UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación web para el juego de cartas The Mind

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Linuesa Garay, Rubén

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación web para el juego de cartas The Mind

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Rubén Linuesa Garay
Tutor: Pedro Valderas

Curso 2022 - 2023

Resum

El propòsit d'aquest projecte és desenvolupar un videojoc web multijugador en línia basat en el joc de cartes The Mind, dissenyat per ser jugat per dues persones. La implementació s'ha dut a terme utilitzant el framework React i l'eina Firebase de Google. A més, aquest projecte abasta una anàlisi tant del mercat actual per a jugar The Mind o altres jocs semblants en línia de forma multijugador com de les eines disponibles per al desenvolupament de videojocs.

Paraules clau: Firebase, React, videojoc web, multijugador en línia, The Mind.

Resumen

El propósito de este proyecto es desarrollar un videojuego web multijugador en línea basado en el juego de cartas The Mind, diseñado para ser jugado por dos personas. La implementación se ha llevado a cabo utilizando el framework React y la herramienta Firebase de Google. Además, este proyecto abarca un análisis tanto del mercado actual para jugar The Mind u otros juegos similares en línea de forma multijugador como de las herramientas disponibles para el desarrollo de videojuegos.

Palabras clave: Firebase, React, videojuego web, multijugador en línea, The Mind.

Abstract

The purpose of this project is to develop an online multiplayer web video game based on the card game The Mind, designed to be played by two people. The implementation has been carried out using the React framework and Google's Firebase tool. Additionally, this project encompasses an analysis of both the current market for playing The Mind or similar games online in a multiplayer format and the tools available for game development.

Key words: Firebase, React, web video game, online multiplayer, The Mind.

Índice general

Apéndices

Índice de figuras

Índice de algoritmos

CAPÍTULO 1
Introducción

*T*he Mind[1] consiste en una sencilla baraja numerada del 1 al 100. Se juega por rondas, y en cada ronda se reparten cartas a cada jugador en función del número de la ronda. Lo intrigante de este juego es que los jugadores no compiten entre sí, sino que colaboran para lograr el objetivo principal: ordenar las cartas de menor a mayor. Sin embargo, los jugadores no pueden comunicarse durante la ronda, excepto para acordar el uso de un comodín, y disponen de un número limitado de vidas y comodines.

1.1 Motivación

Los juegos de mesa son una de las grandes pasiones de muchas personas. Debido a la última pandemia, que impidió que muchas personas pudieran reunirse en persona, los juegos de mesa en línea se han vuelto más buscados. En particular, The Mind es uno de mis juegos de mesa favoritos para pasar el tiempo con amigos. Además quería realizar un proyecto de página web y desarrollarlo full stack.[2]

1.2 Objectivos

El objetivo de este proyecto es desarrollar una versión jugable y multijugador en línea del juego de cartas. La implementación será en formato web y contará con animaciones de juego, además de la capacidad de realizar un seguimiento de las estadísticas por usuario, ya que se implementará un sistema de registro de usuarios en la plataforma.

[1] Juego de cartas cooperativo diseñado por Wolfgang Warsch creado en 2018.

[2] Técnica de desarrollo en la que un único desarrollador tiene habilidades tanto en el frontend como en el backend

CAPÍTULO 2
Estado del arte

Figura 2.1: Imagen promocional de The Mind

The Mind no es un juego ampliamente conocido, lo que limita las opciones disponibles para jugar en línea. Sin embargo, se han identificado dos aplicaciones o juegos que ofrecen esta experiencia:

2.1 Tabletop Simulator

Tabletop Simulator es una plataforma de juego en línea que se encuentra disponible en la popular plataforma de videojuegos llamada Steam[1]. Esta plataforma permite a los usuarios recrear y disfrutar de una amplia variedad de juegos de mesa y cartas en un entorno virtual. Fue desarrollada por la empresa Berserk Games y se puede adquirir por un precio de 19,99€.

La característica principal de Tabletop Simulator es que proporciona a los jugadores una mesa virtual digital en la que pueden realizar actividades similares a las que se lleva-

[1]Steam es una plataforma de distribución digital de videojuegos y software desarrollada por Valve Corporation.

rían a cabo en una mesa de juego real. Esto incluye la capacidad de colocar y manipular componentes de juegos, como cartas, fichas y tableros. Los jugadores pueden interactuar con estos elementos de manera intuitiva, lo que permite una experiencia de juego inmersiva y cercana a la de la vida real.

Figura 2.2: Página de la tienda de Steam de Tabletop Simulator

Esta plataforma es conocida por su flexibilidad y capacidad para adaptarse a una amplia variedad de juegos de mesa y cartas, lo que la convierte en una opción popular para jugar títulos menos conocidos en línea. Mediate Steam Workshop[2] los usuarios pueden acceder a una biblioteca de juegos creados por la comunidad o crear sus propias configuraciones personalizadas para jugar con amigos o jugadores de todo el mundo.
Para este caso, en la biblioteca se encuentra un mod para The Mind diseñado por el usuario «Malixx». Este tiene importados todos los diseños de cartas originales del juego de mesa y dispone de 3 botones, uno de ellos tiene automatizado el reparto de cartas y los otros 2 para subir y bajar de ronda.

[2]Plataforma integrada en Steam que permite a los usuarios crear, compartir y descargar contenido adicional para videojuegos compatibles.

Figura 2.3: Captura mod The mind para Tabletop Simulator

2.2 App móvil

Existe otra opción para jugar The Mind que está disponible tanto en la App Store[3] como en la Play Store[4]. Este juego ha sido desarrollado por «Brettspielwelt GmbH» y está disponible a un precio de 2,99€.A pesar de sus pocas descargas, ha recibido críticas mixtas, con una calificación promedio de 2,2 sobre 5 y numerosas reseñas negativas, especialmente en lo que respecta a aspectos técnicos y de funcionamiento. Destacan sobretodo problemas de estabilidad con cierres inesperados de la aplicación y problemas de rendimiento.

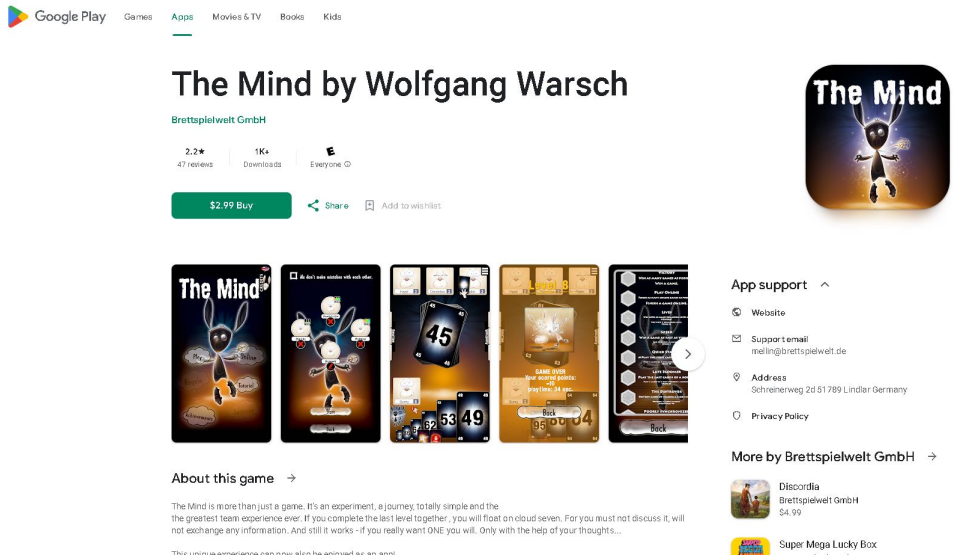


Figura 2.4: Pagina de Play Store de The Mind

[3]Plataforma de distribución de aplicaciones desarrollada por Apple Inc. para dispositivos iOS, como iPhone y iPad.

[4]Plataforma de distribución de aplicaciones desarrollada por Google para dispositivos Android.

CAPÍTULO 3
Metodología

Es esencial establecer una planificación adecuada para guiar el desarrollo de un producto software. Para lograrlo, hemos definido una serie de tareas y objetivos iniciales que servirán como marco de referencia a lo largo del proyecto. A continuación se presentará la evolución del proyecto a lo largo del tiempo.

En cuanto a la metodología de desarrollo, hemos optado por utilizar un enfoque ágil. Esta elección se basa en la flexibilidad y la capacidad de dividir el trabajo en sprints[1], lo que facilita una gestión más efectiva del proyecto. Para el seguimiento y la gestión de las tareas, hemos implementado un tablero Kanban[2] utilizando la herramienta Trello.[3]

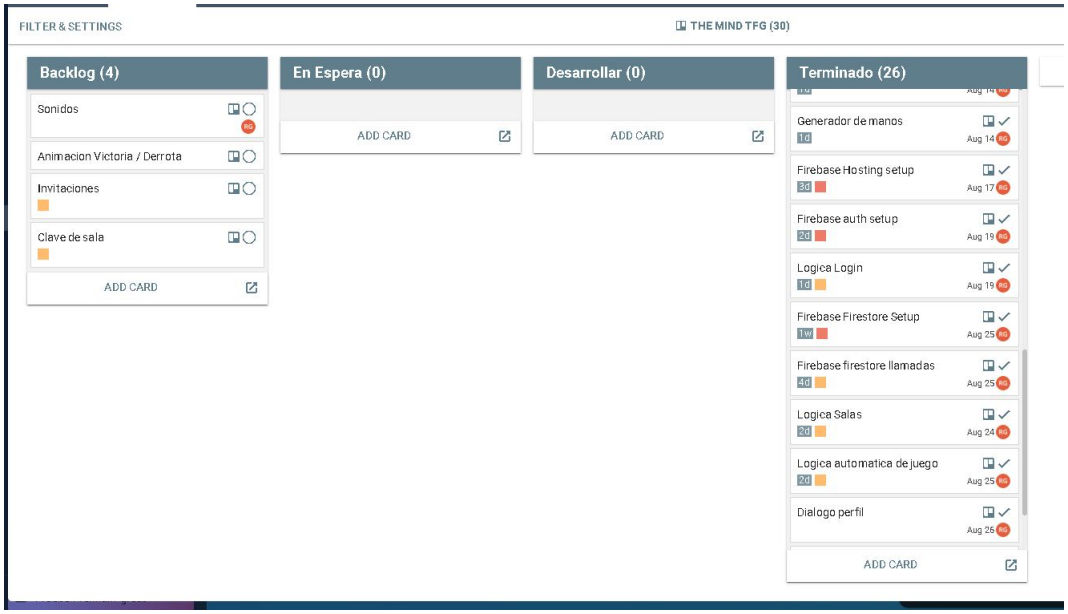


Figura 3.1: Tablero Kanban de The Mind

[1]Período definido en el desarrollo ágil de software para trabajar en tareas específicas en un tiempo limitado.

[2]Método de gestión visual que ayuda a controlar y gestionar el flujo de trabajo, utilizando tarjetas o notas para representar tareas.

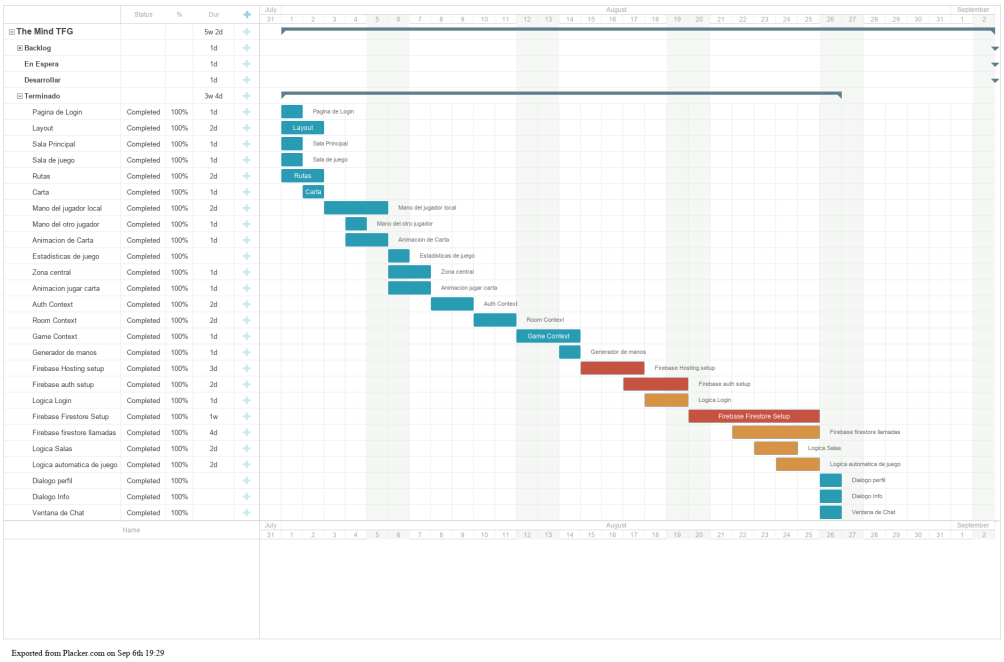
[3]Popular herramienta de gestión de proyectos basada en el método Kanban que utiliza tableros, listas y tarjetas para organizar tareas y colaborar en proyectos.

