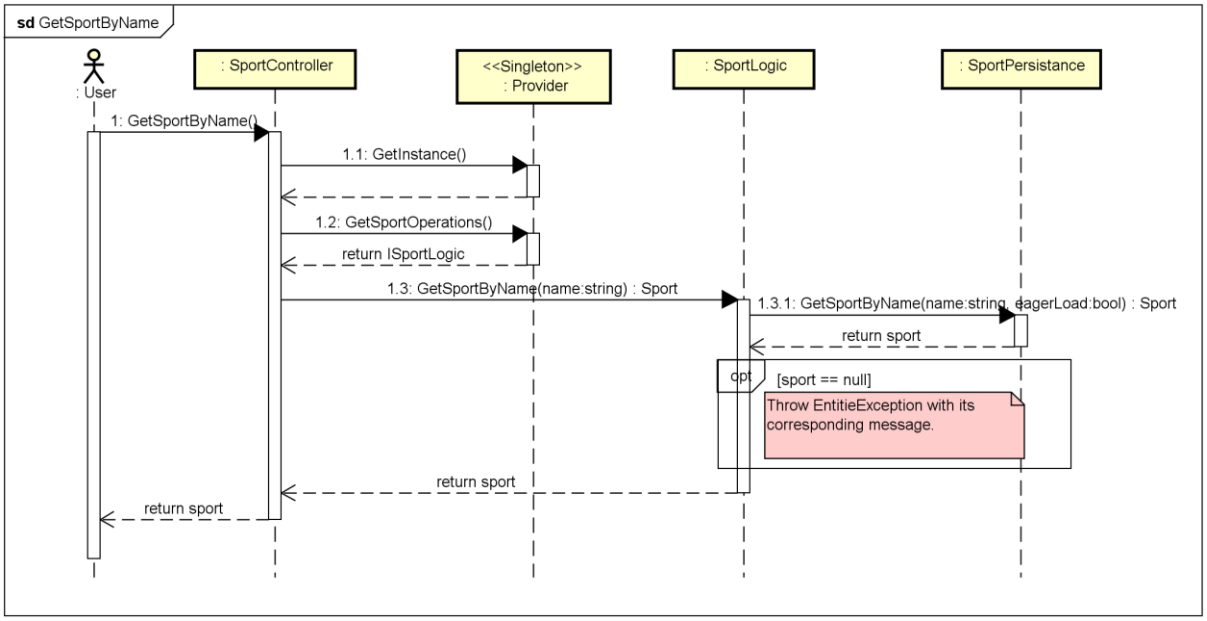
8.1.1 Login

Realizar un Login. (Login_SecDiag.png)



