Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DE SOFTWARE

PokerRun: Desarrollo de una aplicación web de póker online

Autor:
D. Fernando Zamora Díez

Tutor:
D. Joaquín Adiego Rodríguez

Resumen

Ante la ausencia de sitios web de poker online sencillos y que ofrezcan una buena experiencia para el jugador, nace PokerRun.
Una aplicación web desarrollada enteramente utilizando Javascript, tanto para la capa del cliente como para la del servidor lo que comunmente se le conoce como MEAN stack. Para conseguir la actualización de la información en tiempo real se ha hecho uso de Socket.io, una librería basada en websocket que permite una comunicación bidireccional.
Esta aplicación permite la creación y el desarrollo de torneos de póker plenamente funcionales en la modialidad Texas Hold'em.
Siguiendo un estilo que se está haciendo muy popular en Internet para juegos multijugador: interfaz clara y sencilla, facilidad para crear partida con 'dos clicks', compartir el código de la partida con tus amigos y jugar.

Abstract

In the absence of simple online poker websites that offer a good player experience, PokerRun was born.
A web application developed entirely using Javascript, both for the client and server layers, which is commonly known as the MEAN stack. In order to update the information in real time, use has been made of Socket.io, a websocket-based library that allows two-way communication.
This application allows the creation of fully functional poker tournaments in the Texas Hold'em variant.
Following a style that is becoming very popular on the Internet for multiplayer games: clear and simple interface, easy to create a game with 'two clicks', share the game code with your friends and play.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este capítlo hablaremos acerca de las motivaciones que han posibilitado que se desarrolle el proyecto, cuáles han sido los objetivos marcados en este, los usuarios objetivos hacia los que va orientado y un breve resumen de las partes en las que se divide la memoria.

1.1. Motivaciones

La principal motivación que ha hecho posible el desarrollo de este proyecto ha sido la aparición de una necesidad.

Antes solía divertirme con mis amigos jugando partidas de póker entre nosotros, en Marzo de 2020 cuando empezó la pandemia y se inició la cuarentena en España la posibilidad de jugar presencialmente se hizo nula. Buscamos sitios web que nos permitieran jugar cada uno desde nuestra casa pero lo único que encontrábamos eran páginas donde había que hacer un registro previo o casas de póker (como PokerStarts, Winamax, PartyPoker..) que contaban con un modo de juego gratuito pero que al igual que las otras había que registrarse.

Por ello surgió la idea de desarrollar este proyecto, un sitio web sin registros y fácil de usar. En el que creando partida y compartiendo el código del torneo ya pudieras jugar con tus amigos.

1.2. Objetivos

1.2.1. Objetivos personales

Los objetivos personales marcados para este proyecto son principalmente formativos:

- Ampliar los conocimientos en el framework Angular 2+
- Conocer y aplicar conceptos de la gestión de proyectos
- Mejorar y poner a prueba mi organización y autogestión
- Aprender a desarrollar una aplicación web multijugador en tiempo real

1.2.2. Objetivos de desarrollo

El principal objetivo de desarrollo será suplir la necesidad por la que surge este proyecto, proveer de un sitio web que permita la realización de partidas de póker entre amigos lo más sencillamente posible, mediante un código de torneo.

Lo que se busca conseguir es que , para un usuario que quiera crear un torneo, únicamente rellenando 3 campos lo pueda hacer y, para un usuario que quiera unirse al torneo de un amigo solamente tenga que introducir su nombre y el código del torneo. A todo esto se le añadiría un diseño de la interfaz facil y simplista que permita un correcto uso de la aplicación y ayude en la experiencia de juego del usuario.

1.3. Estudio de mercado

El usuario objetivo al que va orientada esta aplicación web tiene el perfil de una persona que juega al póker de manera *amateur*, no contempla la posibilidad de apostar dinero real y quiere evitar registrarse en sitios web. La edad de este perfil de persona abarcaría todo tipo de edades desde los 18 años, edad en la que conoces los riesgos que conllevan este tipo de juegos y la peligrosidad de que, una actividad meramente de ocio en la que no gastas dinero pueda pasar a convertirse en una adicción en la que se pierde dinero.

1.3.1. PokerStars

La conocida sala de póker online, la principal desventaja que se ha encontrado, como pasa con todas las salas de este calibre, es que tienes que realizar un registro previo, enviando una foto del DNI y añadiendo incluso la información de pago aunque no tengas pensado ingresar dinero. Una vez hecho todo el registro, crear un torneo para jugar con tus amigos es complicado, tras haber navegado por la inmensa cantidad de opciones que tiene su menú

llegarás al apartado 'Home Games', donde tendrás que crear un nuevo club, invitar a tus amigos al club, que tus amigos acepten la invitación, una vez aceptada tendrás que crear el torneo a una hora determinada configurándo por lo menos 10 opciones, tras haberlo creado tus amigos tendrán que ir al menú del club para encontrar la partida.

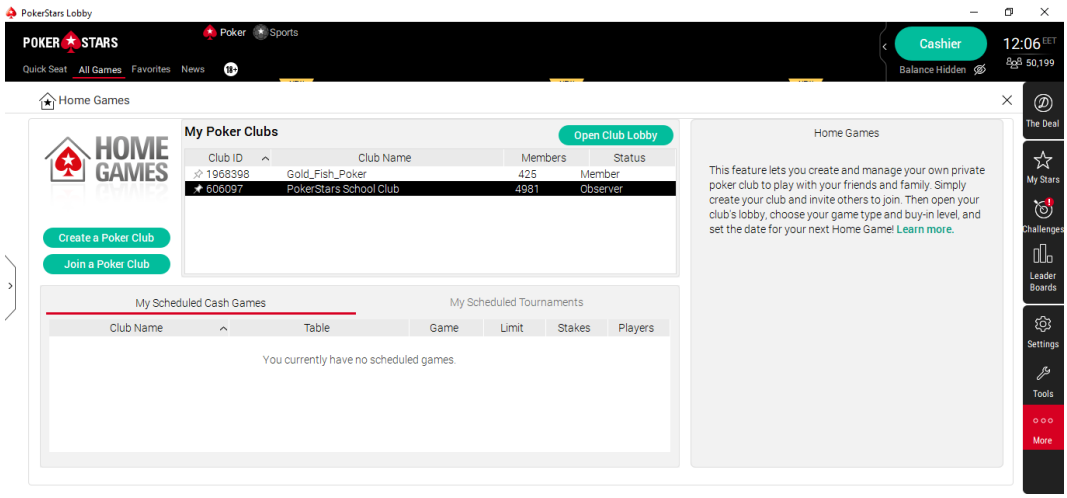

Figura 1.1: Interfaz Pokerstars

1.3.2. Winamax

Una de las mayores salas de póker online, al igual que pasa con PokerStars, tienes que realizar un registro previo en el que tambíen tienes que proporcionar tus datos bancarios, tiene demasiados menús y demasiadas opciones dentro de cada menú. Para poder jugar con tus amigos, todos tus amigos tienen que estar registrados en la plataforma, se tiene que crear un torneo e invitarlos.

Figura 1.2: Interfaz Winamax

1.4. Organización de la memoria

A continuación se realizará una pequeña de los capítulos que tiene esta memoría y que temas se tratan en cada uno:

- **Capítulo 1: Introducción**: Se describen las motivaciones y objetivos del proyecto
- **Capítulo 2: Contexto**: Se explica y da una idea general del juego del póker
- **Capítulo 3: Plan de desarrollo del proyecto**: Se presentan los riesgos del proyecto, la metodología de desarrollo utilizada y el seguiemiento del proyecto.
- **Capítulo 4: Análisis de la aplicación**: Se presentan los requisitos, casos de uso y el modelo de dominio del sistema.
- **Capítulo 5: Tecnologías utilizadas**: Se describen las principales tecnologías utilizadas para desarrollar el proyecto
- **Capítulo 6: Diseño de la aplicación**: Se presentan los diagramas de diseño de la aplicación
- **Capítulo 7: Implementación**: Se explica cómo se ha llevado a cabo la implementación del diseño
- **Capítulo 8: Pruebas**: Se muestras las pruebas que ha pasado el software y sus resultados

- **Capítulo 9: Conclusiones y líneas futuras**: Se comenta la evaluación final del proyecto, si se han conseguido los objetivos marcados y qué se tiene en mente para el futuro del proyecto

Capítulo 2

Contexto

2.1. Qué es el póker

El póker [24] es un juego de cartas cuya dinámica principal son las apuestas, hay una puja inicial a partir de la cual los jugadores apuestan tratando de hacerse con la suma del total, bien teniendo la mejor combinación de cartas o bien haciendo que los demás jugadores abandonen.

2.1.1. Variantes principales

Las variantes de póker más jugadas en el mundo son Texas Hold'em y Omaha, ambas tienen estructuras de apuesta idénticas, la principal diferencia en el número de cartas que tiene el jugador.

En Omaha el jugador dispone de 5 cartas para poder combinar 2 de ellas con las que están en la mesa, en cambio en Texas Hold'em el jugador únicamente dispone de 2 cartas.

La variante que vamos a utilizar para este torneo va a ser Texas Hold'em ya que aparte de ser la más popular entre las 2, es la que conoce y ha jugado el desarrollador

2.1.2. Formatos de juego

Además de variantes, el póker puede ser jugado en 2 formatos principales:

- **Partidas de cash**: En las partidas de cash un jugador puede abandonar la mesa en el momento que quiera, llevandose consigo el dinero que haya ganado o dejando en la mesa lo que haya perdido

- **Torneos**: En los torneos, los jugadores juegan hasta que se quedan sin fichas. Las primeras posiciones reciben premio

2.2. Texas Hold'em

Texas Hold'em [14] es una variante del juego del póker. Dentro de esta hay dos posibilidades: con límite de apuesta máximo, es decir, la máxima cantidad de fichas para apostar esta fijada de antes; y *no limit*, puedes apostar la cantidad que quieras hasta quedarte sin fichas.

Esta última modalidad es la que vamos a seguir en el proyecto.

2.2.1. Acciones

Los jugadores podrán realizar una de las siguientes acciones cuando sea su turno:

- **No ir**: Retirarse de la mano y perder tus cartas
- **Pasar**: Evitar hacer una apuesta pero continuar en la mano
- **Igualar**: Igualar la apuesta más alta hasta el momento
- **Apostar**: Establecer la apuesta más alta
- **Subir**: Aumentar la apuesta más alta

2.2.2. Apuestas

El Texas Hold'em se juega utilizando las apuestas *ciega pequeña* y *ciega grande*. Dichas ciegas son apuestas obligatorias que tienen que hacer los jugadores de forma periódica al comienzo de cada mano. De esta forma se evita que un jugador permanezca inactivo durante el juego, jugando unicamente las mejores cartas, además da fluidez y agilidad a la partida.

- **Botón**: El botón es la posición a partir de la cual el dealer va a repartir las cartas. En la mesa viene indicado con una ficha de plástico en la posición del jugador
- **Ciega pequeña**: Se encarga de ponerla el jugador a la izquierda del *dealer*, es la mitad de una ciega grande
- **Ciega grande**: Se encarga de ponerla el jugador a la izquierda de la ciega pequeña, es la apuesta mínima fijada en el sistema de ciegas del torneo

Dichas posiciones van a ir rotando en la direcciñon de las agujas del reloj a lo largo de la partida. De esta forma el número de apuestas obligatorias se distribuye equitativamente entre todos los jugadores.

2.2.3. Manos de póker

Las posibles combinaciones de cartas ordenadas de mejor a peor son las siguientes:

Escalera real: Cinco cartas del mismo palo del 10 hasta el As

Escalera de color: Cinco cartas consecutivas del mismo palo

Póker: Cuatro cartas del mismo valor

Full: Un trío y una pareja

Color: Cinco cartas del mismo color

Escalera: Cinco cartas consecutivas da igual el palo

Trio: Tres cartas del mismo valor

Doble pareja: Dos parejas

Pareja: Dos cartas del mismo valor

Carta alta: Carta de más valor

Figura 2.1: Ordenación manos de poker

2.2.4. Fases del juego

- **Preflop**: Se establece el botón junto con la ciega pequeña y la ciega grande y se reparten 2 cartas boca abajo a cada jugador, las cuales solo pueden ser vistas por él. Cuando todos lo juagores que no se hayan retirado de la mano igualen la apuesta máxima se pasa a la siguiente fase.
- **Flop**: El repartidor quema una carta y descubre tres cartas en el centro de la mesa. Vuele a haber una ronda de apuestas en el que todos tendrán que igualar la apuesta máxima para continuar jugando y pasar a la siguiente fase.
- **Turn**: El repartidor quema una carta y descubre otra en el centro de la mesa. Ahora hay 4 cartas en el centro.
- **River**: El repartidor vuelve a quemar una carta y descubre la quinta y útlima carta sobre la mesa.
- **Showdown**: Se muestran las cartas para decidir el ganador entre los que hayan igualado la máxima apuesta del River.

Capítulo 3

Plan de desarrolo del proyecto

En este capitulo se habla del plan de desarrollo que ha tenido este proyecto. Empezando por el análisis de riesgos a los que se puede enfrentar el proyecto, pasando por la estimación de los costes que acarreará el desarrollo, la planificación misma del proyecto, explicando el proceso elegido y terminando con el seguiemiento del proyecto para comprobar si toda la planificación anterior a surgido efecto o no, si ha aparecido algún riesgo mas y si se han cumplido los plazos.

3.1. Gestión de riesgos

A la hora de planificar un proyecto inevitablemente surgen riesgos, esto es debido a la falta de información y la incertidumbre que se crean alrededor de este. Ante cualquier mínima adicción o cambio, el proyecto es susceptible a riesgos y con ello peligra la finalización de este o por lo menos sus plazos.

Pero, ¿que es un riesgo? Es una medida de magnitud con la que podemos cuantificar los peligros. Aplicando esto a un proyecto se puede decir que un riesgo es un evento o condición con una determinada probabilidad de ocurrencia que tiene un efecto en el mismo.

Podemos dividir los riegos de dos formas:

- **De proyecto**: Los relacionados con los objetivos marcados dentro del proyecto, ya sean funcionalidades o plazos.
- **De negocio**: Los relacionados con el exito del proyecto en el mercado.

Un proyecto casi perfecto puede despues fracasar a la hora de venderlo y al revés, uno mediocre puede tener mejores resultados de los esperado.

3.1.1. Riesgos encontrados

Para este proyecto se han omitido los riegos de negocio, los cuales no aportan nada porque no se tiene pensado sacar esta aplicación web al mercado. Por lo que nos centraremos únicamente en los riesgos de proyecto.