3.1 Sprints

En el primer sprint, nos hemos centrado en la parte de frontend[4] del proyecto. Esto nos permitió obtener relativamente rápido una aproximación a un estado potencialmente final del proyecto, aprovechando la familiaridad del equipo con la tecnología de React. Durante este sprint, hemos completado tareas relacionadas con la interfaz de usuario y la experiencia del usuario. El producto al final de este sprint era una aplicación local con datos de prueba. lo que nos podía dar una rápida visión a lo que sería el proyecto terminado.

El segundo sprint, que será el último, estará más orientado hacia la parte de backend[5]. En este sprint, abordaremos aspectos como la configuración de la herramienta Firebase en el proyecto, que es crucial para el funcionamiento de la aplicación en línea. Hemos realizado toda la configuración de las herramientas, Firebase Hosting, Authentication y Firestore, así como la creación y uso de la base de datos en la aplicación.

A continuación, podremos ver el diagrama de Gantt[6] con las unidades de trabajo de los 2 Sprints. En azul, las relacionadas con frontend, y en naranja y rojo, las relacionadas con la base de datos y Firebase.



Exported from Placker.com on Sep 6th 19:29

Esta metodología ágil y esta división en sprints nos ha permitido gestionar de manera efectiva las distintas etapas del desarrollo y nos ha brindado un enfoque estructurado y eficiente para el desarrollo del producto software.

[4]Parte de una aplicación web o software que interactúa directamente con el usuario y se encuentra en el lado del cliente. Esto incluye la interfaz de usuario, la experiencia del usuario y la presentación de datos.

[5]Parte de una aplicación web o software que se encarga del procesamiento de datos, la gestión de bases de datos y la lógica de servidor.

[6]Gráfico de barras utilizado en la gestión de proyectos para representar la planificación y programación de tareas a lo largo del tiempo.

CAPÍTULO 4
Análisis de Requisitos

4.1 Requisitos iniciales

4.1.1. Requisitos Funcionales

1. Registro de Usuarios:
   - *a*) Los jugadores deben tener la opción de registrarse o iniciar sesión en la plataforma.
   - *b*) Deben poder usar un apodo[1] (nick) en lugar de información personal.
   - *c*) Los jugadores también pueden entrar al juego de forma anónima, sin necesidad de registrarse.
   - *d*) Al jugar de forma anónima, pueden elegir un apodo (nick) temporal para su sesión actual.
2. Creación de Salas:
   - *a*) Los jugadores pueden crear salas de juego.
   - *b*) Cada sala debe tener un identificador único.
   - *c*) Los jugadores pueden unirse a las salas existentes mediante el numero de sala.
3. Interfaz de Juego:
   - *a*) La interfaz de juego debe proporcionar una representación visual de las cartas de los jugadores.
   - *b*) Debe mostrar las cartas de los jugadores, sin revelar las cartas de otros jugadores.
   - *c*) La interfaz de juego debe proporcionar una representación visual de las cartas jugadas.
   - *d*) Debe tener animaciones a la hora de interactuar con las cartas.
4. Mecánica de Juego:
   - *a*) El juego debe seguir las reglas estándar de The Mind.
   - *b*) Los jugadores deben jugar cartas en orden ascendente sin comunicarse directamente entre sí.

[1] Nombre alternativo utilizado por los jugadores en un juego para identificarse en lugar de su nombre real.

c) Se debe proporcionar una forma de indicar cuándo un jugador ha cometido un error (por ejemplo, sonidos o efectos visuales).

5. Sistema de Puntuación:

   a) El juego debe realizar un seguimiento de las rondas completadas exitosamente.

   b) El juego debe realizar un seguimiento de las vidas y comodines restantes.

   c) El juego debe mostrar un mensaje de victoria o derrota.

4.1.2. Requisitos No Funcionales

1. El juego debe incluir animaciones fluidas para mejorar la experiencia del usuario.
2. Debe implementar medidas de seguridad para proteger los datos del usuario y garantizar la integridad del juego.
3. El juego debe ser compatible con navegadores web modernos y dispositivos móviles.
4. Se debe proporcionar documentación técnica y de usuario para el juego.

4.2 Casos de Uso

4.2.1. Caso de Uso 1: Registro de Usuario

Actor Principal: Usuario no registrado.

Escenario Principal:

1. El usuario accede a la página de inicio.
2. El usuario selecciona la opción de registro.
3. El usuario proporciona un apodo, un email y una contraseña.
4. El sistema verifica los datos y registra al usuario.
5. El usuario queda autenticado y se le redirige a la página principal.

4.2.2. Caso de Uso 2: Inicio de Sesión

Actor Principal: Usuario registrado o anónimo.

Escenario Principal:

1. El usuario accede a la página de inicio.
2. El usuario selecciona la opción de inicio de sesión o inicio anónimo.
3. El usuario proporciona su apodo o email y contraseña si está registrado.
4. El sistema verifica los datos y autentica al usuario.
5. El usuario queda autenticado y se le redirige a la página principal.

4.2.3. Caso de Uso 3: Creación de Sala de Juego

Actor Principal: Usuario autenticado.

Escenario Principal:

1. El usuario inicia sesión en la aplicación.
2. El usuario accede a la página principal.
3. El usuario selecciona la opción para crear una sala de juego.
4. El sistema crea una nueva sala con un identificador único.
5. El sistema coloca al usuario como el creador de la sala y le proporciona un enlace para compartir con otros jugadores.
6. El usuario puede esperar a que otros jugadores se unan a la sala y comenzar el juego cuando esté listo.

4.2.4. Caso de Uso 4: Unirse a una Sala de Juego

Actor Principal: Usuario autenticado.

Escenario Principal:

1. El usuario inicia sesión en la aplicación.
2. El usuario accede a la página principal.
3. El usuario selecciona la opción para unirse a una sala de juego.
4. El usuario ingresa el identificador único de la sala proporcionado por el creador.
5. El sistema verifica la existencia de la sala y la disponibilidad de espacios.
6. El usuario se une a la sala y puede esperar a que comience el juego.

4.2.5. Caso de Uso 5: Jugar una Partida

Actor Principal: Jugadores en una sala de juego.

Escenario Principal:

1. Los jugadores se encuentran en una sala de juego y están listos para comenzar.
2. El juego comienza y se reparten las cartas a los jugadores.
3. Los jugadores deben jugar sus cartas en orden ascendente sin comunicarse verbalmente.
4. El sistema verifica si las cartas se juegan en el orden correcto y si se cometen errores.
5. Si se comete un error, el sistema notifica a los jugadores y registra la puntuación.
6. Los jugadores pueden decidir usar un comodin mediante un chat[2] con el que comunicarse.
7. Los jugadores pueden continuar jugando más rondas hasta finalizar la partida.

[2]Comunicación en tiempo real a través de mensajes de texto entre dos o más personas.

4.3 Diagrama de clases

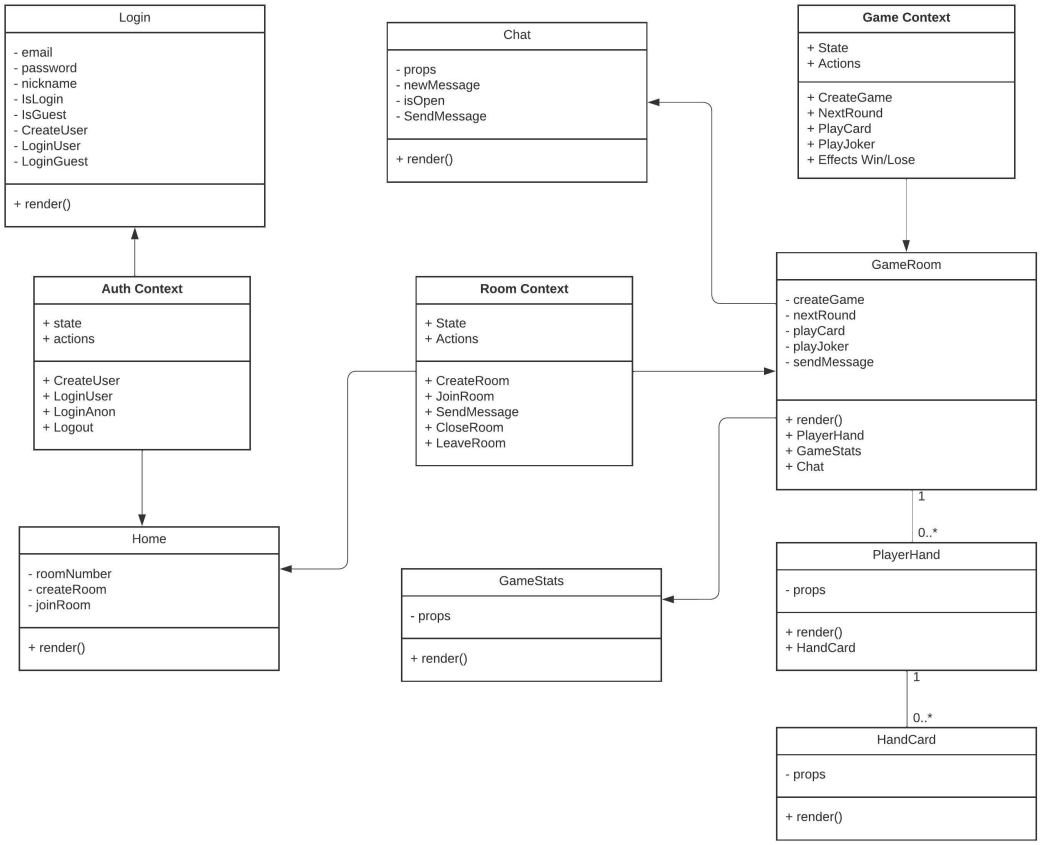


Figura 4.1: Diagrama de clases

CAPÍTULO 5
Diseño

5.1 Modelo de la BD del servidor

En este caso Se ha decidido diseñar la base de datos con estructura noSql basado en colecciones y documentos en cada una ya que es lo que utiliza Firestore.

1. Colección de Usuarios: Esta colección almacena los datos de usuarios.
2. Colección de Salas: Aquí se registran detalles relacionados con las salas de juego, como números de sala y datos de jugadores.
3. Colección de Partidas: Esta colección funciona como una subcolección de cada sala y contiene datos sobre las partidas en curso, incluyendo el estado del juego y estadísticas.

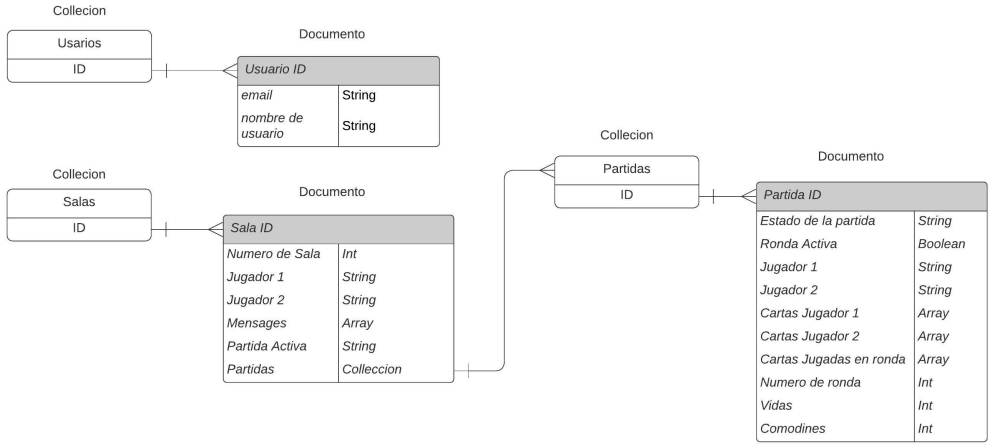


Figura 5.1: Diagrama de la base de datos noSql

5.2 Bocetos de las interfaces de la aplicación

Para la realización de los mockups se ha utilizado la herramienta Balsamiq.

