

Organizador de partidos de fútbol 5 (OPF5)

Contexto general

Existe una comunidad de personas que se juntan periódicamente a jugar al fútbol 5 (5 jugadores por equipo). En el momento del relevamiento la periodicidad era semanal (todos los jueves a las 21 hs) pero al usuario le interesaría poder modificar la frecuencia o el día y horario en el que se juega el partido.

Esta comunidad está formada aproximadamente por 25 personas, no todos juegan regularmente. A veces ocurre que cuesta conseguir los 10 participantes y otras veces hay gente que se queda con las ganas de jugar. El principal problema a resolver es el de generar la lista de los 10 participantes que jugarán un determinado partido. Hay personas que juegan casi todas las semanas, a estas personas se las debe priorizar de alguna manera para garantizarles la plaza. Por otro lado ocurre que ciertas personas que confirman la asistencia a veces no van o cancelan sobre la hora. Se desea castigar a las personas que incurran en estas actitudes en forma frecuente.

Otro problema que debe resolver el sistema a diseñar es que los equipos se armen de la forma más equitativa posible para mitigar las quejas de los perdedores de turno. Veremos más adelante los criterios propuestos por el cliente.

Ciclo de vida de un partido

Un administrador tiene que usar el sistema para indicarle que prepare un nuevo partido, estableciendo fecha y hora del mismo. Entonces los jugadores se inscriben de alguna de las siguientes formas:

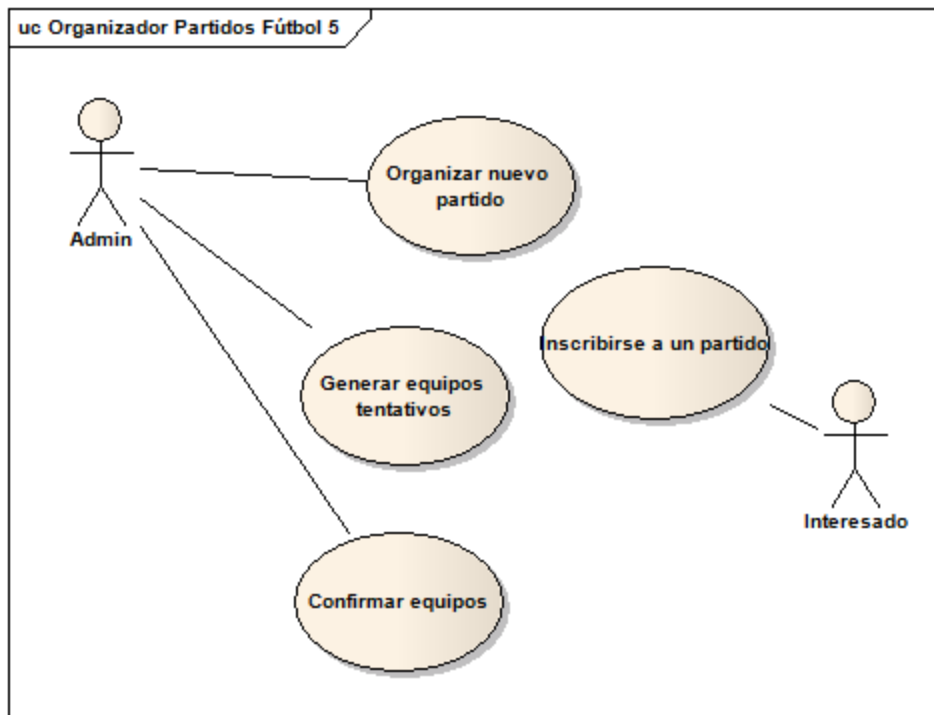
- **estándar:** el jugador confirma su participación en el partido
- **solidaria:** la forma solidaria significa “juego si no hay otro”, con lo cual, si en la lista hay 10 jugadores, pero uno de ellos está en forma solidaria, otro jugador puede agregarse al partido y desplazar de la lista al solidario. En caso de que haya más de un jugador solidario, el que es desplazado es el primero que se agregó a la lista.
- **condicional:** la participación del jugador se hace en base a una condición impuesta sobre el partido. Ej. “que no haya más de 6 jugadores menores de 20 años”, “que el partido se juegue en tal o cual lugar” (siempre son condiciones que se aplican al partido). Como podrán imaginarse el jugador con inscripción condicional no tiene ningún tipo de prioridad, es desplazado por los jugadores de participación estándar o solidaria en el armado del partido.

El sistema no deja agregar más jugadores cuando la lista tiene 10 con la plaza asegurada (modo estándar).

Antes del partido, el administrador genera la posible formación de los equipos. Luego del partido, el administrador confirma los equipos (es decir, puede modificar los equipos definidos en la fase anterior). En este punto también determina los participantes ausentes, a los que se penaliza con una infracción.