A continuación una breve descripción de los atributos que tendrán los riegos:

- **Probabilidad**: Probabilidad de que el riesgo ocurra. Puede ser:
  - **Muy alta**: Mas del 80 %
  - **Alta**: 60 - 80 %
  - **Media**: 40 - 60 %
  - **Baja**: 20 - 40 %
  - **Muy baja**: Menos del 20 %
- **Impacto**: Si el riesgo se produce, que consecuencias negativas tendría para el proyecto. El impacto se mide en relación a los costes y plazos del proyecto:
  - **Muy alto**: Mas del 30 %
  - **Alto**: 15 - 30 %
  - **Medio**: 10 - 15 %
  - **Bajo**: 5 - 10 %
  - **Muy bajo**: Menos del 5 %
- **Plan de mitigación**: Estrategia para reducir la probabilidad de ocurrencia del riesgo
- **Plan de contingencia**: Estrategias que se van a tomar cuando aparezca el riesgo

Título	**Cambios en los requisitos**
Descripción	A lo largo del desarrollo de un proyecto los requisitos iniciales pueden cambiar. En este caso, si los requisitos cambiaran serían minimamente y afectaría poco, debido a que la idea principal de este proyecto es el póker, si eso cambia pues ya sería otro totalmente distinto.
Probabilidad	Muy baja
Impacto	Bajo
Plan de mitigación	Definir bien los requisitos al comienzo del proyecto, para que de haber cambios, estos sean mínimos
Plan de contingencia	Replanificación de los plazos del proyecto e implementación de los nuevos requisitos tanto en el análisis como en el código

Cuadro 3.1: Riesgo por cambiod en los requisitos

Título	**Enfermedad del desarrollador**
Descripción	El desarrollador cae enfermo y el desarrollo del proyecto se ve paralizado
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Seguir las medidas de prevención contra la CoVID-19 y tener preparada una ampliación de plazo para la finalizacion del proyecto
Plan de contingencia	Dependiendo de la indisponibilidad del desarrollador, aprovechar para realizar tareas del proyecto que no supongan un gran esfuerzo, como son la parte de bocetaje y la maquetación html y css. En caso de total indisponibilidad, retrasar todo el proyecto

Cuadro 3.2: Riesgo por enfermedad del desarrollador

Título	**Tiempo de respuesta**
Descripción	Debido a que el juego es online, si el tiempo de respuesta percibido por el usuario es muy grande esto haría a la partida muy lenta, pudiendo causar una mala experiencia de usuario
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Utilización de una base de datos que funcione bien con grandes cantidades de información, como lo es MongoDB; y un servidor de hosting con buenas características de conexión
Plan de contingencia	Contratación de un servidor de hosting de pago que ofrezca mejores servicios que uno gratuito

Cuadro 3.3: Riesgo por tiempo de respuesta

Título	**Avería en el ordenador**
Descripción	El ordenador del desarrollador sufre una avería, imposibilitando la continuación de las tareas del proyecto y la posible pérdida de datos del proyecto
Probabilidad	Muy baja
Impacto	Muy alto
Plan de mitigación	Ir guardando en la nube toda la documentación del proyecto y utilizar un sistema de control de versiones como Git para el código. Contar con un ordenador secundario que pueda suplir al actual
Plan de contingencia	Continuar de forma normal desarrollando el proyecto pero desde el nuevo ordenador

Cuadro 3.4: Riesgo por avería en el ordenador

Título	**Retraso en el desarrollo**
Descripción	La previsión del timpo empleado en las tareas se queda corto
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Ampliación del plazo
Plan de contingencia	Reorganizar las tareas

Cuadro 3.5: Riesgo por restraso en el desarrollo

Título	**Errores en el diseño**
Descripción	Un mal diseño final podría llevar a errores en la implementación
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	Dedicar una gran parte del proyecto al diseño, intentando conseguir algo muy concreto, mejorandolo iterativamente
Plan de contingencia	Reelaborar las partes del diseño erroneas hasta conseguir reflejar lo deseado

Cuadro 3.6: Riesgo por errores en el diseño

Título	**Falta de conocimiento**
Descripción	Falta de conocimiento del desarrollador en las tecnologías utilizadas
Probabilidad	Muy baja
Impacto	Alto
Plan de mitigación	Seleccionar tecnologías que el desarrollador conozca y en las que se sepa desenvolver
Plan de contingencia	Buscar información en internet y realizar algún curso sobre el tema

Cuadro 3.7: Riesgo por errores en el diseño

Título	**Contratación en trabajo externo**
Descripción	El desarrollador comienza a trabajar en una empresa, lo que reduce bastante las horas que puede desempeñar en el proyecto al día
Probabilidad	Alta
Impacto	Alto
Plan de mitigación	Continuar de forma normal hasta que se de el riesgo, entonces pasar al plan de contingencia
Plan de contingencia	Reestructurar la planificación para que se ajuste al tiempo que el desarrollador puede trabajar en el proyecto

Cuadro 3.8: Riesgo por contratación en trabajo externo

3.2. Planificación del proyecto

Debido a que el proyecto se trata de un juego online de póker, el cual va a poder ser jugable desde prácticamente el inicio de su desarrollo, se ha pensado que una buena metodología sería el Proceso Unificado (UP).

3.2.1. El Proceso Unificado

El Proceso Unificado es un marco de desarrollo de software que está dirigido por los casos de uso, es iterativo, es incremental y está enfocado en los riesgos.
Esto nos permite empezar con un prototipo muy simple al que se le van añadiendo funcionalidades para acabar terminando con el producto final. En el caso de este proyecto el prototipo simple sería salas donde usuarios se conectan por medio de un código, las funcionalidades serían ir añadiendo las distintas normas del póker, las apuestas y los turnos, para finalmente tener el juego final.

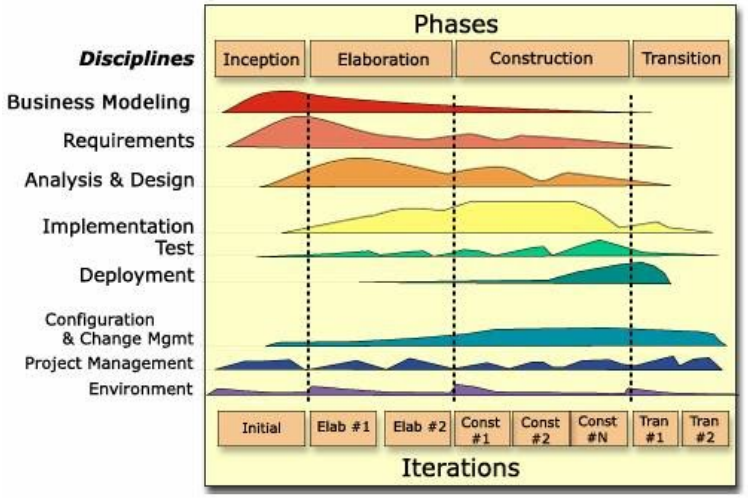

Figura 3.1: El Proceso Unificado

En la figura anterior podemos comprobar el funcionamiento del Proceso Unificado. Todos los flujos de trabajo tienen presencia mientras dure el desarrollo. Durante las primeras iteraciones lo predominante son las actividades de la fase de Inicio y Elaboración. En las iteraciones medias el flujos de trabajo que van a generar los documentos de la fase de Construcción son los que más carga van a tener, pero esto no evita que se trabajen en los demas, los cuales van a permitir que se continue mejorando y refinando el proyecto. Durante las iteraciones finales se emplea el tiempo en los documentos de Transición y al igual que en las iteraciones medias, se sigue trabajando en los demás pero meramente para rematar los últimos detalles.

Consta de 4 fases:

- **Inicio**: En esta fase se deciden cuales son los verdaderos objetivos del proyecto y cual va a ser su alcance
- **Elaboración**: Se realiza el analisis y diseño del proyecto, es decir, la identificación de los requisitos, los casos de uso, el modelo de dominio y los diagramas de secuencia
- **Construcción**: Se lleva a cabo la implementación del analisis y diseño previo. Consta de varias iteraciones en las que se van incorporando los casos de uso
- **Transición**: Fase final, se realizan las pruebas unitarias y se corrigen los ultimos detalles

Cada una de estas fases finaliza con un hito, la disponibilidad de un conjunto de modelos y documentos que han sido consecuencia del trabajo previo. Así los documentos generados con cada fase del desarrollo serían:

- **Inicio**: Gestión de riesgos y la planificación del proyecto
- **Elaboración**: Casos de uso, requisitos, modelo de dominio y diagramas de secuencia
- **Construcción**: Funcionalidades descritas en los casos de uso
- **Transición**: Pruebas y manual de instalación

3.2.2. Iteraciones del proyecto

Fase	Semanas	Fecha Incio	Fecha Fin	Iteraciones
Inicio	1	02/04/2020	08/04/2020	1
Elaboración	4	09/04/2020	06/05/2020	4
Construcción	8	07/05/2020	01/07/2020	4
Transición	2	02/07/2020	15/07/2020	2

Cuadro 3.9: Iteraciones del proyecto

Iteración	**1**
Iteración relativa	1
Fase	Inicio
Tareas	▪ Planificación del proyecto ▪ Gestión de riesgos ▪ Identificación de los requisitos ▪ Inicio del proyecto Angular
Días	7 días
Fecha Inicio	02/04/2020
Fecha Fin	08/04/2020

Cuadro 3.10: Iteración 1

Iteración	**2**
Iteración relativa	1
Fase	Elaboración
Tareas	▪ Detalle de los requisitos ▪ Formación Node y ExpressJS
Días	7 días
Fecha Inicio	09/04/2020
Fecha Fin	15/04/2020

Cuadro 3.11: Iteración 2

Iteración	**3**
Iteración relativa	2
Fase	Elaboración
Tareas	▪ Casos de uso ▪ Bocetos parte front ▪ Inicio proyecto Node para el back
Días	7 días
Fecha Inicio	16/04/2020
Fecha Fin	22/04/2020

Cuadro 3.12: Iteración 1

Iteración	**4**
Iteración relativa	3
Fase	Elaboración
Tareas	▪ Modelo de dominio
Días	7 días
Fecha Inicio	23/04/2020
Fecha Fin	29/05/2020

Cuadro 3.13: Iteración 4

Iteración	**5**
Iteración relativa	4
Fase	Elaboración
Tareas	▪ Diagramas de secuencia
Días	7 días
Fecha Inicio	30/04/2020
Fecha Fin	06/05/2020

Cuadro 3.14: Iteración 5

Iteración	**6**
Iteración relativa	1
Fase	Construcción
Tareas	▪ Maquetación front
Días	14 días
Fecha Inicio	07/05/2020
Fecha Fin	20/05/2020

Cuadro 3.15: Iteración 6

Iteración	**7**
Iteración relativa	2
Fase	Construcción
Tareas	▪ Caso de uso crear torneo ▪ Caso de uso unirse a torneo ▪ Caso de uso iniciar torneo ▪ Pruebas de los casos de uso implementados ▪ Correciones de posibles errores en el diseño
Días	14 días
Fecha Inicio	21/05/2020
Fecha Fin	03/06/2020

Cuadro 3.16: Iteración 7

Iteración	**8**
Iteración relativa	3
Fase	Construcción
Tareas	▪ Casos de uso Fold, Call, Check, Bet y Raise ▪ Pruebas de los casos de uso implementados ▪ Correciones de posibles errores en el diseño
Días	14 días
Fecha Inicio	04/06/2020
Fecha Fin	17/06/2020

Cuadro 3.17: Iteración 8

Iteración	**9**
Iteración relativa	4
Fase	Construcción
Tareas	▪ Caso de uso aumentar ciegas ▪ Caso de uso determinar mano ganadora ▪ Caso de uso finalizar torneo
Días	14 días
Fecha Inicio	18/06/2020
Fecha Fin	01/07/2020

Cuadro 3.18: Iteración 9

Iteración	**10**
Iteración relativa	1
Fase	Transición
Tareas	▪ Pruebas unitarias ▪ Correciones de posibles errores en el diseño
Días	7 días
Fecha Inicio	02/07/2020
Fecha Fin	08/07/2020

Cuadro 3.19: Iteración 10

Iteración	**11**
Iteración relativa	2
Fase	Transición
Tareas	▪ Manual de instalación
Días	7 días
Fecha Inicio	09/07/2020
Fecha Fin	15/07/2020

Cuadro 3.20: Iteración 11

3.2.3. Diagrama de Gantt

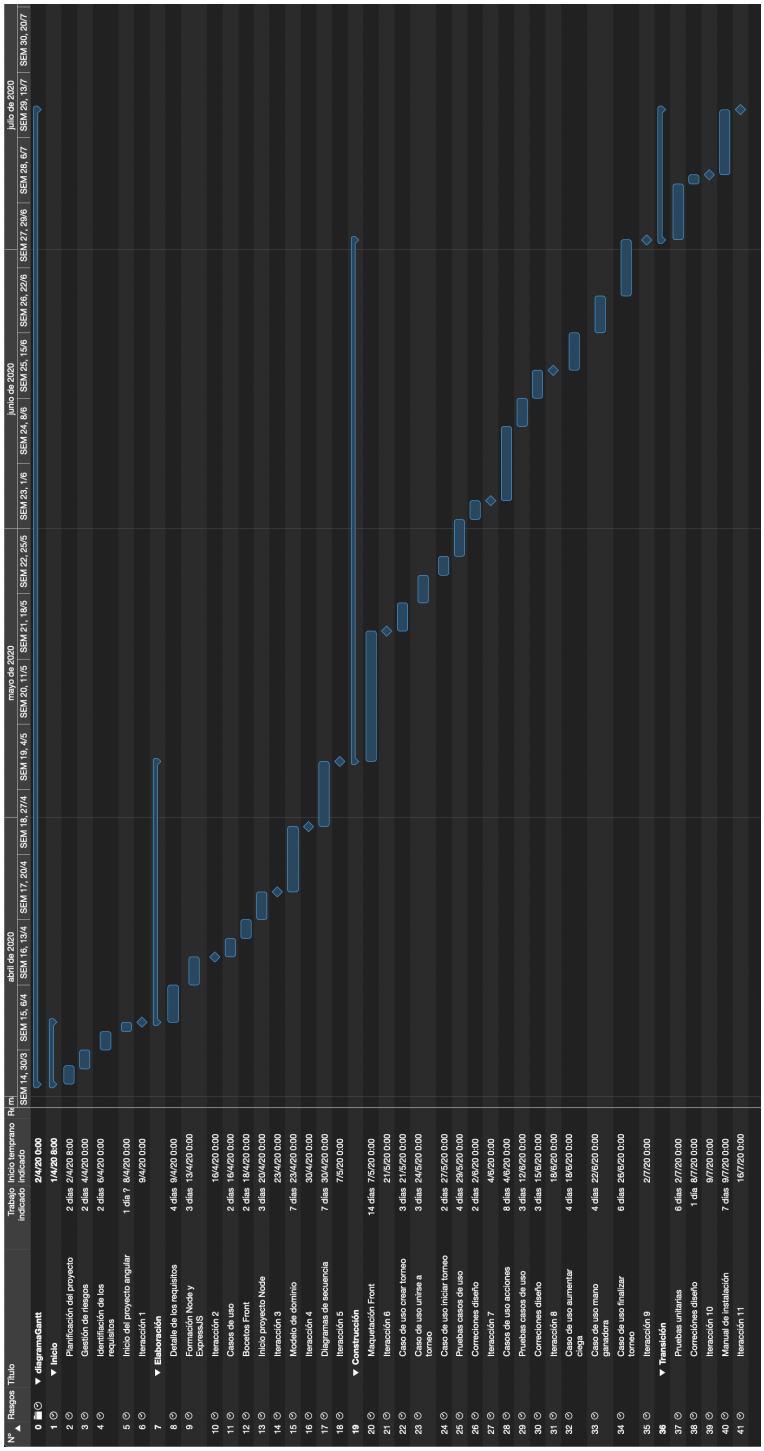

Figura 3.2: Diagrama de Gantt del proyecto

3.3. Estimación de costes

Una vez terminada la planificación del proyecto, explicadas sus fases y desglosadas sus iteraciones, podemos pasar a estimar los costes totales del proyecto.

Se ha estimado que el proyecto se realizará en 15 semanas, dedicando 20 horas en cada una nos daría un total de 300 horas totales. Contamos con que los costes humanos van a ser calculados como si de un titulado en ingeniería informática se tratara. El sueldo medio de un trabajador de este tipo en España es de unos 21.000 euros brutos al año, por lo que se estima que el sueldo a la hora será de unos 12 euros. Multiplicándolo por el total de horas del proyecto saldrían 3600 euros en gastos para el desarrollador.

Además de los costes humanos también hay que tener en cuenta los costes materiales. En nuestro caso no han supuesto un coste para el proyecto debido a que el trabajo se desarrolla desde casa y los gastos fijos en luz y agua corren a cargo de entidades externas al proyecto. En cuanto al gasto en software y herramientas utilizadas, todo lo utilizado ha sido de uso libre o bien se contaba con licencia previa.

Para los diagramas se ha utilizado la licencia de Astah que proporciona la uva. GitHub, Angular, Node... son herramientas gratuitas. Para el diseño de las cartas se ha utilizado una librería con licencia libre de uso. Para el maquetado de la aplicación se ha utilizado Marvel, la cual tiene una sección gratuita

Los costes materiales que si se van a tener en cuenta son los atribuidos al hardware utilizado y la conexión a internet. El valor del ordenador es de 1700 euros y se estima que sus años de vida útiles son 15, con esto sacamos que su amortización anual es de 141 euros al año, 11,75 euros al mes. El coste mensual por la conexión a internet es de 10 euros.

Teniendo en cuenta que la duración total del proyecto va a ser de 4 meses, sacamos que la suma total de todos los costes mencionados anteriormente es de 3687 euros.

Recurso	**Coste**
Conexión a internet	40 euros
Amortización del hardware	47 euros
Equipo de desarrollo	3600 euros
Total	**3687 euros**

Cuadro 3.21: Estimación de costes

3.4. Seguimiento del proyecto

A continuación veremos como ha sido el resultado de la planificación del proyecto, si los plazos han sido cumplidos y si ha surgido alguno de los riesgos mencionados anteriormente.

Desde el inicio del proyecto se contempló que uno de los mayores riesgos y el que más podía causar el retraso de todo el proyecto iba a ser la contratación del desarrollador por parte de una empresa. Esto haría que bajase considerablemente el número de horas diarias empleadas y con ello el aumento del número de días y semanas necesarias para finalizar el proyecto.

Las primeras semanas transcurrieron con normalidad, se emplearon el número de horas fijadas, se cumplieron los plazos fijados en las iteraciones y se obtuvieron los documentos fijados en cada final de iteración. Hasta el final de la quinta iteración, que es cuando el desarrollador fué contratado por una empresa externa, justo coincidiendo con el inicio de la fase de construcción anteriormente explicada.