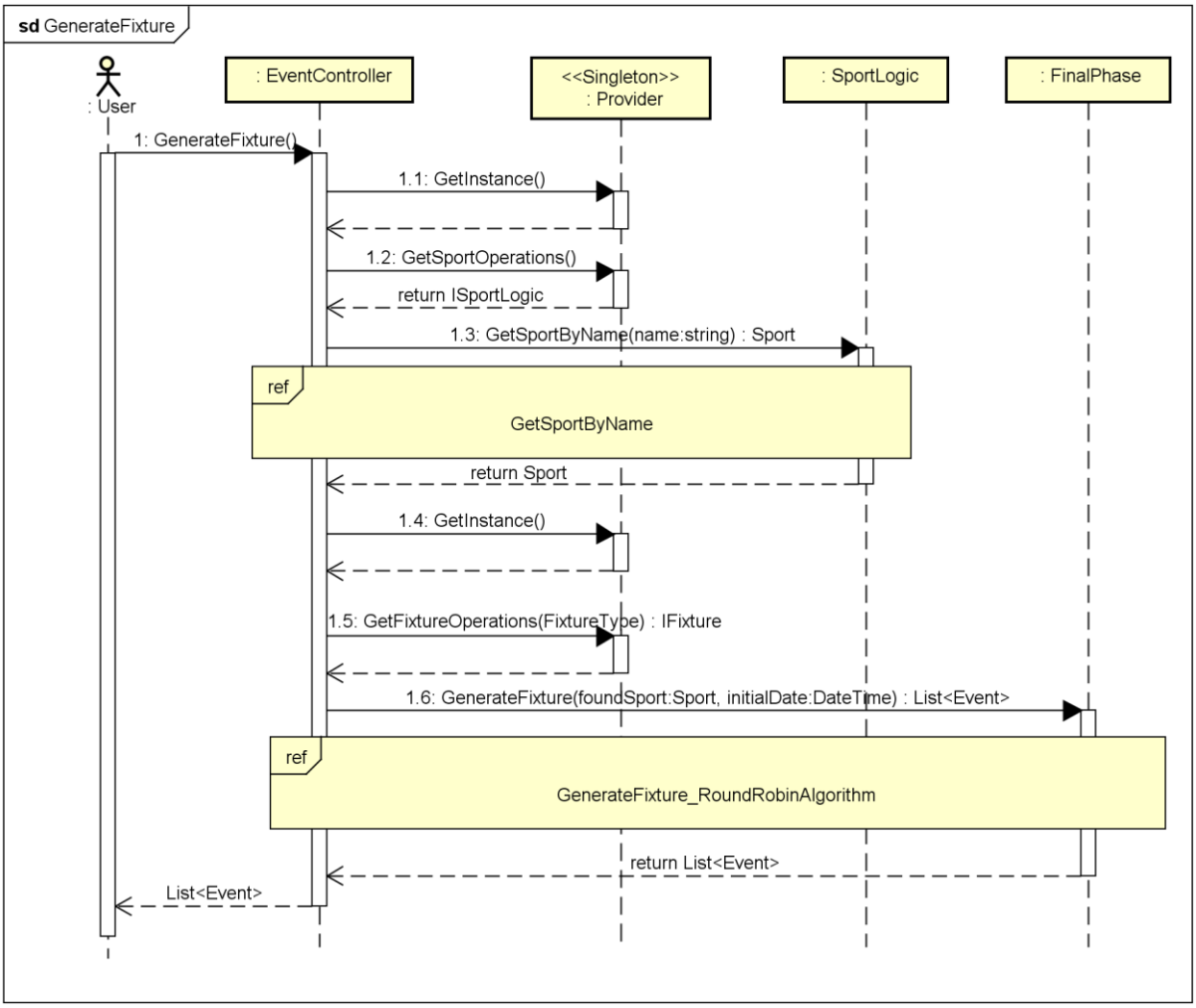
8.1.2 GetSportByName

Obtener un Sport a partir del nombre. (GetSportByName_SecDiag.png)



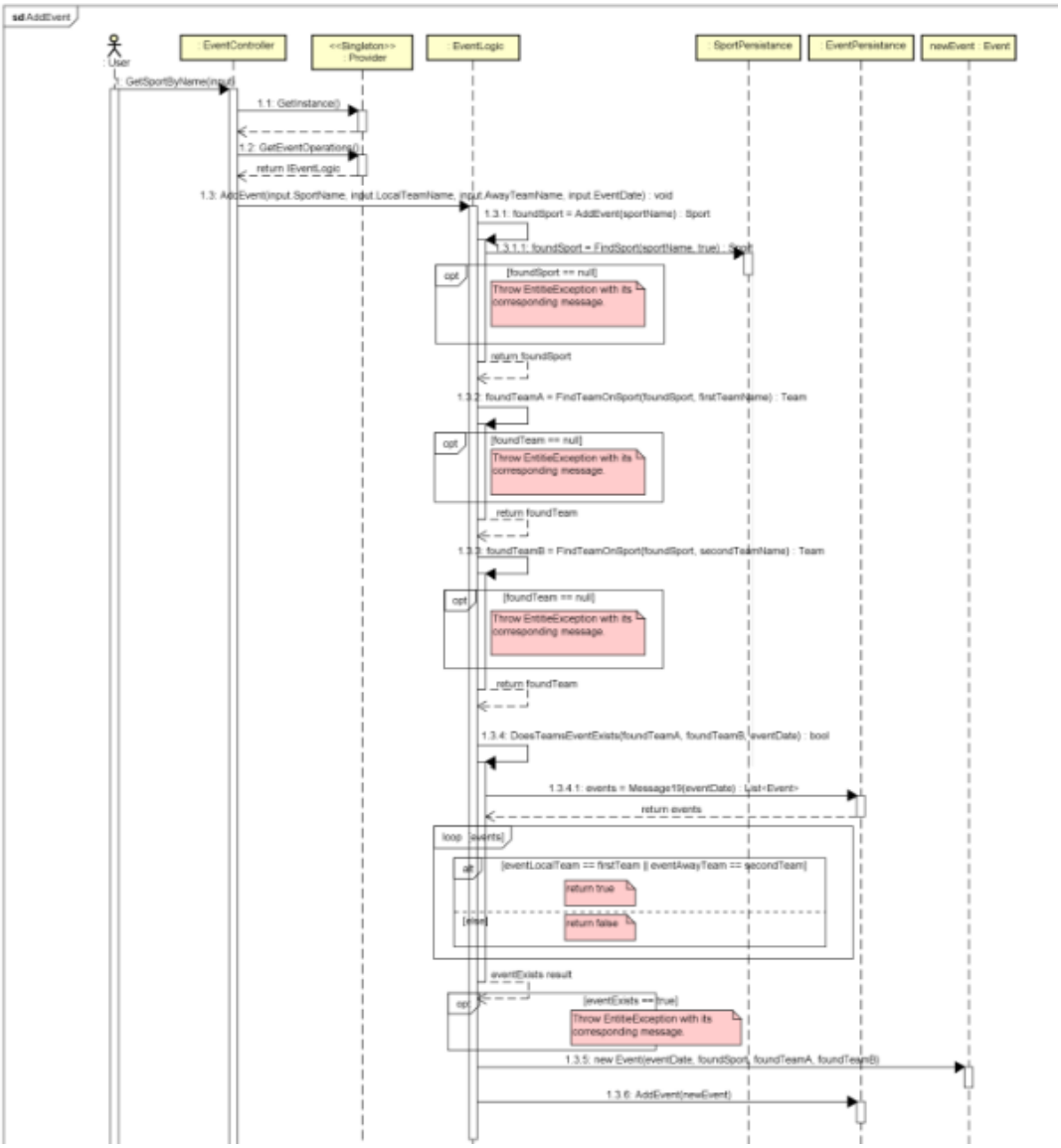
8.1.3 GenerateFixture

Generación de un fixture a partir de un tipo, deporte y fecha. (GenerateFixture_SecDiag.png)



