DP1 2021-2022

Documento de Diseño del Sistema

Proyecto Idus Martii

https://github.com/gii-is-DP1/dp1-2021-2022-g5-04

Miembros:

- Arias Expósito, José Ramón
- Carnero Vergel, Manuel
- Moreno Pérez, Juan Carlos
- Sabugueiro Troya, David
- Santos Pérez, Pablo
- Serrano Mena, Antonio Roberto

Tutor: Manuel Resinas

GRUPO G5-04

Versión 3.0

26/12/2021

Historial de versiones

Fecha	Versión	Descripción de los cambios	Sprint
26/12/2021	V1	● Creación del documento ● Introducción ● Diagrama de capas primera versión	3
04/01/2022	V2	● Añadido diagrama de dominio/diseño ● Explicación de la aplicación del patrón caché	3
06/01/2022	V3	● Puesta a punto del documento	3
21/01/2022	V4	● Arreglos de diagramas y finalización del documento	4

Contents

Añade encabezados (Formato > Estilos de párrafo) y aparecerán en el índice.

Introducción

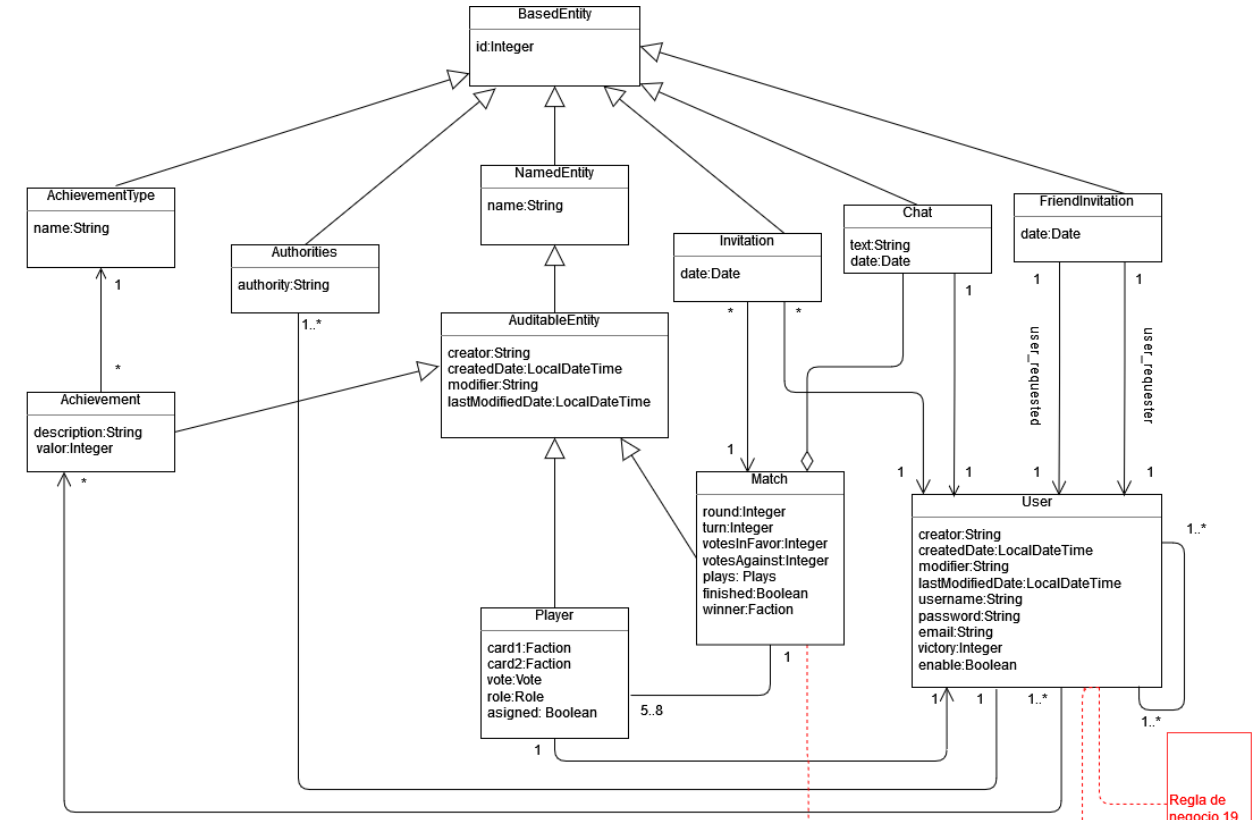
Para este entregable teníamos como objetivo alcanzar el 100% en la implementación de historias de usuarios del proyecto , lo que se puede traducir en que el juego es totalmente funcional, con una interfaz gráfica acorde con el juego y totalmente desvinculados del proyecto base PetClinic. Todo el código producido está totalmente optimizado y junto con las pruebas unitarias de cada historia de usuario, conforman nuestro proyecto completo.

