Camilo Linares - 202122820
Daniel Pedroza - 202123283

# Entrega 1 Proyecto 3

## Alcances:

- Para este proyecto estuvimos contemplando diferentes métodos para poder guardar los datos además de maneras para cargarlos, finalmente nos decidimos en limitar la información es a través del uso de csv para cuando se vaya a ingresar información nueva a través del administrador, como una nueva temporada, partidos y jugadores, en cambio para recargar las temporadas antiguas o juegos antiguos decidimos limitarlo a el método de serialización.
- Una restricción que utilizaremos consiste en que la forma en la que la alineación se creará de manera que el usuario escoja a los suplentes y que los restantes vayan a ser los jugadores principales por lo que no se va a poder escoger un orden de las posiciones entre estos.
- Para mantener un ranking adecuado se creó la clase pair para poder asignar un valor a un objeto esto se logró a través de que se genera una tupla de <int,objeto> y también una clase que se dedica a comprar los resultados para organizarlos de mejor a peor.
- Para solucionar muchas de las restricciones establecidas en el documento de análisis que se relacionaban a la nómina se va a lograr con un método que altera el valor del atributo (Presupuesto) del usuario basado en los métodos de comprarjugador(), venderjugador() y setPresupuesto(), los valores principales se sacarán de los jugadores ya que cuando se creen ya van a tener su valor de compra y de venta.

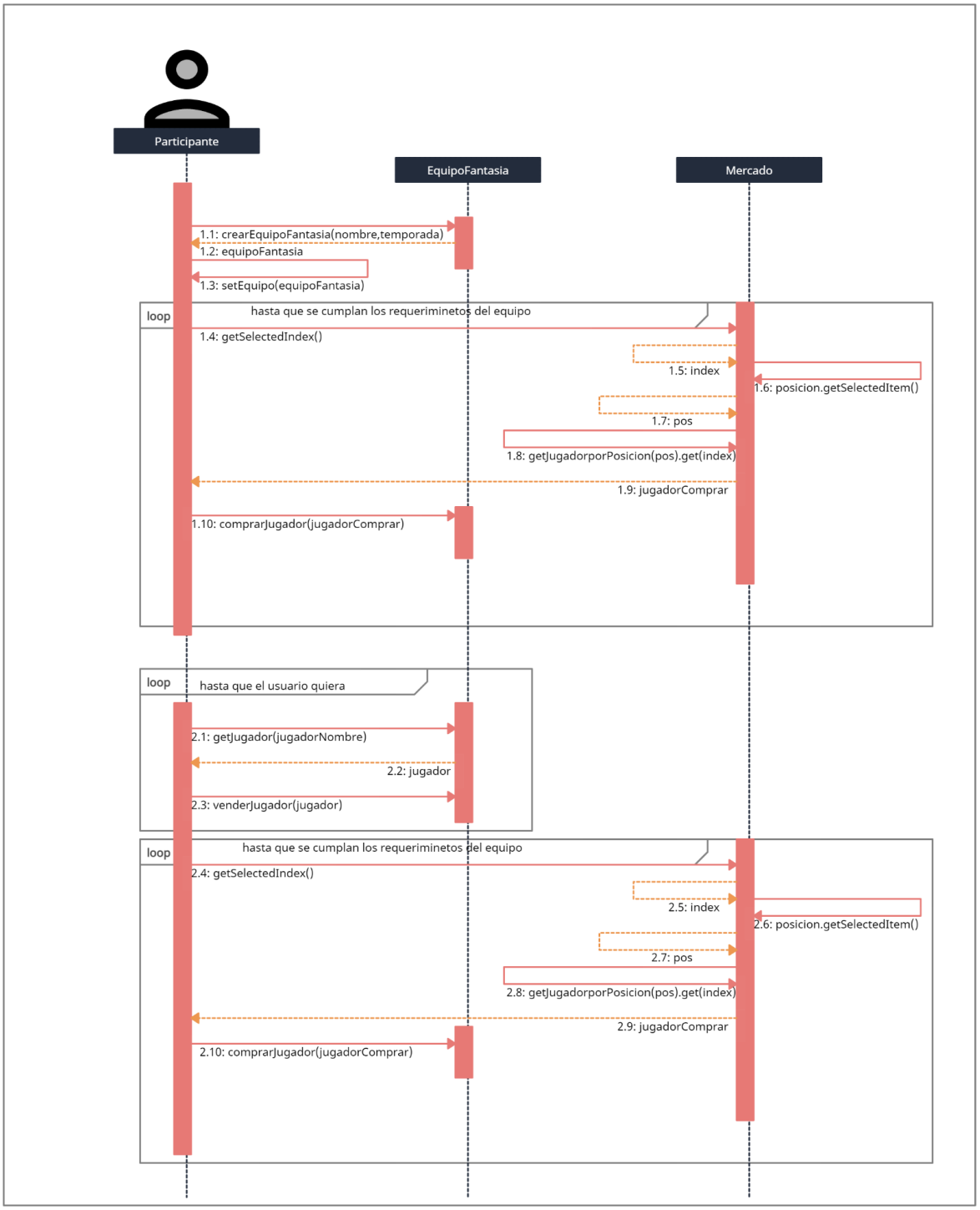
## No-Objetivos:

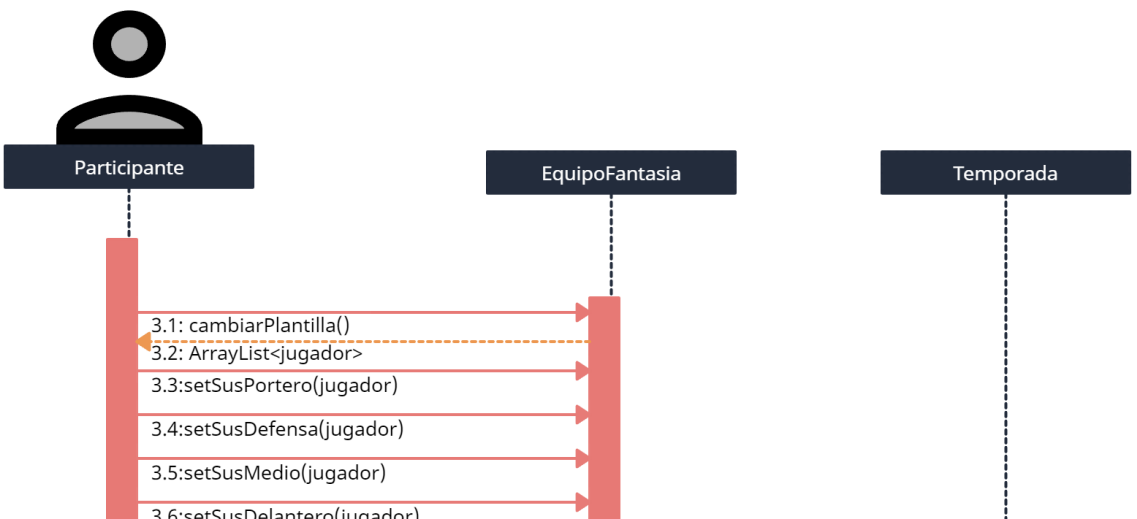
- Algunos aspectos que queremos dejar explícitamente fuera de la solución pueden incluir el hecho de que se utilicen jugadores de diferentes ligas de fútbol.
- Evitar la selección de jugadores en posiciones que no les corresponden, un ejemplo de esto sería utilizar un delantero como arquero para asegurar más puntos en la liga de fantasía.
- Dejar afuera la opción de alterar la alineación de suplentes y jugadores mientras ocurre el partido, el cambio debe tener cierto tiempo de notificación.
- Que un jugador real este en dos equipos diferentes
- Dos partidos se jueguen a la misma hora
- No poder tener dos equipos Fantasía con el mismo nombre.