En el relevamiento surgió la inquietud del cliente de poder automatizar a futuro algunos de estos procesos, aunque se acordó con él comenzar a diseñar el sistema de la forma más sencilla posible.

Ofrecemos un diagrama de casos de uso del circuito:



Diseño orientado a objetos

Entrega 1

El objetivo de esta entrega es diseñar en detalle la inscripción de un jugador al partido, partiendo por los casos de prueba. Para esto se debe presentar:

1. Implemente los casos de prueba, donde debe indicar
 - a. escenario (pre-condición)
 - b. serie de pasos a ejecutar
 - c. resultado esperado (post-condición)

o bien los tests cases que implementen dichos casos de prueba (incluyendo un fixture o juego de datos).

2. Modele dos posibles soluciones alternativas para ese caso de uso. Luego, realice un análisis comparativo de ambas soluciones y elija una de las opciones, justificando dicha decisión.
3. La implementación del código de negocio necesario para resolver lo pedido en los casos de prueba.
4. Una explicación detallada sobre la vinculación entre las decisiones de diseño y la correspondiente implementación/especificación, que ayude a la trazabilidad de la solución. Utilice las herramientas que crea convenientes para explicar las decisiones de diseño que tomó en cada caso (diagramas de clase, secuencia, colaboración, objetos, tests, código o pseudocódigo, prosa castellana, etc.). No olvide indicar qué componentes aparecieron, cuáles son sus responsabilidades y las colaboraciones con otros componentes.
5. Analizar cada una de las soluciones teniendo en cuenta los atributos de diseño simplicidad y mantenibilidad.

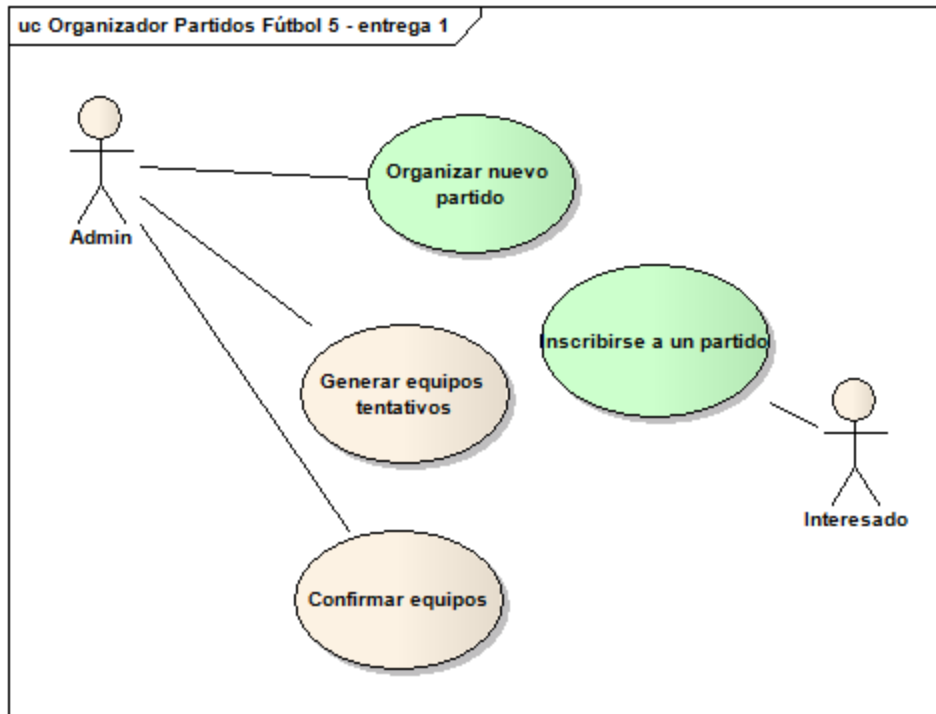


Diagrama de casos de uso - Entrega 1

En verde aparecen los casos de uso nuevos a implementar.

En azul aparecerán los casos de uso a modificar.

Hoja de evaluación del docente

Casos de prueba

- Calidad de los casos de prueba
- Casuística cubierta / Juego de datos preparado

Diseño detallado

- Manejo de errores / Separación de flujo principal y alternativo
- Buenas prácticas de diseño
 - evitar lógica duplicada,
 - mantener la simplicidad de la solución,
 - definición de la interfaz de los componentes que minimice su acoplamiento,
 - consistencia de la solución general
- Utilización del/de los patrones dentro del contexto del enunciado

Comunicación de la entrega

- Trazabilidad de los requerimientos. Vinculación entre el análisis, diseño de alto nivel, diseño detallado, código (si lo hubiere) y pruebas unitarias.
- Calidad general de la documentación presentada
- No pueden faltar los análisis / comparaciones (se piden tres en total)

Entrega 2

Tras ver los primeros avances, el cliente nos pidió que incorporemos algunas mejoras en el ciclo de vida del partido:

- Cada vez que el partido esté confirmado (esto es, que los 10 jugadores confirmaron su participación según la modalidad en la que se anotaron, *no implica que el administrador confirma los que efectivamente jugaron, cosa que ocurre en otro momento*), se debe notificar al administrador del sistema.
- Incorporar como caso de uso la baja de un jugador a un partido. Cuando esto ocurre
 - el jugador debe indicar qué jugador lo reemplazará, en caso contrario se le generará una infracción. Como el sistema de infracciones es por tiempo limitado, nos interesa poder discriminar infracciones de diferentes momentos y por distintos motivos.
 - si el partido deja de tener los 10 jugadores confirmados, se debe notificar al administrador del sistema.
- Cada vez que un jugador se inscriba al partido se debe notificar a sus amigos.