Ante la aparición de este riesgo se optó por realizar su plan de contingencia.
Teniendo en cuenta que ahora no se iban a poder cumplir los plazos fijados ni se tendría una fecha límite para poder terminarlo como eran las convocatorias ordinaria y extraordinaria, se reorganizó el tiempo empleado en el proyecto de tal manera que fuera viable para el desarrollador, al trabajar por la mañana hasta media tarde y realizar el proyecto por la tarde noche.
Las 20 horas semanales pasaron a 5 horas semanales, dejando consigo una nueva tabla de iteraciones del proyecto y un nuevo diagrama de Gantt:

Fase	**Semanas**	**Fecha Incio**	**Fecha Fin**	**Iteraciones**
Inicio	1	02/04/2020	08/04/2020	1
Elaboración	4	09/04/2020	06/05/2020	4
Construcción	22	07/05/2020	07/10/2020	4
Transición	8	08/10/2020	01/12/2020	2

Cuadro 3.22: Iteraciones seguimiento del proyecto

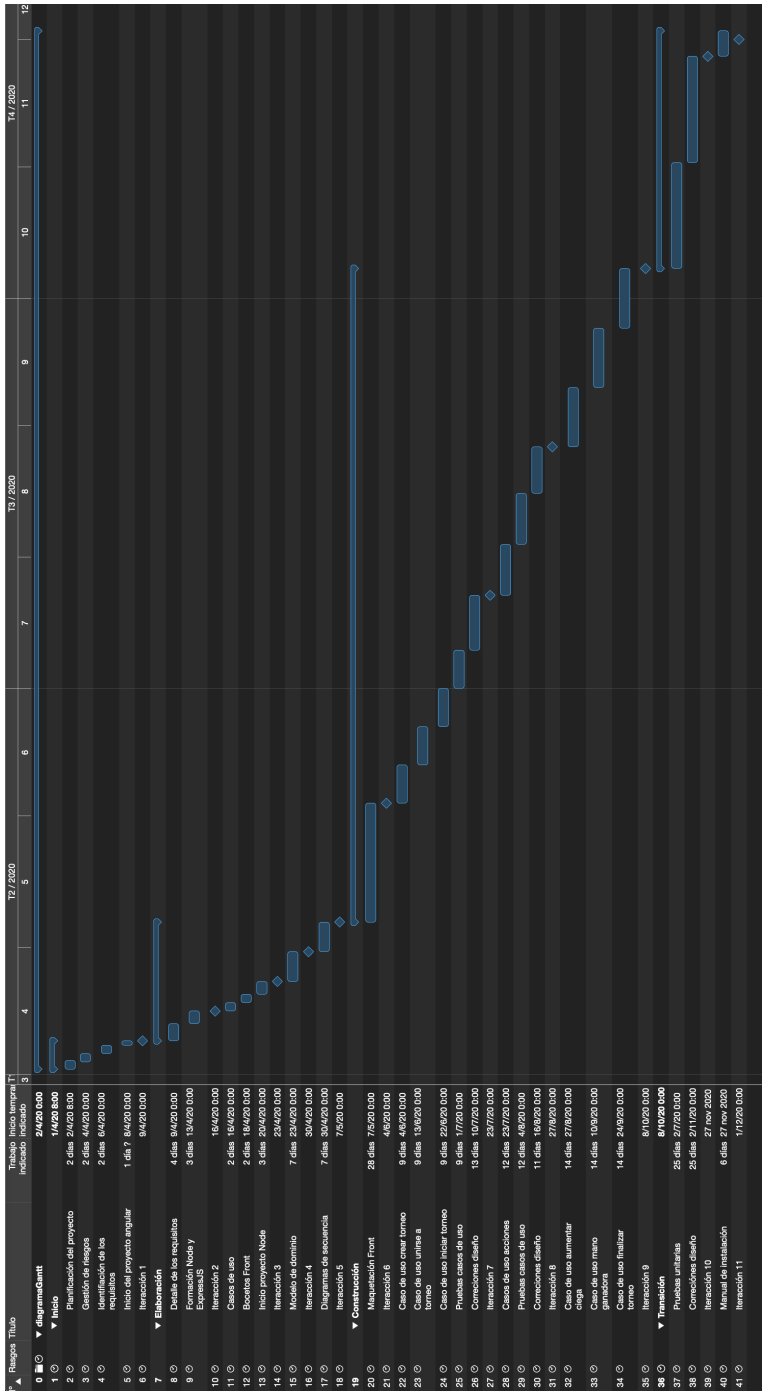

Figura 3.3: Diagrama de Gantt final del proyecto

Capítulo 4

Análisis de la aplicación

4.1. Requisitos

4.1.1. Requisitos Funcionales

ID	Nombre	Descripción
RF-01	Crear torneo	El sistema deberá permitir al usuario crear un torneo seleccionando el stack inicial y la velocidad del mismo
RF-02	Unirse a torneo	El sistema deberá permitir al usuario unirse a un torneo creado
RF-03	Iniciar torneo	El sistema permitir al jugador 1 iniciar el torneo pulsando en el botón Start
RF-04	Fold	El sistema deberá permitir al usuario deshechar sus cartas
RF-05	Check	El sistema deberá permitir al usuario pasar su turno
RF-06	Call	El sistema deberá permitir al usuario igualar la máxima apuesta
RF-07	Bet	El sistema deberá permitir al usuario apostar parte de su stack si no hay ninguna apuesta previa
RF-08	Raise	El sistema deberá permitir al usuario apostar parte de su stack para superar la apuesta previa

RF-09	Finalizar torneo	El sistema deberá finalizar el torneo cuando solo haya un jugador con fichas
RF-10	Modificar fichas	El sistema deberá aumentar o reducir las fichas de los jugadores según las vayan ganando o apostando
RF-11	Aumentar ciegas	El sistema deberá aumentar las ciegas gradualmente, según la velocidad del torneo
RF-12	Feedback de acciones	El sistema deberá registrar y mostrar mediante texto las acciones realizadas por cada jugador
RF-13	Feedback de ganador(es)	El sistema deberá mostrar mediante texto el ganador o ganadores de cada ronda
RF-14	Determinar mano ganadora	El sistema deberá determinar cual es la mano ganadora de entre todas las de los jugadores

Cuadro 4.1: Requisitos funcionales

4.1.2. Requisitos No Funcionales

Identificador	Nombre	Descripción
RNF-01	Usabilidad	La aplicación será fácil de usar. Permitiendo que hasta un 95 % de usuarios la sepan utilizar desde el primer momento que la utilicen
RNF-02	Idioma	La aplicación estará disponible en castellano
RNF-03	Acceso a los datos	El sistema gesitionará el acceso a los datos mediante MongoDB
RNF-04	Interfaz de usuario	El sistema presentará una interfaz de usuario basada en Angular 2+
RNF-05	Arquitectura	El sistema estará diseñado siguiendo el patrón MVC
RNF-06	Tiempo de respuesta	El sistema responderá en menos de 1 segunda a las acciones del jugador
RNF-07	Formato UTF-8	El sistema utilizará el formato de caracteres UTF-8

Cuadro 4.2: Requisitos no funcionales

4.1.3. Requisitos de Información

Identificador	Nombre	Descripción
RDI-01	Nombre del jugador	El sistema deberá almacenar el nombre del jugador
RDI-02	Stack inicial	El sistema deberá almacenar el stack inicial introducido por el jugador
RDI-03	Velocidad del torneo	El sistema deberá almacenar la velocidad selecionada para el torneo de entre las siguientes: Normal, Turbo y HyperTurbo

Cuadro 4.3: Requisitos de información

4.2. Casos de uso

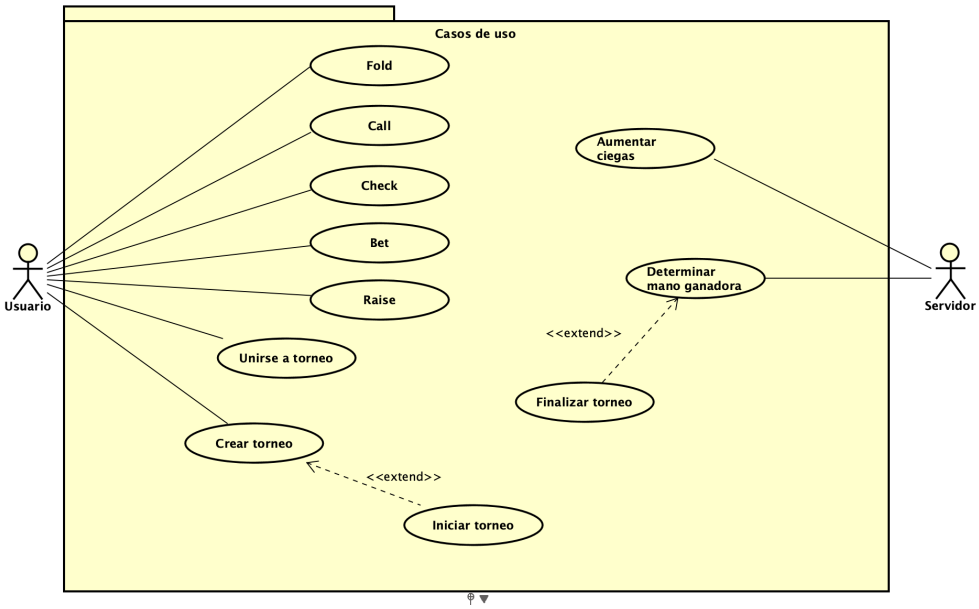

Figura 4.1: Casos de uso

4.2.1. Especificación de los casos de uso

CU-01	**Crear torneo**
Descripción	Crear un torneo de póker
Actor	Usuario
Precondicion	El usuario está en la aplicación y ha clickado en el botón 'Create Game'
Flujo normal	1. El usuario introduce su nombre, el stack inicial del torneo y selecciona su velocidad entre 'Normal', 'Turbo' e 'Hyperturbo' y pulsa el botón 'Create' 2. El sistema envía la petición al servidor, este crea el torneo, crea el usuario y añade al usuario al mismo
Postcondiciones	El usuario ha creado un torneo y es llevado al lobby del mismo
Flujo alternativo	

Cuadro 4.4: CU-01 Crear torneo

CU-02	**Unirse a torneo**
Descripción	Unirse a un torneo de poker previamente creado
Actor	Usuario
Precondiciones	El usuario está en la aplicación y ha clickado en el botón 'Join game'
Flujo normal	1. El usuario introduce su nombre, el código del torneo y pulsa el botón 'Join' 2. El sistema envía la petición al servidor, este crea el usuario y lo añade al torneo
Postcondiciones	El usuario se une al lobby del torneo
Flujo alternativo	

Cuadro 4.5: CU-02 Unirse a torneo

CU-03	**Iniciar torneo**
Descripción	Iniciar un torneo previamente creado
Actor	Usuario
Precondiciones	Se ha realizado el CU-01
Flujo normal	1. El usuario pulsa en el botón 'Start' 2. El sistema envía la petición al servidor, este crea el sistema de ciegas e inicia el torneo 3. El sistema coloca a cada usuario en su posición en la mesa
Postcondiciones	El torneo se inicia
Flujo alternativo	

Cuadro 4.6: CU-03 Iniciar torneo

CU-04	**Fold**
Descripción	Descartar tu mano
Actor	Usuario
Precondiciones	Es el turno del usuario y tiene cartas
Flujo normal	1. El usuario pulsa en el botón 'Fold' 2. El sistema envía la petición al servidor, este le quita el turno al usuario para toda la mano, le desasigna sus cartas y se lo comunica al sistema 3. El sistema deja de mostrar las cartas del usuario
Postcondiciones	El usuario deja de participar en la mano
Flujo alternativo	▪ 2b El sistema envía la petición al servidor, este le quita el turno al usuario para toda la mano, detecta que solo queda 1 jugador más en la mano, finaliza la mano y se lo comunica al sistema ▪ 3b El sistema muestra quien es el ganador de la mano

Cuadro 4.7: CU-04 Fold

CU-05	**Check**
Descripción	Pasar turno sin aumentar la apuesta
Actor	Usuario
Precondiciones	▪ La apuesta más alta de la mesa y la apuesta del usuario son iguales ▪ Es el turno del usuario
Flujo normal	1. El usuario pulsa en el botón 'Check' 2. El sistema envía la petición al servidor, este le quita el turno al usuario, se lo asigna al siguiente jugador y se lo comunica al sistema 3. El sistema muestra a los demás jugadores que el usuario ha checkeado
Postcondiciones	El usuario pasa el turno y continúa en la mano
Flujo alternativo	▪ 2b El sistema envía la petición al servidor, este le quita el turno al usuario, detecta que es el último jugador que quedaba por jugar de la ronda, pasa a la siguiente ronda y se lo comunica al sistema ▪ 3b El sistema muestra a los demás jugadores que el usuario ha checkeado y muestra las cartas de la siguiente ronda

Cuadro 4.8: CU-05 Check

CU-06	**Call**
Descripción	Igualar la máxima apuesta
Actor	Usuario
Precondiciones	▪ El usuario tiene cartas ▪ Es el turno del usuario ▪ El usuario no ha igualado la máxima apuesta
Flujo normal	1. El usuario pulsa en el botón 'Call' 2. El sistema envía la petición al servidor, este le quita el turno al usuario, se lo asigna al siguiente jugador, registra la apuesta y se lo comunica al sistema 3. El sistema muestra a los demás jugadores que el usuario ha igualado la máxima apuesta
Postcondiciones	▪ El usuario pierde el turno ▪ El stack del usuario se reduce la cantidad apostada
Flujo alternativo	▪ 2b El sistema envía la petición al servidor, este detecta que la acción del jugador le deja en posición de 'All-in' y le quita el turno para el resto de la mano. Se lo comunica al sistema ▪ 3b El sistema muestra a los demás jugadores que el usuario esta en posición de 'All in'

Cuadro 4.9: CU-06 Call

CU-07	**Bet**
Descripción	Apostar una cantidad de fichas
Actor	Usuario
Precondiciones	▪ El usuario tiene cartas ▪ Es el turno del usuario ▪ No hay apuestas previas en la mesa
Flujo normal	1. El usuario selecciona la cantidad a apostar y pulsa en el botón 'Bet' 2. El sistema envía la petición al servidor, este le quita el turno al usuario, se lo asigna al siguiente jugador, registra la apuesta y se lo comunica al sistema 3. El sistema muestra a los demás jugadores que el usuario ha checkeado y la cantidad que ha apostado
Postcondiciones	▪ El usuario pierde el turno ▪ El stack del usuario se reduce la cantidad apostada ▪ La apuesta máxima de la ronda aumenta
Flujo alternativo	

Cuadro 4.10: CU-07 Bet

CU-08	**Raise**
Descripción	Aumentar la máxima apuesta de la ronda
Actor	Usuario
Precondiciones	▪ El usuario tiene cartas ▪ Es el turno del usuario ▪ Hay al menos una apuesta previa en la mesa
Flujo normal	1. El usuario selecciona la cantidad a apostar y pulsa en el botón 'Raise' 2. El sistema envía la petición al servidor, este le quita el turno al usuario, se lo asigna al siguiente jugador, registra la apuesta, aumenta la apuesta máxima de la mesa y se lo comunica al sistema 3. El sistema muestra a los demás jugadores que el usuario ha hecho raise y la cantidad de fichas que ha aumentado
Postcondiciones	▪ El usuario pierde el turno ▪ El stack del usuario se reduce la cantidad apostada ▪ La apuesta máxima de la ronda aumenta
Flujo alternativo	▪ 2b El sistema envía la petición al servidor, este detecta que la acción del jugador le deja en posición de 'All-in' y le quita el turno para el resto de la mano. Se lo comunica al sistema ▪ 3b El sistema muestra a los demás jugadores que el usuario esta en posición de 'All in'

Cuadro 4.11: CU-08 Raise

CU-09	**Aumentar ciegas**
Descripción	Las ciegas grande y pequeña van aumentando a medida que transcurre el torneo
Actor	Servidor
Precondiciones	Hay un torneo en curso
Flujo normal	1. Al finalizar cada mano, el sistema comprueba cuanto tiempo ha transcurrido desde que se inició el torneo y establece las ciegas acorde la cantidad que aparezca en la tabla de ciegas
Postcondiciones	▪ La ciga grande aumenta ▪ La ciega pequeña aumenta
Flujo alternativo	

Cuadro 4.12: CU-09 Aumentar ciegas

CU-10	**Determinar mano ganadora**
Descripción	Determinar quien ha ganado la mano
Actor	Servidor
Precondiciones	Se ha realizado CU-04 o CU-05 o CU-06 o CU-07
Flujo normal	1. El servidor detecta que está en fase de final de mano, se lo comunica al sistema y espera 10 segundos para terminar la mano 2. El sistema muestra tanto las cartas de la mesa como las de los jugadores que han llegado showdown 3. El sistema termina su espera de 10 segundos, da el bote al ganador o ganadores, restablece la apuesta máxima, restablece la baraja, restablece las ciegas, calcula la posición del nuevo dealer, da comienzo a la siguiente mano y se lo comunica al sistema. 4. El sistema muestra a los jugadores las acciones llevadas acabo por el servidor
Postcondiciones	Da cominezo una nueva mano
Flujo alternativo	▪ 3b El servidor detecta que ya solo queda 1 jugador con fichas, da comienzo el CU-11

