PROYECTO:

ChessGame

Manual de código

Realizado por: ***Miguel Ortega Álvarez***

Contenido

Manual de código

El manual de código es una guía fundamental para entender y trabajar con el código fuente de la aplicación. Proporciona una visión detallada de la estructura, organización y funcionamiento del código, lo que permite a los desarrolladores comprender rápidamente cómo interactúan los diferentes componentes y cómo realizar contribuciones efectivas al proyecto. En esta sección, se presentará una descripción general del manual de código, destacando su importancia y el alcance de la información proporcionada para facilitar el desarrollo, mantenimiento y colaboración en el proyecto.

API Backend (PHP)

Esta API se encarga de redirigir las peticiones a la base de datos dependiendo del endpoint que contenga la solicitud que proviene del API del front.

```php
// Funcion para conectarse a la base de datos
22 references
function get_connection()
{
    $dsn = 'mysql:host=localhost;dbname=chessdb';
    $user = 'root';
    $pass = '1234';
    $opciones = [];
    try {
        $con = new PDO($dsn, $user, $pass);
        $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    } catch (PDOException $e) {
        echo "Fallo la conexion: " . $e->getMessage();
    };
    return $con;
}
```

Esta es la función principal que asegura una conexión con la base de datos en local.

```php
// Función para autenticar usuario
1 reference
function autenticarUsuario($user, $contraseña)
{

    $con = get_connection();
    // Consulta SQL para autenticar al usuario
    $sql = "SELECT idusuarios, nombre, contraseña, perfil FROM usuarios WHERE (nombre=:user OR correo=:user) AND estado=1";
    $statement = $con->prepare($sql);
    $statement->bindParam(':user', $user);
    $statement->execute();
    $result = $statement->fetch(PDO::FETCH_ASSOC);

    if ($result) {
        // Verificar la contraseña
        if (password_verify("$contraseña", $result["contraseña"])) {
            $response = array("success" => true, "message" => "Inicio de sesión exitoso", "userId" => $result["idusuarios"]);
        } else {
            $response = array("success" => false, "message" => "Credenciales incorrectas");
        }
        header('Content-Type: application/json');
        echo json_encode($response);
    } else {
        $response = array("success" => false, "message" => "Credenciales incorrectas");
        header('Content-Type: application/json');
        echo json_encode($response);
    }
}
```

```php
//Funcion para subir un nuevo avatar
1 reference
function subirAvatar($avatar, $idUsuario)
{
    $con = get_connection();

    $parts = explode(',', $avatar);
    // Tomar la segunda parte que contiene la parte base64
    $base64Content = $parts[1];
    // Decodificar el avatar base64 para obtener los datos binarios
    $avatarBinary = base64_decode($base64Content);

    $sql = "UPDATE usuarios SET avatar = :avatar WHERE idusuarios = :idusuarios";
    $statement = $con->prepare($sql);
    $statement->bindParam(':avatar', $avatarBinary, PDO::PARAM_LOB);
    $statement->bindParam(':idusuarios', $idUsuario);
    $success = $statement->execute();

    if ($success) {
        // Avatar actualizado correctamente
        $response = array("success" => true, "message" => "Avatar actualizado correctamente");
    } else {
        // Error al subir el avatar
        $response = array("success" => false, "message" => "Error al subir el avatar");
    }

    header('Content-Type: application/json');
    echo json_encode($response);
}
```

```php
//Funcion obtener datos por id
1 reference
function obtenerDatos($idUsuario)
{
    $conn = get_connection();

    // Consulta para obtener los datos del usuario y el avatar por ID de usuario
    $sql = "SELECT nombre, contraseña, correo, fechaCreacion, avatar FROM usuarios WHERE idusuarios = :idUsuario";
    $stmt = $conn->prepare($sql);
    $stmt->bindValue(':idUsuario', $idUsuario);
    $stmt->execute();
    $userData = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($userData) {
        // Convertir el avatar blob a base64
        $avatarBlob = $userData['avatar'];
        $base64Avatar = base64_encode($avatarBlob);

        // Agregar el avatar en formato base64 al array de datos del usuario
        $userData['avatar'] = $base64Avatar;

        // Devolver los datos del usuario junto con el avatar en formato base64
        header('Content-Type: application/json');
        echo json_encode($userData);
    } else {
        // Usuario no encontrado
        http_response_code(404);
        echo json_encode(["message" => "Usuario no encontrado"]);
    }
}
```

```php
//Buscador de usuarios
1 reference
function buscarAmigosPorNombre($nombreAmigo)
{
    $conn = get_connection();
    $stmt = $conn->prepare("SELECT nombre, avatar, idusuarios FROM usuarios WHERE nombre LIKE :nombreAmigo");
    $stmt->bindValue(':nombreAmigo', "%$nombreAmigo%", PDO::PARAM_STR);
    $stmt->execute();

    // Inicializa un array para almacenar los amigos encontrados
    $amigos = array();

    // Itera sobre los resultados y agrega cada amigo al array de amigos
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        $nombre = $row["nombre"];
        $avatar = $row["avatar"];
        $id = $row["idusuarios"];
        $base64Avatar = base64_encode($avatar);

        // Agrega el amigo actual al array de amigos
        $amigos[] = array(
            "nombre" => $nombre,
            "avatar" => $base64Avatar,
            "id" => $id
        );
    }

    // Verifica si se encontraron resultados
    if (!empty($amigos)) {
        $response = array("success" => true, "message" => "Se encontraron resultados", "amigos" => $amigos);
    } else {
        $response = array("success" => false, "message" => "No se encontraron resultados para el nombre proporcionado");
    }

    // Devuelve los resultados como JSON
    header('Content-Type: application/json');
    echo json_encode($response);
}
```