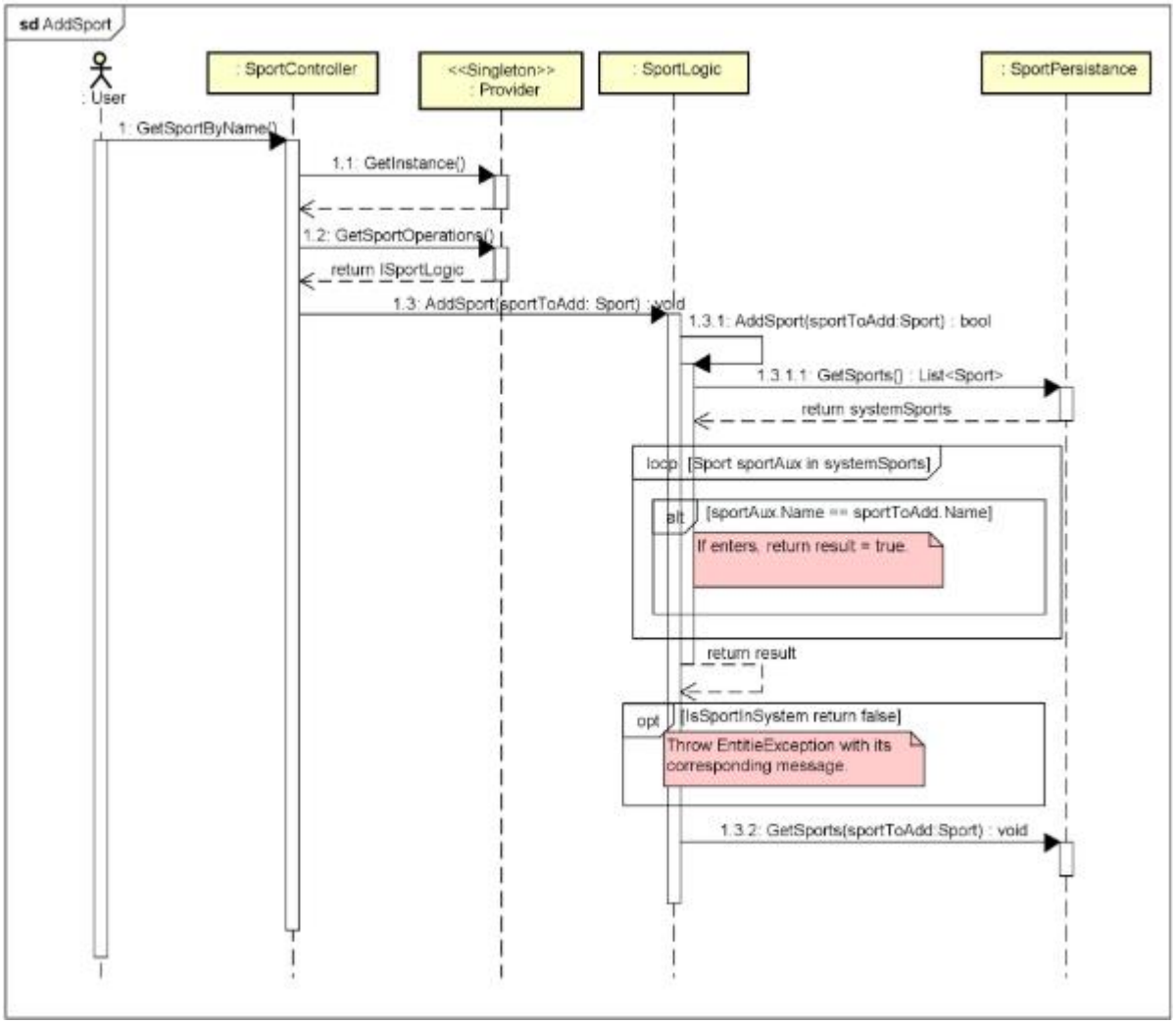
8.1.4 AddEvent

Alta de un evento.