Cuadro 4.13: CU-10 Determinar mano ganadora

CU-11	**Finalizar torneo**
Descripción	Finalizar el torneo
Actor	Servidor
Precondiciones	Se ha realizado CU-10
Flujo normal	1. El servidor detecta que solamente queda 1 jugador con fichas en el torneo, le da como ganador, finaliza el torneo y se lo comunica al sistema 2. El sistema muestra a los jugadores quien es el ganador del torneo
Postcondiciones	Finaliza el torneo
Flujo alternativo	

Cuadro 4.14: CU-11 Finalizar torneo

4.3. Modelo de dominio

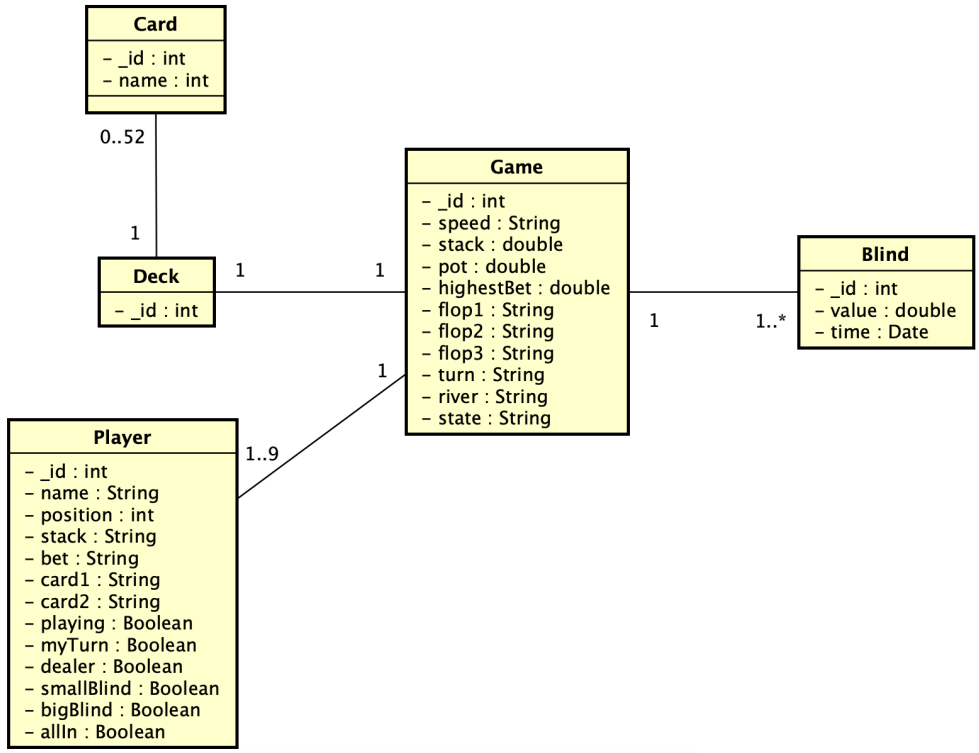

Figura 4.2: Modelo de dominio

4.4. Diagramas de secuencia

A continuación se muestran unos diagramas de secuencia con el objetivo de enseñar el funcionamiento de la aplicación. Los dos primeros representan la conexión de la aplicación cliente directamente con la base de datos y los dos últimos la conexión mediante sockets entre el cliente y el servidor.

Para simplificar la arquitectura de la aplicación se ha optado por la conexión entre cliente y base de datos de la forma "tradicional", sin pasar por los sockets, para los casos de uso más simples como ya son Crear un torneo o Unirse a un torneo

4.4.1. Crear torneo

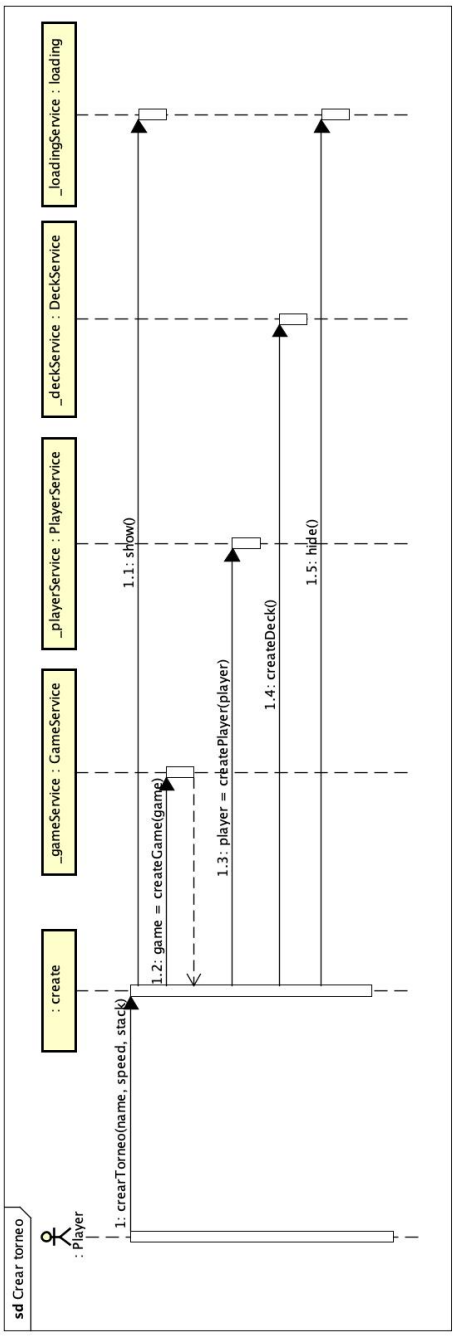

Figura 4.3: Diagrama de secuencia: Crear torneo

4.4.2. Unirse a torneo

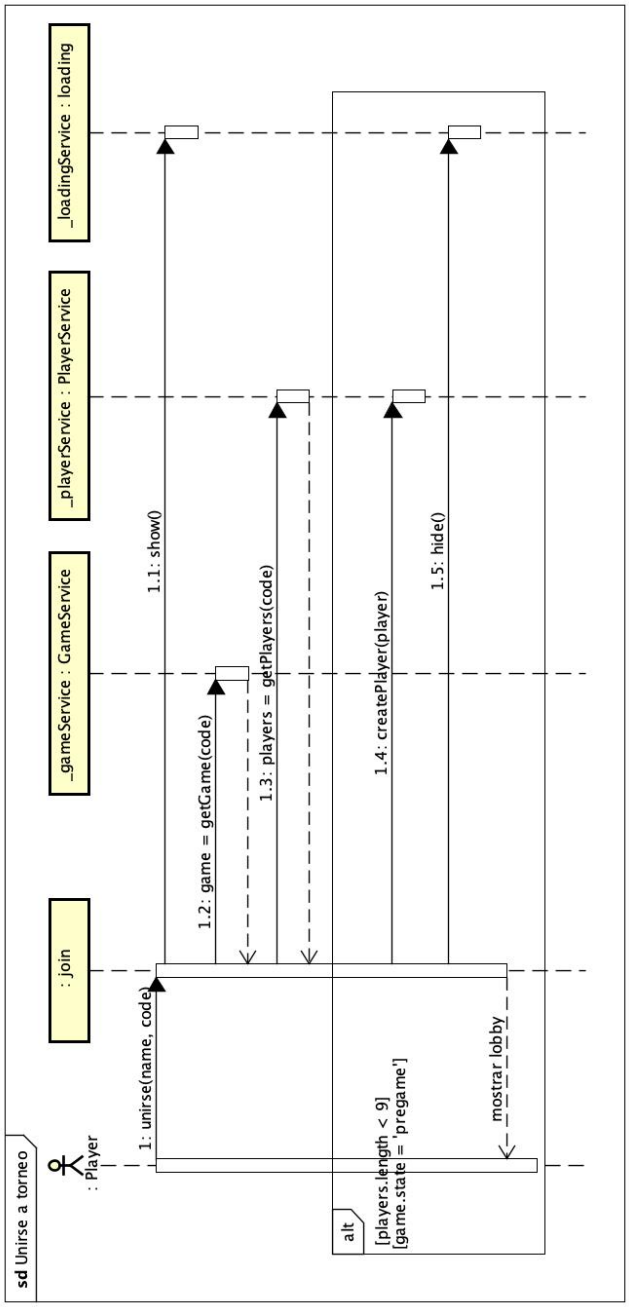

Figura 4.4: Diagrama de secuencia: Unirse a torneo

4.4.3. Iniciar torneo en el cliente

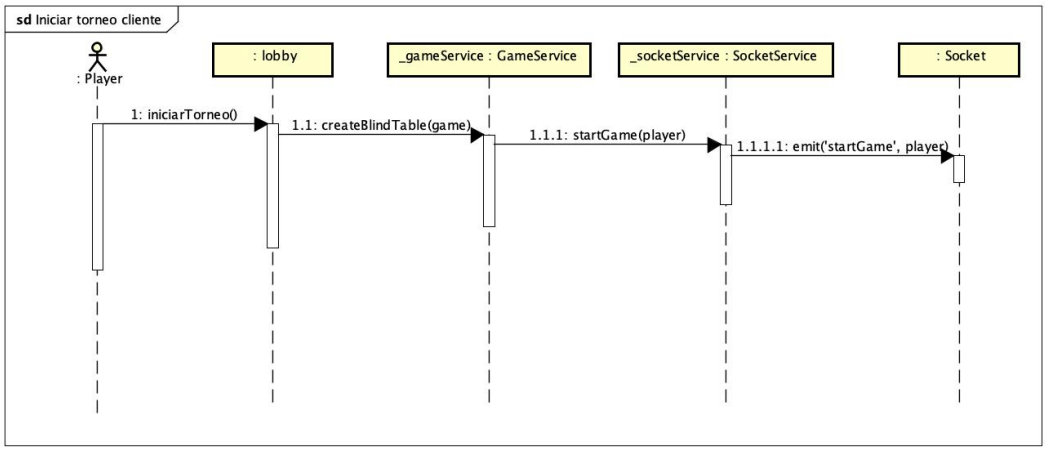

Figura 4.5: Diagrama de secuencia: Iniciar torneo en el cliente

4.4.4. Iniciar torneo en el servidor

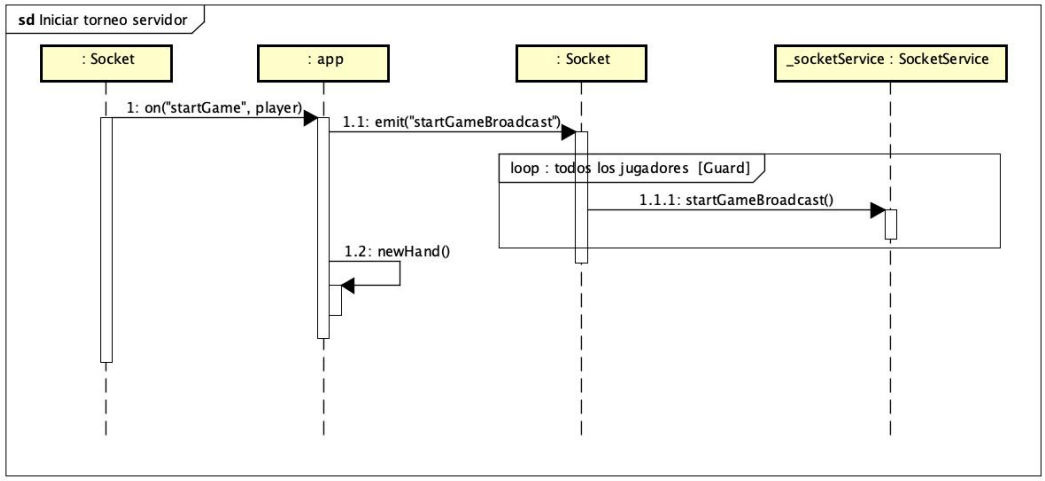

Figura 4.6: Diagrama de secuencia: Iniciar torneo en el servidor

4.4.5. Fold en el cliente

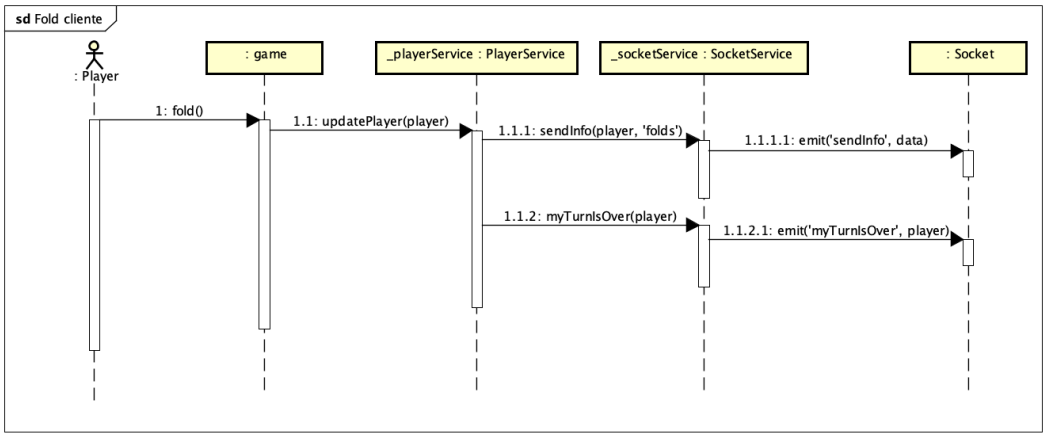

Figura 4.7: Diagrama de secuencia: Fold en el cliente

4.4.6. Fold en el servidor

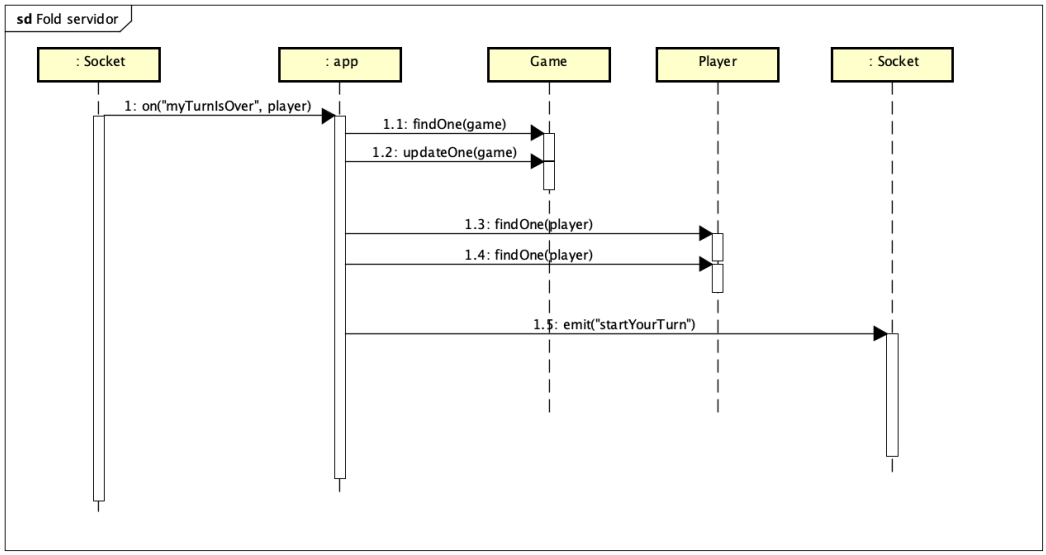

Figura 4.8: Diagrama de secuencia: Fold en el servidor

Explicados los tres tipos de diagramas de la aplicación:

- Cliente se comunica con el servidor para modificar la base de datos sin utilizar los socket.
- El cliente se comunica con el servidor mediante sockets para que este actualice la información de los demás clientes.
- El cliente se comunica con el servidor para que este además de informar a los demás clientes de que el estado ha cambiado, modifique la base de datos.

Los demás casos de uso se realizarán de la misma manera, ya que coincidirán con uno de estos tres.

Capítulo 5

Tecnologías utilizadas

Para el desarrollo de esta aplicación web se ha optado por utilizar MEAN [5], un framework o conjunto de subsistemas de software para el desarrollo de aplicaciones y páginas web dinámicas, estas basadas en JavaScript. El nombre MEAN es acrónimo formado por las iniciales de MongoDB, ExpressJS, Angular y NodeJS. Gracias al conjunto de estas cuatro tecnologías se pueden crear aplicaciones utilizando Javascript en todas las capas.

5.1. Tecnologías frontend

5.1.1. Angular

Angular [8] es un framework para aplicaciones web desarrollado en Typescript, un superconjunto de Javascript el cual añade tipado y clases de objetos. Los principales motivos para elegir este framework en lugar de otros como VueJS o ReactJS han sido la robustez que presenta, al estar mantenido por Google y la experiencia previa en el mismo.