## Decisiones de diseño:

Para desarrollar muchos de los diagramas de diseño que se pueden ver en futuras páginas, tomamos en cuenta el diagrama desarrollado en la fase de análisis de la primera entrega, puede que estos no tengan cambios mayores ya que en general se tendrá la misma funcionalidad de todos los requerimientos, pero se añadirán los cambios por parte de la interfaz gráfica ya que tiene una gran cantidad de clases que son importantes para satisfacer la necesidad de poder interactuar con el usuario, estos cambios en la interfaz fueron una interfaz por responsabilidad entonces se mantiene una interacción sencilla para cada responsabilidad. Con los nuevos cambios permitidos los ingresos de más de un equipo por usuario, también una forma eficiente de poder comparar las estadísticas de la temporada con la clase pair para así poder tener una instancia de cómo van los puntajes mientras ocurre cada partido con el fin de que el usuario pueda recrear y saber su progreso en la tabla. Se ingresó un controller a la par de las interfaces para poder mantener el modelo MVC más claro ya que se percató que el proyecto tenía falta de controladores para poder simplificar el control de la vizualizacion entonces ahora que se maneja un nivel más avanzado de interfaz si es necesario la creación de una clase controller.

## Diagrama de secuencia Participante:

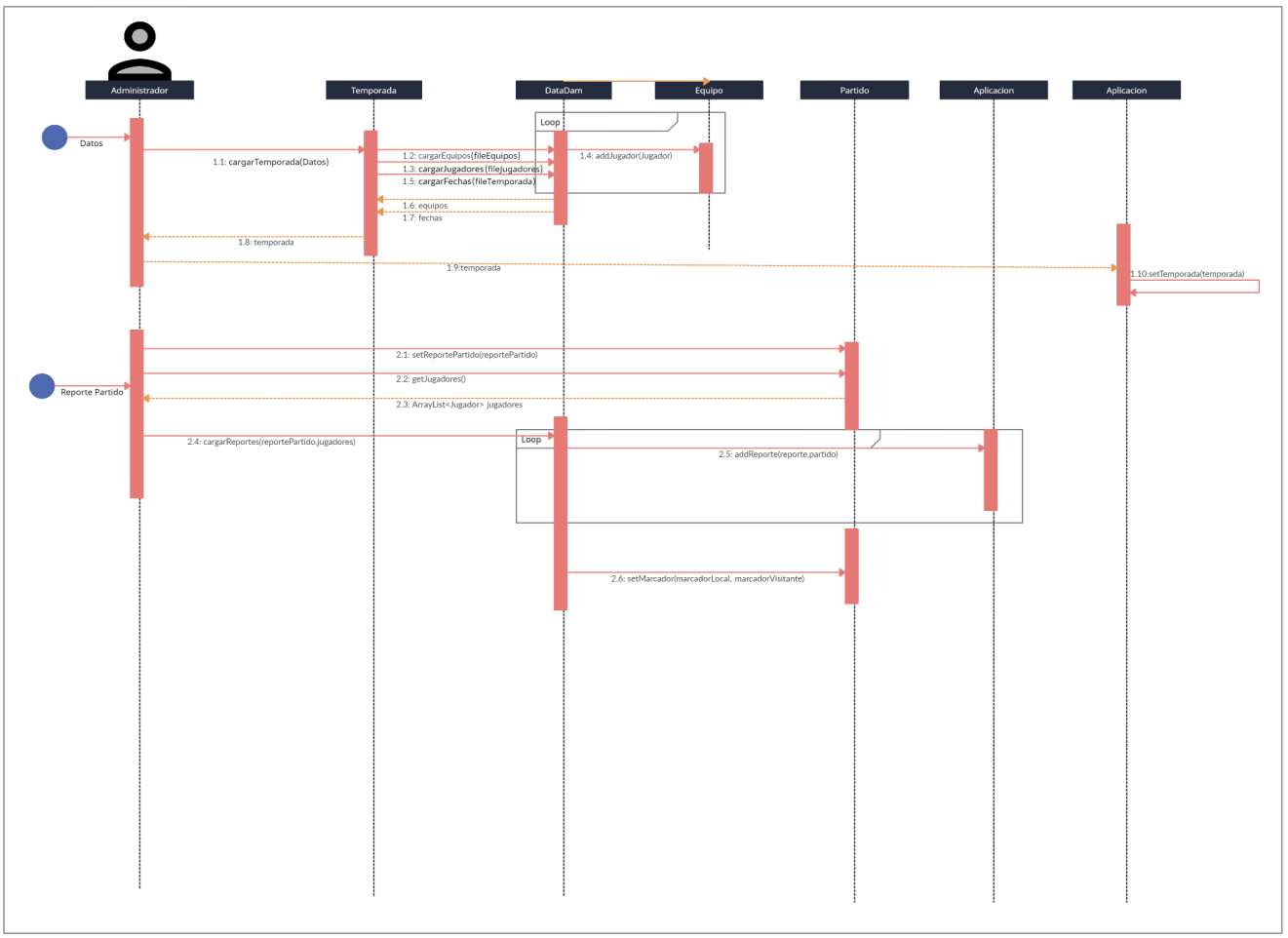

Participante
EquipoFantasia
Mercado
1.1: crearEquipoFantasia(nombre,temporada)
1.2: equipoFantasia
1.3: setEquipo(equipoFantasia)
loop
hasta que se cumplan los requeriminetos del equipo
1.4: getSelectedIndex()
1.5: index