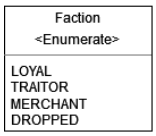
Diagrama(s) UML:

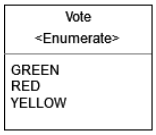
Diagrama de Dominio/Diseño

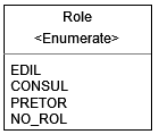
DP1 2021/22
Idus Martii

Grupo: G5-04









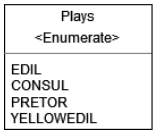
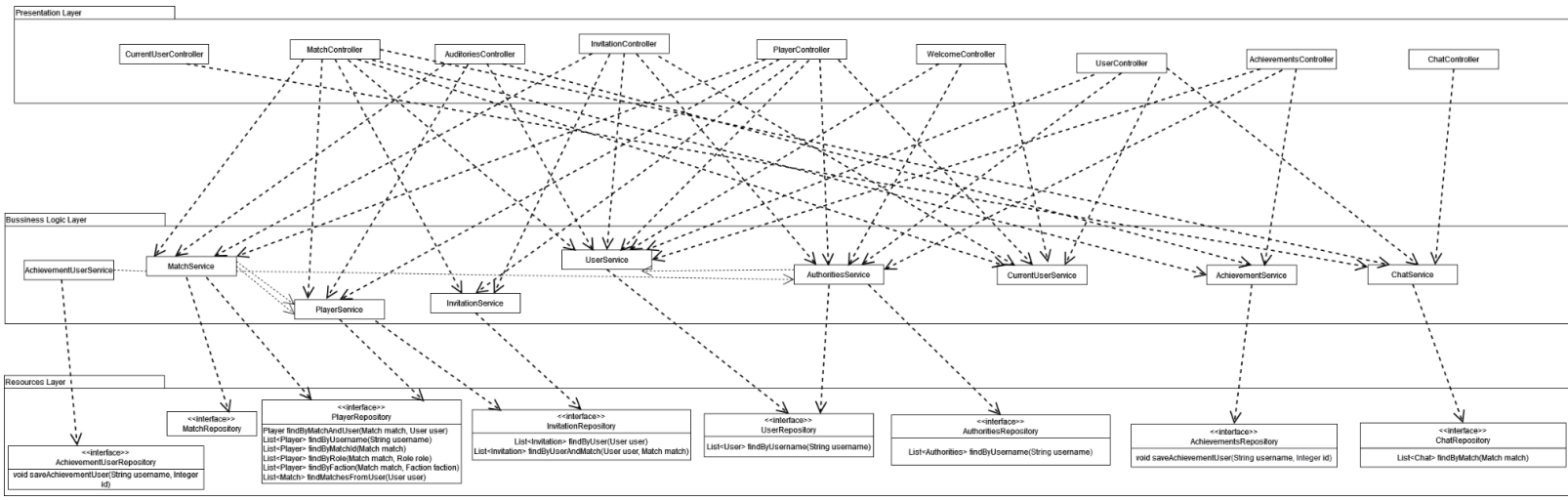
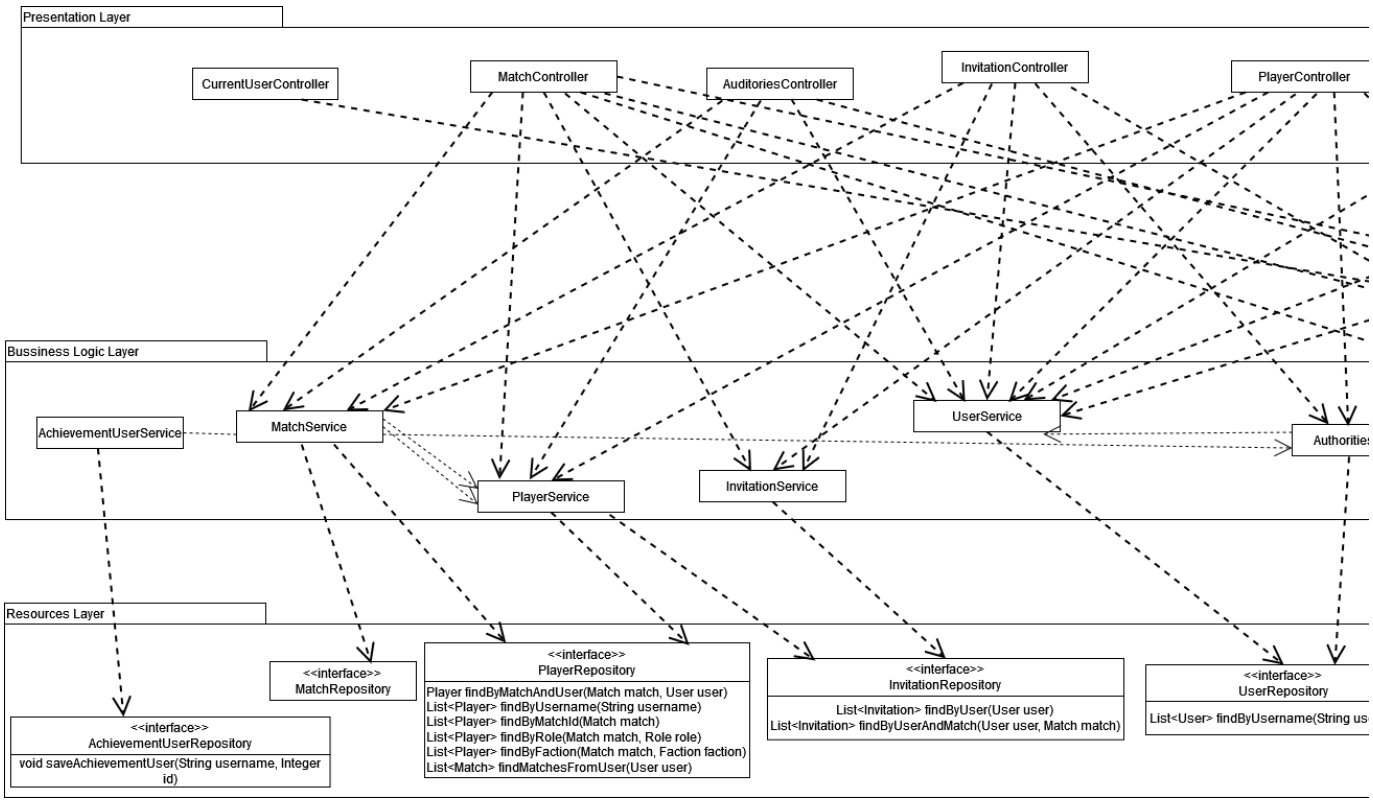
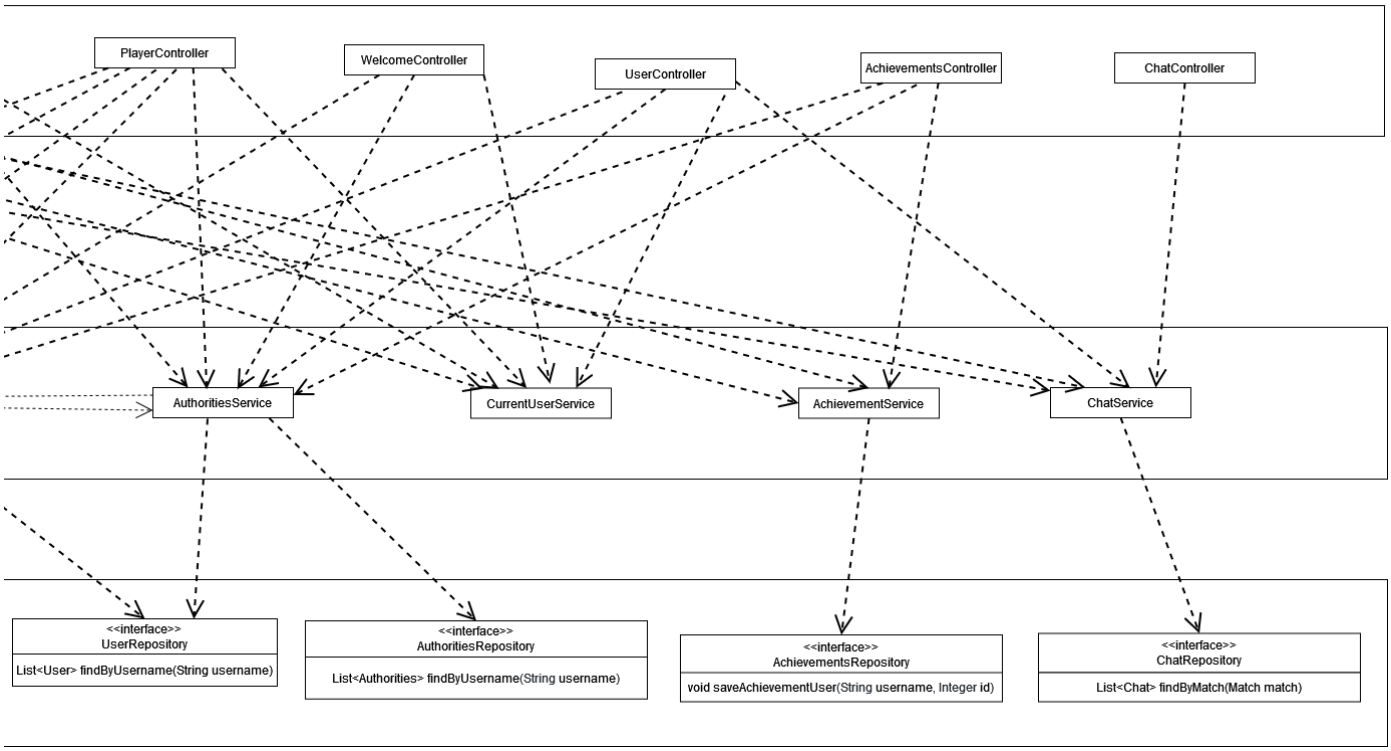