Figura 5.1: Logo de Angular

5.2. Tecnologías backend

5.2.1. MongoDB

MongoDB [11] es un sistema de base de datos NoSQL, almacena los datos en documentos, los cuales están definidos con notación JSON (JavaScript Object Notation), esto favorece en gran medida la inclusión con otras tecnologías que utilicen o esten basadas en JavaScript. Los principales motivos que me han impulsado a utilizar una base de datos NoSQL en lugar de una relacional han sido: la flexibilidad que tienen, mayor rendimiento frente a las relacionales, carácter descentralizado. Además se ha elegido MongoDB por la posible desmesurada cantidad de datos que tendrá que albergar el sistema.

Figura 5.2: Logo de MongoDB

5.2.2. NodeJS

NodeJS [9] es un entorno de código abierto que permite el desarrolo de aplicaciones JavaScript del lado del servidor. Su arquitectura para la entrada y salida de datos es asíncrona, es decir, sin bloqueos. Además nos proporciona una gran biblioteca con módulos JavaScript

Figura 5.3: Logo de NodeJS

5.2.3. ExpressJS

ExpressJS [10] es un framework para NodeJS. Está orientado en el desarrollo de aplicaciones web y APIs. Gracias a ExpressJS se pueden manejar las peticiones y solicitudes que se hacen por medio de los métodos GET, POST, PUT y DELETE. Además ofrece el sistema de enrutamiento para el backend.

Figura 5.4: Logo de ExpressJS

Capítulo 6

Diseño de la aplicación

6.1. Diseño de la base de datos

Para la base de datos se ha utilizado MongoDB, una base de datos NoSQL, es decir, no relacional. Esto implica que no existen relaciones entre una tabla y otra y por tanto tampoco claves foráneas ni primarias.
MongoDB relaciona sus objetos de la base de datos por medio de referencias. Estas referencias son atributos del objeto de la base de datos que bien pueden representar únicamente su *_id* por defecto o directamente todo el objeto haciendo uso del método *populate* que implementa la librería mongoose.

En la siguiente imagen podemos ver cómo son estas referencias:

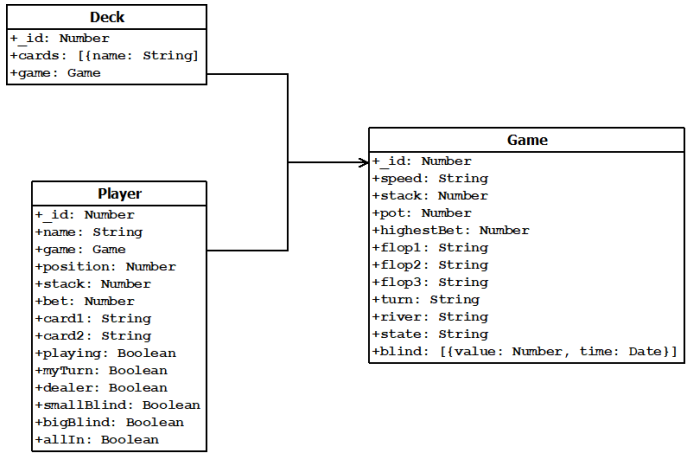

Figura 6.1: Diagrama de diseño de la base de datos

Tanto el objeto Player como el Deck tienen una referencia al Game que pertenece. Con una sola consulta a la base de datos podemos traer de vuelta en forma de JSON la información de un Player junto con el *_id* del Game al que pertenece o podemos traernos el objeto Game entero directamente.

Además, podemos observar tanto en Deck como en Game que tienen los atributos cards y blinds en los cuales aparece como tipo de atributo un objeto.
Esto son los *subdocuments* de mongoose, documentos anidados dentro de otro documento.

En el caso de Deck, el atributo cards representa un conjunto de objetos con un único atributo name, el nombre de la carta en formato String.
En el caso de Game, blind representa un conjunto de objetos con los atributos value y time, la tabla de ciegas que tiene cada torneo.

6.2. Arquitectura

6.2.1. Despliegue

En cuanto al despliegue, en la parte del servidor se ha utilizado la base de datos NoSQL basada en documentos MongoDB, para la lógica se ha hecho uso de NodeJS junto con su framework de aplicaciones web ExpressJS.

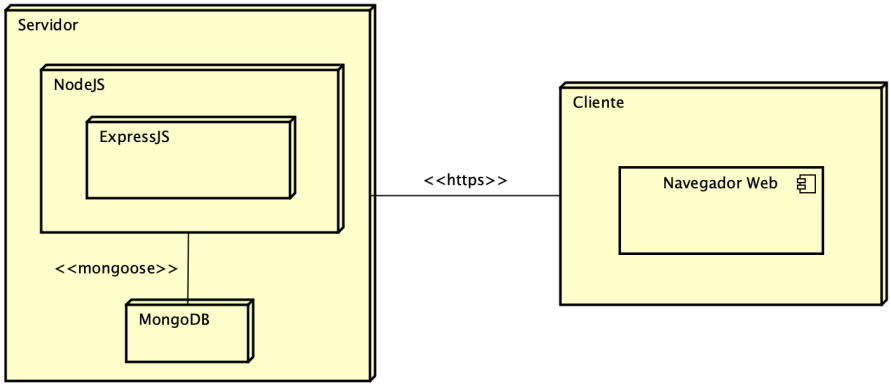

Figura 6.2: Diagrama de despliegue

6.2.2. Arquitectura del cliente

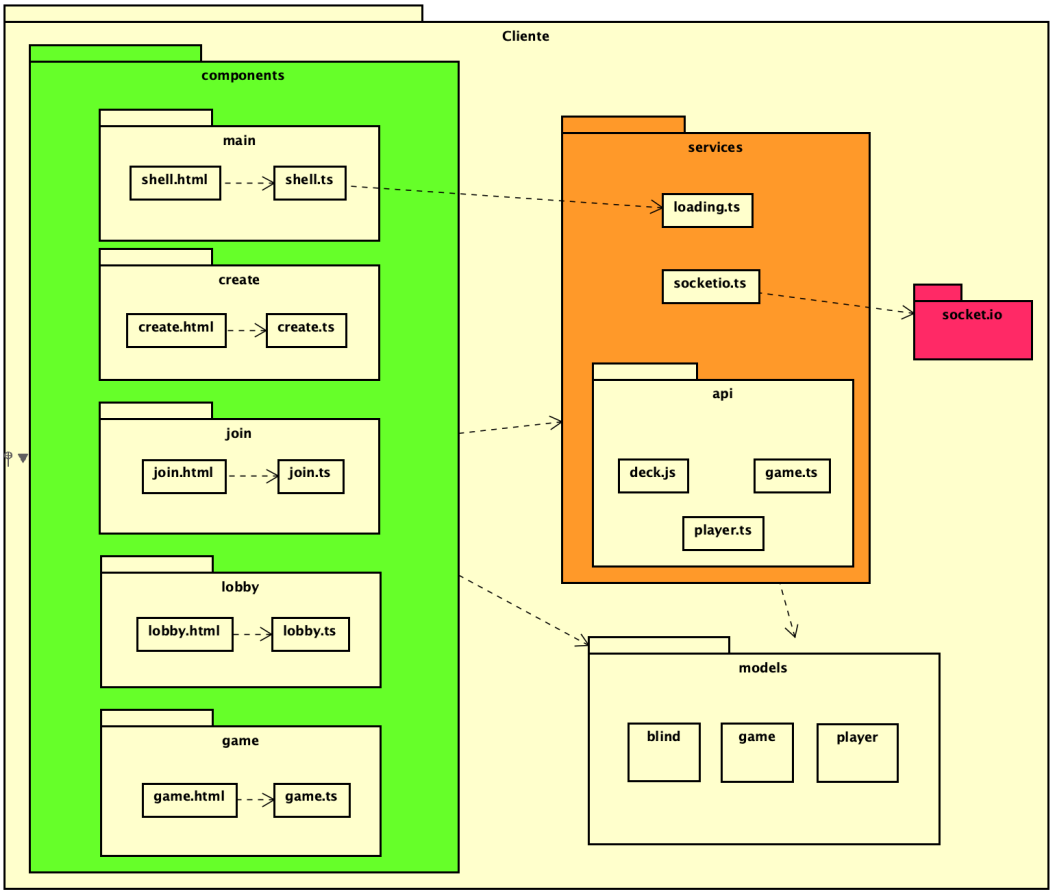

Figura 6.3: Arquitectura del cliente

6.2.3. Arquitectura del servidor

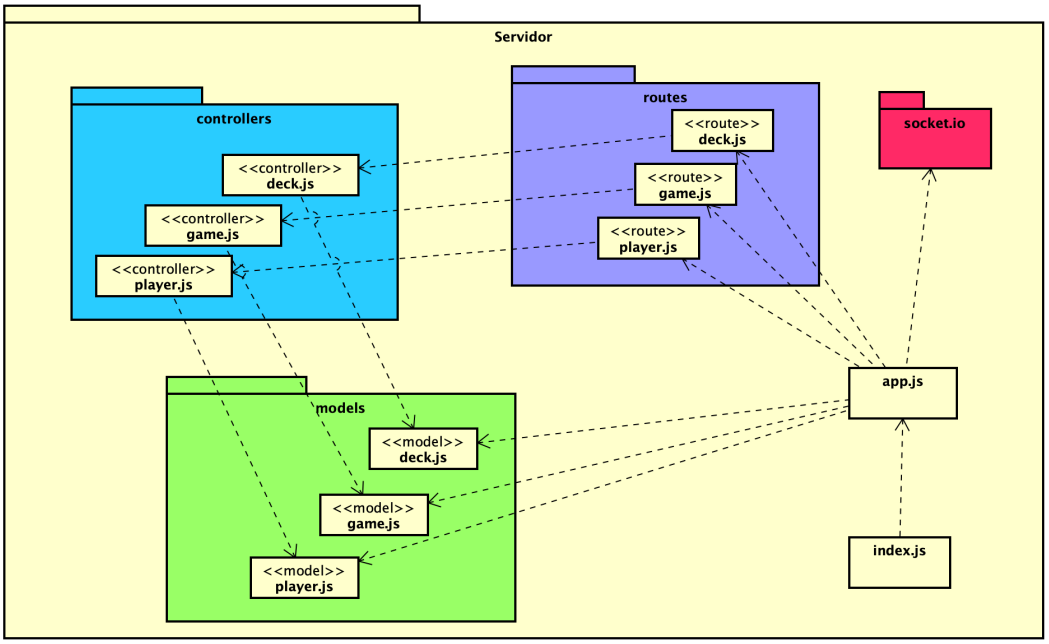

Figura 6.4: Arquitectura del servidor

6.3. Patrones de diseño

Cuando encuentras un problema en tu software, lo más probable es que mucha gente antes que tú tambien se haya encontrado de cara con el mismo problema y haya dado una solución. Si esta solución se puede extraer, explicar y reutilizar entonces es que estamos ante un patrón de diseño.

Existen muchos patrones de diseño software, cada uno orientado a resolver un problema específico. Para el desarrollo de este proyecto se han utilizado los siguientes:

6.3.1. Patrón MVVM

Angular sigue el modelo *two way data binding*, en el modelo podemos modificar la vista y en la vista podemos modificar el modelo, algo que hace a modelo y vista totalmente dependientes el uno del otro. La independencia que crea el patrón de diseño Modelo Vista Controlador (MVC) no la podemos conseguir, por ello se llama Modelo-Vista Vista-Modelo (MVVM)

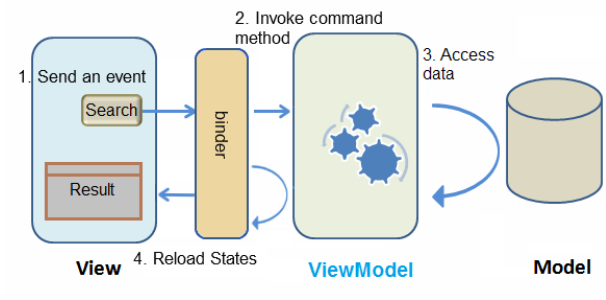

Figura 6.5: Patrón de diseño MVVM

Con el patrón MVVM desacoplamos los datos de la aplicación de la interfaz de usuario

Vista

Define como se va a representar la información gráficamente. En el caso de Angular, la vista serían los archivos HTML y CSS

Modelo

Representa el modelo de dominio de la aplicación. En Angular está alojada en los archivos typescript asociados a cada html y css, y sería el conjunto de variables y constantes declarados dentro de estos. Además de los datos globales de la aplicación.

Vista-Modelo

Comunica el Modelo con la Vista y contiene la lógica de presentación de la Vista. Serían las funciones localizadas en los archivos typescript, responsables de las actualizaciones de vista o modelo.

6.3.2. Patrón Singleton

El patrón Singleton establece que solamente debe de haber una instancia de un objeto a la vez y debe de ser accesible a los clientes desde un punto de acceso conocido.

En nuestro caso, cuando dos componentes hacen uso de un mismo servicio sería muy mala idea tener dos instancias diferentes de ese servicio, en cambia se utilizan los servicios compartidos de Angular. Clases de typescript en las que se agrupan funciones comunes.

Se ha utilizado el patrón Singleton en los siguiente puntos de la aplicación:

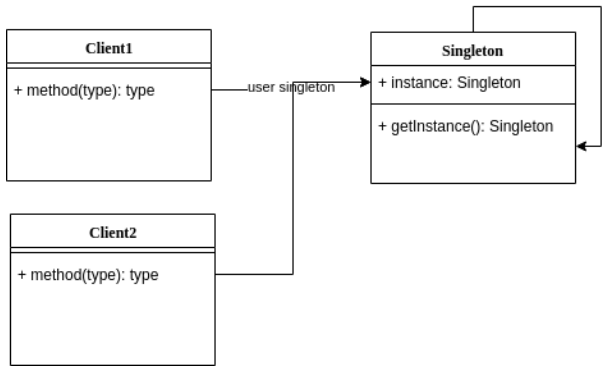

Figura 6.6: Patrón de diseño Singleton

Servicio Loading

Un servicio para activar o desactivar el spinner que aparece en medio de la pantalla e indicar que la aplicación esta cargando

Servico API

Para hacer la conexión con la parte back, separado en la API para la baraja de cartas (deck), la partida (game) y el jugador (player). Cada una con un objeto HTTPClient encargado de hacer las consultas HTTP.

Servicio Socket

Contiene toda la lógica de la conexión mediante sockets entre el servidor y los usuarios. Las clases utilizan este servicio para emitir o recibir mensajes procedentes del back o repetidos por este.

6.3.3. Patrón Estado

El patrón Estado es utilizado cuando el comportamiento de un objeto cambia en función del estado de este.

Este patrón se ha utilizado para los objetos Game, los cuales tienen una propiedad que hacen referencia a la fase en la que se encuentra la partida (preflop, flop, turn, river, showdown) y según esta fase la lógica es distinta. También se puede ver el patrón en los Players, pueden estar jugando, no jugando, en estado de all-in o no, puede ser su turno o según la posición en la que estén se realizará una acción u otra.

6.3.4. Patrón Observador

El patron Observador define una dependencia de uno a muchos entre objetos. Los observadores permanecen escuchando a la notificación de un cambio en el estado del sujeto.

En nuestro caso podemos encontrarlo en el funcionamiento de la librería javascript socket.io, cada cliente tiene abiertos diferentes canales de escucha pendientes de un cambio en el estado de los objetos.

Por ejemplo, cuando hay un cambio de fase en la mano, esta es notificada a todos los jugadores desde el servidor por medio del socket. Cuando los jugadores lo reciben actualizan esta información en su vista.

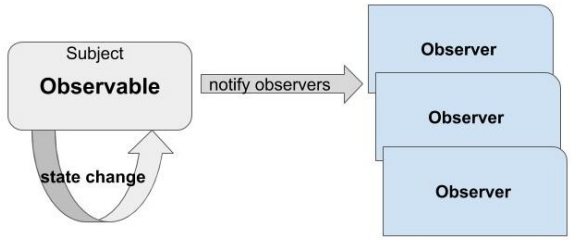

Figura 6.7: Patrón de diseño Observador

6.4. Bocetos

Para realizar los bocetos de la página web se ha utilizado la herramienta Marvel Design, una plataforma de diseño de prototipos tanto web como móvil que cuenta con un apartado gratuito del cual nos hemos beneficiado.

La idea principal de la interfaz de usuario ha sido hacerla lo más sencilla posible, a diferencia de los sitios de póker más famosos actualmente donde la interfaz cuenta con cientos de opciones y menús. Nuestra interfaz contará con poca información por pantalla para que incluso las personas menos experimentadas en el póker o que directamente están aprendiendo las normas sepan utilizarla y no se pierdan.

Cabe recordar que lo mostrado a continuación son bocetos simplificados del resultado final, el cual podrá variar y al cual se le añadirá contenido gráfico que mejore la apariencia de la página y la experiencia del jugador.