Se pide:

1. Diseñe la solución aplicando dos soluciones posibles. Recuerde utilizar las herramientas que crea convenientes para documentar las decisiones de la misma manera que en la entrega 1.
2. Implemente los casos de prueba automatizados que permitan garantizar la consistencia de las soluciones a. y b, para lo cual
3. Modelar un objeto impostor (mock / stub) que simule el envío de mails. Indique cómo este objeto facilita la realización de pruebas.
4. Haga un análisis comparativo de ambas soluciones en base a:
 - cómo se incorpora cada funcionalidad pedida a lo desarrollado anteriormente
 - qué consecuencias trae en cada solución el manejo de la identidad de los objetos existentes
 - cómo ayuda cada solución a aumentar la cohesión de los componentes.

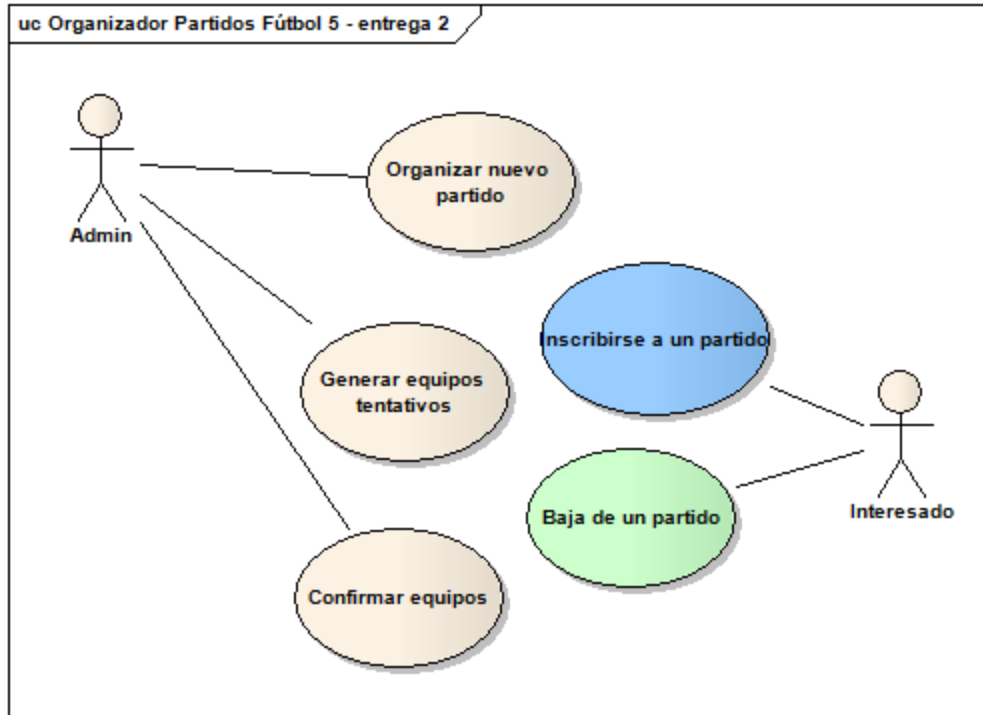


Diagrama de casos de uso - entrega 2

Entrega 3

Se pide incorporar las siguientes funcionalidades al diseño:

Nuevos jugadores

El circuito para incorporar nuevos jugadores es el siguiente: un jugador propone un amigo al administrador, que decide si aprueba o no su moción. En caso afirmativo se cargan los datos del jugador (nombre, fecha de nacimiento, amigos, etc.) y se define la modalidad de participación que va a tener. En caso negativo el administrador debe justificar el motivo del rechazo y se debe registrar esa denegación con la fecha del día.

Calificaciones

Una vez jugado el partido, cada jugador que participó puede calificar a cada uno de los jugadores con una nota numérica del 1 al 10 y agregar un texto a modo de crítica a cada jugador.