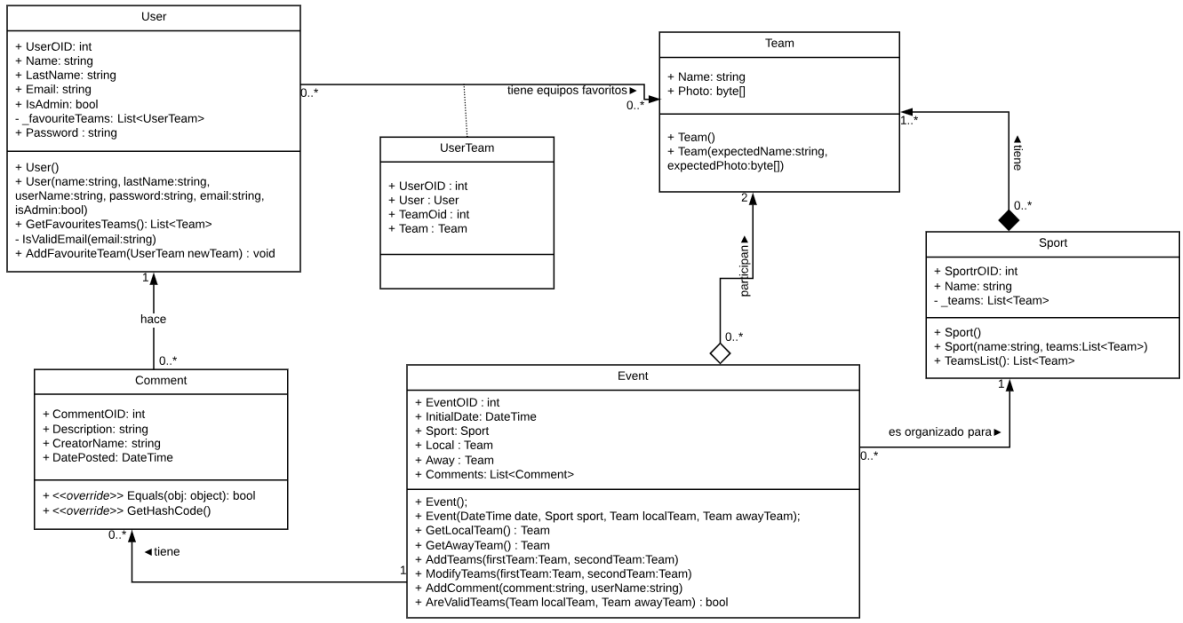
8.1.4 AddSport

Alta de un nuevo deporte. (AddSport_SecDiag.png)



8.2. Diagrama de clase del dominio

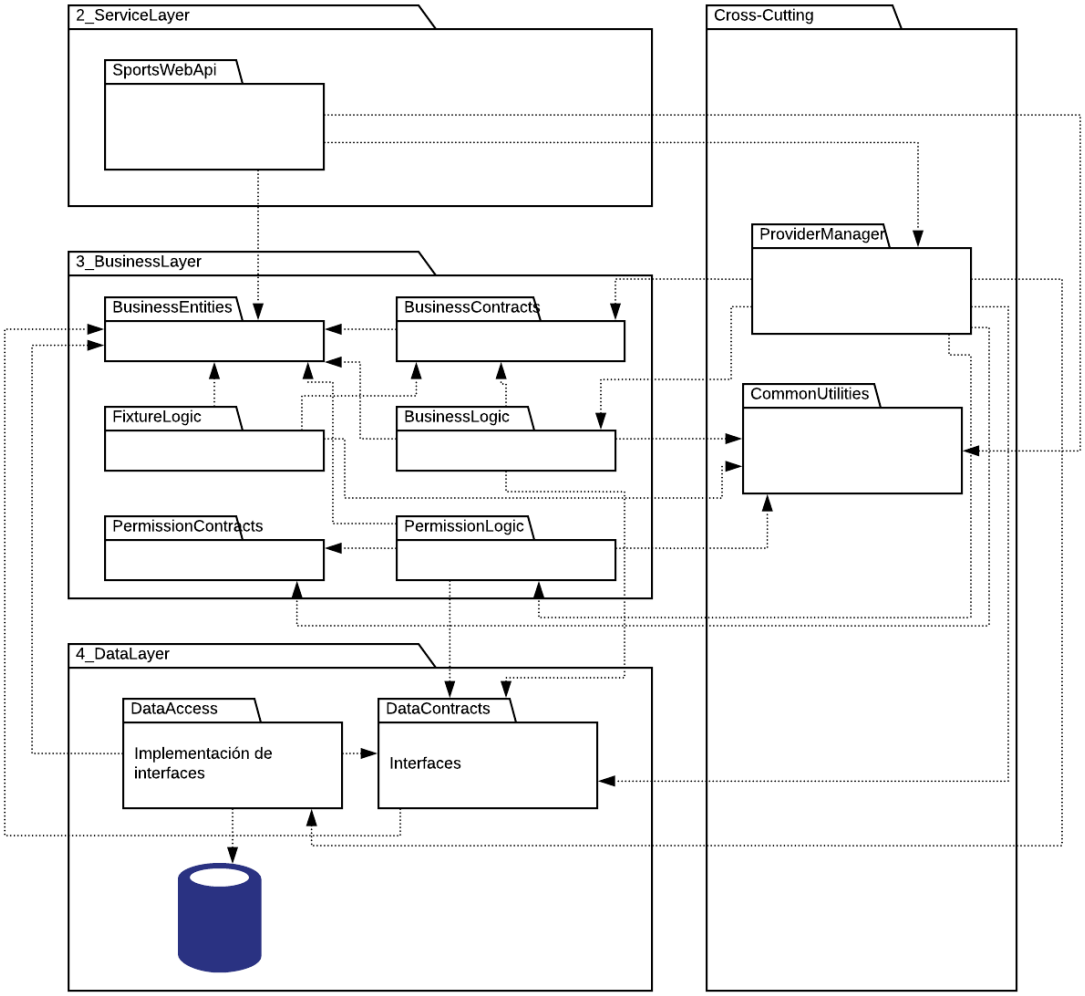
Se adjunta el diagrama de clases del dominio de la solución (Diagrama de clases del Dominio.png)