6.4.1. Pantalla principal

La pantalla principal lo más simple posible, el nombre de la aplicación y los botones de crear o unirse a torneo. Va a ser lo primero que vea un usuario nuevo, si de primeras hubiera muchas opciones esto podría hacer que el usuario se perdiera y abandonara la página

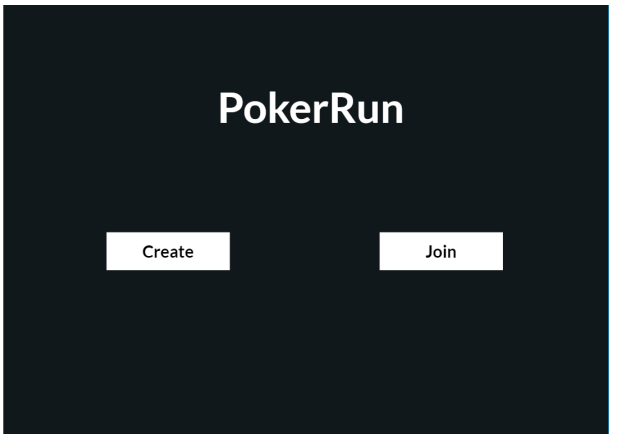

Figura 6.8: Boceto pantalla principal

6.4.2. Crear torneo

Al pulsar en el botón de crear se accederá a la pantalla de crear torneo, en la que podremos introducir nuestro nombre, la velocidad del torneo entre 'Normal', 'Turbo' o 'HyperTurbo' y el stack inicial con el que van a empezar los jugadores.

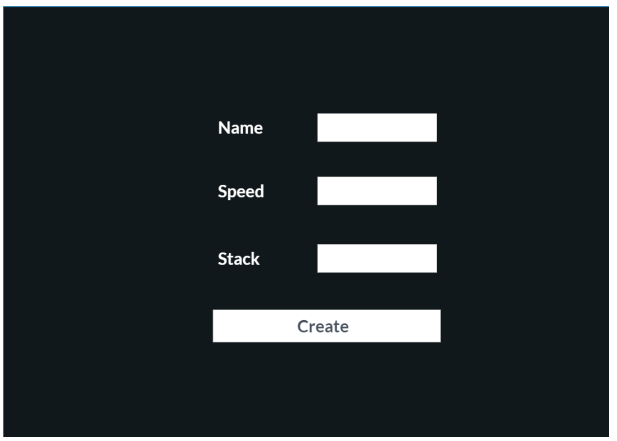

Figura 6.9: Boceto crear torneo

6.4.3. Unirse a torneo

Si en la pantalla pricipal en lugar de pulsar el botón de crear torneo pulsamos el de unirse a torneo accederemos a esta pantalla. Se nos pedirá un nombre y el código del torneo existente. Abajo estará el botón para unirnos a dicho torneo.

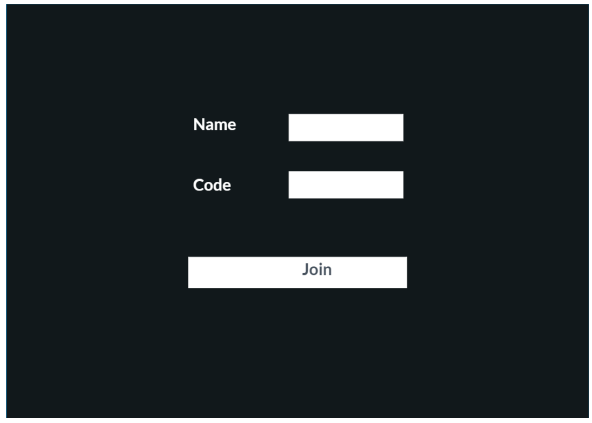

Figura 6.10: Boceto unirse a torneo

6.4.4. Lobby del torneo

Bien cuando creamos un torneo o cuando nos unimos a uno accederemos a esta pantalla. En ella se mostrarán los jugadores que se han unido, la información del torneo, el código del torneo para que más jugadores se puedan unir y el botón de empezar el torneo, el cual solo puede ver el jugador número 1.

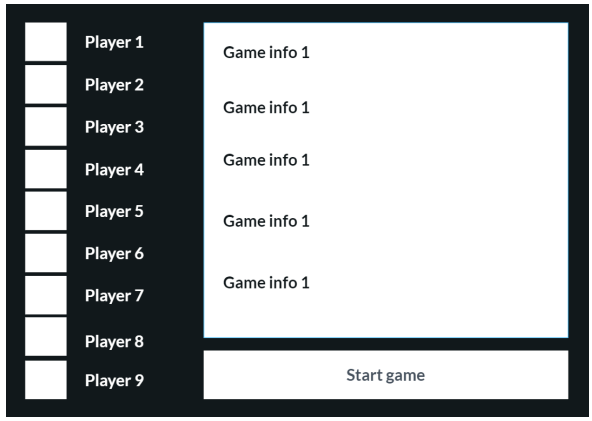

Figura 6.11: Boceto lobby del torneo

6.4.5. Pantalla del torneo

Cuando se inicia el torneo todos los jugadores pasarán a esta pantalla. En ella tendremos las acciones posibles como botones abajo.
En el centro la mesa, donde estarán dispuestos los jugadores alrededor de ella. Cada jugador tendrá a la izquierda el espacio de sus cartas y la derecha las fichas que poseen en ese momento. En el caso propio, las cartas estarán visibles.
En el centro de la mesa irán apareciendo las cartas del flop, turn y river respectivamente

Se prevee mostrar también las acciones que realiza cada jugador como mensaje de texto abajo a la izquierda de la pantalla y la estructura de las ciegas del torneo, en que nivel de ciegas se encuentra abajo a la derecha de la pantalla

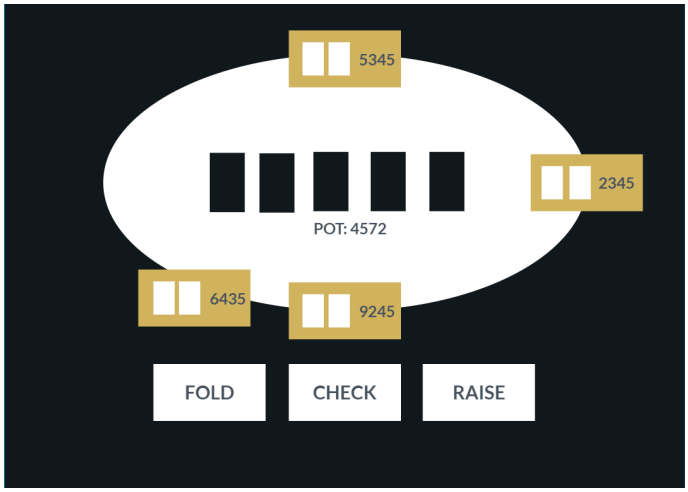

Figura 6.12: Boceto pantalla del torneo

Capítulo 7

Implementación

Este proyecto además de ser una aplicación web es un juego, esto conlleva a que la tarea de programarlo, aun siendo una de las partes fundamentales sino la más importante, no sirva de nada si la interfaz y la experiencia de juego no son atractivas para los jugadores. A continuación se describirán las librerias utilizadas para conseguir tanto una programación de codigo como una interfaz y experiencia de juego óptimas.

7.1. Socket.io

Socket.io es una libreria JavaScript que nos permite la creación de aplicaciones web en tiempo real. Y, ¿qué nos permite esto? Nos permite una comunicación bidireccional entre cliente y servidor, ambas partes están continuamente escuchando. Todo esto lo hace sobre un único socket TCP.

7.1.1. Conexión

Lo primero de todo, socket.io necesita ser instalado tanto en cliente como en servidor, hay una versión para cada parte pero la API es prácticamente la misma. Una vez instalada, en el servidor ligamos el servidor http que hemos creado a la variable que importa socket.io y en el cliente apuntamos al puerto en el que esté desplegado el servidor http. Por último para que el servidor escuche cuando se realiza una conexion, lo conseguiremos con la función 'on' con el parámetro 'connection'.

7.1.2. Envio de mensajes

Para el envio de mensajes entre cliente y servidor de utiliza el siguiente sistema, igual para ambos lados sea quien sea el emisor/receptor:

- socket.emit('Nombre identificativo', mensajeEmitido)
- socket.on('Nombre identificativo', mensajeRecibido)

Para la aplicación se han utilizado los siguientes:

Nombre	**playerConnection**
Descripción	Detecta que un nuevo jugador se ha conectado
Emisor	Cliente
Receptor	Servidor

Cuadro 7.1: Mensaje socket: playerConnection

Nombre	**playerConnectionBroadcast**
Descripción	Comunica a todos los clientes que se ha conectado un nuevo jugador
Emisor	Servidor
Receptor	Cliente

Cuadro 7.2: Mensaje socket: playerConnectionBroadcast

Nombre	**startGame**
Descripción	Comunica al servidor que el jugador 1 ha comenzado el torneo
Emisor	Cliente
Receptor	Servidor

Cuadro 7.3: Mensaje socket: startGame

Nombre	**startGameBroadcast**
Descripción	Comunica a los clientes que el torneo empieza
Emisor	Servidor
Receptor	Cliente

Cuadro 7.4: Mensaje socket: startGameBroadcast

Nombre	**startYourTurn**
Descripción	Comunica al cliente que es su turno
Emisor	Servidor
Receptor	Cliente

Cuadro 7.5: Mensaje socket: startYourTurn

Nombre	**myTurnIsOver**
Descripción	Comunica al servidor que el cliente ha acabado su turno
Emisor	Cliente
Receptor	Servidor

Cuadro 7.6: Mensaje socket: myTurnIsOver

Nombre	**showdown**
Descripción	Comunica al cliente que se ha llegado a la fase de 'showdown'
Emisor	Servidor
Receptor	Cliente

Cuadro 7.7: Mensaje socket: showdown

Nombre	**sendInfo**
Descripción	Comunica al servidor la jugada que ha hecho el cliente
Emisor	Cliente
Receptor	Servidor

Cuadro 7.8: Mensaje socket: sendInfo

Nombre	**getInfo**
Descripción	Comunica al cliente la jugada que ha hecho un jugador
Emisor	Servidor
Receptor	Cliente

Cuadro 7.9: Mensaje socket: getInfo

7.1.3. Rooms

Las 'rooms' o salas de socket.io son canales a los que un socket puede entrar o salirse y por los que un socket puede enviar y recibir mensajes exclusivamente de los demás sockets que estén la misma sala. Para la aplicación las salas han sido creadas mediante el identificador único de torneo, por lo que todos los jugadores se unen a dicha sala y asi se evitan colisiones de mensajes con las demás partidas.

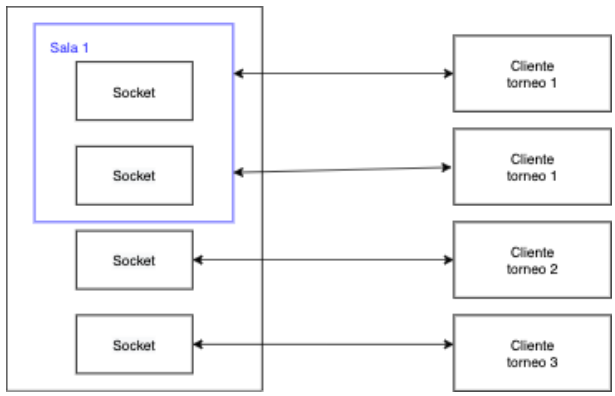

Figura 7.1: Salas de socket.io

7.2. Pokersolver

Pokersolver [26] es una librearía escrita en JavaScript que permite comparar manos de Poker y decidir el ganador o ganadores. No es necesario que estén las 5 cartas sobre la mesa, puede comparar manos desde 2 hasta 7 cartas. Además de esto, puede devolver la descripción exacta de cual ha sido la mano ganadora, como por ejemplo: Two Pair, A's 8's, que indicaría que el ganador tiene dobles parejas de ases y ochos.

7.3. Ngx-flip

Librería JavaScript utilizada para la animación de voltear y mostrar las cartas.

7.4. SVG-cards

Se ha utilizado la librería SVG-cards para dar diseño tanto a la cara como al torso de las cartas de la baraja francesa. La principal ventaja que he encontrado de esta librería en comparación con otras que por ejemplo directamente son imágenes de las caras, es que esta ha utilizado SVG, lo cual nos permite modificar a nuestro antojo tanto el tamaño como el color de los elementos que forman la carta.

Figura 7.2: Cartas de SVG-cards

7.5. Angular Material

La librería de componentes web por excelencia de Angular. Para este proyecto se ha hecho concretamente de los componentes: mat-slider para seleccionar la cantidad de fichas a apostar

y mat-dialog para el pop-up que avisa del final de partida. Para el resto de los elementos visuales que salen en la aplicación no se ha utilizado ninguna otra librería.

7.6. Estructura de ciegas

Todos los torneos de póker tienen asociada una estructura de ciegas mediante una tabla de ciegas. Dicha tabla de ciegas suele tener los siguiente campos:

- **Nivel**: nivel del torneo en el que nos encontramos, los niveles van aumentando conforme a un tiempo preestablecido que suele variar de 2 minutos para los torneos más rapidos a 20 minutos en los más lentos.
- **Tamaño de la ciega**: Cantidad a la que se establece el valor de la ciega.
- **Ante**: Valor de la apuesta obligatoria que tienen que hacer todos los jugadores de la mesa en cada mano. El ante es una apuesta obligatoria pero utilizarla para un toreno es algo opcional, da más dinamismo a la partida.

Para los torneos reales, las salas de póker crean una estructura de ciegas acorde a la velocidad del torneo que se desea y a los jugadores que van a jugarlo.
Para nuestro proyecto es algo más complejo, tanto el número de jugadores como el stack inicial es algo variable y seleccionable en cada torneo, por ello se ha automatizado esta estructura de ciegas acorde a las opciones introducidas.

Para ello nuestra tabla va a tener dos campos:

- **Tiempo**: Cuando se inicia un torneo, se establece como primer valor la hora en la que se ha iniciado, a partir de ahí los siguientes valores van a ir aumentando según la velocidad del torneo selecionada en 5, 10 y 15 minutos para *Normal*, *Turbo* e *HyperTurbo* respectivamente.
- **Ciega**: Para las ciegas se va a partir de con un valor de 1/100 del stack inicial, a partir de ahí el tamaño va a ir aumentando según un parámetro que se ha llamado *blindIncreaser*, el cual inicialmente va a ser 1/200 del stack inicial pero que cada 3 aumentos de ciega se va a doblar su valor. De esta forma se va a conseguir que el aumento de ciegas sea algo exponencial.

Cuando finaliza una mano, según la hora que sea se comprueba en esta tabla cual es el valor de la ciega que se tiene que establecer. La tabla tiene un límite, cuando el valor de la ciega sea mayor que el número de fichas del torneo.

Capítulo 8

Pruebas

Durante el ciclo de vida de un desarrollo de software van a surgir errores que imposibilitarán el correcto funcionamiento de este. El mejor escenario sería que todos estos errores fueran detectados durante las fases de creación del proyecto, pero también hay errores que no son detectados hasta que la aplicación es lanzada al mercado. Debido a esto es necesario dedicar una fase entera únicamente a la correción y detección de errores.

Dentro del desarrollo de un proyecto, la realización de las pruebas es una de las partes más importantes. Constituye la parte central del control de calidad y ayuda a determinar el correcto funcionamiento del código. Además previene la aparición de nuevos errores y aumenta el nivel de confianza hacía el propio software.

Vamos a realizar dos tipos diferentes de pruebas: pruebas unitarias, para comprobar el correcto funcionamiento de partes de código de manera independiente; y pruebas de integración, una vez hayamos pasado las unitarias, para comprobar el funcionamiento de los componentes en conjunto.