Se pide

1. Implemente los siguientes casos de uso
 - a. Nuevos jugadores, siguiendo todo el circuito descrito.
 - b. Calificaciones
2. Implemente los casos de prueba automatizados que permitan garantizar la consistencia de las soluciones para
 - a. Nuevos jugadores
 - b. Calificaciones
3. Indique qué conceptos del diseño permitieron bajar el acoplamiento entre componentes. Justifique a partir de ejemplos concretos para cada concepto.
4. Utilice las herramientas que crea convenientes para explicar las decisiones de diseño que tomó en cada caso (diagramas de clase, secuencia, colaboración, objetos, tests, código o pseudocódigo, prosa castellana, etc.). No olvide indicar qué componentes aparecieron, cuáles son sus responsabilidades y las colaboraciones con otros componentes.

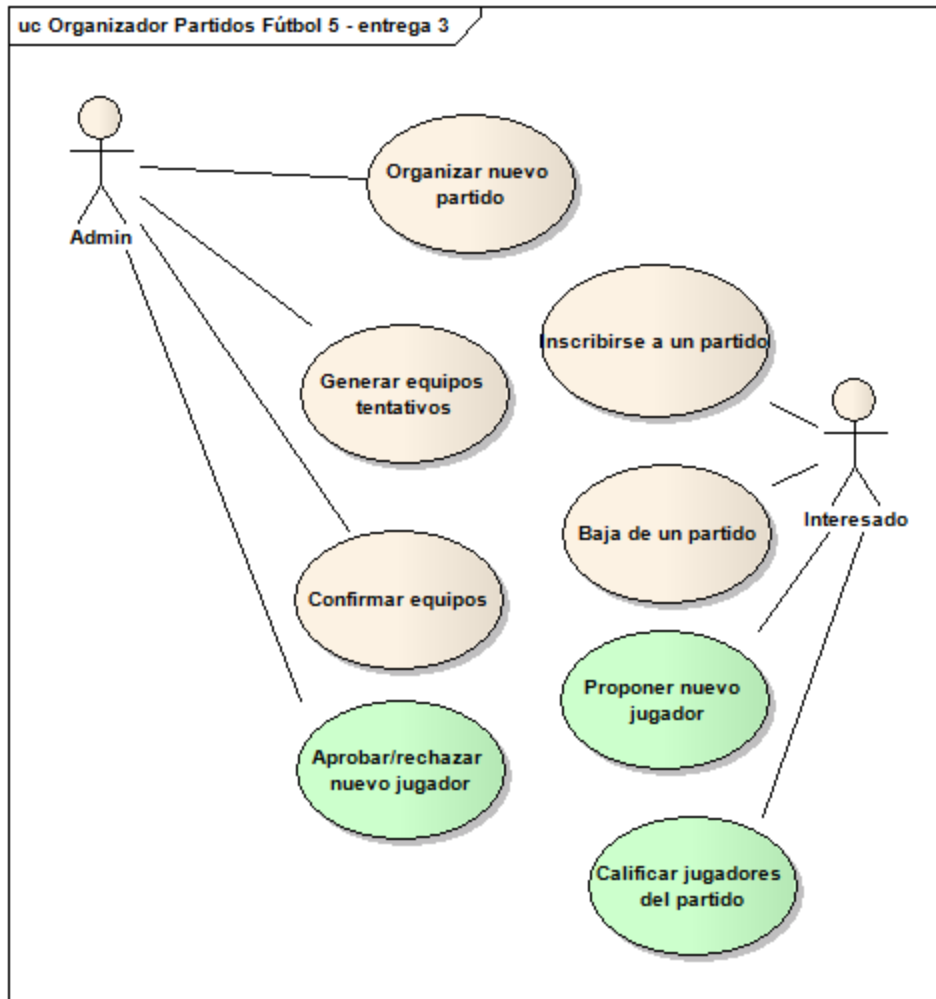


Diagrama de casos de uso - entrega 3

Hoja de evaluación del docente

Diseño general de la solución

- manejo del asincronismo del requerimiento *Nuevos jugadores*
 - diferenciación entre los casos de uso de negocio y los de sistema. Impacto en el diseño de la solución.
- expansión de requerimientos: incorporar funcionalidades a diseños existentes
 - grado de acoplamiento entre componentes nuevos y existentes

Comunicación general del diseño

- Nivel de detalle / profundidad de la solución
- Calidad de la documentación
- Calidad de la justificación presentada para minimizar el grado de acoplamiento entre componentes

Entrega 4

Generar equipos tentativos

El administrador puede pedir al sistema que organice los jugadores de la lista en dos equipos. La aplicación ordena la lista según uno de los siguientes criterios:

- por handicap: el administrador define un “nivel de juego” o handicap para cada jugador, que es un número de 1 (peor) a 10 (mejor).
- tomando en cuenta el promedio de calificaciones del último partido. *Ejemplo:* si a Juan lo calificaron en el último partido con 6, 8, 8 y 7, su promedio da 7,25.
- tomando el promedio de los últimos n partidos, donde el n es configurable.
- utilizar un mix de criterios (debe evitarse lógica duplicada): cada criterio devuelve un número y se considera un promedio entre todos.

Se desea con el tiempo desarrollar nuevos criterios.