8.3. Diagrama de paquetes

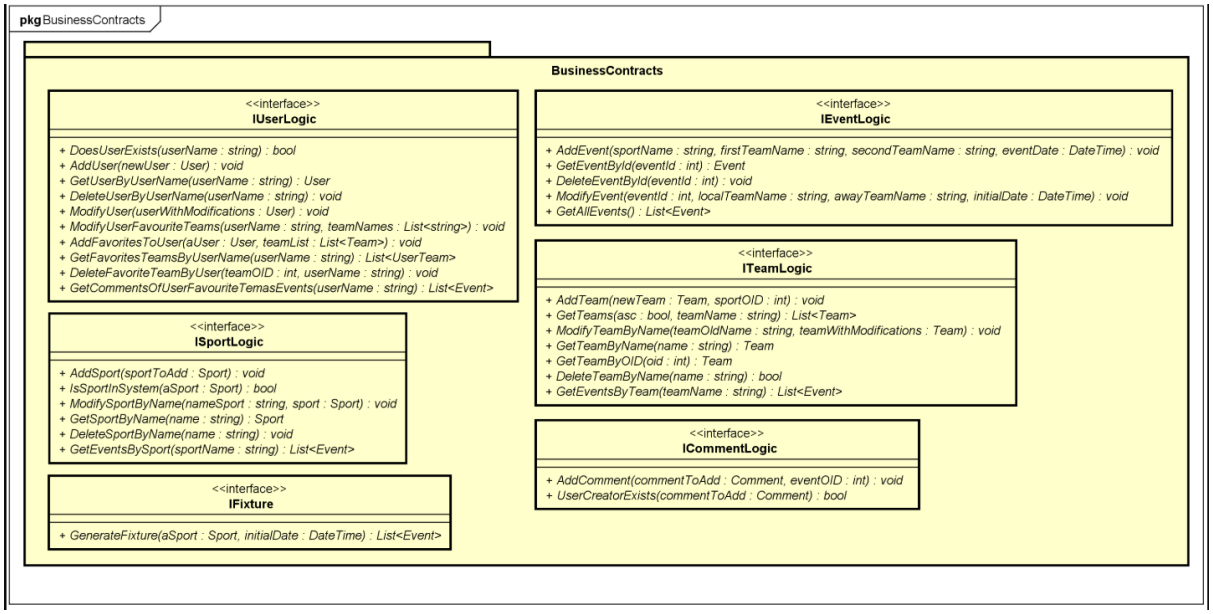
8.3.1 Diagrama de paquetes general

Se adjunta el diagrama de paquetes general de la solución con todas sus referencias entre los paquetes. (Diagrama de paquetes.png)



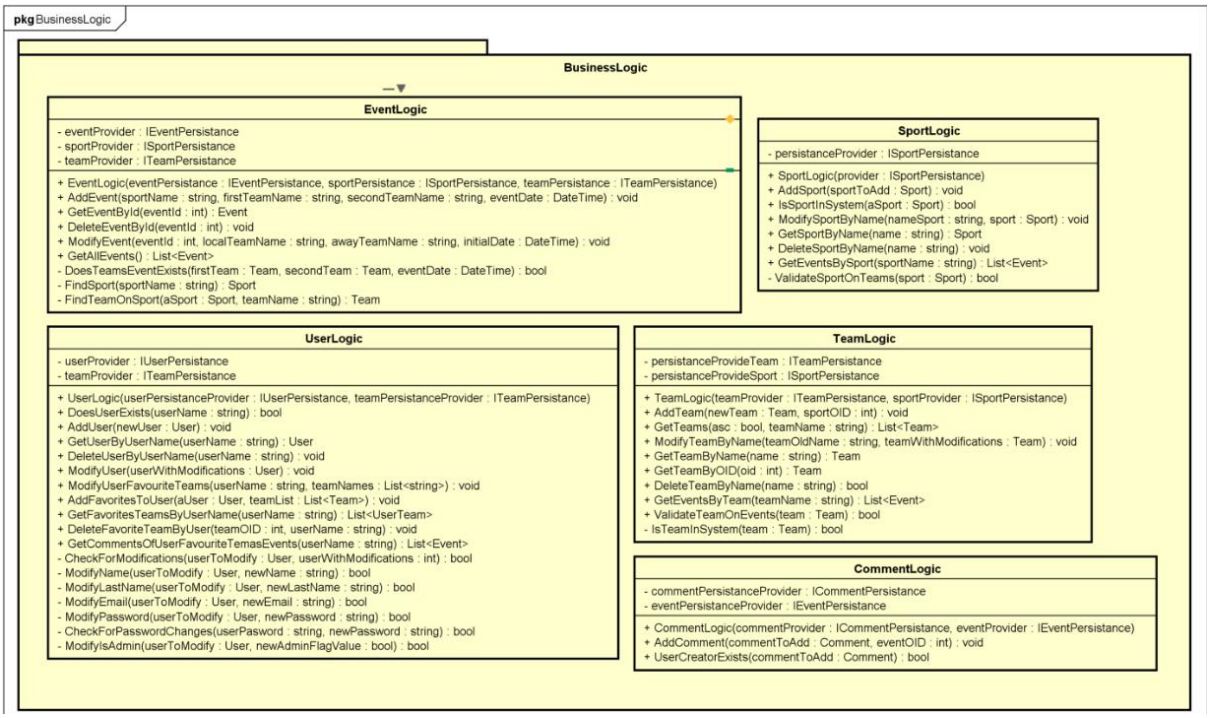
8.3.2 Diagrama de paquetes de BusinessContracts