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)




Presentation Layer
CurrentUserController
MatchController
AuditoriesController
InvitationController
PlayerController
Bussiness Logic Layer
AchievementUserService
MatchService
PlayerService
InvitationService
UserService
Authorities
Resources Layer
<<interface>>
AchievementUserRepository
void saveAchievementUser(String username, Integer id)
<<interface>>
MatchRepository
<<interface>>
PlayerRepository
Player findByMatchAndUser(Match match, User user)
List<Player> findByUsername(String username)
List<Player> findByMatchId(Match match)
List<Player> findByRole(Match match, Role role)
List<Player> findByFaction(Match match, Faction faction)
List<Match> findMatchesFromUser(User user)
<<interface>>
InvitationRepository
List<Invitation> findByUser(User user)
List<Invitation> findByUserAndMatch(User user, Match match)
<<interface>>
UserRepository
List<User> findByUsername(String us


PlayerController
WelcomeController
UserController
AchievementsController
ChatController
AuthoritiesService
CurrentUserService
AchievementService
ChatService
<<interface>>
UserRepository
List<User> findByUsername(String username)
<<interface>>
AuthoritiesRepository
List<Authorities> findByUsername(String username)
<<interface>>
AchievementsRepository
void saveAchievementUser(String username, Integer id)
<<interface>>
ChatRepository
List<Chat> findByMatch(Match match)

Patrones de diseño y arquitectónicos aplicados

Patrón: Estado

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado parcialmente en algunos aspectos de la aplicación relacionados con la partida (entidad “Match”).