¿Cómo se debe dividir los equipos?

- En un equipo van los jugadores que ocupan la posición impar (1, 3, 5, 7 y 9) y en el otro los jugadores que ocupan la posición par (2, 4, 6, 8 y 10)
- En un equipo quedan los jugadores que ocupan la posición 1°, 4°, 5°, 8° y 9° mientras que en el otro el 2°, 3°, 6°, 7° y 10°

Se pide contemplar la posibilidad de agregar nuevos algoritmos a futuro.

Este proceso puede repetirse tantas veces como el administrador desee. Una vez que esté satisfecho con los equipos generados el administrador podrá confirmar el armado de equipos. Esto implica que ya no se permiten altas ni bajas.

Se pide modelar e implementar una solución posible donde

1. el sistema ordene los jugadores y los elija según criterios, siendo posible agregar nuevos criterios a futuro. Indique qué conceptos del diseño utiliza para lograr este objetivo, y qué consecuencias tendría no aplicar esos conceptos
2. se aplique algún pattern, para diferenciar los momentos en que el administrador solicita armar los equipos y cuando efectivamente lo confirma.

Implemente los casos de prueba automatizados que permitan garantizar la consistencia de las soluciones propuestas.

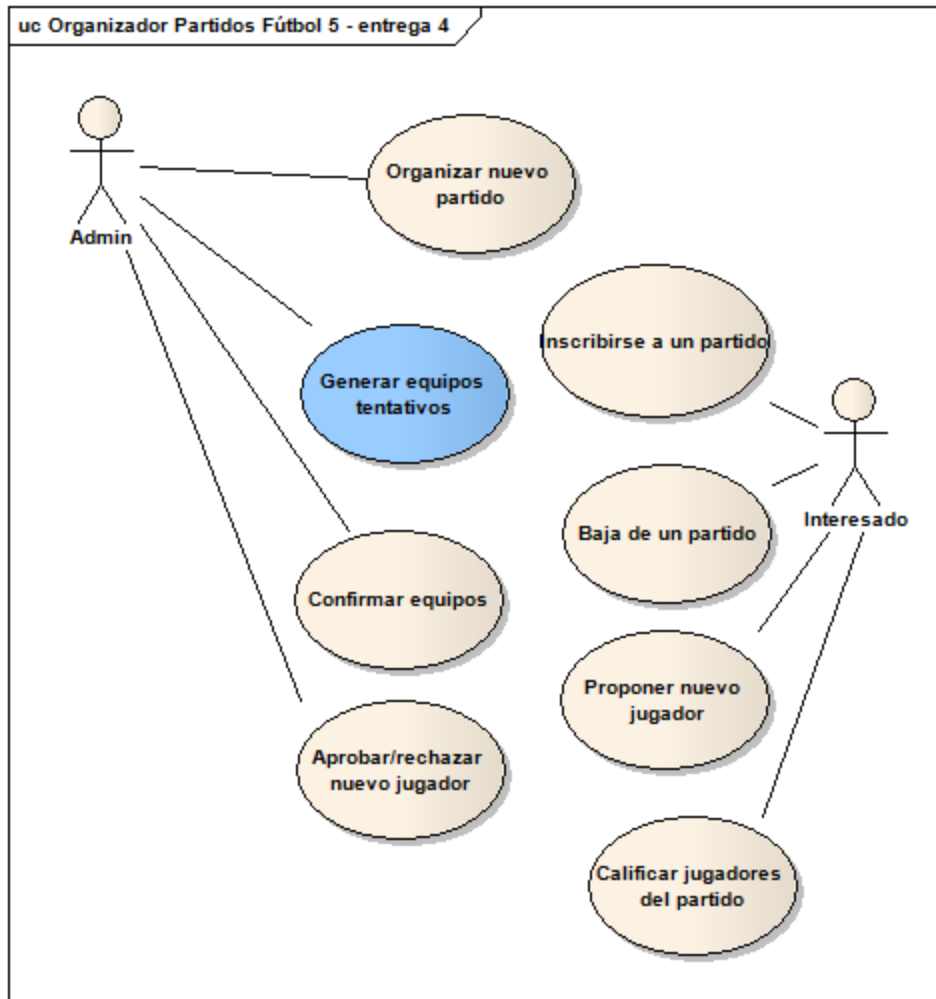


Diagrama de casos de uso - entrega 4

Hoja de evaluación del docente

Diseño general de la solución

- expansión de requerimientos: incorporar funcionalidades a diseños existentes
 - grado de acoplamiento entre componentes nuevos y existentes
- flexibilidad en la elección de criterios para elegir / ordenar jugadores
- aplicación de conceptos de diseño/patronos dentro del contexto pedido

Comunicación general del diseño

- Nivel de detalle / profundidad de la solución
- Calidad de la documentación
- Calidad de la justificación presentada para minimizar el grado de acoplamiento entre componentes

Entrega 5

Se publica el desarrollo del caso de uso *Generar equipos*. El grupo debe:

1. Realizar ingeniería reversa del desarrollo recibido, infiriendo el diseño a partir del código.
2. Cada uno de los integrantes del grupo debe
 - a. Identificar al menos un “bad smell” y explicar a su criterio cuáles son los problemas de diseño de la solución implementada. Recordar que el foco debe estar puesto en el diseño, no en la programación (usar un while en lugar de un for *no es un bad smell*, por poner un ejemplo).
 - b. Proponer para ese bad smell refactorizaciones que no modifiquen la funcionalidad existente.
 - c. Implementar la refactorización
3. Correr los tests automatizados para probar la consistencia de la solución.