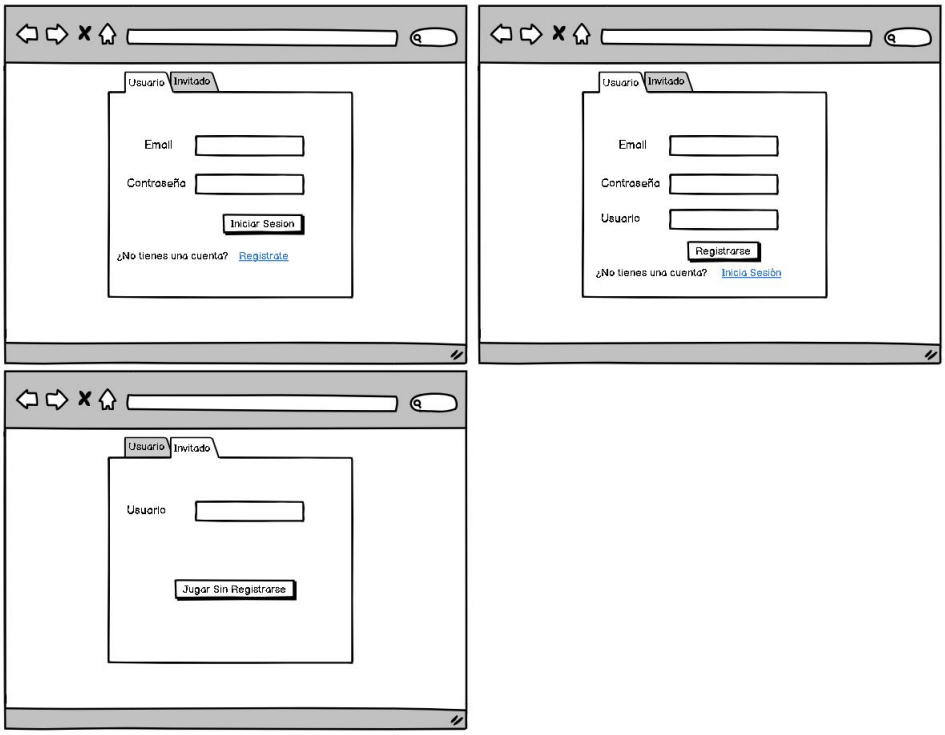


Figura 5.2: Mockups de inicio de sesion

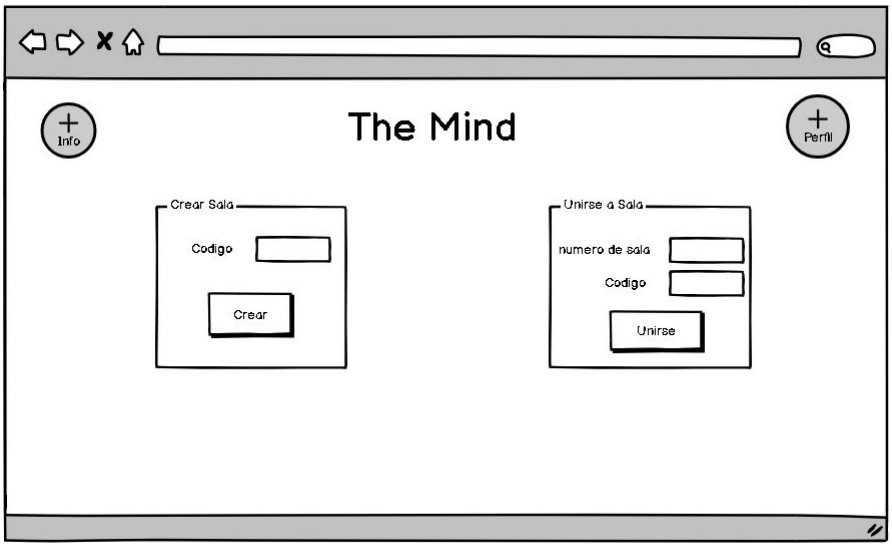


Figura 5.3: Mockup de pantalla principal

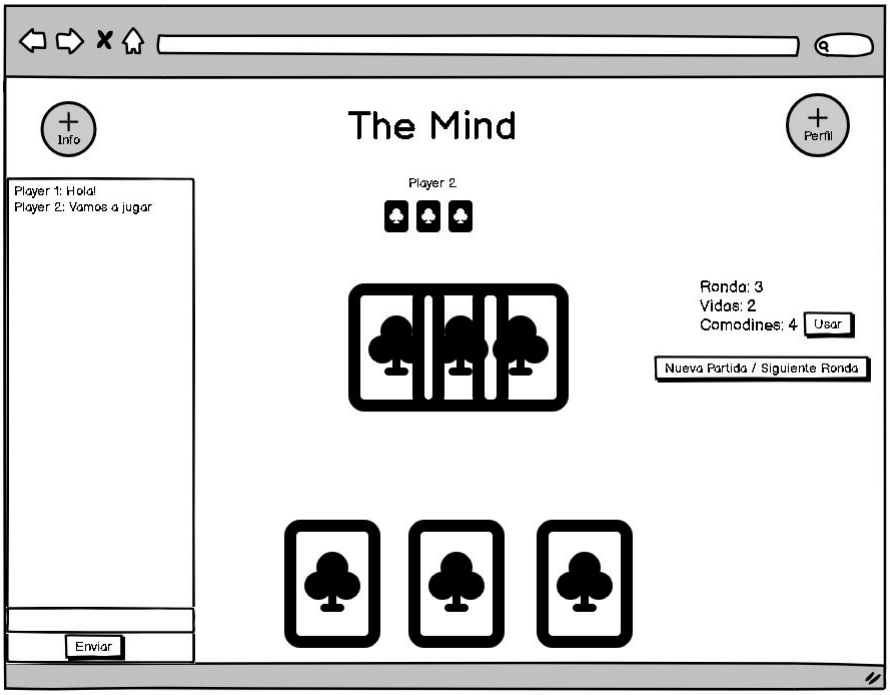


Figura 5.4: Mockups de sala de juego

CAPÍTULO 6
Desarrollo de la solución

6.1 Arquitectura Y Context Tecnológico

Los elementos esenciales que participan en la arquitectura de la aplicación son React con Material UI y Framer Motion en el frontend, y Firebase en la parte del servidor para alojar la página web, la gestión de registro e inicio de sesión de usuariosy la administración de la base de datos.

Comunicación entre Elementos:

1. **React**:
   - En el frontend, la aplicación web desarrollada en React hace uso de las bibliotecas Material UI y Framer Motion para crear una experiencia de usuario atractiva y animada.
   - Interactúa directamente con Firebase Firestore utilizando el SDK de Firebase para JavaScript.
2. **Firebase**: Firebase pone a disposición multitud herramientas de las cuales se han elegido las siguientes.
   - Firebase Hosting se utiliza para alojar la página web y permitir su acceso a través de la web.
   - Firebase Authentication gestiona el proceso de registro e inicio de sesión de los usuarios, garantizando la seguridad y autenticidad de los mismos.
   - Firebase Firestore se encarga de administrar la base de datos, lo que incluye el almacenamiento y la recuperación de datos relacionados con la aplicación.

Comunicación y Datos:

- La comunicación entre el frontend y Firebase se realiza a través del SDK de Firebase para JavaScript, lo que permite enviar cambios a la base de datos y recibir actualizaciones en tiempo real.
- Al enviar datos a Firebase Firestore, el sistema automatiza su almacenamiento en formato JSON, lo que simplifica la gestión de datos. Sin embargo, es crucial seguir una estructura adecuada de datos, siguiendo las pautas documentadas por Firebase. Se envía un objeto ya estructurado que contiene los campos que se desean añadir o modificar en la base de datos.

Firestore, como una base de datos NoSQL basada en formato JSON, permite la actualización de datos existentes o la adición de nuevos datos como nuevos documentos o colecciones, dependiendo de la estructura de tu base de datos. Esta flexibilidad brinda un control preciso sobre cómo se organizan y gestionan los datos en tu aplicación

Bibliotecas:

- Material-UI es una biblioteca de componentes[1] de interfaz de usuario diseñada para React que sigue las pautas de diseño de Material Design, desarrolladas por Google. Esta biblioteca ofrece una amplia gama de componentes predefinidos, como botones, barras de navegación, tarjetas y formularios, lo que facilita la creación de interfaces de usuario atractivas y coherentes en aplicaciones web de React. Uno de los puntos fuertes de Material-UI es su capacidad de personalización y tematización, lo que permite a los desarrolladores adaptar los componentes según las necesidades específicas de sus proyectos. Sin embargo, debido a que Material-UI se utiliza comúnmente para diseñar interfaces de menús y otros elementos estándar, uno de los desafíos fue utilizar sus componentes para diseñar los elementos del juego.
- Framer Motion es una biblioteca de animación diseñada para React que simplifica la creación de animaciones y transiciones fluidas en componentes de React. Proporciona una serie de componentes y utilidades que facilitan la creación de animaciones tanto simples como complejas en respuesta a eventos o cambios de estado. Framer Motion es particularmente útil para dar vida a elementos de la interfaz de usuario, como animar la entrada o salida de componentes, crear transiciones de página o añadir interactividad a elementos como botones o tarjetas. Esta biblioteca es altamente personalizable y permite definir animaciones basadas en propiedades CSS[2] y transiciones entre diferentes estados de manera eficiente.

[1] Elementos de interfaz de usuario predefinidos para React.

[2] Siglas de 'Cascading Style Sheets' en inglés. Se refiere a un lenguaje de marcado utilizado para definir el estilo y la presentación visual de documentos web.

CAPÍTULO 7
Implementacion

7.1 Front End

En esta fase, se llevó a cabo la organización y estructuración de la parte correspondiente al frontend del proyecto. Además, se realizó el diseño visual de la aplicación, abarcando aspectos como los componentes, los assets[1] utilizados, la estructura visual y las animaciones desarrolladas. A continuación, se presentarán los componentes más relevantes, destacando sus funciones en el contexto de la aplicación. La estructura de presentación seguirá un orden jerárquico, comenzando desde los componentes de nivel superior y descendiendo hacia los componentes más específicos.

7.1.1. Index.js y App.js

Estos componentes son fundamentales para dar estructura a la aplicación React. Comenzando con `index.js`, se configura la renderización inicial en el elemento 'root'. Luego, en `App.js`, se desempeña un papel central al actuar como el contenedor principal que se adapta según la ruta actual.

La aplicación utiliza React Router para gestionar las rutas y la navegación. Cada ruta se define mediante el componente `<Route>` y se asocia con un componente específico, como `<Home />` o `<GameRoom />`. Además, el contexto de autenticación se administra a través de `<AuthProvider>`, mientras que las rutas protegidas se controlan con `<ProtectedRoute>`, garantizando que solo los usuarios autenticados puedan acceder a ciertas partes de la aplicación.

```
function App() {
  return (
    <AuthProvider>
      <Routes>
        <Route
          element={
            <ProtectedRoute>
              <RoomProvider>
                <Layout />
              </RoomProvider>
            </ProtectedRoute>
          }
        >
          <Route exact path="/Home" element={<Home />} />
          <Route
```

[1] Recursos multimedia y archivos utilizados en un juego o aplicación, como imágenes, videos y fuentes

```
            exact
            path="/:roomId"
            element={
              <GameProvider>
                <GameRoom />
              </GameProvider>
            }
          />
          <Route path="/" element={<Navigate to="/Home" replace />} />
        </Route>

        {<Route path="/Login" element={<Login />} />}
      </Routes>
    </AuthProvider>
  );
}
```

Algoritmo 7.1: Estructura de App.js

7.1.2. Contextos

En una aplicación de React, un contexto cumple la función de proporcionar un espacio centralizado para compartir y gestionar datos y funciones que deben estar disponibles en toda la aplicación, independientemente de la profundidad de la jerarquía de componentes. Esto resulta especialmente útil cuando se requiere mantener un estado global. El contexto permite que los componentes accedan a estos datos o funciones de manera eficiente, sin depender de la propagación de propiedades a través de múltiples niveles de componentes hijos. Esta separación de la lógica de estado de la representación visual de los componentes simplifica el código y mejora su legibilidad. Esto facilita el mantenimiento y la escalabilidad de la aplicación, ya que los cambios en el estado global pueden gestionarse en un solo lugar, y los componentes individuales pueden centrarse en su propia funcionalidad sin preocuparse por la gestión de datos compartidos.

