

**GRUPO**

N°3

**CURSO**

5K2

**DOCENTES**

Ing. Zohil, Julio

Ing. Liberatori, Marcelo

Ing. Jaime, Natalia

**ALUMNOS**

Allemand, Facundo leg. 58971

Herrera, Antonio leg. 57824

Pedrosa, Paula Melania leg. 58822

Rojas Amaya, M. Florencia leg. 58577

### 

**UNIVERSIDAD TECNOLÓGICA NACIONAL**

**INGENIERÍA EN SISTEMAS DE INFORMACIÓN** **PROYECTO FINAL**

**PROYECTO**



**Que Golazo!**

**Sistema de Gestión de Torneos de Fútbol**

**Documento de Arquitectura**

|  |  |  |  |
| --- | --- | --- | --- |
| HISTORIAL DE VERSIONES | | | |
| VERSION | FECHA | RESPONSABLE | OBSERVACION |
| 1.0 | 11/06/2014 | Florencia Rojas | Creación Documento |
| 1.1 | 11/08/2014 | Florencia Rojas | Se agregaron vistas |
| 1.2 | 06/09/2014 | Florencia Rojas | Se agregó patrones de diseño y generación de fixture. |
| 1.3 | 25/10/2014 | Florencia Rojas | Se actualizó Diagrama de Clases y se agregó el modelo de datos. |

Tabla de contenido

[Introducción 3](#_Toc401996951)

[Arquitectura 4](#_Toc401996952)

[Patrón N-Tier Cliente - Servidor 4](#_Toc401996953)

[Arquitectura a nivel de Hardware 5](#_Toc401996954)

[Estructura de la Solución 6](#_Toc401996955)

[Relación entre las clases 7](#_Toc401996956)

[Modelos 8](#_Toc401996957)

[Diagrama de Clases de dominio 8](#_Toc401996958)

[Diagramas de Estado 9](#_Toc401996959)

[Clase Partido 9](#_Toc401996960)

[Clase Fecha 9](#_Toc401996961)

[Clase Fase 10](#_Toc401996962)

[Diagramas de Interacción 11](#_Toc401996963)

[Registrar Configuración de Edición 11](#_Toc401996964)

[Consulta de Jugadores 13](#_Toc401996965)

[Diseño 15](#_Toc401996966)

[Uso de Patrones 15](#_Toc401996967)

[Generación de Fixture 16](#_Toc401996968)

[Todos contra todos 16](#_Toc401996969)

[Todos contra todos Ida y Vuelta 16](#_Toc401996970)

[Eliminatoria por Fases 16](#_Toc401996971)

[Mapeo a la Base de Datos 17](#_Toc401996972)

[Prototipo de interfaz 18](#_Toc401996973)

[Login 18](#_Toc401996974)

[Mis Torneos 18](#_Toc401996975)

Documento de Arquitectura

# Introducción

En el presente documento se detalla la arquitectura del producto que se desarrollará. Se presentará la estructura, las vistas arquitectónicas , los modelos que se hayan desarrollado y los prototipos de interfaz realizados.

# Arquitectura

## Patrón N-Tier Cliente - Servidor

**Aplicación:** Se utiliza el patrón para implementar el sistema de administración de Que golazo vía web.

**Motivaciones:** Separación de los distintos intereses en varios niveles lógicos, facilitando las modificaciones y extensibilidad del sistema.

Capa de Presentación

Cliente Web

Servidor Web

Servidor de Aplicaciones

Servidor de Base de Datos

Servidor de Correo

Cliente Web

Cliente Web

Capa de Servicios Web

Capa de Lógica de Negocio

Capa de Administración de Datos

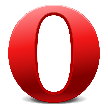
## Arquitectura a nivel de Hardware

A continuación se presenta a nivel de componentes de hardware, como será la arquitectura del sistema.

Se utilizará un servidor alquilado, el cual deberá proveer todos los servidores necesarios:

* Web (IIS 7)
* De aplicación
* Base de Datos (SQL Server 2012)
* Correo

Cualquier dispositivo que posea de conexión web, podrá acceder a la aplicación a través de una URL.



**Servidor**

-Web

-Aplicaciones

-Base de Datos

-Correo

**Cliente Web**

**Internet**

## Estructura de la Solución

Cada proyecto agrupa un conjunto de clases que se comportan de manera similar. Los proyectos definidos son los siguientes:

* **Web:** Agrupa las clases de presentación, que están formadas por el código HTML, y los componentes ASP, el CodeBehind, donde se incluye el código en C#. También se incluyen los archivos de configuración web (Global Asax, Web. Config), recursos de imágenes, de estilos css, y archivos de JavaScript.
* **Lógica:** Es una biblioteca de clases de C#, que son las contiene la lógica de negocio. Cada clase de estas es un gestor, de alguno de los módulos del sistema. Ejemplo: GestorPartido, GestorEdicion, GestorUsuario, entre otros.
* **Utilidades:** Es una biblioteca de clases de C#, que contiene clases que dan soporte a las clases de la capa de lógica de negocio, como lo son clases de soporte para el envío de mails, formato de imágenes, validadores de datos, entre otros.
* **Entidades**: Es una biblioteca de clases de C#, que contiene las clases de entidad, es decir, las del modelo de dominio, como lo son las clases Equipo, Torneo, Usuario.
* **Acceso a Datos:** Es una biblioteca de clases de C#, que contiene las clases que realizan el acceso a datos, es decir, que realizan la materialización y desmaterialización de los objetos de las clases de entidad. Las clases se nombran igual que las clases de entidad, pero se antepone la pablabra DAO a estas clases, como lo son las clases DAOUsuario, DAOTorneo. DAOEdicion. En estas clases se encuentran todas las sentencias SQL para manipular la base de datos, dentro de los métodos definidos.