Clases o paquetes creados

Los paquetes y sus clases creados con este patrón son:

- paquetes -> “org.springframework.samples.IdusMartii.enumerates”
- clases -> paquete.enumerates - {“Plays.java”}

Ventajas alcanzadas al aplicar el patrón

Con este patrón hemos podido controlar los cambios producidos en el estado de la partida a través de “Plays.java” y la adición de un atributo con dicho tipo en la entidad “Match”, consiguiendo así eliminar muchas condiciones de nuestro juego a la hora de implementar la lógica del mismo además de poder agregar nuevos estados sin tener que modificar los ya existentes o el contexto de ellos.

Patrón: Modelo-Vista-Controlador (MVC)

Tipo: Arquitectónico

Contexto de Aplicación

El patrón se ha aplicado en prácticamente toda la aplicación:

- Modelo: carpetas “src/main/java” (excepto el paquete que termina con .web, el que termina en IdusMartii y el que termina en .exceptions) y “src/main/resources”
- Vista: carpeta “src/main/webapp”
- Controlador: carpeta “src/main/java” (únicamente el paquete que termina con .web)

Clases o paquetes creados

Los paquetes y sus clases creados con este patrón son:

- paquetes -> “org.springframework.samples.IdusMartii.configuration”, “org.springframework.samples.IdusMartii.enumerates”, “org.springframework.samples.IdusMartii.model”, “org.springframework.samples.IdusMartii.repository”, “org.springframework.samples.IdusMartii.service”, “org.springframework.samples.IdusMartii.util” y “org.springframework.samples.IdusMartii.web”
- clases -> paquete.configuration - {“ExceptionHandlerConfiguration.java”, “GenericIdToEntityConverter.java”, “SecurityConfiguration.java” y “WebConfig”} paquete.enumerates - {“Faction.java”, “Plays.java”, “Role.java” y “Vote.java”} paquete.model - {“Achievement.java”, “AuditableEntity.java”, “AuditorAwareImpl.java”,

“Authorities.java”, “BaseEntity.java”, “Invitation.java”, “Match.java”, “NamedEntity.java”; “Owner.java”, “Player.java”, “User.java” y “Chat.java”}
paquete.repository - {AchievementRepository.java”, AchievementUserRepository.java”,“AuthoritiesRepository.java”, “FriendInvitationRepository.java“,”FriendRepository.java”,InvitationRepository.java”, “MatchRepository.java”, “PlayerRepository.java”, “UserRepository.java” y “ChatRepository.java”}
paquete.service - {“AchievementService.java”, “AuthoritiesService.java”, “CurrentUserService.java”, “InvitationService.java”, “MatchService.java”, “PlayerService.java”, “UserService.java” y “ChatService.java”}
paquete.util - {“CallMonitoringAspect.java” y “EntityUtils.java”}
paquete.web - {“AchievementController.java”, “AuditoriesController.java”, “CrashController.java”, “CurrentUserController.java”, “InvitationController.java”, “MatchController.java”, “PlayerController.java”, “UserController.java”,“WelcomeController.java” y “ChatController.java”}

Ventajas alcanzadas al aplicar el patrón

Con este patrón hemos podido separar las responsabilidades de forma que se nos ha hecho más fácil el implementar el juego, pues de esta forma, tenemos un controlador específico para cada entidad, eliminando a su vez el acoplamiento y favoreciendo la lectura del código y la localización de errores en el mismo.

Patrón: Layer Supertype

Tipo: Diseño

Contexto de Aplicación

El patron se ha aplicado con las clases BaseEntity y NamedEntity

Clases o paquetes creados

Los paquetes y sus clases creados con este patrón son:

- paquetes -> “org.springframework.samples.IdusMartii.model”
- clases -> paquete.model - {“BaseEntity.java” y “NamedEntity.java”}

Ventajas alcanzadas al aplicar el patrón

Con este patrón nos hemos ayudado con los id de nuestras clases, ya que con BaseEntity o NamedEntity nos genera el id sin necesidad de definirlo en cualquier clase, ya que nuestras clases extienden a ellas.