Al definir un contexto, se establece un estado inicial y se definen acciones para modificar ese estado. Estos elementos pueden ser accedidos desde cualquier parte de la aplicación que esté incluida dentro del proveedor del contexto, lo que incluye tanto el estado como los métodos públicos.Para mantener un estándar en las acciones y garantizar una gestión coherente del estado, se deben definir métodos públicos. Dentro de estos métodos es el único lugar donde se deben utilizar las acciones del «dispatch» para realizar cambios en el estado.

```

const DispatchContext = React.createContext();
const StateContext = React.createContext();

const initialState = {
  token: null,
  email: null,
  username: null,
  error: null,
};

function authReducer(state, action) {
  switch (action.type) {
    case 'USER_LOGIN_SUCCESS': {
      return {
        ...state,
```

```
        ...action.payload,
      };
    }

    case 'USER_LOGIN_ERROR': {
      return {
        ...state,
        error: action.payload.error,
      };
    }

    case 'USER_NOT_LOGGED': {
      return {
        ...state,
        token: null,
        email: null,
        username: null,
        error: null,
      };
    }

    default: {
      return state;
    }
  }
}
```

Algoritmo 7.2: Estados y acciones de Auth Context

Auth Context

El contexto de autenticación (Auth Context) tiene como función principal almacenar y proporcionar el estado de autenticación del usuario en toda la aplicación de React. Esto abarca detalles como el token de autenticación, la dirección de correo electrónico del usuario, su nombre de usuario y, en caso de ocurrir, un error relacionado con la autenticación. Este contexto se encarga de mantener disponibles y actualizados estos datos para todos los componentes que los requieran en la aplicación.

```
/**
 * Hook to access and manage the state of the Auth Context
 */
const useAuthContext = () => {
  const authState = useContext(StateContext);
  const authDispatch = useContext(DispatchContext);

  // EFFECTS  ////////////////////////

  /**
   * Effect to update when there is a change in the auth state
   * For the logged user it gets the data from DB
   */
  useEffect(() => {
    ...
  }, []);

  // PUBLIC METHODS  ////////////////////////

  /** Method to sign in anonymously and save username */
  const loginAsGuest = async (username) => {
    ...
```

```
  };

  /** Method to sign in with email and password */
  const createUser = async (email, password, username) => {
    ...
  };

  /** Method to log in with email and password */
  const login = async (email, password) => {
    ...
  };

  /** Method to log out */
  const logout = async () => {
    signOut(auth)
      .then(() => {
        // Sign-out successful.
        authDispatch({
          type: 'USER_NOT_LOGGED',
        });
      })
      .catch((error) => {});
  };

  return {
    loginAsGuest,
    createUser,
    login,
    logout,
    authState,
  };
};

export { AuthProvider, useAuthContext };
```

Algoritmo 7.3: Metodos y efectos de auth context

Room Context

El contexto de sala (Room Context) se utiliza para mantener y distribuir información relacionada con las salas de juego. Esto implica almacenar datos como el identificador de la sala, el número de la sala, los jugadores conectados a la sala, los mensajes enviados en la sala y el identificador de la partida actual. Además, este contexto identifica si el usuario local es el anfitrión de la sala, es decir, el jugador que creó la partida.

```
const initialState = {
  roomNumber: '',
  roomId: '',
  player1: '',
  player2: '',
  host: false,
  messages: [],
  actualGame: '',
};
```

Algoritmo 7.4: Estados de Room Context

Game Context

El contexto de juego (Game Context) se dedica a gestionar el estado de la partida actual. Esto comprende elementos críticos como el identificador, el estado de la partida (si ha comenzado o finalizado teniendo en cuenta si ha sido victoria o derrota), detalles sobre las manos de los jugadores, las cartas jugadas en la mesa durante la ronda, el número de ronda y la cantidad de vidas y comodines disponibles en la partida. En resumen, este contexto se encarga de mantener un seguimiento detallado de todos los aspectos relacionados con el progreso y las estadísticas de la partida en curso.

```
const initialState = {
  gameId: '',
  gameStatus: false,
  roundNumber: 0,
  player1Cards: [],
  player2Cards: [],
  playedCardsInRound: [],
  lives: lives,
  jokers: jokers,
};
```

Algoritmo 7.5: Estados de Game Context

```
/** Effect to control game loss */
  useEffect(() => {
    if (roomState.host && gameState.lives === 0) {
      lossGame();
    }
  }, [lossGame, gameState.lives, roomState.host]);

  /** Effect to control game win */
  useEffect(() => {
    if (
      roomState.host &&
      gameState.roundNumber === max_round &&
      gameState.player1Cards.length === 0 &&
      gameState.player2Cards.length === 0
    ) {
      winGame();
    }
  }, [winGame, gameState.roundNumber, roomState.host, gameState.player1Cards,
      gameState.player2Cards,]);
```

Algoritmo 7.6: Efectos para controlar si hay victoria o derrota

7.1.3. Páginas

Login

El componente «Login» adapta la interfaz permitiendo al usuario iniciar sesión, registrarse o jugar como invitado. Además, gestiona las acciones introducidas por el usuario y administra el contexto de autenticación, redirigiendo automáticamente a usuarios autenticados.

Home

El componente «Home» es la página principal de un usuario autenticado. Permite a los usuarios crear o unirse a salas de juego. Los usuarios pueden crear una sala aleatoria o unirse a una sala existente ingresando su número de sala. Este número se lo tendrá que proporcionar el jugador que haya creado la sala y que podrá ver en la dirección del navegador como /XXX. Además, el componente proporciona información sobre posibles errores al unirse a una sala.

Game Room

El componente «GameRoom» es la página principal de la aplicación donde se llevan a cabo las partidas. Este componente facilita la interacción del usuario con la partida y está compuesto por varios subcomponentes para lograr una estructura organizada y escalable. Estos son las manos de ambos jugadores, un espacio central para mostrar las cartas jugadas, estadísticas de la partida, acciones del juego, que se situan en la parte derecha y un elemento desplegable en la parte izquierda para el chat.

7.1.4. Componentes

Layout

El layout visualmente es un componente que será común en todas las páginas de la aplicación. En este caso, tendrá botones flotantes para abrir los diálogos de información sobre cómo jugar y el perfil, así como un texto en el centro que diga The Mind. Para lograr esto, se utiliza `outlet`, que es un marcador de posición en el enrutamiento de React proporcionado por react-router-dom. Permite que el contenido de una ruta específica se renderice en un punto particular de un diseño común compartido por múltiples páginas. Es útil para mantener elementos consistentes en todas las páginas mientras cambias solo el contenido principal según la ruta.

```
<Box sx={{ ... }}>
  <Box
    sx={{
          display: 'flex',
          flexDirection: 'row',
          alignItems: 'center',
          justifyContent: 'space-between',
          width: 'auto',
          marginBottom: 15,
    }}>
    {/* Info Floating Icon (Omitido) */}
    <Typography variant="h1">The Mind</Typography>
    {/* Profile Floating Icon (Omitido) */}
  </Box>

  {/* Dialog de Informacion (Omitido) */}
  {/* Dialog de Perfil (Omitido) */}

  {/* Contenido Principal */}
  <Outlet />
</Box>
```

Algoritmo 7.7: Estructura de Layout.jsx

Room Stats

`GameStats` es un componente de React que despliega información importante sobre el estado de la partida actual en un juego. Además, permite que el jugador anfitrión realice ciertas acciones

- **Información de la Partida:** `GameStats` muestra datos relevantes de la partida, como la ronda actual, el número de vidas restantes y la cantidad de comodines disponibles. En caso de que el juego haya terminado con victoria o derrota, se actualizan los mensajes para reflejar estos estados.

- **Acciones para el Anfitrión:** Si el jugador es el anfitrión de la sala de juego y ambos jugadores están conectados, se presentan botones adicionales:

  - **Empezar Partida / Siguiente Ronda:** Dependiendo del estado actual del juego, se muestra un botón que permite al anfitrión iniciar una nueva partida si aún no ha comenzado o avanzar a la siguiente ronda si el juego ya está en curso.
  - **Usar Comodín:** Si hay suficientes comodines disponibles, se muestra un botón que permite al anfitrión usar un comodín. En caso de que ocurra un error al intentar usarlo, como la falta de cartas en las manos de los jugadores, se mostrará un mensaje de error en el chat bajo el nombre «system».

- **Abandonar la Sala:** Para asegurar que los jugadores tengan la opción de salir de la sala en cualquier momento, se proporciona un botón de «Salir de la Sala». Este botón es visible para ambos jugadores en todo momento y se ha implementado debido a ciertas limitaciones relacionadas con la desconexión de un jugador y la actualización de la base de datos.

Player Hand

`PlayerHand` es un subcomponente diseñado para mostrar las manos de los jugadores. Su apariencia y comportamiento varían según la propiedad local que se le pase. Cuando se establece como verdadera, representa las cartas de manera destacada para el jugador local, mostrando las cartas en su tamaño real con la cara frontal visible. Cuando se establece como falsa, las cartas se representan en un tamaño más pequeño y muestran la cara trasera para así conservar la privacidad del otro jugador. Además, también establece si las cartas son seleccionables o no. Para mostrar cada carta en la mano del jugador se utiliza el subcomponente HandCard.

Hand Card

El componente `HandCard` es una representación de una carta de juego. La carta en sí está contenida en un componente `Card` de Material-UI y muestra una imagen de la carta base. Sobre esta imagen, se superpone el número de la carta (`card.id`) en las esquinas y en el centro.

7.1.5. Estilos y Animaciones

FlexBox

Flexbox es un modelo de diseño en CSS que permite organizar elementos en una página web de manera flexible y eficiente. Se utiliza para alinear elementos en un contenedor en una dirección principal y controlar cómo se distribuye el espacio disponible entre ellos. Es una herramienta con la que se pueden crear diseños responsivos y ordenados con facilidad y que simplifica la creación de diseños complejos y permite un mayor control sobre la disposición de los elementos en una página. Se puede ver un ejemplo de su uso en la Figura 7.7

Framer Motion

Primera Animación (Para Cartas Jugables en la Mano del Jugador): Esta animación utiliza Framer Motion para dar respuesta visual a la interacción del usuario con las cartas jugables. Cuando el cursor se coloca sobre una carta, la tarjeta se agranda ligeramente y se eleva, lo que proporciona un efecto de «resalte» y muestra que la carta es interactiva. Cuando se hace clic en la tarjeta, esta se reduce de tamaño con una transición suave, creando una animación de clic suave y fluida que mejora la experiencia del usuario al jugar.

```
<motion.div
  whileHover={{ scale: 1.1, y: -10 }}
  whileTap={{ scale: 0.9, transition: { duration: 0.3 } }}
>
```

Algoritmo 7.8: Código de animación de curson sobre carta

Segunda Animación (Para Cartas Jugadas en la Mesa de Juego): Esta animación se aplica a las cartas que se juegan y se añaden a la mesa de juego. Cuando una carta se juega, aparece en el centro de la mesa desde abajo. Inicialmente, la carta tiene una opacidad de cero y se anima para volverse completamente visible y deslizarse hacia el centro de la mesa con una transición de opacidad y movimiento. Esto crea un efecto visual atractivo que muestra las cartas jugadas de manera gradual y elegante en el contexto del juego.

```
<motion.div
  key={card.id}
  initial={{ opacity: 0, y: '100%' }}
  animate={{ opacity: 1, y: 0 }}
  transition={{ duration: 0.5 }}
  style={{
    position: 'absolute',
    left: `calc(50% - ${index * 20}px)`,
  }}
>
```

Algoritmo 7.9: Código de animación de jugar carta

7.2 Back End

7.2.1. Firebase Hosting

Firebase Hosting ofrece un servicio sencillo para alojar tu aplicación web. El proceso comienza registrando tu aplicación en Firebase, donde configuras los detalles del proyecto. Luego, debes incorporar el SDK de Firebase en tu proyecto, permitiendo la interacción con los servicios de Firebase desde tu aplicación. Firebase te proporcionará configuraciones específicas para tu aplicación, como claves de API y otros detalles necesarios. Una vez que hayas desarrollado tu aplicación y la hayas construido, puedes desplegarla en Firebase Hosting con un simple comando`"firebase deploy"`. Esto hace que tu aplicación esté disponible en línea a través de un dominio proporcionado por Firebase. Para ello previamente debes construir tu aplicación utilizando el comando `npm run build`.

7.2.2. Firebase Authentication

Firebase Authentication simplifica la autenticación de usuarios en aplicaciones web y móviles, ofreciendo múltiples métodos de inicio de sesión.