Web

Lógica

Entidades

Acceso a Datos

Utilidades



Solución

Proyectos

Base de Datos



Servidor de Correo

# Relación entre las clases

A continuación se presenta cómo interactuan las clases dentro de los proyectos. Cada pantalla web (Clase .aspx) se relaciona con uno o más gestores de la capa de lógica. Cada gestor se relaciona con las clases de entidad, y sus clases DAO correspondiente. En el gestor por lo general se crear los objetos, y se los manipula. La lógica necesaria se realiza en las clases gestores. Se debe mantener estas relaciones entre las distintas clases de los proyectos, evitando el acoplamiento entre los proyectos de entidad y capa de datos, con la capa de presentación.



Web

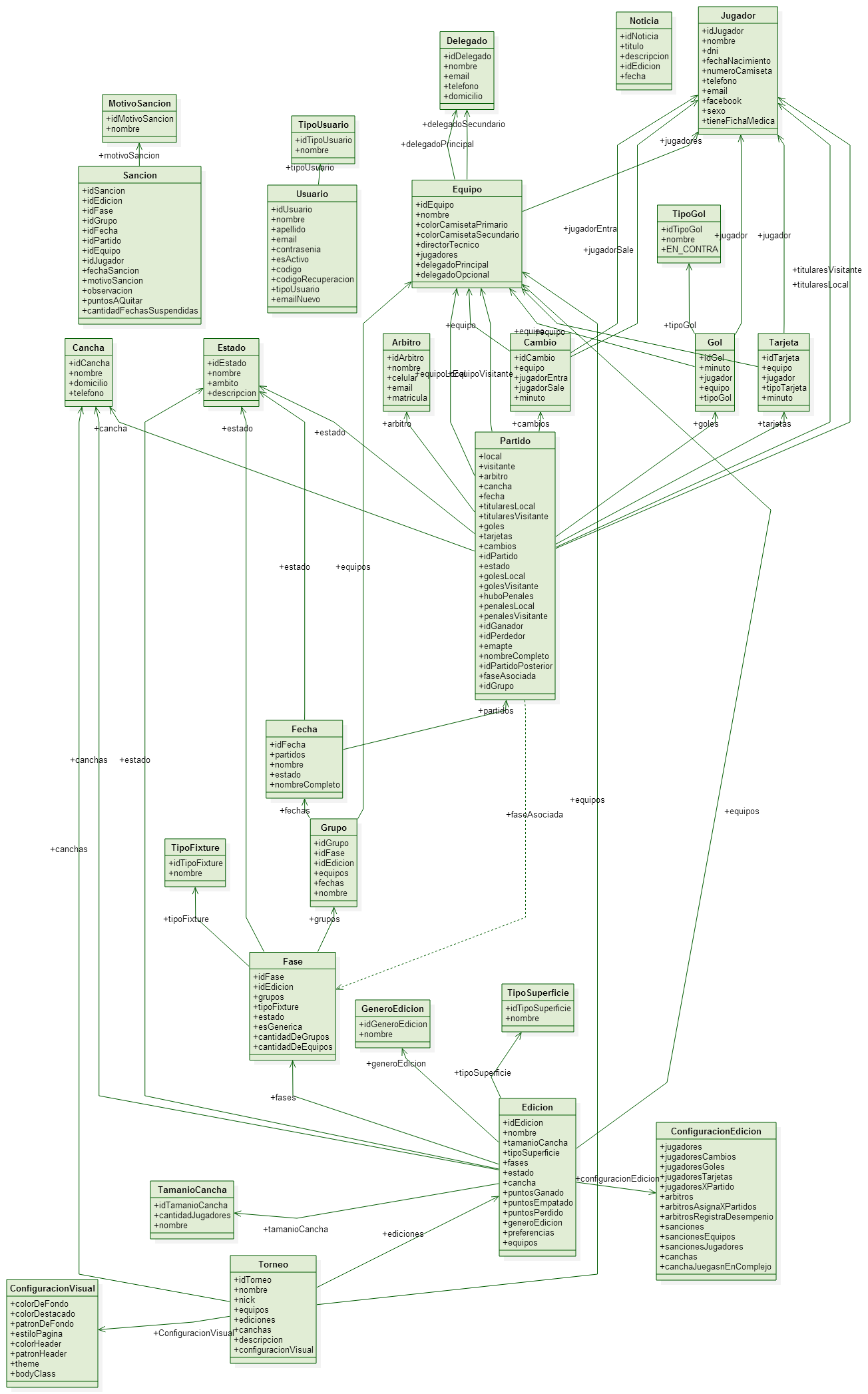
Lógica

Entidad / Acceso a Datos

# Modelos

## Diagrama de Clases de dominio





## Diagramas de Estado

A continuación se presentan los diagramas de transición de estados de los objetos: Fase, Fecha y Partido.

### Clase Partido



**Estados:**

* **Diagramado:** Se pasa a este estado cuando se crea la diagramación de la edición.
* **Programado:** Cuando se le asigna Fecha y Hora.
* **Cancelado:** Se cancela el partido por cualquier motivo.
* **Jugado:** Se pasa a este estado cuando el partido ya se jugó (Se registró el resultado del mismo).

|  |  |
| --- | --- |
| **Métodos** | **Clase a la que pertenece** |
| confirmarEdicion() | GestorEdicion |
| modificarPartido() | GestorPartido |
| cancelarEdicion() | GestorEdicion |

### Clase Fecha



**Estados:**

* **Diagramado:** Se pasa a este estado cuando se crea la diagramación de la edición.
* **Completa:** Cuando se jugaron todos los partidos.
* **Incompleta:** Cuando ya tiene partidos jugados (pero no todos).

|  |  |
| --- | --- |
| **Métodos** | **Clase a la que pertenece** |
| confirmarEdicion() | GestorEdicion |
| modificarPartido() | GestorPartido |