Hoja de evaluación del docente

Implementación de ideas de diseño

- Buenas prácticas de diseño de la nueva solución generada
 - evitar lógica duplicada
 - mantener la simplicidad de la solución
 - definición de la interfaz de los componentes que minimice su acoplamiento
 - consistencia de la solución general
- Refactoring. Estrategias para realizar mejoras en forma iterativa al diseño de un sistema que ya se encuentra en funcionamiento.

Comunicación de la entrega

- Justificación de potenciales problemas de diseño
- Refactorizaciones propuestas para esos problemas.
- Calidad general de la documentación presentada

Diseño de Arquitectura y UI

Entrega 6

Se pide definir la arquitectura que utilizará en las siguientes entregas, especificando

- si elegirá una interfaz de usuario web o una cliente-servidor,
- si la arquitectura funcionará en forma centralizada o distribuida.

Justifique cada una de las decisiones.

Además debe implementar la interfaz de usuario de los siguientes casos de uso:

Generar equipos

- Establecimiento de criterios para generar equipos
 - Criterio de selección
 - Par/impar
 - 1,4,5,8,9
 - Criterio de ordenamiento
 - Por hándicap
 - Por promedio de notas del último partido
 - Por promedio de notas de los últimos n partidos
 - Mixto
- Botón Generar equipos, que actualiza...
- ...Resultado: equipos generados
 - Equipo 1, lista de jugadores
 - Equipo 2, lista de jugadores.
 - Al hacer click sobre un jugador de alguno de los dos equipos deben visualizarse en una pantalla aparte todos los datos del mismo
- Confirmación de equipos generados

Búsqueda de jugadores

- Criterios de búsqueda
 - Por nombre “comienza con”
 - Por apodo “contiene”
 - Búsqueda por fecha de nacimiento “anterior a”
 - Por rango desde/hasta del hándicap (puede ingresarse sólo desde, o sólo hasta)
 - Por rango desde/hasta del promedio de último partido
 - Filtrar sólo los que tuvieron infracciones, sólo los que no tuvieron infracciones, todos
- Información a mostrar de cada jugador
 - Nombre
 - Apodo

- Hándicap
 - Promedio
- Al hacer click sobre un jugador se visualizará en una pantalla aparte todos los datos del mismo

Pantalla de visualización de datos de un jugador

- Datos a visualizar
 - Nombre
 - Apodo
 - Hándicap
 - Promedio del último partido
 - Promedio de todos los partidos que jugó
 - Fecha de nacimiento (en formato dd/mm/yyyy)
 - Amigos (grilla, reutilizar la misma grilla definida en la búsqueda de jugadores)
 - Infracciones (grilla con fecha y hora y motivo)
 - Cantidad de partidos que jugó
- Botón Volver

Nota importante para aprobar el TP: no se debe duplicar la lógica para resolver los siguientes requerimientos

- Si un jugador tiene más de 8 puntos de hándicap, debe visualizarse en color azul en todos los casos de uso.
- Grilla que muestra resultados de la búsqueda de jugadores y grilla que muestra los amigos de un jugador
- Caso de uso Generar equipo: grillas con las formaciones de ambos equipos
- Ventana de visualización de datos de un jugador: debe estar definida en un solo lugar