Estas son algunas de las funciones mas sencillas de la aplicación. A continuación, mostrare las funciones relacionadas con la solicitud de partida, recuperación de la partida, abandonar partida, etc.

```php
// Funcion para enviar solicitud de partida
1 reference
function enviarSolicitudPartida($amigoId, $usuarioId, $tiempoPorJugada, $colorPiezasUsuario, $partidaPuntuada)
{
    try {
        if ($colorPiezasUsuario === 'Blancas') {
            $jugadorBlancas = $usuarioId;
            $jugadorNegras = $amigoId;
        } else {
            $jugadorNegras = $usuarioId;
            $jugadorBlancas = $amigoId;
        }
        $tiempoPorMovimiento = obtenerTiempoPorJugada($tiempoPorJugada);
        $estado_inicial = array(
            "position" => [
                [
                    ["wr", "wn", "wb", "wq", "wk", "wb", "wn", "wr"],
                    ["wp", "wp", "wp", "wp", "wp", "wp", "wp", "wp"],
                    ["", "", "", "", "", "", "", ""],
                    ["", "", "", "", "", "", "", ""],
                    ["", "", "", "", "", "", "", ""],
                    ["", "", "", "", "", "", "", ""],
                    ["bp", "bp", "bp", "bp", "bp", "bp", "bp", "bp"],
                    ["br", "bn", "bb", "bq", "bk", "bb", "bn", "br"]
                ]
            ],
            "turn" => "w",
            "candidateMoves" => [],
            "movesList" => [],
            "promotionSquare" => null,
            "status" => "Ongoing",
            "castleDirection" => ["w" => "both", "b" => "both"]
        );
        $estado = json_encode($estado_inicial);

        if ($partidaPuntuada == '') {
            $partidaPuntuada = 0;
        }
        $fechaPartida = date('Y-m-d H:i:s');
        $cancelada = 0;
        $terminada = 0;

        $conn = get_connection();
        // Insertar una nueva solicitud de amistad
        $stmt = $conn->prepare("INSERT INTO partidas (jugador_blancas, jugador_negras, estado, tiempo_por_jugada, clasificada, fechaPartida, canc
        $stmt->bindParam(':jugador_blancas', $jugadorBlancas);
        $stmt->bindParam(':jugador_negras', $jugadorNegras);
        $stmt->bindParam(':estado', $estado);
        $stmt->bindParam(':tiempo_por_jugada', $tiempoPorMovimiento);
        $stmt->bindParam(':clasificada', $partidaPuntuada);
        $stmt->bindParam(':fechaPartida', $fechaPartida);
        $stmt->bindParam(':cancelada', $cancelada);
        $stmt->bindParam(':terminada', $terminada);

        $stmt->execute();
        $idpartida = $conn->lastInsertId();

        if ($stmt->rowCount() > 0) {
            $partida[] = array(
                "idpartida" => $idpartida,
                "estado" => $estado_inicial,
            );
            $response = array("success" => true, "message" => "Solicitud de partida enviada correctamente", "partida" => $partida);
        } else {
            $response = array("success" => false, "message" => "Error al enviar la solicitud de partida");
        }
    } catch (PDOException $e) {
        $response = array("success" => false, "message" => "Error al enviar la solicitud de partida: " . $e->getMessage());
    }
    // Devuelve los resultados como JSON
    header('Content-Type: application/json');
    echo json_encode($response);
}
```