Patrón: Repository

Tipo: Diseño

Contexto de Aplicación

El patron se ha aplicado para encapsular la lógica de negocio necesaria para acceder a los datos.

Clases o paquetes creados

Los paquetes y sus clases creados con este patrón son:

- paquetes -> “org.springframework.samples.IdusMartii.repository”

- clases -> paquete.repository - {AchievementRepository.java", AchievementUserRepository.java","AuthoritiesRepository.java", "FriendInvitationRepository.java","FriendRepository.java",InvitationRepository.java", "MatchRepository.java", "PlayerRepository.java", "UserRepository.java" y "ChatRepository.java"}

Ventajas alcanzadas al aplicar el patrón

Con este patrón nos hemos ayudado de las Queries que nos deja usar este patrón, con ello podemos acceder a los datos de la base de datos de una manera más fácil, y con ello, implementar métodos del servicio, ya que el servicio llama al repositorio.

Decisiones de diseño

En esta sección describiremos las decisiones de diseño que se han tomado a lo largo del desarrollo de la aplicación que vayan más allá de la mera aplicación de patrones de diseño o arquitectónicos.

Decisión 1: Realización de pruebas de la partida

Descripción del problema:

Como grupo nos gustaría poder probar algunos estados de la partida, significando dichos estados, un conjunto de condiciones que se pueden dar en una partida en un determinado momento, porque nos resultaba necesario para poder asegurar la jugabilidad del juego. El problema es que hay ciertas pruebas que son muy complejas para realizarlas como tests.

Alternativas de solución evaluadas:

Alternativa 1.a: Incluir las condiciones en el propio script de inicialización de la BD (data.sql).

Ventajas:

- Simple, no requiere nada más que modificar el SQL que genera la partida.
- No afecta la elaboración de pruebas ni el trabajo diario.

Inconvenientes:

- Se necesita un tiempo extra para poder probar todos los posibles estados.

Alternativa 1.b: Crear un controlador y un servicio que permita incluir las condiciones sin tener que cambiar el script de inicialización de la BD (data.sql).

Ventajas:

- Permite reutilizar los datos ya existentes en (data.sql).

Inconvenientes:

- Afecta la elaboración de pruebas, ya que habría que realizar más pruebas para el controlador y el servicio.
- Supone añadir más complejidad al código.

Justificación de la solución adoptada

Como no queremos renunciar a trabajar de forma ágil, escogimos la alternativa 1.a, pues modificar la base de datos según nos convenga era más rápido que crear un controlador y un servicio y las pruebas asociadas a ellos.

Decisión 2: Refresco puntual de la partida

Descripción del problema:

Como grupo y posibles jugadores, nos gustaría que cuando el estado de una partida en curso cambie sea mostrado dicho cambio en las ventanas de todos los jugadores presentes para que por ejemplo cuando los ediles terminen de votar, se le muestren automáticamente al pretor los botones de revisar los votos y hacer de la experiencia de juego algo más amena.

Alternativas de solución evaluadas:

Alternativa 2.a: Añadir un autorefresco a la página de duración moderada, por ejemplo 5-10 segundos para que los jugadores no tengan que estar refrescando la página para saber si es su turno o no.

Ventajas:

- Dificultad baja, es solo añadir poca cantidad de código en el controlador de la partida en curso
- No afecta la elaboración de pruebas ni el trabajo diario.

Inconvenientes:

- Si el tiempo de autorefresco es de 10 segundos y el estado de la partida cambia muy poco después de haberse refrescado, el jugador debe esperar 10 segundos para que se le actualicen los cambios o refrescar manualmente.

Alternativa 2.b: Implementar un RestController que se encargue de actualizar la página cuando suceda un cambio.

Ventajas:

- La experiencia de usuario mejora considerablemente al no tener que esperar al autorefresco.
- optimización a la hora de probar funcionalidades desde el punto de vista del jugador.