### Clase Fase



**Estados:**

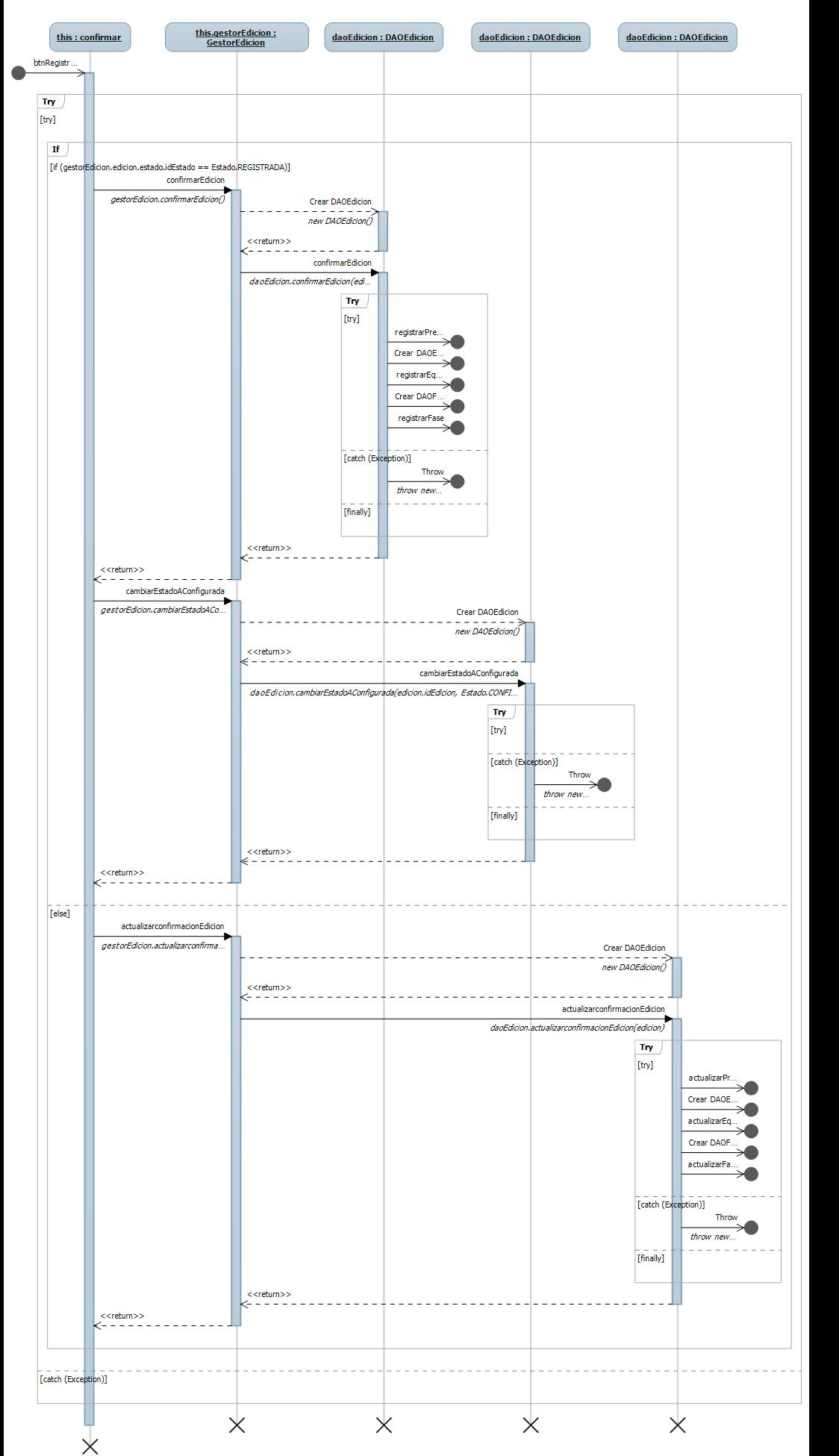
* **Creada:** Cuando se crea la fase.
* **Diagramada:** Cuando se registra la edición configurada.
* **En Juego:** Cuando se registra como jugado un partido.
* **Cerrada:** Cuando se completaron todas las fechas.

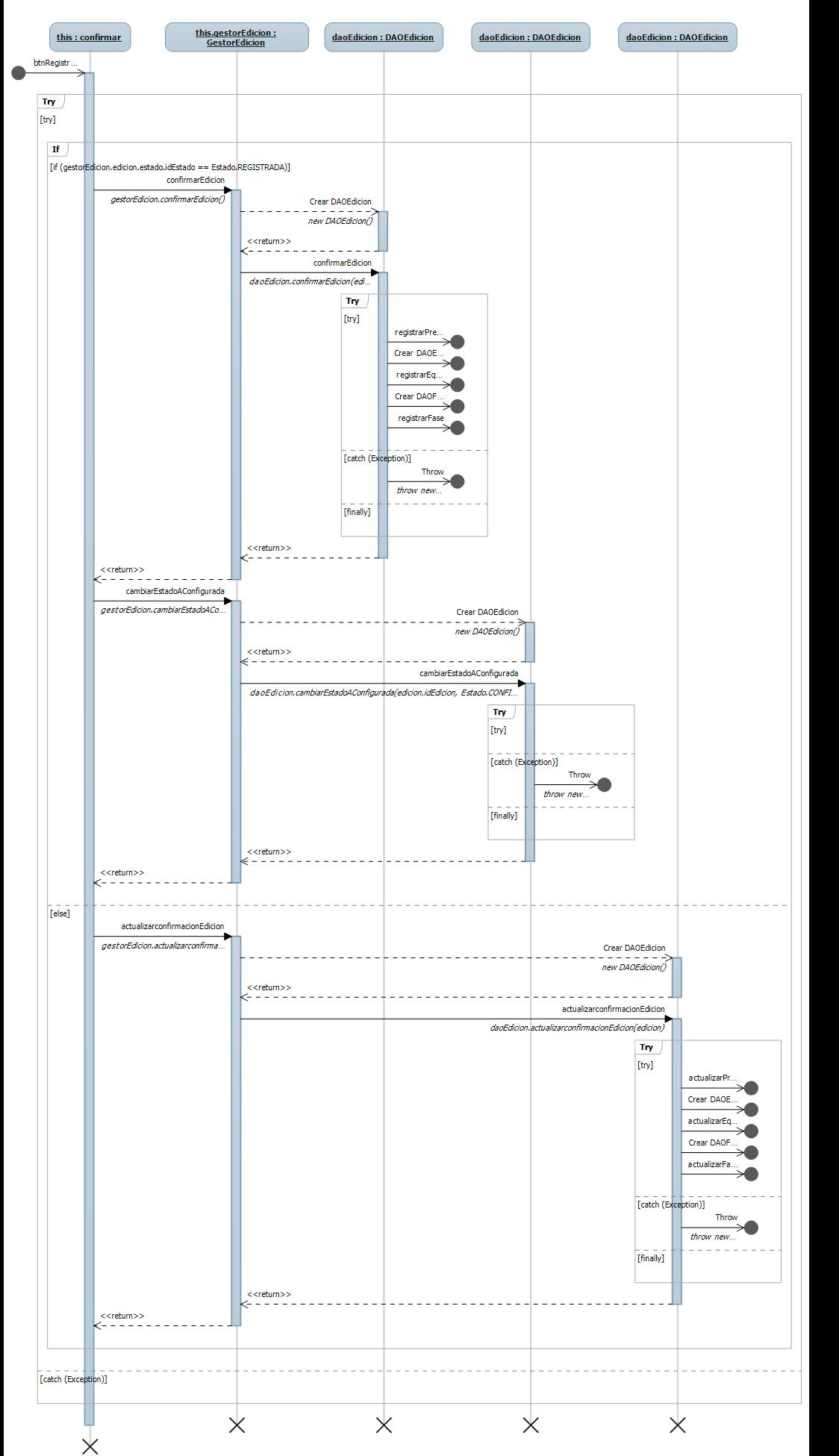
|  |  |
| --- | --- |
| **Métodos** | **Clase a la que pertenece** |
| Crear() | Fase |
| confirmarEdicion() | GestorEdicion |
| modificarPartido() | GestorPartido |