1.6: posicion.getSelectedItem()
1.7: pos
1.8: getJugadorporPosicion(pos).get(index)
1.9: jugadorComprar
1.10: comprarJugador(jugadorComprar)
loop
hasta que el usuario quiera
2.1: getJugador(jugadorNombre)
2.2: jugador
2.3: venderJugador(jugador)
loop
hasta que se cumplan los requeriminetos del equipo
2.4: getSelectedIndex()
2.5: index
2.6: posicion.getSelectedItem()
2.7: pos
2.8: getJugadorporPosicion(pos).get(index)
2.9: jugadorComprar
2.10: comprarJugador(jugadorComprar)


Participante
EquipoFantasia
Temporada
3.1: cambiarPlantilla()
3.2: ArrayList<jugador>
3.3:setSusPortero(jugador)
3.4:setSusDefensa(jugador)
3.5:setSusMedio(jugador)
3.6:setSusDelantero(jugador)

Los diagramas de secuencia de los usuarios se tienen que dividir en dos por el programa que estábamos usando que solo limita una cantidad de 60 elementos por diagrama, también el diagrama estaba quedando muy largo por lo que no se vería.

Muchas de las acciones que se ven son solo acciones que tiene un efecto directamente con el usuario entonces se evitan creaciones de clases como la de mercado y de sus listas de los jugadores, se hace una suposición de que ya se creó con éxito. También las entradas de las listas que se retornan al usuario se pueden ver como información que será mostrada al usuario a través de la vista.

## Diagrama de secuencia Administrador:




## Roles:

| Roles |
| --- |
| R1: Administrador |

| |
|---|
| R2: Participante |
| R3: DataDam |
| R4: Equipo |
| R5: Jugador |
| R6: EquipoFantasia |
| R7: ReporteJugador |
| R8: Temporada |
| R9: Fecha |
| R10: Mercado |
| R11: Aplicación |
| R12: Alineación |
| R13: Partido |