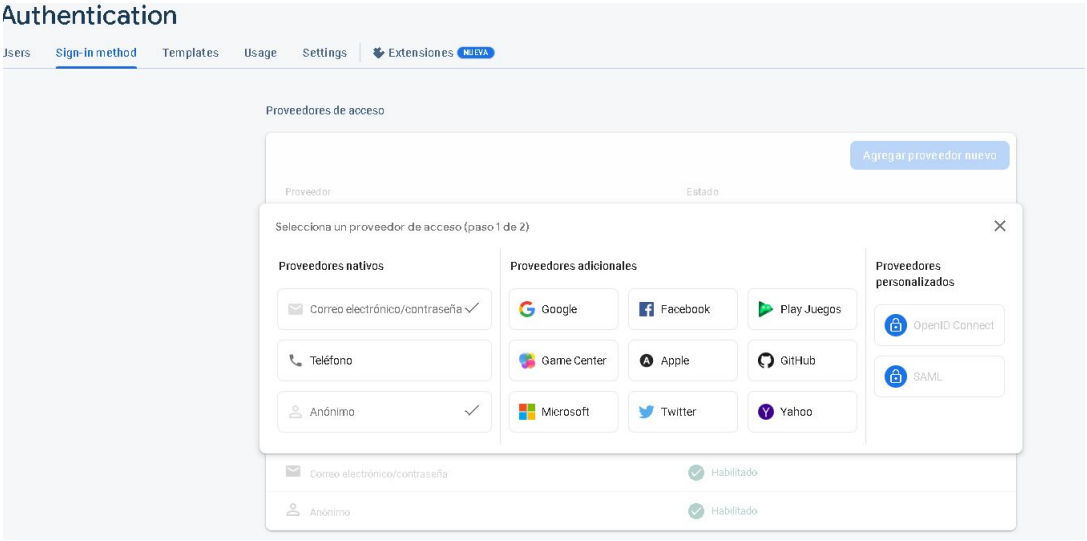


Figura 7.1: Opciones disponibles en Firebase Authentication

En este caso, se han seleccionado dos mecanismos: inicio de sesión con correo y contraseña, y autenticación anónima.

Los usuarios pueden crear cuentas con su correo electrónico y contraseña mediante `createUserWithEmailAndPassword(auth, email, password)`, o iniciar sesión con estas credenciales usando `signInWithEmailAndPassword(auth, email, password)`. Estos mecanismos se han implementado en el authContext de la aplicación, lo que simplifica la gestión de la autenticación y la interacción de los usuarios con la aplicación. Si el usuario elige jugar de manera anonima, se utiliza el método `signInAnonymously(auth)` para authenticarlo pero este luego no se guardara en la base de datos. Todo esto ese ha implementado en el `authContext`

```
const loginAsGuest = async (username) => {
  signInAnonymously(auth)
    .then((userCredential) => {
      const user = userCredential.user;
      // Save to Auth State
      authDispatch({
        type: 'USER_LOGIN_SUCCESS',
        payload: {
          token: user.uid,
          username: username,
          error: null,
        },
      });
    })
    .catch((error) => {
      authDispatch({
        type: 'USER_LOGIN_ERROR',
        payload: { error: error.message },
      });
    });
};
```

Algoritmo 7.10: Método para iniciar sesión de manera anónima

Además, se ha implementado un efecto que se suscribe a cambios en el estado de autenticación mediante `onAuthStateChanged`. Esta función actúa como un observador que detecta cuándo un usuario inicia o cierra sesión en la aplicación. En este caso, se utiliza para recuperar el nombre de usuario de la base de datos cuando un usuario inicia sesión (si no es anónimo) dado que al iniciar sesion no lo solicitamos y Firebase Authentication solo se encarga de los campos necesearios para ello.

```
useEffect(() => {
  const unsubscribe = onAuthStateChanged(auth, async (user) => {
    if (user) {
      const uid = user.uid;
      const exists = await userExists(user.uid);

      if (exists) {
        const loggedUser = await getUserData(uid);
        authDispatch({
          type: 'USER_LOGIN_SUCCESS',
          payload: {
            token: uid,
            username: loggedUser.username,
            email: loggedUser.email,
            error: null,
          },
        });
      } else {
        // save anon to db
        // saveUser(uid, { email: authState.email, username: authState.username
            });
      }
    } else {
      logout();
    }
  });

  return () => {
    // Desuscribirse del listener al desmontar el componente
```

```
    unsubscribe();
  };
}, []);
```

Algoritmo 7.11: Efecto para observar cuando cambia el estado del usuario

7.2.3. Firebase Firestore Database

Dado que Firestore utiliza un modelo de base de datos NoSQL, se ha establecido la creación de tres colecciones: una para usuarios, otra para salas y una tercera para partidas. Aunque se mencionan tres colecciones, en realidad son dos: usuarios y salas, ya que cada sala contiene una subcolección de partidas.

Usuarios

En la colección de usuarios, cada documento tiene campos para el correo electrónico y el nombre de usuario. Gracias a la posibilidad de establecer el identificador del documento de cada usuario con el identificador proporcionado por Firebase Authentication, no es necesario crear un campo adicional para el ID del usuario.

```
createUserWithEmailAndPassword(auth, email, password)
      .then((userCredential) => {
        const user = userCredential.user;
        saveUser(user.uid, { email: email, username: username });
        ...
        }
export async function saveUser(user, data) {
  try {
    const usersRef = collection(db, usersCollection);
    await setDoc(doc(usersRef, user), data);
  } catch (error) {
    throw error;
  }
}
```

Algoritmo 7.12: Funcion para crear usuario y guardar en BD

Salas

En la colección de salas, cada documento representa una sala de juego y contiene campos como «roomNumber», «player1», «player2», «actualGame» y «messages». El formato de cada mensaje en el campo «messages» incluye el «nick» y el «message» en sí. Para evitar la creación de salas duplicadas con el mismo número de sala, se realiza una comprobación, recopilando todas las salas y filtrar por el número de sala.

```
export async function apiCreateRoom(roomNumber, player1) {
  const roomRef = collection(db, roomCollection);
  try {
    const roomDocs = await getDocs(roomRef);
    let roomDoc = null;
    roomDocs.forEach((doc) => {
      if (doc.data().roomNumber === roomNumber) {
        roomDoc = doc;
```

```
      }
    });
    if (roomDoc) {
      // La sala ya existe, actualizar sala
      await updateDoc(doc(roomRef, roomDoc.id), {
        roomNumber: roomNumber,
        player1: player1,
        player2: '',
        messages: [],
        actualGame: '',
      });
      return { roomId: roomDoc.id };
    } else {
      // La sala no existe, crear sala
      const docRef = await addDoc(roomRef, {
        roomNumber: roomNumber,
        player1: player1,
        messages: [],
        actualGame: '',
      });
      return { roomId: docRef.id };
    }
  } catch (error) {
    throw error;
  }
}
```

Algoritmo 7.13: Llamada para crear/abrir sala

A la hora de unirse a una sala se tiene en cuenta que la sala exista, que esté abierta (tenga jugador 1) y que no esté llena (que no tenga jugador 2).

```
export async function apiJoinRoom(roomNumber, player2) {
  const roomRef = collection(db, roomCollection);

  try {
    const roomDocs = await getDocs(roomRef);
    let roomDoc = null;
    roomDocs.forEach((doc) => {
      if (doc.data().roomNumber === roomNumber) {
        roomDoc = doc;
      }
    });

    if (roomDoc) {
      const roomData = { ...roomDoc.data(), roomId: roomDoc.id };

      if (!roomData.player1) {
        // La sala existe pero no tiene player1
        throw new Error('La sala no esta creada.');
      }

      if (!roomData.player2) {
        // La sala existe y no tiene player2, unirse a la sala
        await updateDoc(doc(roomRef, roomDoc.id), {
          player2: player2,
        });
        return roomData;
      } else {
        // La sala ya tiene player2
        throw new Error('La sala ya tiene dos jugadores.');
```

```
      }
    } else {
      // La sala no existe
      throw new Error('La sala no esta creada.');
    }
  } catch (error) {
    throw error;
  }
}
```

Algoritmo 7.14: Llamada para unirse a una sala

Partidas

En la colección de partidas, cada documento representa una partida en curso y contiene campos como «gameStatus», «isRoundActive», «player1», «player2», «player1Cards», «player2Cards», «playedCardsInRound», «roundNumber», «lives» y «jokers». Esta colección, que es una subcolección de cada sala, permite mantener un historial de todas las partidas jugadas en esa sala y proporciona datos para futuras estadísticas de jugadores, como el número de partidas jugadas, entre otros.

```
export async function createGame(roomId, data) {
  const gamesRef = collection(db, roomCollection, roomId, gamesCollection);
  try {
    const gameRef = await addDoc(gamesRef, data);
    return { gameId: gameRef.id };
  } catch (error) {
    throw error;
  }
}

export async function updateGame(roomId, gameId, data) {
  const gamesRef = collection(db, roomCollection, roomId, gamesCollection);
  try {
    await updateDoc(doc(gamesRef, gameId), data);
  } catch (error) {
    throw error;
  }
```

Algoritmo 7.15: Llamadas para crear y actualizar partida

A la hora de obtener los cambios registrados, se utiliza nuevamente onSnapshot sobre el documento de la partida actual. En este caso, se ha implementado un filtro para gestionar únicamente los campos que han experimentado cambios. Esto se debe a que onSnapshot escucha el documento completo, y Firestore no ofrece la opción de escuchar un campo específico de un documento.

Por esta razón, se ha introdujo la creación del campo .actualGame.en el documento de la sala. Para el jugador 2, que no tiene conocimiento de las colecciones y documentos creados, sería imposible saber cuál es la partida que se ha creado. Sin embargo, gracias a la creación y actualización de este campo por parte del anfitrión al iniciar la partida, el jugador 2 puede detectar el cambio y escucharlo a través del ID proporcionado.

```
useEffect(() => {
    if (gameState.gameId) {
```

```
      const unsubscribe = onSnapshot(
        doc(db, 'rooms', roomState.roomId, 'games', gameState.gameId),
        (doc) => {
          const newData = doc.data();

          const changedFields = [];

          if (newData.gameStatus !== gameState.gameStatus) {
            changedFields.push('gameStatus');
          }

          if (newData.roundNumber !== gameState.roundNumber) {
            changedFields.push('roundNumber');
          }

          if (newData.lives !== gameState.lives) {
            changedFields.push('lives');
          }

          if (newData.jokers !== gameState.jokers) {
            changedFields.push('jokers');
          }

          if (newData.player1Cards !== gameState.player1Cards) {
            changedFields.push('player1Cards');
          }

          if (newData.player2Cards !== gameState.player2Cards) {
            changedFields.push('player2Cards');
          }

          if (newData.playedCardsInRound !== gameState.playedCardsInRound) {
            changedFields.push('playedCardsInRound');
          }

          if (changedFields.length > 0) {
            // Actions for the changed fields
            changedFields.forEach((field) => {
              switch (field) {...})}}
```

Algoritmo 7.16: Efecto para controlar los cambios en las partidas

CAPÍTULO 8
Producto desarrollado

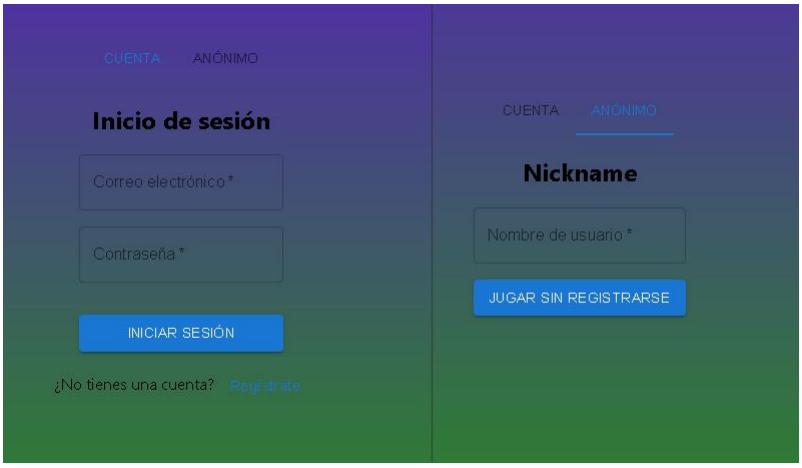


Figura 8.1: Captura de la página de Iniciar sesión.