## Diagramas de Interacción

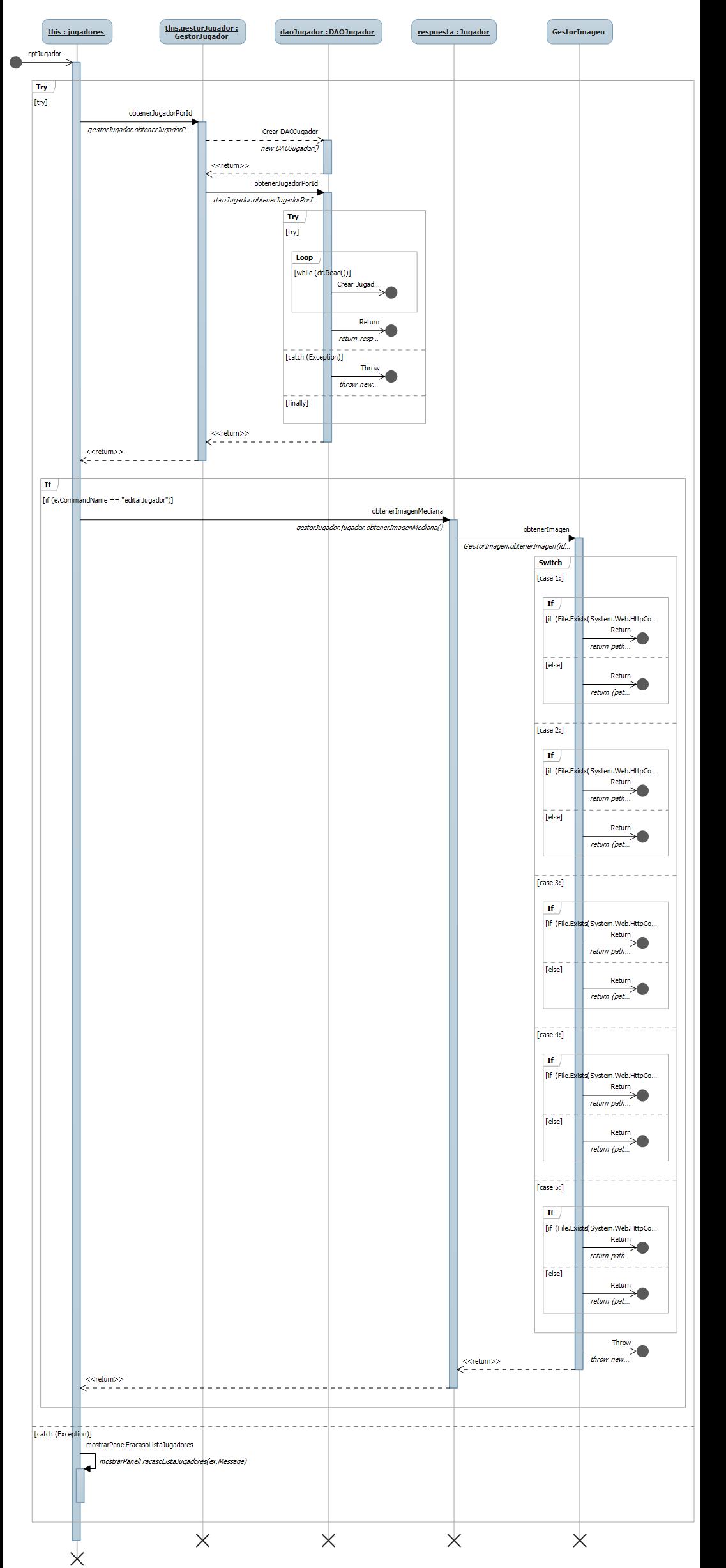
A continuación se presentan los diagramas de secuencia referidos al registro de las características de una edición y la consulta a los jugadores registrados para un equipo.