8.1. Pruebas unitarias

PU-01	**Crear partida**
Descripción	Pulsar en crear partida con los campos nombre, velocidad y stack inicial completados
Resultado	OK
Resultado esperado	La partida se crea en la base de datos y el jugador es llevado a la pantalla del lobby

Cuadro 8.1: PU-01: Crear partida

PU-02	**Crear partida sin introducir algún campo**
Descripción	Pulsar en crear partida con alguno de los campos sin completar
Resultado	OK
Resultado esperado	La partida no se crea debido a que son campos obligatorios

Cuadro 8.2: PU-02: Crear partida sin introducir algún campo

PU-03	**Crear partida introduciendo un valor incorrecto como stack**
Descripción	Pulsar en crear partida introduciendo letras o un valor negativo en el campo stack
Resultado	OK
Resultado esperado	La partida no se crea debido a que el valor introducido es erróneo

Cuadro 8.3: PU-03: Crear partida introduciendo un valor incorrecto como stack

PU-04	**Unirse a partida**
Descripción	Pulsar en unirse a partida tras haber introducido los campos nombre y código de la partida
Resultado	OK
Resultado esperado	El jugador es añadido a la partida y pasa a la pantalla del lobby

Cuadro 8.4: PU-04: Unirse a partida

PU-05	**Unirse a partida sin introducir algún campo**
Descripción	Dar a unirse a partida sin haber introducido los campos nombre o código de partida
Resultado	OK
Resultado esperado	El jugador no se une a partida, debido a que el campo es obligatorio

Cuadro 8.5: PU-05: Unirse a partida sin introducir algún campo

PU-06	**Unirse a partida introduciendo un código de partida no válido**
Descripción	Dar a unirse a partida introduciendo un código de partida erróneo
Resultado	OK
Resultado esperado	El jugador no se une a partida, debido a que no se ecuentra ninguna partida con ese código

Cuadro 8.6: PU-06: Unirse a partida introduciendo un código de partida no válido

PU-07	**Recolocación de jugadores al abandonar el lobby**
Descripción	Cuando un jugador abandona el lobby, todos lo demás jugadores son recolocados en sus nuevas posiciones
Resultado	OK
Resultado esperado	Los jugadores que están por debajo de él en posición, aumentan todos una posición

Cuadro 8.7: PU-07: Recolocación de jugadores al abandonar el lobby

PU-08	**Empezar partida**
Descripción	El jugador número uno da a empezar partida
Resultado	OK
Resultado esperado	Todos los jugadores pasan a la pantalla del torneo, cada uno con el stack incial

Cuadro 8.8: PU-08: Recolocación de jugadores al abandonar el lobby

PU-09	**Fold**
Descripción	Un jugador pulsa en Fold
Resultado	OK
Resultado esperado	Su mano es descartada

Cuadro 8.9: PU-09: Fold

PU-10	**Check**
Descripción	Un jugador pulsa en Check
Resultado	OK
Resultado esperado	El jugador pasa de turno y continúa en la partida

Cuadro 8.10: PU-10: Check

PU-11	**Call**
Descripción	Un jugador pulsa en Call
Resultado	OK
Resultado esperado	El jugador iguala la máxima apuesta de la mesa

Cuadro 8.11: PU-11: Fold

PU-12	**Bet**
Descripción	Un jugador pulsa en Bet
Resultado	OK
Resultado esperado	El jugador apuesta la cantidad indicada

Cuadro 8.12: PU-12: Bet

PU-13	**Raise**
Descripción	Un jugador pulsa en Raise
Resultado	OK
Resultado esperado	El jugador aumenta la apuesta en la cantidad indicada

Cuadro 8.13: PU-13: Raise

PU-14	**Cambio de turno**
Descripción	El turno se cambia correctamente
Resultado	OK
Resultado esperado	Independientemente de la acción realizada, el turno pasa al siguiente jugador que esté jugando la mano

Cuadro 8.14: PU-14: Cambio de turno

PU-15	**Aumento de ciegas**
Descripción	Las ciegas aumentan automáticamente a lo largo de la partida
Resultado	OK
Resultado esperado	Cuanda hayan pasado x minutos, las ciegas aumentarán según el valor que haya en la tabla de ciegas

Cuadro 8.15: PU-15: Aumento de ciegas

PU-16	**Interfaz y posición de las ciegas y el botón**
Descripción	Finaliza una mano y se pasa a la siguiente
Resultado	OK
Resultado esperado	Las posiciones de las ciegas y el botón se recolocan correctamente en función de las agujas del reloj teniendo en cuenta que puede haber jugadores que ya han sido eliminados

Cuadro 8.16: PU-16: Interfaz y posición de las ciegas y el botón

PU-17	**Jugador eliminado**
Descripción	Un jugador pierde todas sus fichas
Resultado	OK
Resultado esperado	Este jugador pasa a ser un espectador, no se le reparten cartas, no se le da turno y las posiciones de ciegas y botón le saltan

Cuadro 8.17: PU-17: Jugador eliminado

PU-18	**Un jugador en all-in**
Descripción	Un jugador apuesta todas sus fichas, hace un all-in
Resultado	OK
Resultado esperado	El jugador no podrá apostar más fichas y se le pasará el turno hasta el momento del showdown a menos que todos los demás jugadores se retiresn

Cuadro 8.18: PU-18: Un jugador en all-in

PU-19	**Todos los jugadores en all-in**
Descripción	Todos los jugadores que siguen en la mano van all-in
Resultado	OK
Resultado esperado	La mano muestra todas las cartas que faltan por salir y pasa directamente a estado de showdown

Cuadro 8.19: PU-19: Todos los jugadores en all-in

PU-17	**Stack menor que una ciega pequeña**
Descripción	Un jugador se queda con un stack menor que una ciega pequeña
Resultado	OK
Resultado esperado	Si le toca poner ciega, pone las fichas que le quedan y automáticamente entra en all-in.

Cuadro 8.20: PU-17: Stack menor que una ciega pequeña

PU-18	**Orden de fases**
Descripción	Termina una fase de la mano
Resultado	OK
Resultado esperado	Cada mano sigue el orden de fases Preflop, Flop, Turn , River y Showdown. Si la mano termina antes de llegar al Showdown, se vuelve al Preflop

Cuadro 8.21: PU-18: Orden de fases

PU-19	**Fin de partida**
Descripción	Un jugador se hace con todas las fichas del torneo
Resultado	OK
Resultado esperado	Se indica que el jugador ha ganado a todos los jugadores y toda la información del torneo en base de datos es borrada

Cuadro 8.22: PU-19: Fin de partida

8.2. Pruebas de integración

PI-01	**Prueba de integración lobby**
Caso de prueba	▪ Jugador 1 crea partida ▪ Jugador 2 se une a la partida introduciendo el código ▪ Jugador 3 se une a la partida introduciendo el código ▪ Jugador 1 se va de la partida ▪ El jugador 2 pasa a estar en la posición 1 y el jugador 3 pasa a estar en la posisción 2 ▪ Jugador 1 se vuelve a unir a la partida y entra en la posición 3
Resultado	OK

Cuadro 8.23: PI-01: Prueba de integración lobby

PI-02	**Prueba de integración mano normal sin llegar al showdown**
Caso de prueba	▪ La ciega pequeña, la ciega grande y el botón son asignados a los jugadores 2, 3 y 1 respectivamente. ▪ El jugador 1 hace Bet de 1000 fichas ▪ El jugador 2 hace Fold ▪ El jugador 3 hace Fold ▪ Se indica que el jugador 1 ha ganado ▪ Se suma el pot al stack del jugador 1
Resultado	OK

Cuadro 8.24: PI-02: Prueba de integración mano normal sin llegar al showdown

PI-03	**Prueba de integración mano normal llegando al showdown**
Caso de prueba	▪ La ciega pequeña, la ciega grande y el botón son asignados a los jugadores 2, 3 y 1 respectivamente. ▪ El jugador 1 hace Call ▪ El jugador 2 hace Call ▪ El jugador 3 hace Check ▪ Se muestra el Flop en la mesa y juntan las apuestas en el pot ▪ El jugador 2 hace Bet de 200 fichas ▪ El jugador 3 hace Raise de 400 fichas ▪ El jugador 1 hace Fold ▪ El jugador 2 hace Call ▪ Se muestra el Turn en la mesa y se juntan las apuestas en el pot ▪ El jugador 2 hace Check ▪ El jugador 3 hace Check ▪ Se muestra el River en la mesa y se juntan las apuestas en el pot ▪ El jugador 2 hace Bet de 400 fichas ▪ El jugador 3 hace Call ▪ Se muestran las cartas de ambos jugadores y se indica que el jugador 2 ha ganado ▪ Se suma el pot al stack del jugador 2
Resultado	OK

Cuadro 8.25: PI-03: Prueba de integración mano normal llegando al showdown

PI-04	**Prueba de integración mano con all-in**
Caso de prueba	▪ La ciega pequeña, la ciega grande y el botón son asignados a los jugadores 2, 3 y 1 respectivamente. ▪ El jugador 1 hace Call ▪ El jugador 2 hace Call ▪ El jugador 3 hace Check ▪ Se muestra el Flop en la mesa y juntan las apuestas en el pot ▪ El jugador 2 hace Bet de 200 fichas ▪ El jugador 3 hace Raise de 400 fichas ▪ El jugador 1 hace all-in de 5000 fichas ▪ El jugador 2 hace Call ▪ El jugador 3 hace Call ▪ Se muestran el turn y el river en la mesa y se pasa ha estado de showdown ▪ Se muestran las cartas de los tres jugadores y se indica que el jugador 2 ha ganado ▪ Se suma el pot al stack del jugador 2
Resultado	OK

Cuadro 8.26: PI-04: Prueba de integración mano con all-in

Capítulo 9

Conclusiones y lineas futuras

9.1. Conclusiones

Una vez se acabado el proyecto, se puede decir que se han cumplido los objetivos marcados al inicio de este y que me siento muy satisfecho con el trabajo realizado.

Uno de los principales objetivos cumplidos ha sido el de ampliar los conocimientos en el framework Angular. Se tenian conociminetos previos sobre este framework, pero realmente se aprende cuando se realiza un proyecto real en el que anteriormente se marca lo que se desea hacer sin tener mucha idea de cómo se va a hacer, entonces te ves forzado a buscar información, 'pegarte' y aprender sobre ello.

También se han cumplido los objetivos de conocer y aplicar conceptos de la gestión de proyectos. Se ha podido comprobar de forma práctica lo importante que es tener una buena gestión de proyecto para poder realizar un desarrollo de calidad, pasando por todas las fases, desde el primer análisis de la aplicación hasta la última prueba de integración.

Ha mejorado en gran medida la autogesitión y organización. Primeramente se elaboró un plan de desarrollo del proyecto teniendo en cuenta los riesgos, uno de estos riesgos se materializó, lo que provocó la elaboración de un segundo plan de desarrollo para adaptarse a los nuevos inconvenientes.

Además, se ha podido realizar una aplicación web multijugador en tiempo real, una de las principales motivaciones para este proyecto. Lo realizado hasta ahora en la universidad habian sido páginas web prácticamente estáticas, idóneas para iniciarse en el desarrollo web, pero siempre habían estado ahí las ganas de embarcarse en el desarrollo de una más compleja que permitiera cambios en tiempo real.

En cuanto a los objetivos de proyecto se ha conseguido lo propuesto, suplir la necesidad por la que surge este proyecto, que era tener un sitio web que permita la realización de partidas de póker entre amigos.

9.2. Líneas futuras

Respecto a mejoras futuras que se podrían añadir se ha pensado en las siguientes:

- Chat entre los jugadores de la partida
- Modo espectador por si se unieran más personas que sitios hay en el torneo
- Torneos de varias mesas
- Sistema de encriptación para la baraja
- Desplegar el proyecto en un dominio

Apéndice A

Manual de instalación

A.1. Requisitos

- Repositorio del proyecto
- MondoDB Server v4.2+
- NodeJS v12+
- npm v6+
- Angular CLI v9+

A.2. Instalación

Esta aplicación ha sido ejecutada en un sistema macOS, aun así esta preparada para hacerlo tambien tanto en Linux como en Windows siguiendo las siguientes instrucciones:

A.2.1. Clonación del repositorio

Primero necesitaremos el código fuente del proyecto, para ello clonaremos el repositorio del proyecto que se encuentra en GitHub introduciendo el siguiente comando:

```
$ git clone https://github.com/ferzamo/tfg_ferzamo.git
```

A.2.2. Instalación de MongoDB

A continuación procederemos a instalar MongoDB en nuestro ordenador, para ello según nuestro sistema operativo tendremos que seguir las instrucciones proporcionadas por los desarrolladores de MongoDB en la documentación oficial:

- `https://docs.mongodb.com/manual/installation/`

Una vez instalado MongoDB, en el caso de Linux y macOS tendremos que iniciar el servicio.

Para macOS utilizaremos el siguiente comando:

```
$ brew services start mongodb-community@4.4
```

Para Linux utilizaremos el siguiente comando:

```
$ sudo systemctl start mongod
```

A.2.3. Instalación de NodeJS y npm

Para instalar NodeJS en nuestro sistema tenemos dos opciones, si tenemos Linux podremos instalarlo facilmente con la siguiente línea en la consola de comandos

```
$ sudo apt install nodejs
```

O podremos hacerlo con el instalador proporcionado en su página web, eligiendo el sistema operativo correspondiente:

- `https://nodejs.org/es/download/`

Al instalarse NodeJS tambien se instalará npm.

A.2.4. Instalación de Angular CLI

Una vez tenemos npm installado en nuestro ordenador procederemos a installar Angular CLI con el siguiente comando:

```
$ npm install -g @angular/cli
```

A.3. Ejecución

Antes de ejectuar el proyecto tendremos que instalar las depencias, tanto del servidor como de la aplicación Angular. Para ello tendremos que movernos a las carpetas */poker-back* y */poker-tfg* y ejectuar el siguiente comando:

```
$ npm install
```

Para ejecutar el proyecto, lo primero que tendremos que hacer será inicializar el servidor, para ello nos moveremos a la carpeta */poker-back* dentro del repositorio del proyecto y escribiremos el siguiente comando:

```
$ npm run start
```

Después tendremos que inicializar la parte front, para ello nos moveremos a la carpeta */poker-tfg* y escribiremos el siguiente comando:

```
$ ng serve
```

Esto inicializará el servidor de despliegue de angular en el puerto 4200 de nuestro ordenador. Ahora accediendo a *localhost:4200* podremos utilizar la aplicación.

Apéndice B

Manual de usuario

B.1. Pantalla principal

Lo primero que ve el usuario al entrar en la aplicación es el título y dos botones, el de crear un nuevo torneo y el de unirse a uno ya existente.
Se ha optado por esta sencillez en la pantalla principal para cumplir uno de los objetivos del proyecto, que era hacer la aplicación lo más fácil e intuitiva posible para el jugador.
De esta forma, simplificando la primera pantalla que ve el usuario no le sobrecargamos y ayudamos a que permanezca en la aplicación.

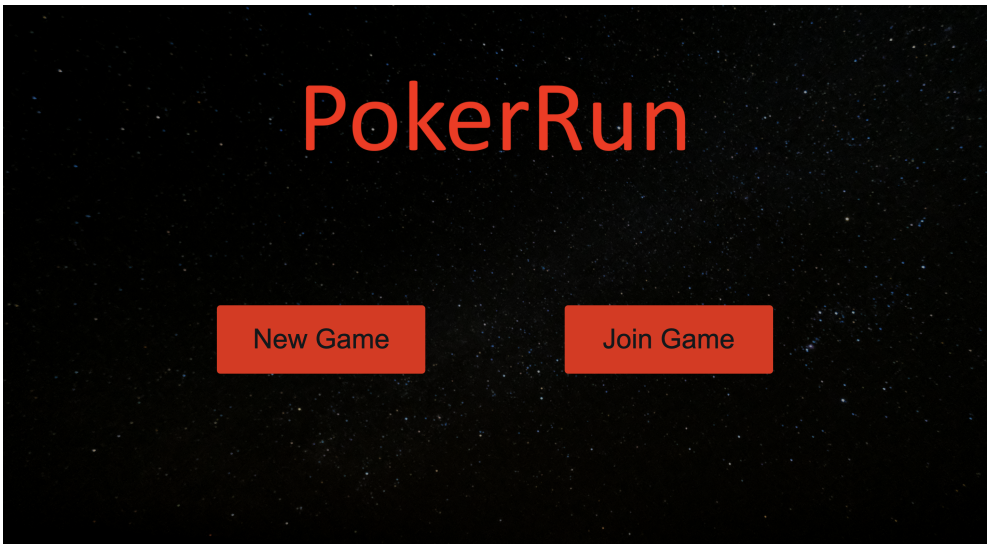

Figura B.1: Pantalla principal

B.2. Pantalla de creación de torneo

Si el usuario pulsa en el botón *New Game*, será llevado a la pantalla de crear un nuevo torneo donde podrá introducir su nombre en el apartado *Your name*, la velocidad del torneo entre Normal, Turbo o Hyperturbo en *Speed* y el stack inicial con el que empezarán todos los jugadores en el apartado *Stack*.
Una vez introducidos todos los campos pulsando en el botón *Create* creará la partida. Si no ha introducido todos los campos o si alguno de ellos no tiene el formato esperado, la partida no se creará.