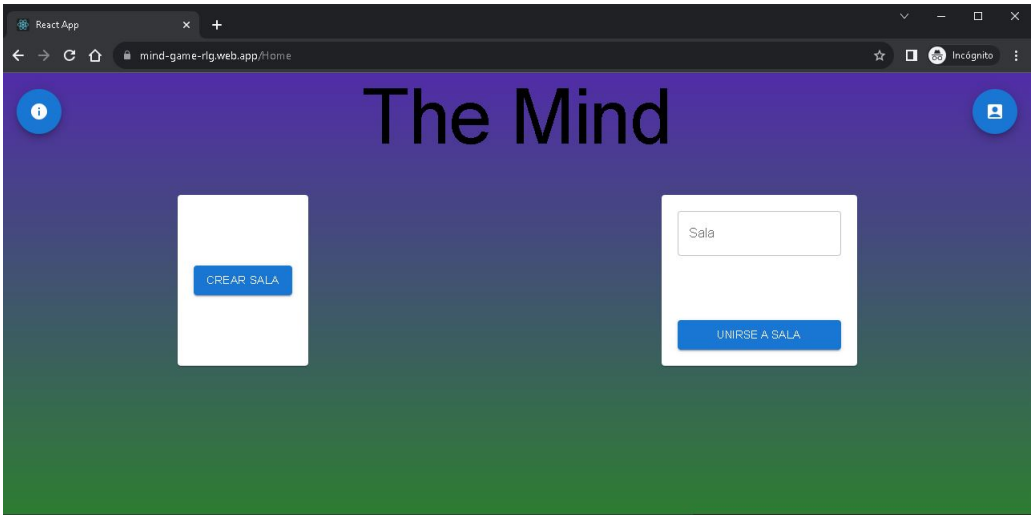


Figura 8.2: Captura de la página de Inicio.

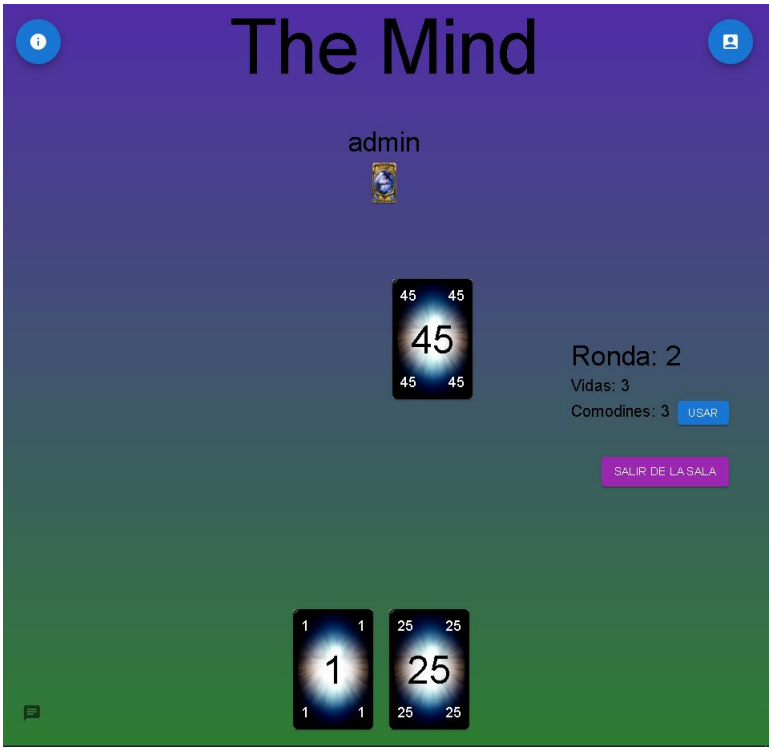


Figura 8.3: Captura de la sala de juego vista host.

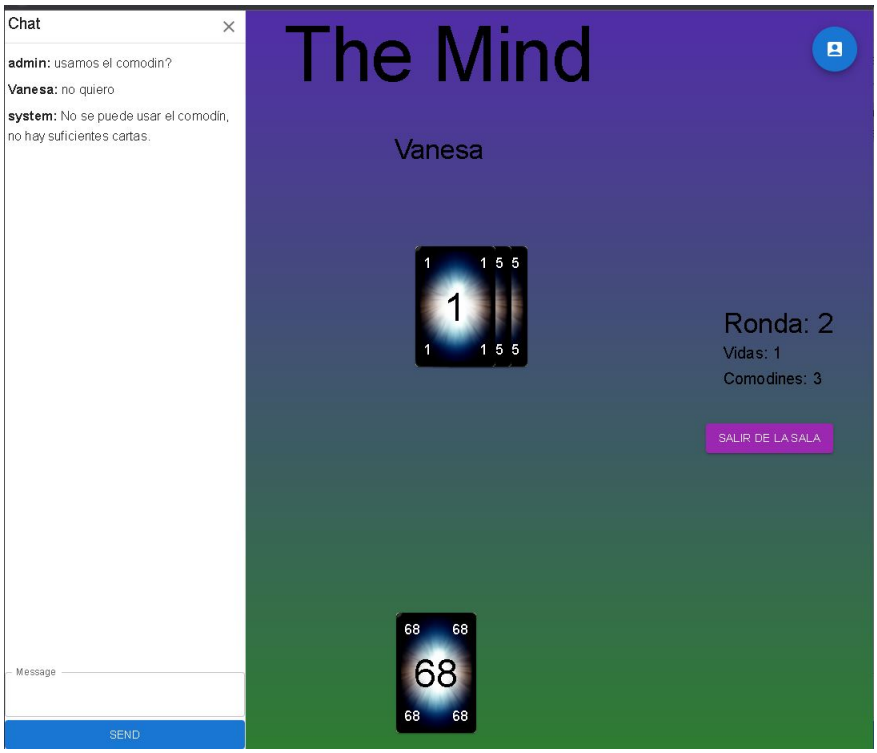


Figura 8.4: Captura de la sala de juego.

Figura 8.5: Animación del cursor sobre una carta.

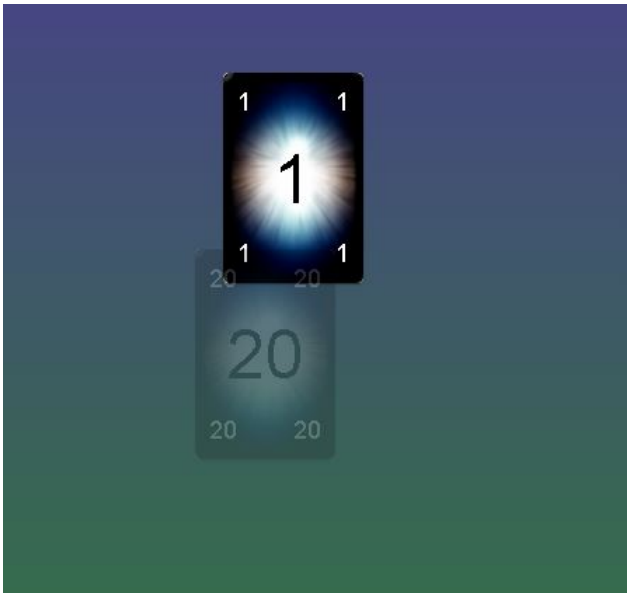


Figura 8.6: Animación de jugar carta.

8.1 Gráficas de uso

A continuacion se mostrarán imagenes de las gráficas que ofrece Firebase al desarrollador sobre el uso de su aplicación.

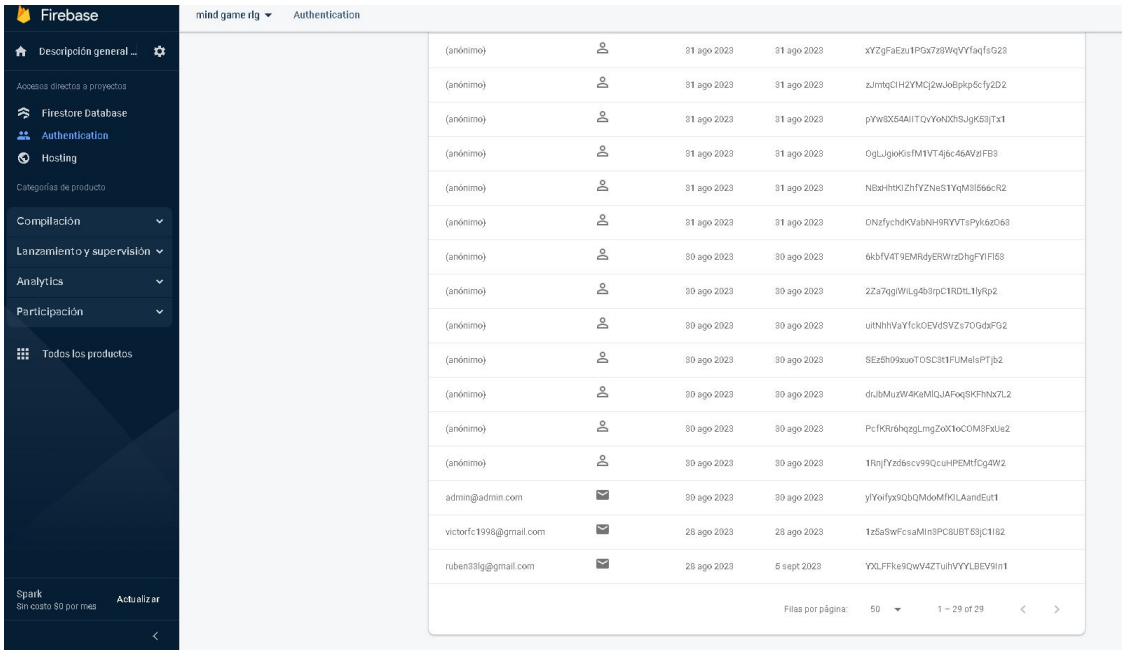


Figura 8.7: Datos de registro de usuarios.

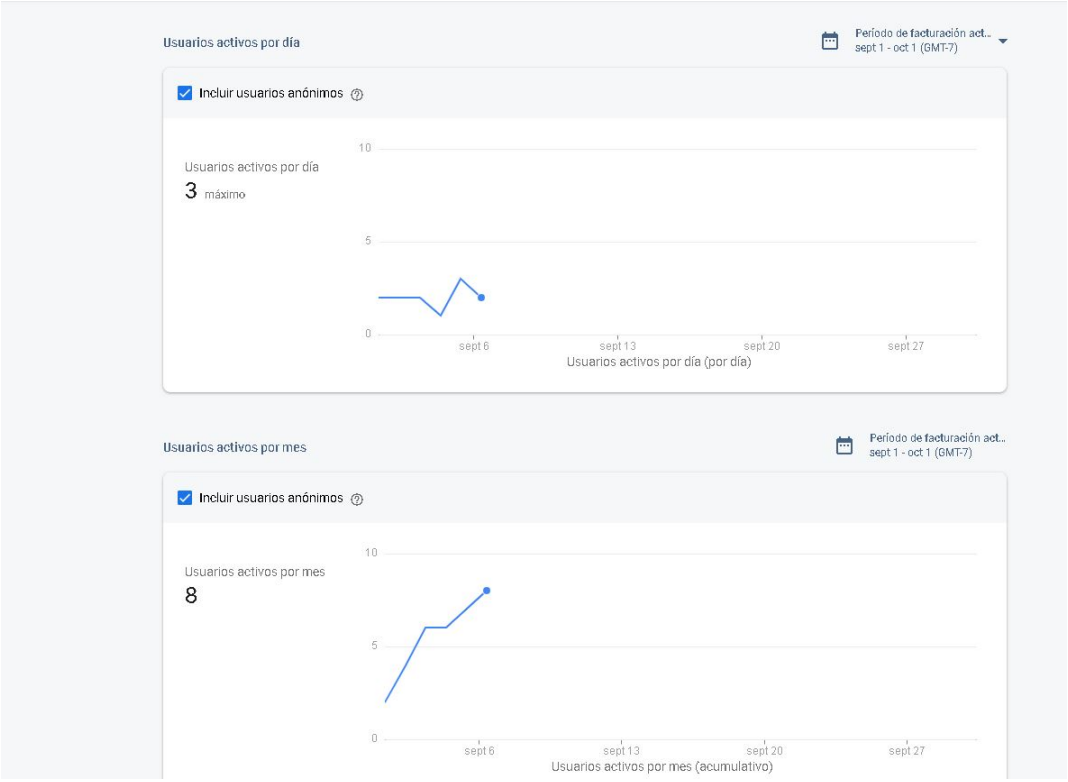


Figura 8.8: Graficas de uso de usuarios.

Figura 8.9: Grafica de uso de la página web.

Figura 8.10: Grafica de uso de la base de datos.

8.2 Ejemplo de partida

En esta sección se realizará un ejemplo mediante capturas de los diferentes estados por los que pasa una partida, incluida la victoria y la derrota. También se podrá ver cómo queda guardada dicha partida en la base de datos.

Figura 8.11: Vista anfitrion crear partida.