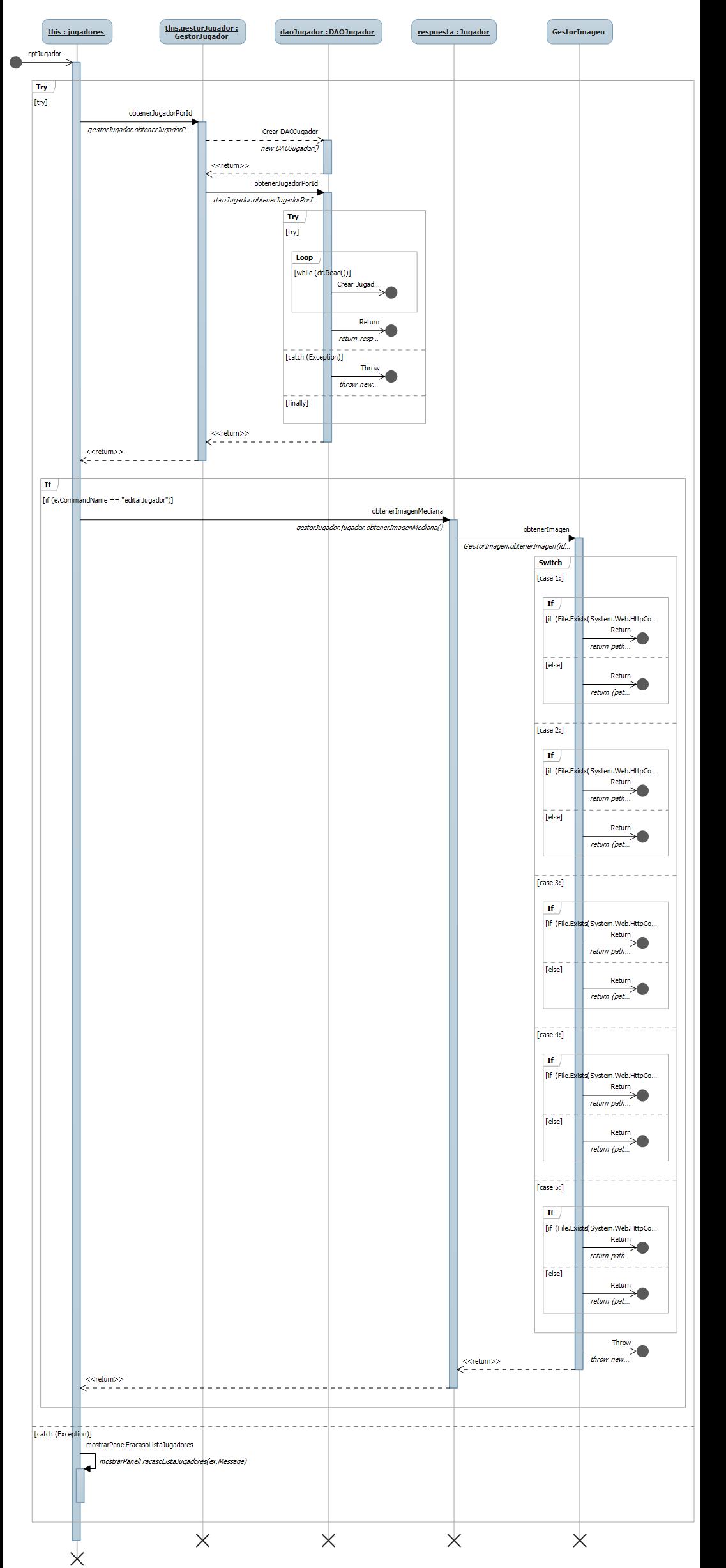
### Registrar Configuración de Edición





### Consulta de Jugadores





## Diseño

### Uso de Patrones

Para resolver la generación de los distintos tipos de fixture, se implementará el patrón Strategy. A continuación se presenta el diagrama de cómo se implementará la solución.

#### Diagrama de Clases

* generarFixture(): Este método recibe la lista de equipos de un determinado Grupo, y a partir de estos, genera las fechas con sus correspondientes partidos.
* setCantidadRondas() : este método setea la cantidad de rondas que se disputarán, Puede valer 1 (cuando es sólo ida) y 2(cuando es ida y vuelta).
* getCantidadRondas(): devuelve la cantidad de rondas que se disputarán. (Se llama a este método internamente dentro del generarFixture().

## Generación de Fixture

Los tipos de torneo que se busca poder diagramar a partir de nuestro sistema son:

* Todos contra todos
* Todos contra todos Ida y Vuelta (considerando locales y visitantes)
* Eliminatorias

A continuación se explica a partir de los algoritmos obtenidos cómo se pretende resolver la problemática de cada tipo de torneo:

### Todos contra todos

Se utilizará un algoritmo basado en Round Robin, que se realizará en el método generarFixture() a partir de la lista de equipos de un grupo. Puede aplicarse a varios Grupos.

### Todos contra todos Ida y Vuelta

Se realizará una adaptación del algoritmo anterior, teniendo en cuenta la satisfacción de la condición de local y visitante. Puede aplicarse a varios Grupos.

### Eliminatoria por Fases

Se realizará implementando el método generarFixture(), donde ya vendrán determinados desde la pantalla, los partidos a jugarse en la primera ronda. La cantidad de llaves a generar estará dada por la cantidad de equipos seleccionados.

# Mapeo a la Base de Datos

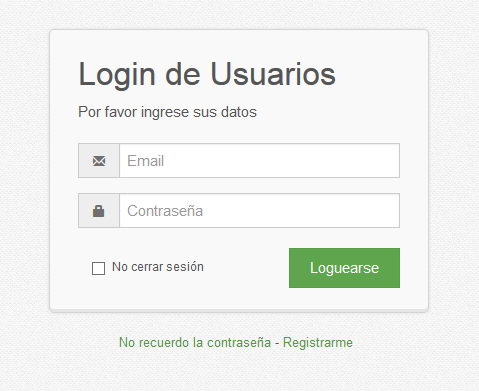
A continuación se presenta el modelo de tablas de la base de datos (Sólo tablas).