Inconvenientes:

- Dificultad alta, al momento de conocer la existencia de esta alternativa, desconocemos la manera de implementarla.
- El desconocimiento presente implica que si queremos decantarnos por esta solución debemos destinar parte de nuestro tiempo a investigar su funcionamiento e implementación.

Justificación de la solución adoptada

Para no perder demasiado tiempo en una solución cuyo impacto tampoco es tan grande si el tiempo de autorefresco es reducido, escogimos la alternativa 2.b para poder centralizar nuestro tiempo a labores y tareas de mayor relevancia en el proyecto

Decisión 3: Eliminación de usuarios en el lobby de la partida

Descripción del problema:

Queremos decidir una manera eficaz de eliminar a los jugadores de la partida como host sin tener oportunidad de expulsarse a sí mismo.

Alternativas de solución evaluadas:

Alternativa 3.a: Implementar una excepción cuya función consiste en devolver un error en el caso de intentar eliminarse a uno mismo

Ventajas:

- Evita poder realizar esta acción a través de la url del navegador.

Inconvenientes:

- Añade complejidad al código
- Peor rendimiento

Alternativa 3.b: Eliminar de la vista el botón de eliminar que aparecería en el host

Ventajas:

- Simplicidad en el código
- Mejor rendimiento

Inconvenientes:

- Posibilidad de acceder al método mediante peticiones HTTP

Justificación de la solución adoptada

Aunque realizar el borrado mediante una petición HTTP, vemos reducida la posibilidad de que los jugadores conozcan la petición correspondiente a ese método porque en ningún momento es visible para el usuario, por lo que la alternativa 3.b nos pareció una solución cómoda y eficaz para que el juego siga su curso sin errores graves.

Decisión 4: Chat de la partida

Descripción del problema:

Como jugador de la partida, me gustaría que el chat se pudiera ver bien, sin que nada lo obstaculice, para que se puedan ver bien los mensajes de los demás jugadores.

Alternativas de solución evaluadas:

Alternativa 4.a: Crear el chat por separado a la partida, con lo cual, se mostrará un enlace del chat en la propia partida, para tener las 2 cosas separadas y que no se pisen.

Ventajas:

- Baja dificultad, ya que su implementación es algo muy básico y que se ha realizado antes.
- No afecta al transcurso de la partida

Inconvenientes:

- Los jugadores tienen que estar cambiando de ventana para cuando quieran chatear o jugar.

Alternativa 2.b: Implementar el chat dentro de la partida.

Ventajas:

- La experiencia de usuario mejora considerablemente al no tener que cambiar de ventana para chatear y jugar

Inconvenientes:

- Dificultad alta, ya que tendríamos que implementar todo lo que conlleva la partida con su lógica junto con el chat.
- Demasiado cargada la vista de la partida, ya que nosotros utilizamos cartas, la vista de la partida se vería demasiado llena de elementos

Justificación de la solución adoptada

Para ir a lo seguro, hemos implementado el chat en una ventana aparte a la de la partida, así nos aseguramos de que ambas partes funcionen correctamente.

Decisión 5: Revisión de voto nulo

Descripción del problema:

Como equipo, queremos encontrar una manera adecuada de hacerle saber a los jugadores que el Pretor ha descubierto un voto nulo a la hora de revisarlo.

Alternativas de solución evaluadas:

Alternativa 5.a: Redirigir a todos los jugadores a una nueva vista especializada para este anuncio

Ventajas:

- Solución bastante llamativa que da mucho juego por las posibilidades de aviso disponibles

Inconvenientes:

- Complejidad de código aumentada
- Creación de una nueva vista

Alternativa 5.b: Añadir un mensaje en la vista de la partida en curso

Ventajas:

- Extremadamente simple
- Claro y conciso
- Simplicidad en el código

Inconvenientes:

- Pérdida de oportunidad de hacer la experiencia más divertida

Justificación de la solución adoptada

Debido a decisiones de equipo, aunque nuestra alternativa favorita era dedicarle una vista individual a este aviso para dar más emoción a la partida, nos decantamos por la alternativa 5.b para mejorar nuestra organización y poder dedicar más tiempo a otros aspectos del proyecto