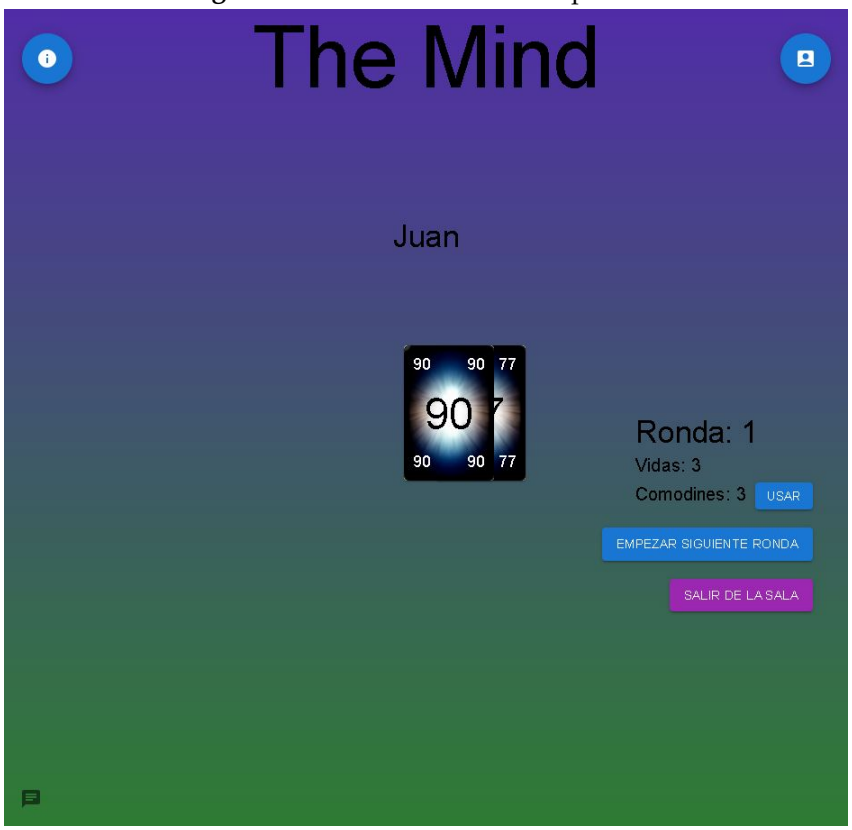


Figura 8.12: Vista anfitrion final ronda.

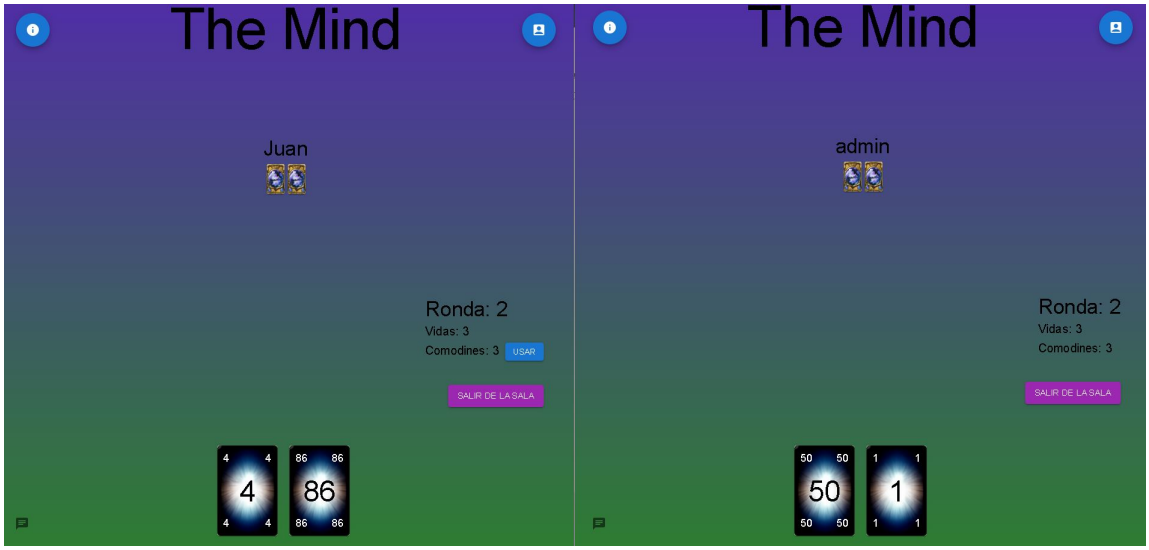


Figura 8.13: Estado anterior a comodin.

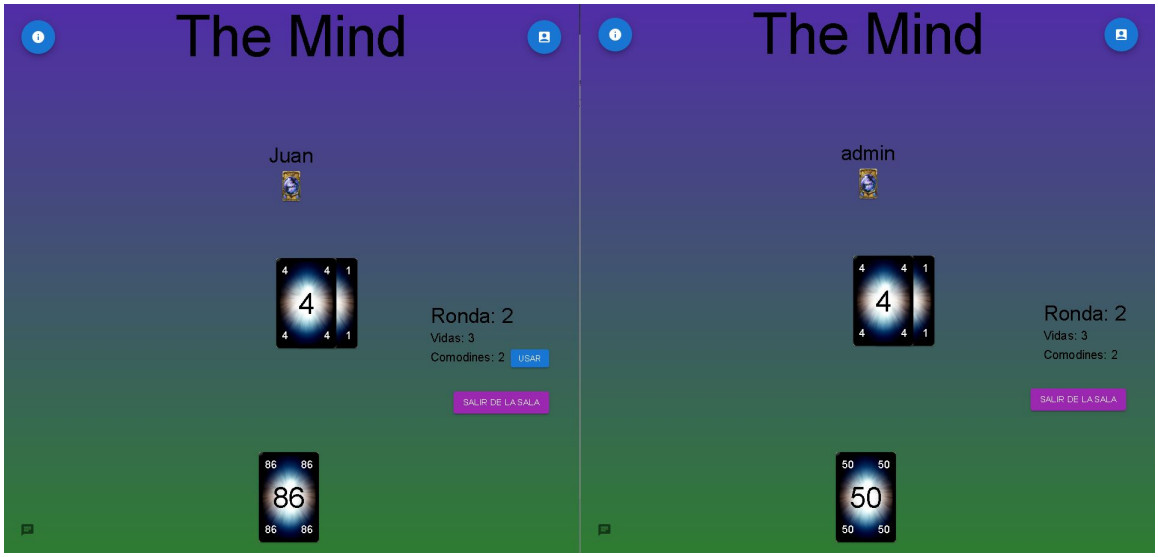


Figura 8.14: Uso de comodin.

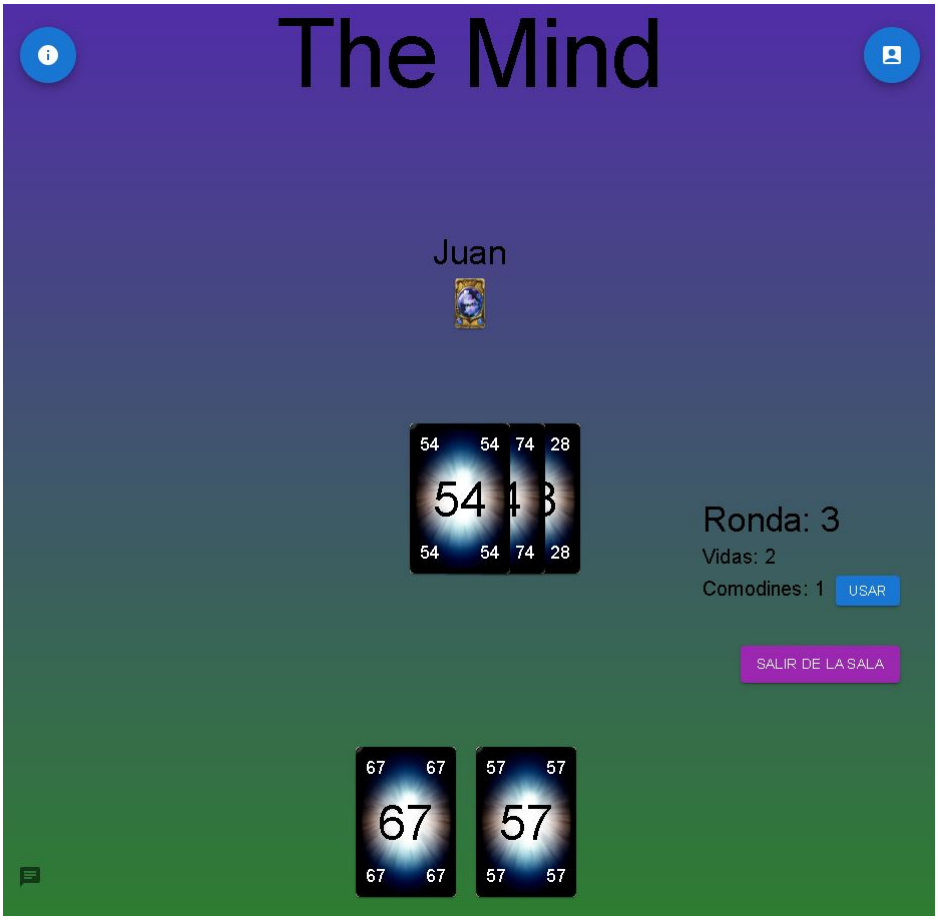


Figura 8.15: Perdida de vida.

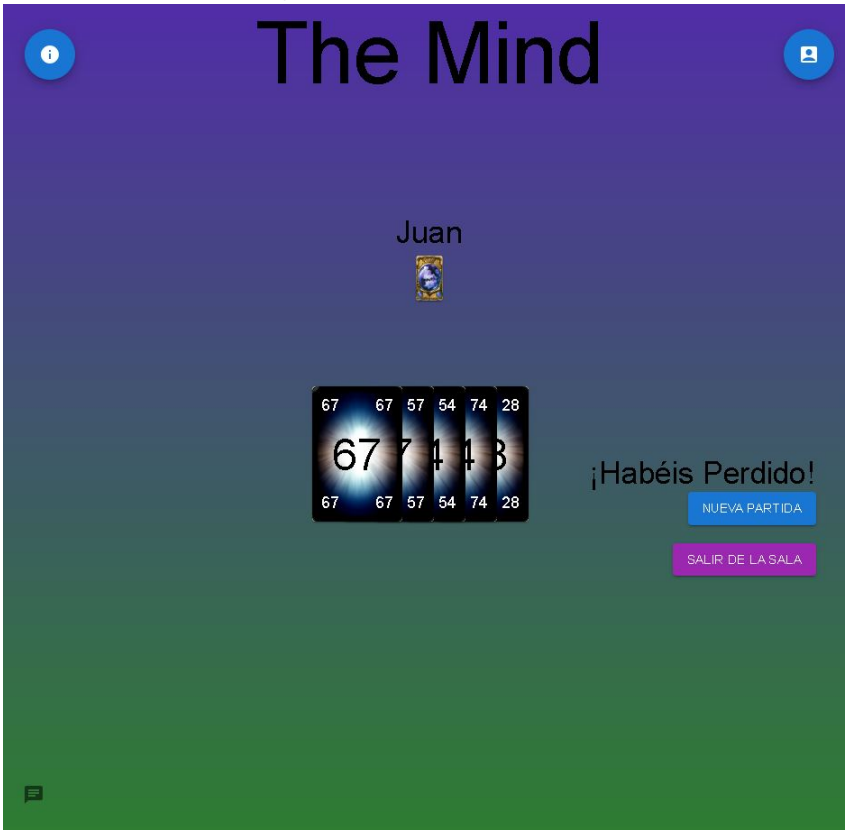


Figura 8.16: Partida perdida.

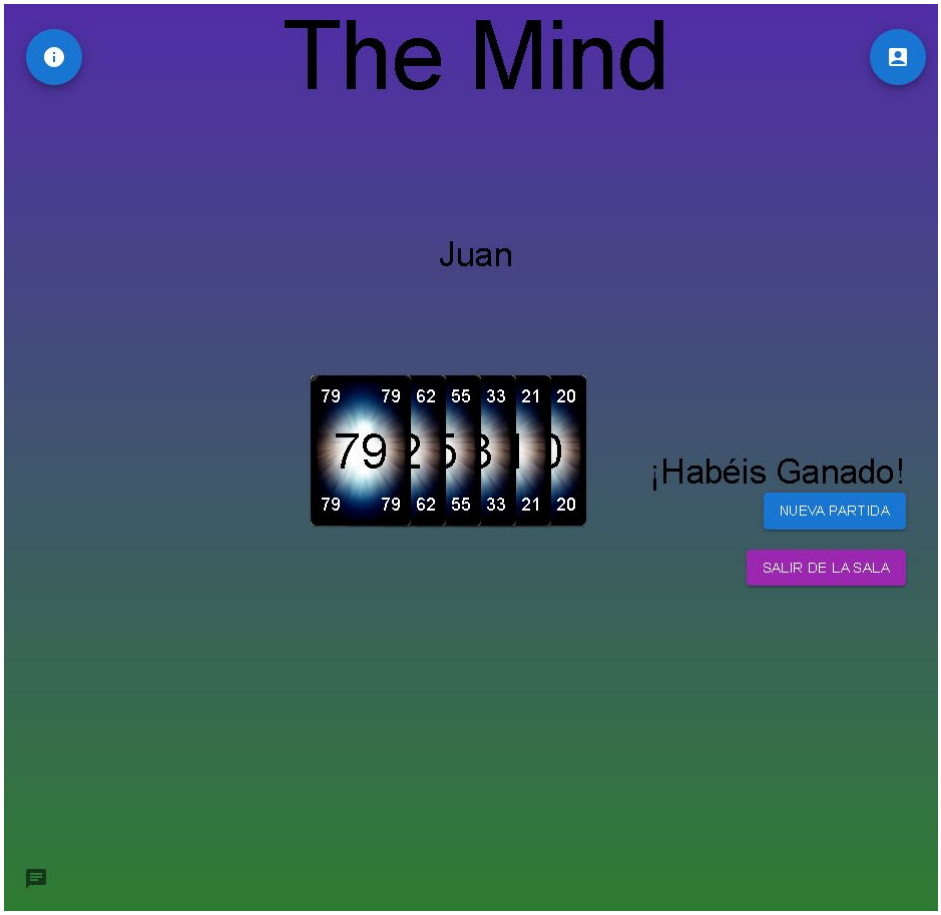


Figura 8.17: Partida ganada.