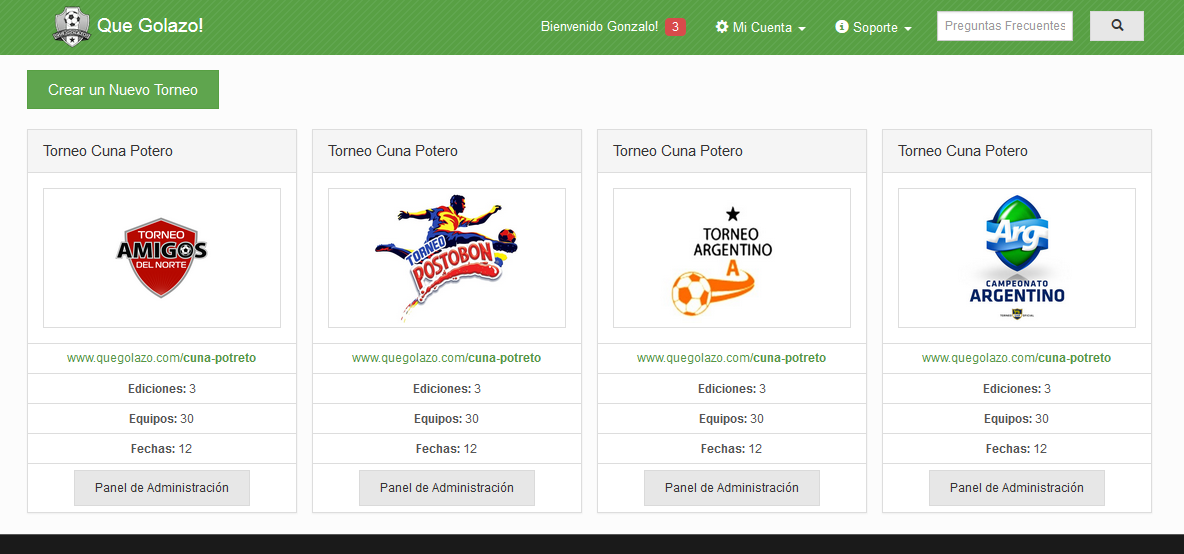
*\*Ver Modelo de DER Detallado en los documentos de DER de cada Sprint.*