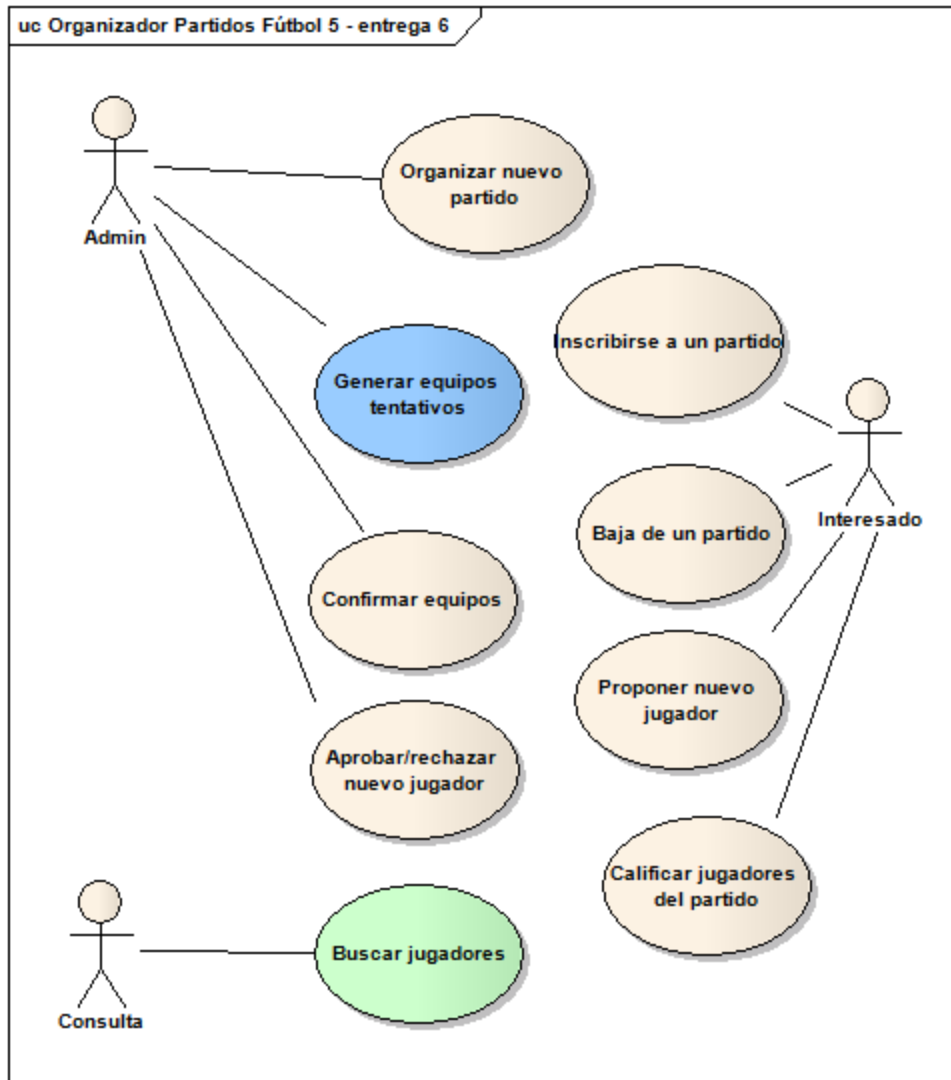


Diagrama de casos de uso - Entrega 6

Hoja de evaluación del docente

Diseño de la interfaz de usuario

- Atributos de calidad de la interfaz de usuario
 - Consistencia
 - Usabilidad: cómo guía al usuario a través del caso de uso
 - Robustez: cómo evita que el usuario cometa errores
 - Manejo de errores de usuario y de sistema
- Existencia de estrategias que permitan evitar la duplicación de código, con el objetivo de garantizar correctitud, minimizando la posibilidad de ocurrir errores durante la programación.
- Buenas prácticas de diseño
 - evitar lógica duplicada
 - mantener la simplicidad de la solución
 - definición de la interfaz de los componentes que minimice su acoplamiento
 - consistencia de la solución general
- Interacción entre componentes vista, controller, modelo, home
 - cohesión y acoplamiento entre dichos componentes

Comunicación de la entrega

- Calidad general de la documentación presentada

Entrega 7 - Diseño de Interfaz de Usuario

En esta entrega se pide

1. Realizar las correcciones y mejoras pedidas por su docente corrector.
2. Muestre un ejemplo de cómo resolvió en el TP cada uno de los siguientes temas
 - layout ⇒ disposición visual de controles
 - separación entre modelo y vista ⇒ grado de acoplamiento entre ambos concerns
 - manejo del estado conversacional de los datos que se cargan en la vista
 - binding unidireccional o bidireccional, comenten con cuál de los dos trabajaron y por qué.
 - manejo de eventos
 - navegación
 - cómo es el flujo de una vista a otra
 - cómo se pasa información de una vista a otra

No debe teorizar en abstracto, debe relacionar los conceptos con su intervención en el trabajo práctico.

Hoja de evaluación del docente

Diseño de la interfaz de usuario

- Revisión de las correcciones de la entrega anterior
- Relación entre cada uno de los siguientes temas y su correspondiente implementación
 - layout
 - separación entre modelo y vista
 - manejo del estado conversacional de los datos que se cargan en la vista
 - tipo de binding utilizado
 - manejo de eventos
 - navegación
 - cómo es el flujo de una vista a otra
 - cómo se pasa información de una vista a otra

Comunicación de la entrega

- Calidad general de la documentación presentada

Diseño Lógico de datos

Entrega 8 - Modelo relacional

En base a los requerimientos diseñados hasta el momento, se pide:

1. Implemente un script que defina el modelo de datos siguiendo el esquema relacional. Utilice las herramientas que crea conveniente (DER, Diccionario de datos, etc.) para comunicar
 - a. las entidades,
 - b. sus atributos y dominios (tipos de datos) y
 - c. las relaciones entre entidades, incluyendo restricciones para garantizar la consistencia del modelo.
2. Implemente un script SQL que realice la carga de un juego de datos o fixture sobre el cual trabajar
3. Defina componentes (vistas, funciones o procedimientos almacenados, triggers, etc.) que resuelvan cada uno de los siguientes requerimientos. Justifique su elección.
 - a. Saber los jugadores malos, con un handicap de 5 ó menos
 - b. Saber qué jugadores son traicioneros, los que tuvieron más de 3 infracciones el último mes
 - c. Saber cuántos jugadores podrían “mejorar” con el tiempo, son los jugadores malos que tienen menos de 25 años (**nota: debe evitar lógica duplicada**)
 - d. Reflejar que un jugador se da de baja a un partido.
 - e. Cada vez que un jugador se baje de un partido se le debe agregar una infracción si no ofrece reemplazante.

Hoja de evaluación del docente

Diseño lógico de datos

- Definición de entidades y relaciones, incluyendo
 - atributos de las entidades y sus dominios / tipos de datos
 - relaciones uno a muchos y muchos a muchos
- Buenas prácticas del modelo de datos definido
 - Consistencia: de qué manera garantiza la integridad
 - Grado de normalización / redundancia

Componentes que resuelven los requerimientos

- Justificación de los componentes elegidos para resolver cada uno de los requerimientos
- Buenas prácticas de diseño
 - evitar lógica duplicada
 - mantener la simplicidad de la solución
 - definición de la interfaz de los componentes que minimice su acoplamiento

Comunicación de la entrega

- Calidad general de la documentación presentada

Entrega 9 - Parte 1: Object / Relational Mapping

Partiendo de la implementación o especificación existente, ud. debe diseñar el mapeo entre el modelo de objetos de dominio (jugadores, infracciones, partidos, amigos, etc.) y el esquema relacional,

- considerando cada uno de los siguientes ejemplos de impedance mismatch debe explicar para un caso concreto qué decisión de diseño tomó
 - implementación relacional de la herencia en objetos
 - manejo de identidad
 - relaciones
 - uno a muchos
 - muchos a muchos
 - recursivas hacia la misma entidad

Se pide explicar un ejemplo de cada tipo de relación.

 - tipos de dato (Ej: String sin especificar longitud máxima vs. CHAR(20), objeto Date y tipo primitivo Date, etc.)
- Implementar la solución utilizando un framework O/R M que soporte la especificación JPA. Explicar
 - qué componentes del sistema se vieron afectados
 - cómo ayudó la arquitectura presentada en el trabajo práctico 2.1 para minimizar el impacto de los cambios. Relacionarlo con el concepto acoplamiento.

En el caso de implementarse el resultado debe ser

1. ingresar al sistema
2. generar equipos
3. cerrar la aplicación
4. volver a abrir la aplicación y visualizar los equipos anteriormente generados.

En el caso de especificar la solución debe hacerlo asegurando que al desarrollarla se persista la información cuando se generen los equipos.

Hoja de evaluación del docente (Parte 1)

Diseño del mapeo entre el modelo de objetos y el relacional

- Justificación de la estrategia de mapeo definida para cada uno de los siguientes casos
 - herencia
 - manejo de identidad
 - relaciones
 - uno a muchos
 - muchos a muchos
 - recursivas
 - definición de tipos de dato
- Relación de la implementación o especificación del mapeo O/R y del componente de acceso a datos con las buenas prácticas de diseño
 - evitar lógica duplicada
 - mantener la simplicidad de la solución
 - definición de la interfaz de los componentes que minimice su acoplamiento

Comunicación de la entrega

- Justificación del impacto de los cambios pedidos en la arquitectura existente y su relación con el concepto acoplamiento
- Calidad general de la documentación presentada

Entrega 9 - Parte 2: NoSQL

Se pide que diseñe e implemente un componente que garantice la persistencia de la aplicación a través de un esquema de persistencia no relacional.

Como resultado de esta implementación debe ser posible

1. ingresar al sistema
2. generar equipos
3. cerrar la aplicación
4. volver a abrir la aplicación y visualizar los equipos anteriormente generados.

En el caso de especificar la solución debe hacerlo asegurando que al desarrollarla se persista la información cuando se generen los equipos.

Hoja de evaluación del docente (parte 2)

Diseño del esquema de persistencia

- Relación de la implementación / especificación del esquema de persistencia y del componente de acceso a datos con las buenas prácticas de diseño
 - evitar lógica duplicada
 - mantener la simplicidad de la solución
 - definición de la interfaz de los componentes que minimice su acoplamiento

Comunicación de la entrega

- Justificación del impacto de los cambios pedidos en la arquitectura existente y su relación con el concepto acoplamiento
- Calidad general de la documentación presentada