Se adjunta el diagrama de paquetes de BusinessContracts, la cual contiene todas las interfaces de la lógica de negocios. (BusinessLogic_Package.png)



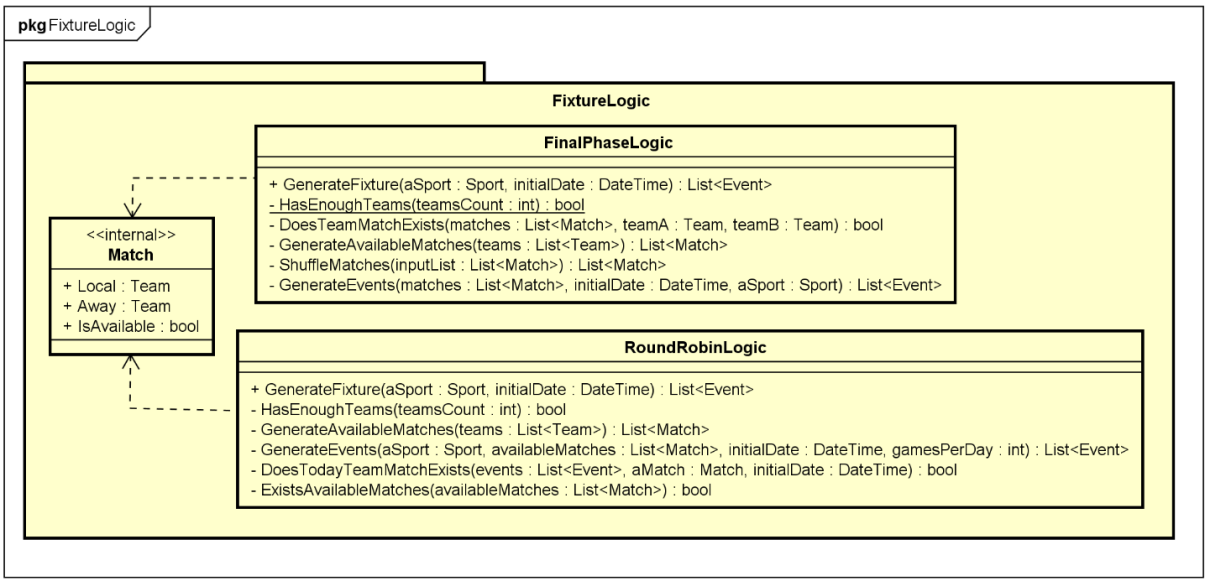
8.3.3 Diagrama de paquetes de BusinessLogic

Se adjunta el diagrama de paquetes de BusinessLogic, la cual contiene todas las implementaciones posibles para las interfaces mencionadas anteriormente. (BusinessLogic_Package.png)