# Prototipo de interfaz

## Login

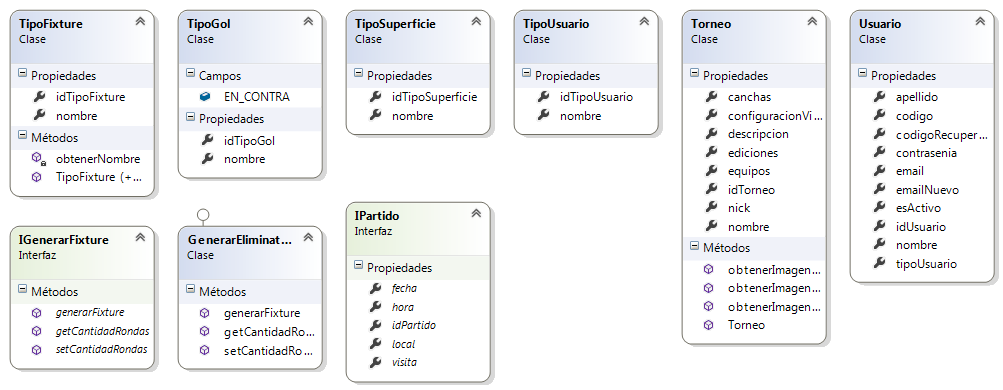


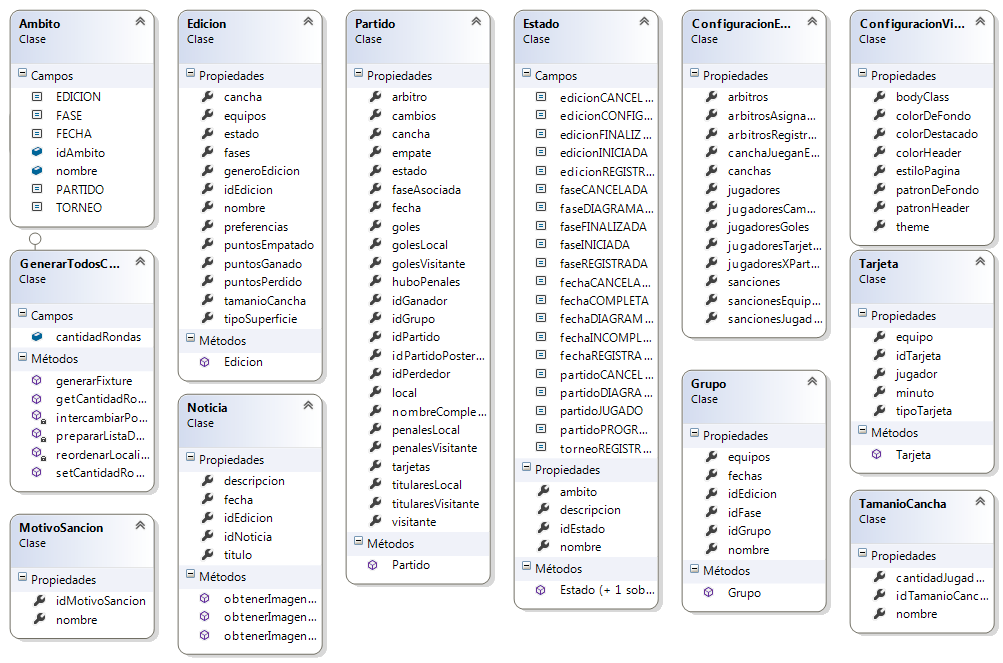
## Mis Torneos

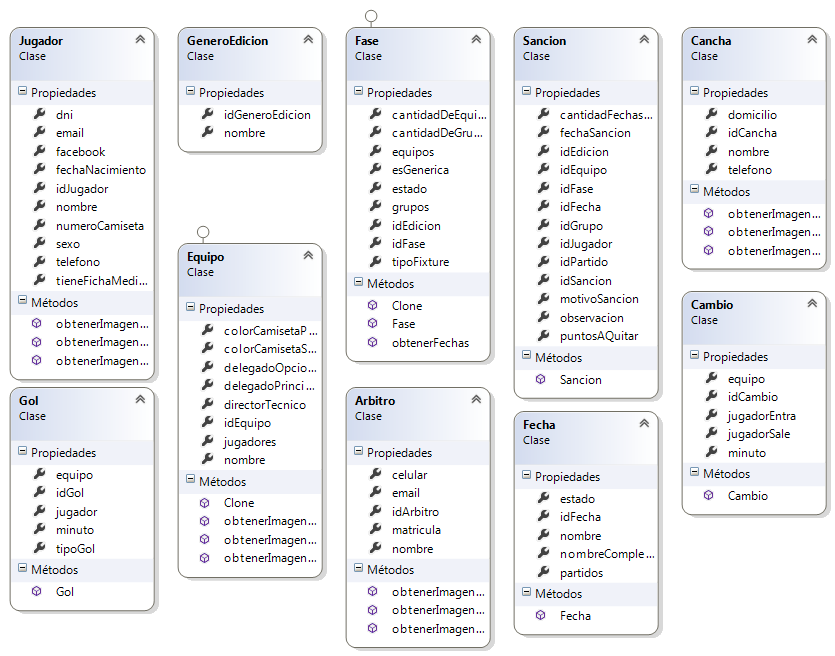


# Implementación

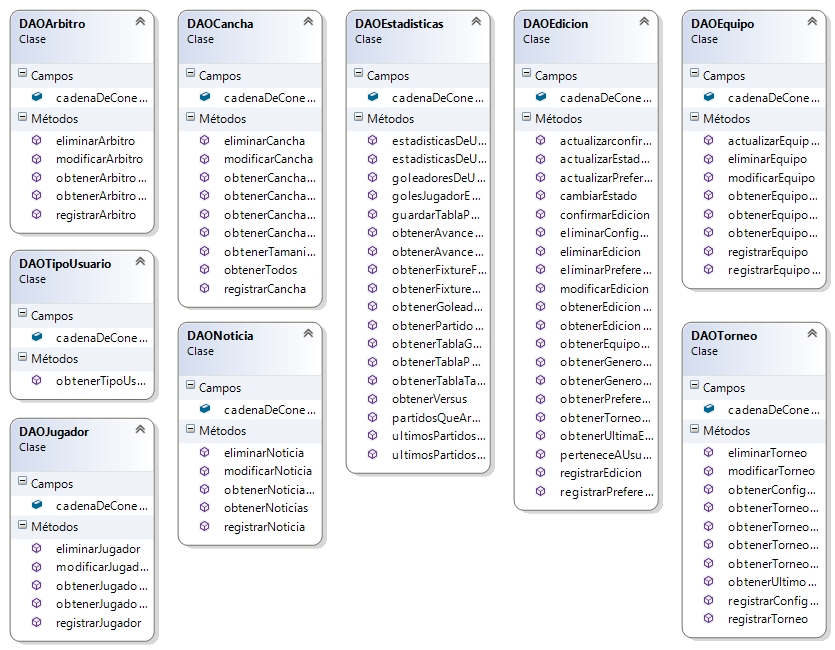
## Entidades (Biblioteca de Clases)