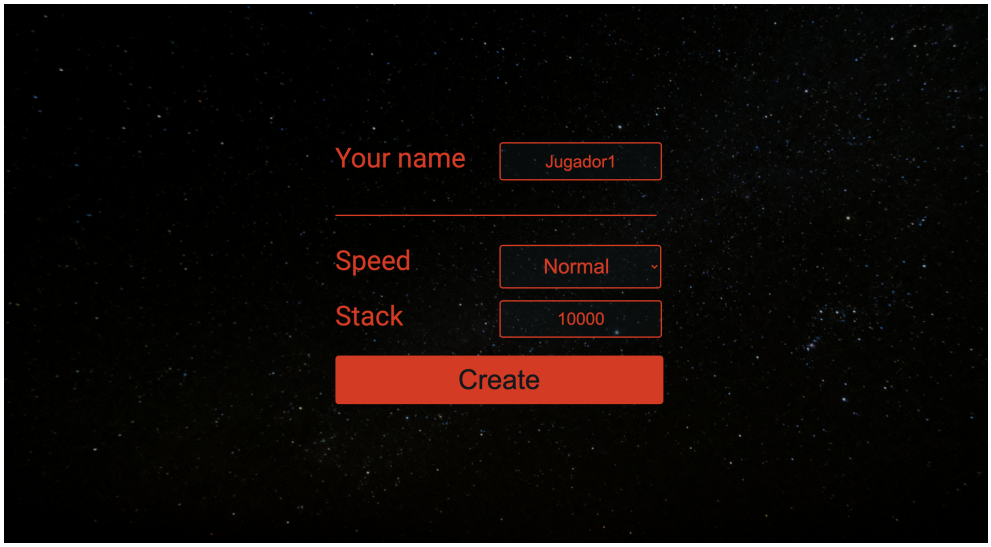

Figura B.2: Pantalla de creación de torneo

B.3. Pantalla de lobby del torneo con un jugador

Cuando el usuario cree un torneo accederá a la pantalla del lobby del torneo, en la que podrá ver en la columna de la izquierda los usuarios que están conectados, a la derecha arriba las características del torneo, modalidad de póker que es (en nuestro caso siempre va a ser Texas Hold'em Sin límite pero se ha añadido este campo para posible futuras modificaciones), el número de jugadores conectados hasta el momento, la velocidad del torneo y el stack con el que cada jugador comienza, seleccionados previamente en la pantalla de crear torneo. Abajo del torneo encontrará el botón de empezar la partida *Start*.

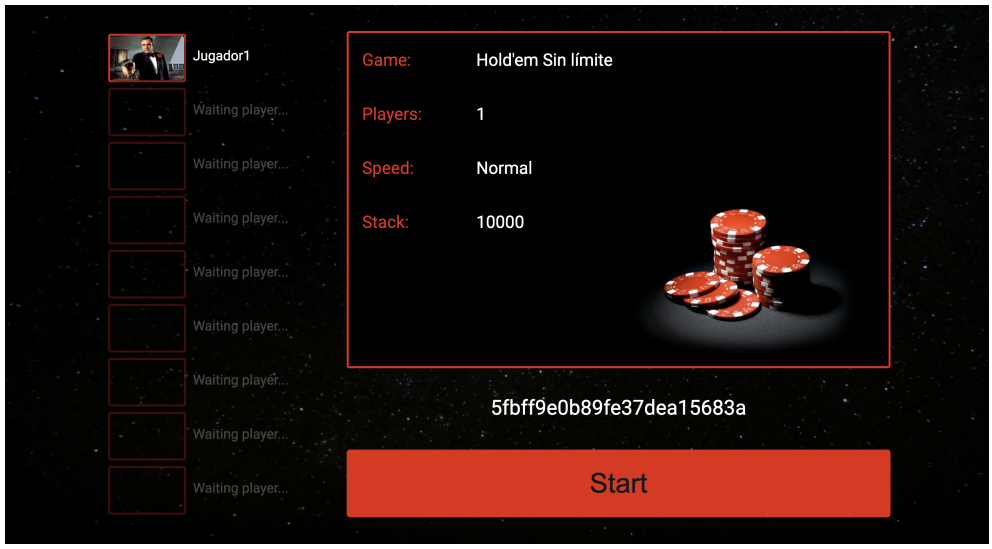

Figura B.3: Pantalla de lobby del torneo con un jugador

B.4. Pantalla de unirse a torneo

Si en vez de seleccionar *New Game* en la pantalla pricipal, el usuario selecciona *Join Game*, accederá a la pantalla de unirse a un torneo existente.
En ella encontrará dos campos para rellenar, su nombre en *Your name* y el código del torneo al que se quiere unir en *The game code*. Una vez rellenados los dos campos el usuario podrá unirse al torneo pulsando en el botón *Join*.
Si no rellena todos los campos o si el código del torneo no es válido, no se pasará a la siguiente pantalla.

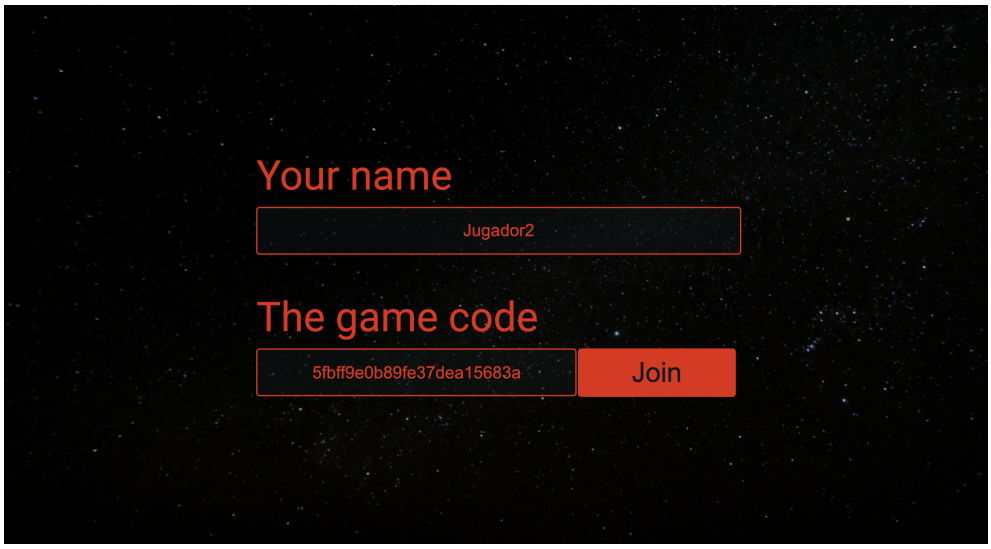

Figura B.4: Pantalla de unirse a torneo

B.5. Pantalla del lobby del torneo tras unirse

Una vez el usuario es introducido en el lobby del torneo, verá lo mismo que hemos comentado anteriormente, jugadores conectados y carácterísticas del torneo, pero en este caso al no ser el jugador número uno no tendrá acceso al botón *Start*.

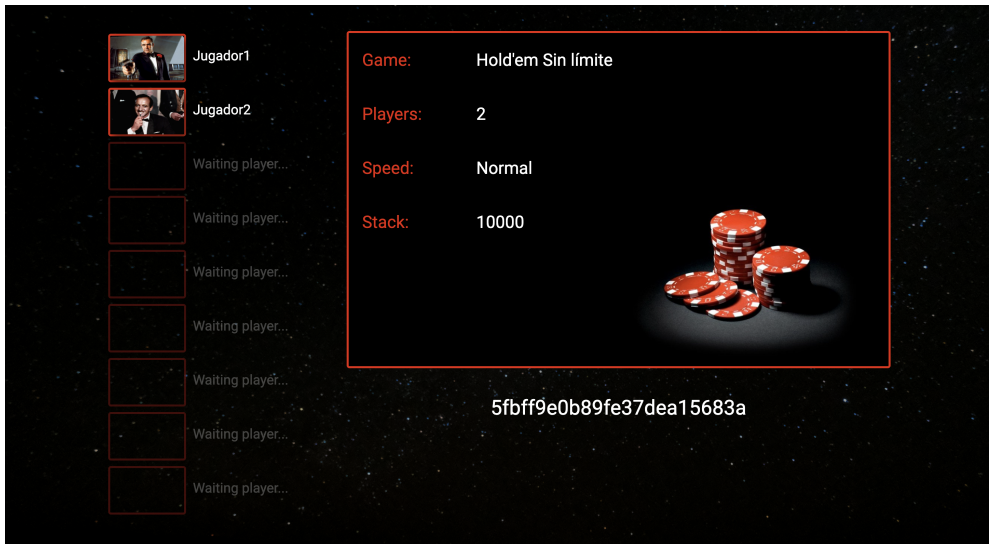

Figura B.5: Pantalla de lobby del torneo tras unirse

B.6. Pantalla del lobby preparado para iniciar torneo

Cuando el jugador número uno decida podrá dar inicio al torneo pulsando en el botón *Start*. Esto hará que todos los jugadores pasen a la pantalla del torneo.
Si el primer jugador se sale del lobby sin haber iniciado, el jugador dos pasará a ocupar su puesto y tendrá acceso al botón *Start*

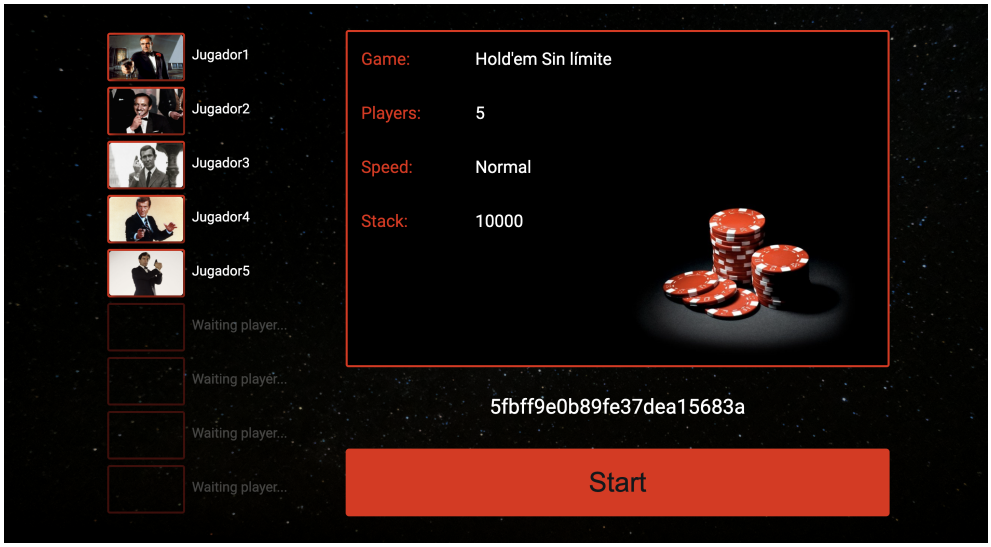

Figura B.6: Pantalla del lobby preparado para iniciar partida

B.7. Pantalla del torneo

Una vez iniciado el torneo todos los jugadores pasarán a esta pantalla, en ella podemos ver en el centro la mesa de juego, donde se dispondrán alrededor de ella los jugadores, donde se irán mostrando las cartas y donde aparecerán las apuestas de cada jugador junto con el *pot* total de la mano.
Mas abajo encontramos el *panel de acciones del jugador*, tres espacios para los botones que irán cambiando segun se pueda hacer *fold*, *call*, *check*, *bet* o *raise*. Encima de estos botones está el slider donde podremos seleccionar la cantidad de fichas a apostar.
Al lado izquierdo de la pantalla se situa el *log* de la partida, donde irán apareciendo los distintos movimientos de cada jugador.
A la derecha de la pantalla encontramos la tabla de ciegas, que irá mostrando en rojo el nivel actual en el que se encuentra en torneo.

Figura B.7: Pantalla del torneo

B.8. Pantalla del torneo estado de showdown

Cuando la mano pase a estar en estado de *showdown* (fase en donde los jugadores muestran sus cartas), las cartas de cada jugador giraran revelando su valor. En ese momento aparecerá en el *log* quien es el ganador de la mano y con qué combinación de cartas ha ganado

Figura B.8: Pantalla del torneo estado de showdown

Bibliografía

[1] *Build a Chat App With MongoDB & Socket.io*,
https://www.youtube.com/watch?v=8Y6mWhcdSUM&t=993s&ab_channel=TraversyMedia

[2] *Mean (MongoDB, Express, Angular & Node.js) Socket.IO app - Section Three (Node Express & Mongoose)*,
https://www.youtube.com/watch?v=VUovDOjLZBU&ab_channel=SureshC

[3] *Desarrollo web con JavaScript, Angular, NodeJS y MongoDB*,
https://www.udemy.com/course/desarrollo-web-con-javascript-angular-nodejs-y-mongodb/

[4] *Angular - Mastering the basics*,
https://www.udemy.com/course/angular-mastering-the-basics/learn/lecture/10121086#overview

[5] *MEAN Stack Quick Guide. Guide to become a Full Stack developer using M.E.A.N*,
https://medium.com/@onejohi/mean-stack-quick-guide-f50351ba56eb

[6] *Introduction to MongoDB*,
https://medium.com/@saivittalb/introduction-to-mongodb-859ed4426994

[7] *NodeJS + MongoDB*,
https://medium.com/200-response/nodejs-mongodb-78e1fc1445de

[8] *Documentación oficial de Angular*,
https://angular.io/docs

[9] *Documentación oficial de NodeJS*,
https://nodejs.org/es/docs/

[10] *Documentación oficial de ExpressJS*,
https://expressjs.com/es/guide/routing.html

[11] *Documentación oficial de MongoDB*,
https://docs.mongodb.com/manual/

[12] *Bold, italics and underlining - Overleaf, Online LaTeX Editor*,
https://www.overleaf.com/learn/latex/bold,_italics_and_underlining

[13] *How to write URLs in Latex? - Stack Overflow*,
https://stackoverflow.com/questions/2894710/how-to-write-urls-in-latex

[14] *Texas hold 'em - Wikipedia*,
https://es.wikipedia.org/wiki/Texas_hold_%27em

[15] *Lists - Overleaf, Editor de LaTeX online*,
https://es.overleaf.com/learn/latex/lists

[16] *Manejar eventos Socket.io y Angular. — by Leifer Mendez — Medium*,
https://medium.com/@leifer33/manejar-eventos-socket-io-y-angular-ad8c8f340be1

[17] *Socket.io (Node.js) Beginners Guide — Mediumz*,
https://medium.com/@icbrewery007/socket-io-node-js-beginners-guide-97fbc4f8ccbd

[18] *Introduction to Socket.IO*,
https://medium.com/@chathuranga94/introduction-to-socket-io-600025322cd2

[19] *Entendiendo NodeJs. What is Node ? — by delgadotrueba — Medium*,
https://medium.com/@delgadotrueba/understanding-nodejs-d526e8e5313

[20] *MongoDB 16 most important commands to start using this NoSQL database*,
https://itnext.io/mongodb-from-zero-to-rock-16-most-important-commands-to-start-using-this-nosql-database-b46728bc0e41

[21] *GitHub - htdebeer/SVG-cards: A set of playing cards in SVG (now also with a rendering in PNG)*,
https://github.com/htdebeer/SVG-cards

[22] *Make font of \subsubsection bold in amsart - TeX - LaTeX Stack Exchange*,
https://tex.stackexchange.com/questions/326498/make-font-of-subsubsection-bold-in-amsart

[23] *Line breaking - Forcing linebreaks in \url - TeX - LaTeX Stack Exchange*,
https://tex.stackexchange.com/questions/3033/forcing-linebreaks-in-url

[24] *Póker - Wikipedia*,
https://es.wikipedia.org/wiki/P%C3%B3quer

[25] *Reglas del póker - PokerStarts*,
https://www.pokerstars.es/poker/games/rules/

[26] *GitHub - goldfire/pokersolver: Javascript poker hand solver*,
https://github.com/goldfire/pokersolver

[27] *GitHub - ritsrivastava01/ngxFlip: Simple Flip component based on angular*,
https://github.com/ritsrivastava01/ngxFlip

[28] *Angular Material UI component library*,
https://material.angular.io/

[29] *Software Project Management 5Th Edition By Mike Cotterell And Bob Hughes, 2009*

[30] *Unified Process - an overview — ScienceDirect Topics*,
https://www.sciencedirect.com/topics/computer-science/unified-process

[31] *Proceso unificado de desarrollo - EcuRed*,
https://www.ecured.cu/Proceso_unificado_de_desarrollo

[32] *Tema 3. El Método de desarrollo. El Proceso Unificado*,
https://www.infor.uva.es/~mlaguna/cd/cd6

[33] *Rational Unified Process - IBM*,
https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

[34] *https://css-tricks.com/snippets/css/a-guide-to-flexbox/*,
ACompleteGuidetoFlexbox|CSS-Tricks

[35] *Making Multiplayer HTML5 Game: MongoDB Database. NodeJs Tutorial Guide*,
https://www.youtube.com/watch?v=E-YYRLQSVDI&t=701s&ab_channel=ScriptersWar

[36] *A Complete Guide to Grid — CSS-Tricks*,
https://css-tricks.com/snippets/css/complete-guide-grid/

[37] *Patrones de diseño - Departamento de Informática*,
https://www.infor.uva.es/~felix/datos/priii/tr_patrones-2x4.pdf

[38] *Patrones de Diseño*,
http://siul02.si.ehu.es/~alfredo/iso/06Patrones.pdf

[39] *Cómo relacionar tus modelos en MongoDB*,
https://carlosazaustre.es/como-relacionar-tus-modelos-en-mongodb