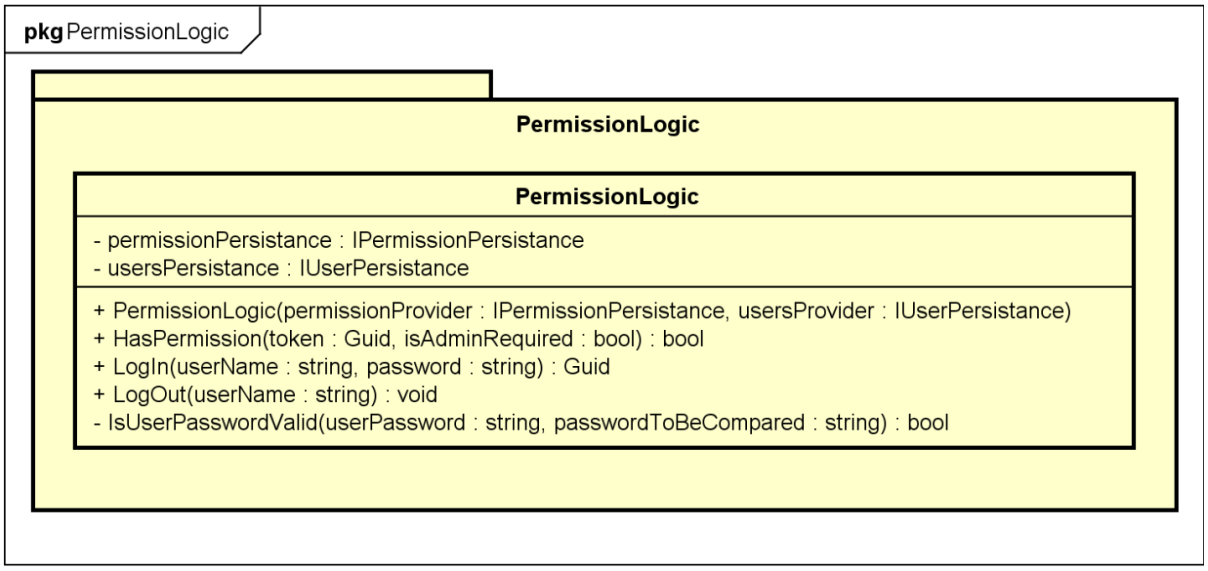
8.3.4 Diagrama de paquetes de FixtureLogic

Se adjunta el diagrama de paquetes de FixtureLogic, la cual contiene todas las distintas logicas y clases para la implementación del patrón strategy para la generación de los fixtures y distintos algoritmos. (FixtureLogic_Package.png)



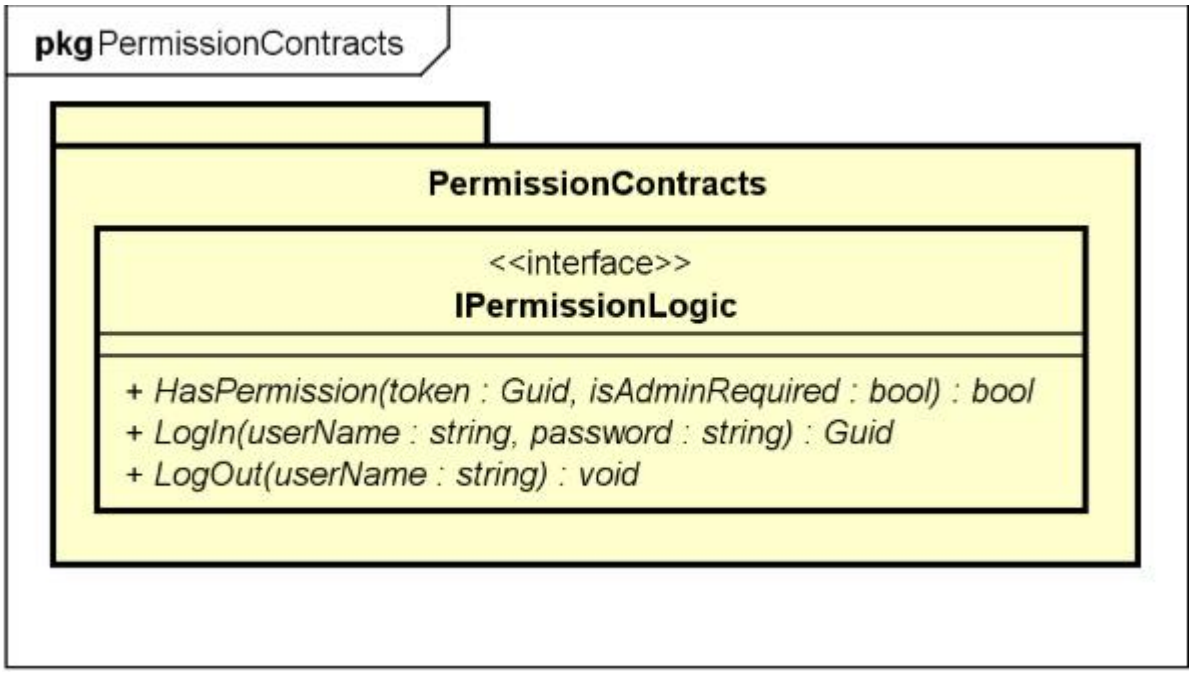
8.3.5 Diagrama de paquetes de PermissionLogic

Se adjunta el diagrama de paquetes de PermissonLogic, la cual contiene la lógica para la validación de permisos, Login y LogOff de la solución. (PErmissionLogic_Package.png)