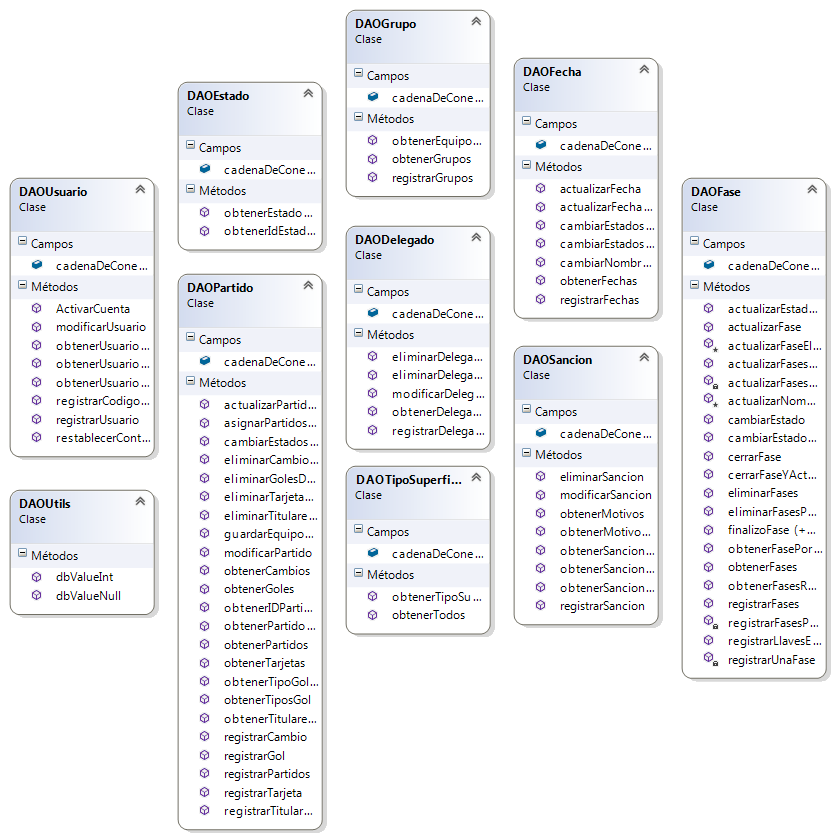




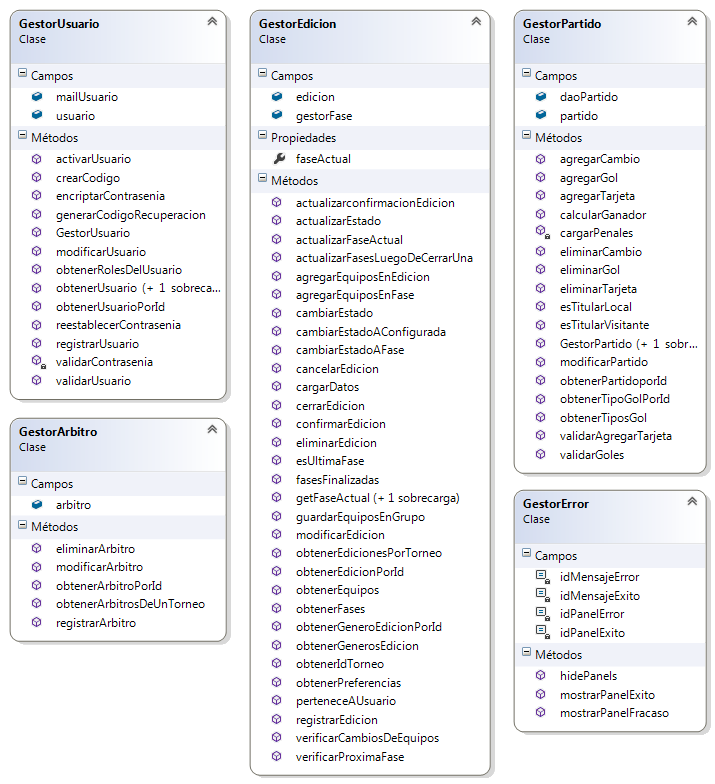


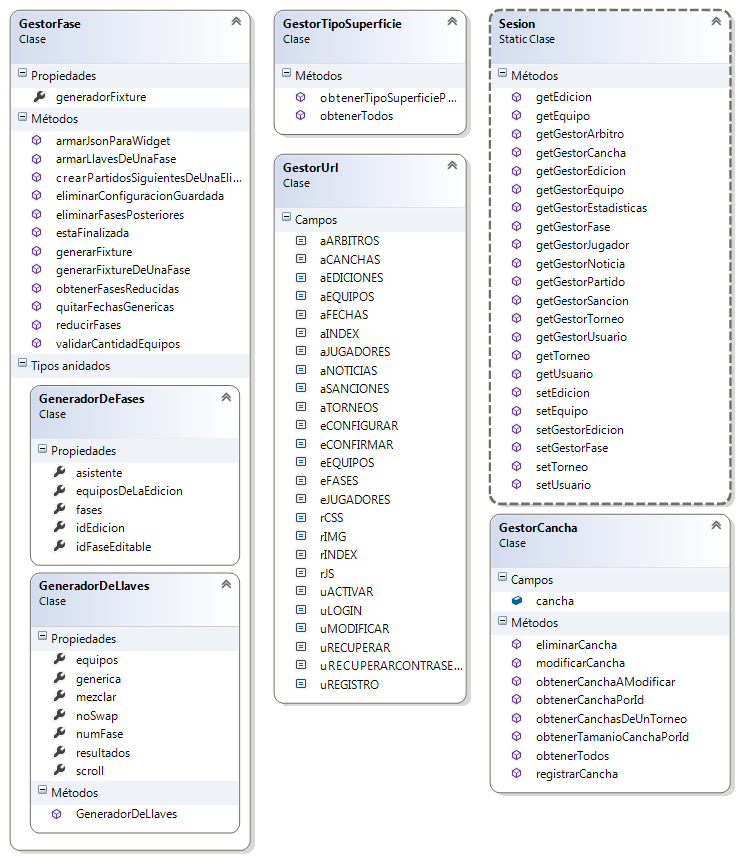
## Acceso a Datos (Biblioteca de Clases)

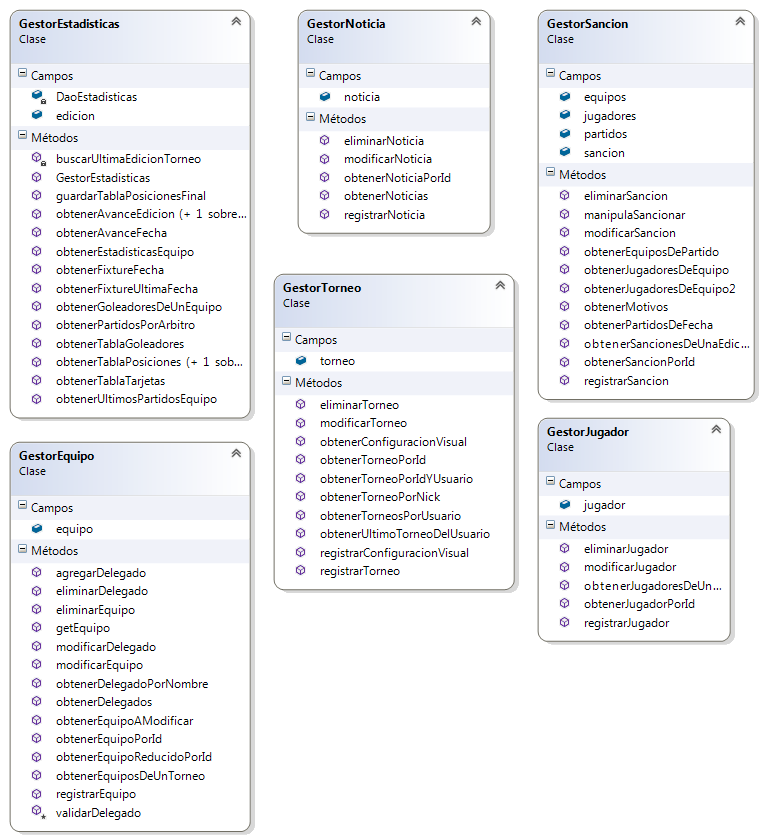




## Lógica (Biblioteca de Clases)







## Utils (Biblioteca de Clases)