## Responsabilidades:

| Responsabilidades |
|---|
| r1 Registro de Participante [R2,R3,R8] |
| r2 Registro de Administrador [R1,R3,R8] |
| r3 Cargar Temporada [R1,R3,R4,R5,R9,R11,R13] |
| r4 Crear Equipo [R3,R4,R5] |
| r5 Crear Jugador [R3,R4,R5] |
| r6 Crear Fecha [R3,R9,R8, R13] |
| r7 Crear Equipo Fantasia [R2,R6] |
| r8 Comprar Jugador |

| |
|---|
| [R2,R6,R10,R5] |
| r9 Vender Jugador<br>[R2,R6,R5] |
| r10 Crear/cambiar alineación<br>[R2,R6,R5] |
| r11 Ver Estadísticas<br>[R2,R8] |
| r12 Cerrar Partido<br>[R1, R3, R8, R12, R13] |
| r13 Crear Mercado<br>[R8] |

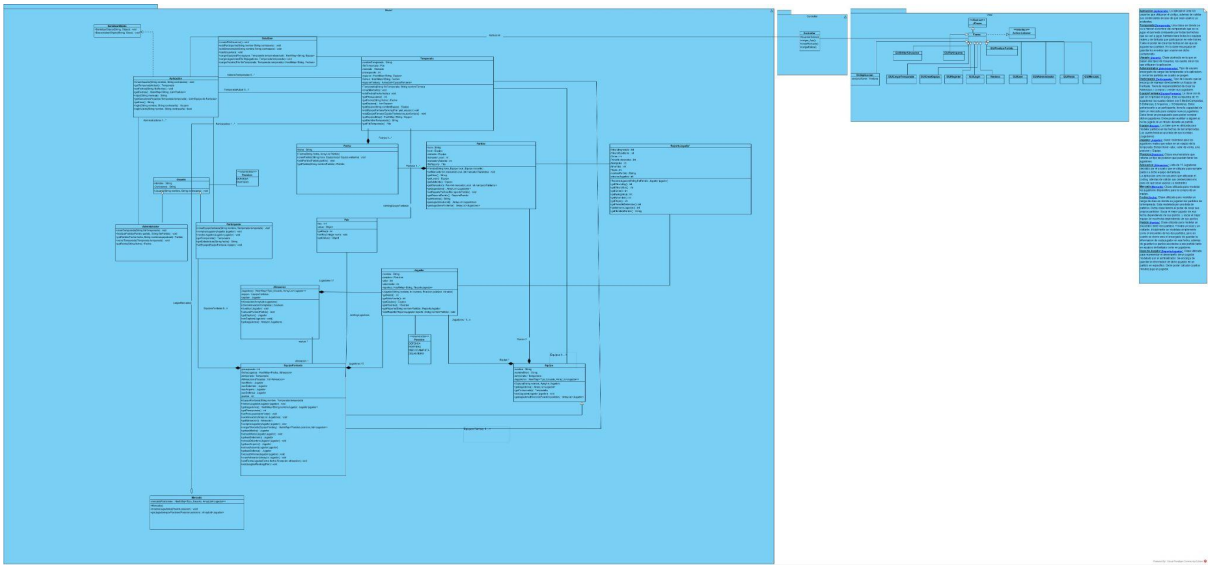
## Colaboraciones:

| Colaboraciones |
|---|
| C1: [R1,R11] |
| C2: [R2,R6] |
| C3: [R2, R11] |
| C4: [R3,R11] |
| C5: [R4, R8] |
| C6: [R5,R7] |
| C7: [R6, R8] |
| C8:[R8, R11] |
| C9: [R10, R8] |

## Diagrama UML:

Este nuevo diagrama de UML tuvo algunos cambios gracias a la integración de la interfaz. Se hicieron cambios más centrados en los controladores del programa ya que en la anterior versión del código teníamos controladores que no estaban bien definidos por lo que las interacciones entre la vista y el modelo eran directas. Se agregó un controlador que conecta

las nuevas interfaces y el modelo anterior, para este proyecto generamos muchas clases de GUI(clases de interfaces) ya que se hizo una para cada acción que puedan hacer los usuarios.



Se subió el UML en la carpeta dado a que no se ve muy bien por el tamaño en el documento de diseño.