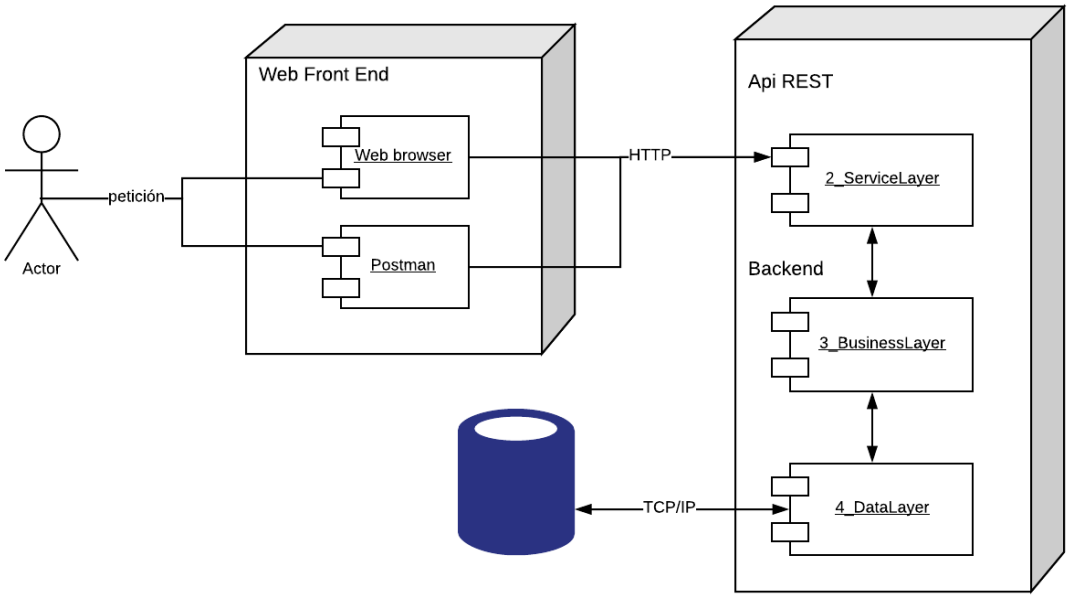
8.3.6 Diagrama de paquetes de PermissionContracts

Se adjunta el diagrama de paquetes de PermissonContracts, la cual contiene la interfaz para la implementación de IPermissionLogic. (PermissionContracts_Package.png)



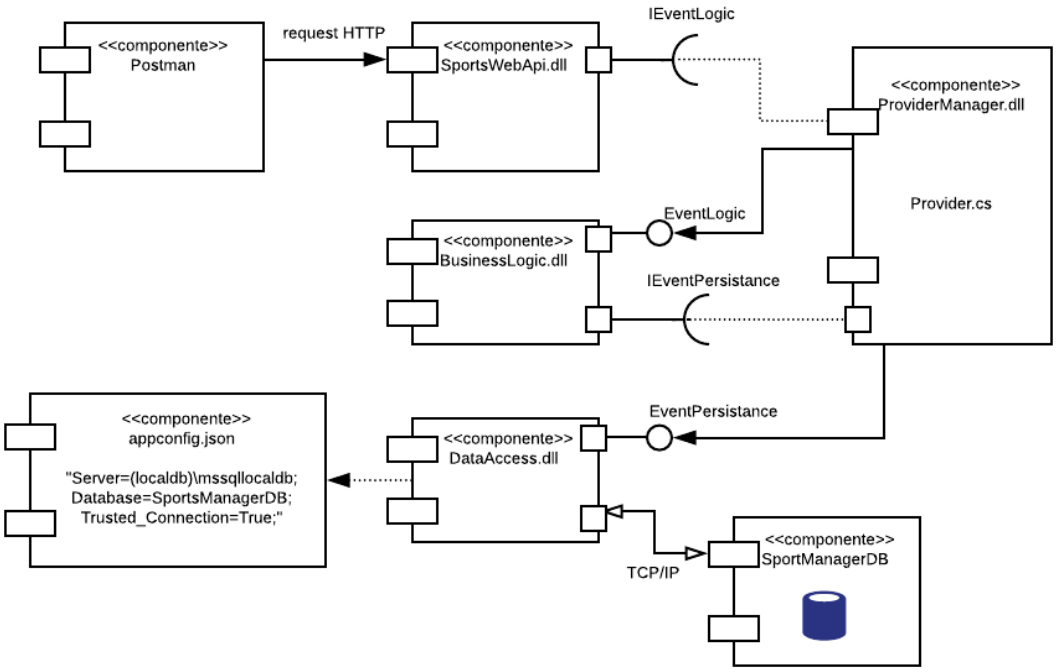
8.9. Diagrama de deploy

Se presenta el diagrama de deploy/entrega de la solución, en el mismo se supone que ya se cuenta con un actor interactuando desde un browser o postman. Este realiza peticiones HTTP a el servidor de aplicaciones, en este caso “Api REST”, la cual resuelve la petición HTTP interactuando entre sus componentes, y con la BD si es necesario. (Diagrama de Deploy.png)



8.10. Diagrama de componentes

Se adjunta un ejemplo de la interacción entre componentes al consumir una operación del módulo de Events. (Diagrama de componentes.png)



8.11. Diccionario de servicios

En el archivo ***Documentación/Diccionario.xls***, se cuenta con un diccionario de servicios en formato excel donde se podrá visualizar las operaciones disponibles con sus respectivos EndPoints y referencias al requerimiento funcional y permisos. Si bien el Administrador no tiene marcada todas las funcionalidades, porque el análisis del diccionario fue a partir de la letra del obligatorio, el mismo cuenta con todas las funcionalidades habilitadas.