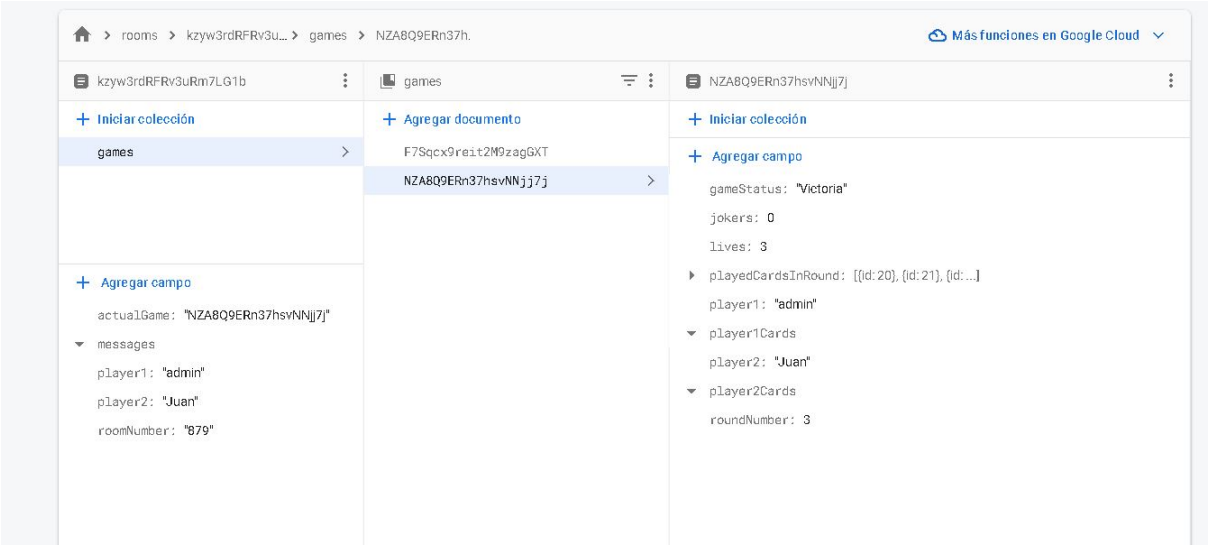


Figura 8.18: Imagen de la partida de ejemplo en la base de datos.

CAPÍTULO 9
Conclusiones

En el marco de este Trabajo de Fin de Grado (TFG), se ha logrado exitosamente el objetivo de desarrollar una aplicación web utilizando React y Firebase. La aplicación ha demostrado ser un Producto Mínimo Viable (MVP) funcional al proporcionar la capacidad de alojar la aplicación web y permitir que dos usuarios puedan crear salas y jugar en línea de manera eficiente.

9.1 Relación con el estudio y trabajo

En relación a mi formación académica, este proyecto me ha brindado la oportunidad de aplicar diversas técnicas que aprendí durante mi carrera. En el ámbito del backend, la gestión de bases de datos y llamadas, y la construcción de la misma. Por otro lado, pude poner en práctica distintas estrategias para el frontend. Por ejemplo, se implementó el patrón Singleton en los contextos diseñados, garantizando una gestión eficiente de los datos compartidos en la aplicación; o un método fábrica para la generación, barajado y distribución automática de cartas, entre otros patrones.

Durante mis prácticas en la empresa, tuve la oportunidad de trabajar en el mantenimiento de software existente y agregar pequeñas funcionalidades utilizando el framework de React. Esta experiencia fue fundamental para su elección como tecnología principal en este proyecto, y me ha permitido fortalecer mis habilidades y conocimientos en el uso de esta herramienta. En conjunto, mi formación académica y experiencia laboral han contribuido significativamente al éxito de este proyecto.

9.2 Opinión Personal

Mi experiencia con las herramientas utilizadas ha sido muy positiva. Aunque ya estaba familiarizado con React, este proyecto me brindó la oportunidad de perfeccionar significativamente mis conocimientos en esta tecnología. En cuanto a Firebase, mi impresión ha sido sumamente positiva. Ofrece servicios de alta calidad y su facilidad de uso es destacable. Firebase proporciona una amplia gama de herramientas que permiten a los desarrolladores tener un control detallado sobre muchos aspectos de la aplicación. Además, su documentación y las estadísticas de uso brindadas, como las gráficas, son de gran ayuda, enriqueciendo significativamente mi experiencia de desarrollo en este proyecto.

Si bien se ha conseguido lograr el objetivo de poder jugar en línea, ciertas tareas menos esenciales pero aún así importantes en un juego se han dejado atrás. Si tuviera que hacer el proyecto de nuevo, cambiaría algunas cosas, sobre todo habría tenido en cuenta

otra manera de desarrollar un backend más propio para así no tener ciertas limitaciones de Firebase. Tampoco habría separado tanto el desarrollo de frontend y backend, habiéndolos llevado un poco más a la par y desarrollando en ambos los elementos en común.

Para futuras mejoras y expansiones de la aplicación, se pueden considerar varias opciones. En primer lugar, es importante abordar tareas pendientes, como la implementación de sonidos y animaciones de victoria, así como la creación de un sistema de invitaciones y estadísticas de jugadores. Además, es posible explorar mejoras adicionales, como la incorporación de un sistema de ranking y la introducción de diferentes modos de juego. Dado que React y Firebase ofrecen una escalabilidad óptima, estas mejoras no requerirían un esfuerzo de desarrollo significativo y podrían enriquecer aún más la experiencia del usuario, permitiendo, por ejemplo, cambiar el orden de juego, habilitar partidas con múltiples montones y aumentar el número de jugadores, entre otras posibilidades.

Bibliografía

[1] Documentación de Firebase. Consultar en https://firebase.google.com/docs.

[2] Documentación de Firebase Hosting. Consultar en https://firebase.google.com/docs/hosting.

[3] Documentación de Firebase Authentication. Consultar en https://firebase.google.com/docs/auth.

[4] Documentación de Firebase Firestore. Consultar en https://firebase.google.com/docs/firestore.

[5] Documentación de React. Consultar en https://react.dev/.

[6] Documentación de Material ui. Consultar en https://mui.com/material-ui/.

[7] Documentación de Framer motion. Consultar en https://www.framer.com/motion/.

[8] Documentación de FlexBox. Consultar en https://css-tricks.com/snippets/css/a-guide-to-flexbox/.

[9] Juan Carlos Bravo Valentín. Estudio de las sinergias entre videojuegos y juegos de mesa. TFG. 2021. http://hdl.handle.net/20.500.12880/235.

[10] Tabletop Simulator. Página de Steam de Tabletop Simulator https://store.steampowered.com/app/286160/Tabletop_Simulator/.

[11] Mod The Mind para Tabletop Simulator. Página de Steam Workshop del mod. https://steamcommunity.com/sharedfiles/filedetails/?id=2270872298&searchtext=the+mind.

[12] The Mind by Wolfgang Warsch. Página de AppStore y PlayStore de The Mind https://apps.apple.com/us/app/the-mind/id1436132672. https://play.google.com/store/apps/details?id=de.brettspielwelt.themind&hl=en&gl=US.

[13] Ronald Humberto Chipana Wariste. Comparativa de las Librerías de Componentes de Bootstrap y Material-UI para React. 13-sep-2022. http://hdl.handle.net/123456789/33397.

[14] Luis Fernando Litano Ramos. Desarrollo de una aplicación móvil utilizando Flutter y Firebase para realizar el seguimiento de los tratamientos farmacológicos de un paciente. 2021. http://repositorio.unp.edu.pe/handle/20.500.12676/3014.

[15] Maida, EG, Pacienzia, J. Metodologías de desarrollo de software. Tesis de Licenciatura en Sistemas y Computación. Facultad de Química e Ingeniería "Fray Rogelio Bacon". Universidad Católica Argentina, 2015. Disponible en: https://repositorio.uca.edu.ar/handle/123456789/522

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**			x	
ODS 4. **Educación de calidad.**			x	
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**			x	
ODS 8. **Trabajo decente y crecimiento económico.**				x
ODS 9. **Industria, innovación e infraestructuras.**				x
ODS 10. **Reducción de las desigualdades.**				x
ODS 11. **Ciudades y comunidades sostenibles.**				x
ODS 12. **Producción y consumo responsables.**	x			
ODS 13. **Acción por el clima.**			x	
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

*E*n primera instancia, un proyecto como este puede parecer distante de los ODS, ya que se centra en la recreación y el entretenimiento más que en abordar cuestiones de sostenibilidad directamente. Sin embargo, al mirar más profundamente, se puede identificar una conexión sólida con el ODS 12: Producción y Consumo Responsables.

El ODS 12 busca promover prácticas de producción y consumo que sean más sostenibles y eficientes. Se trata de reducir el desperdicio, gestionar de manera responsable los recursos naturales y adoptar enfoques de producción y consumo que minimicen el impacto ambiental. En este contexto, la adaptación de juegos de mesa tradicionales a plataformas en línea se alinea perfectamente con este objetivo.

La creación de una versión web de un juego de cartas puede generar un aumento en su uso en línea, tanto por parte de jugadores habituales como de nuevos entusiastas del juego. Esto puede tener un impacto positivo en términos de consumo responsable. Al fomentar la utilización de la versión en línea del juego, se reduce la demanda de producir copias físicas del juego de cartas. Esto a su vez puede ayudar a prevenir la sobreproducción de estos juegos de mesa físicos, lo que, desde una perspectiva sostenible, es un paso en la dirección correcta. Evitar la tala innecesaria de árboles y el desperdicio de recursos utilizados en la fabricación de juegos de mesa es una contribución valiosa al ODS 12.

El impacto positivo no se limita a la reducción de la demanda de recursos naturales. La versión en línea de un juego de cartas también elimina la necesidad de embalajes de plástico y cartón utilizados en los juegos físicos. Esto, a su vez, ayuda a reducir la generación de residuos y disminuye la contaminación ambiental Esto último se podría vincular con otros ODS, como el ODS 14 (Vida Submarina) y el ODS 15 (Vida de Ecosistemas Terrestres), al contribuir a la preservación de los océanos y los hábitats terrestres.

Sin embargo, es importante reconocer que no todos los ODS tienen una conexión directa con este tipo de proyecto. Por ejemplo, los ODS 3 (Salud y Bienestar) o 4 (Educación de Calidad) pueden tener una conexión menos directa con la creación de una versión web de un juego de cartas. Aunque los juegos en línea pueden tener aspectos educativos y promover la interacción social, su contribución directa a estos objetivos puede ser limitada en comparación con otros enfoques.

Además, algunos ODS, como el ODS 7 (Energía Asequible y No Contaminante) y el ODS 13 (Acción por el Clima), pueden no tener una conexión inmediata con un juego en línea, ya que no afecta directamente al uso de energía o a la reducción de emisiones de carbono.

En conclusión, la creación de una versión web de un juego de cartas puede no parecer inicialmente relacionada con los Objetivos de Desarrollo Sostenible de las Naciones Unidas, pero al analizarla con mayor detalle, encontramos conexiones sólidas con el ODS 12 y otros objetivos relacionados con la sostenibilidad. Este proyecto ilustra cómo incluso las actividades de entretenimiento y recreación pueden contribuir de manera significativa a la promoción de prácticas más sostenibles en la industria y cómo cada esfuerzo, por pequeño que parezca, puede sumar en la búsqueda de un mundo más sostenible.