Esta función es la encargada de crear una partida con el estado inicial.

```php
// Funcion para obtener la partida
1 reference
function obtenerPartida($idpartida)
{
    $response = array();
    try {
        $conn = get_connection();

        // Obtener la partida seleccionada
        $stmt = $conn->prepare("SELECT idpartida, jugador_blancas, jugador_negras, tiempo_por_jugada, clasificada, estado, fechaPartida, ultimoMo
        $stmt->bindParam(':partida_id', $idpartida);
        $stmt->execute();

        // Verificar si se actualizó correctamente
        if ($stmt->rowCount() > 0) {
            while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
                // Obtener id de partida
                $idpartida = $row['idpartida'];
                // Decodificar el estado de la partida
                $estado_partida = json_decode($row['estado'], true);

                // Obtener jugadores
                $jugador_blancas = $row['jugador_blancas'];
                $jugador_negras = $row['jugador_negras'];

                // Obtener la fecha actual
                $fecha_actual = new DateTime();

                // Calcular la fecha límite de movimiento para el jugador
                $fecha_limite = new DateTime($row['fechaPartida']);
                $fecha_limite->add(new DateInterval('P' . $row['tiempo_por_jugada'] . 'D'));

                if (!empty($row['ultimoMovimientoBlancas']) && !empty($row['ultimoMovimientoNegras'])) {
                    $ultimo_movimiento = max($row['ultimoMovimientoBlancas'], $row['ultimoMovimientoNegras']);

                    // Calcular la fecha límite basada en el último movimiento
                    $fecha_limite = new DateTime($ultimo_movimiento);
                    $fecha_limite->add(new DateInterval('P' . $row['tiempo_por_jugada'] . 'D'));
                }

                // Calcular los días restantes para el jugador
                $dias_restantes = $fecha_actual->diff($fecha_limite)->days;
                $partida = array(
                    "idpartida" => $idpartida,
                    "estado" => $estado_partida,
                    "dias_restantes" => $dias_restantes,
                    "jugador_blancas" => $jugador_blancas,
                    "jugador_negras" => $jugador_negras
                );
            }
            if (!empty($partida)) {
                $response = array("success" => true, "message" => "Visualizar partida correcto", "partidas" => $partida);
            } else {
                $response = array("success" => false, "message" => "Error al cargar las partidas activas");
            }
        } else {
            $response = array("success" => false, "message" => "El usuario no tiene partidas activas");
        }
    } catch (PDOException $e) {
        $response = array("success" => false, "message" => "Error al aceptar usuario: " . $e->getMessage());
    }
    // Devuelve los resultados como JSON
    header('Content-Type: application/json');
    echo json_encode($response);
}
```

Esta función es la encargada de obtener una partida al pulsar sobre la partida en el listado de partidas.

```php
// Funcion al confirmar un movimiento
1 reference
function actualizarPartida($idpartida, $usuarioId, $nuevoEstado)
{
    $response = array();
    try {
        $conn = get_connection();

        // Seleccionar la partida que se está jugando
        $stmt = $conn->prepare("SELECT idpartida, jugador_blancas, jugador_negras, tiempo_por_jugada, clasificada, estado, fechaPartida, ultimoMo
        $stmt->bindParam(':partida_id', $idpartida);
        $stmt->execute();

        if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            // Obtener jugadores y último movimiento
            $jugadorBlancas = $row['jugador_blancas'];
            $jugadorNegras = $row['jugador_negras'];
            $ultimoMovimientoBlancas = $row['ultimoMovimientoBlancas'];
            $ultimoMovimientoNegras = $row['ultimoMovimientoNegras'];

            // Actualizar el estado de la partida en la base de datos
            $estado = json_encode($nuevoEstado);
            $stmt = $conn->prepare("UPDATE partidas SET estado = :nuevoEstado WHERE idpartida = :partida_id");
            $stmt->bindParam(':partida_id', $idpartida);
            $stmt->bindParam(':nuevoEstado', $estado);
            $stmt->execute();

            // Verificar si se actualizó correctamente
            if ($stmt->rowCount() > 0) {
                // Actualizar el último movimiento según el usuario
                if ($usuarioId == $jugadorBlancas) {
                    $ultimoMovimientoBlancas = date('Y-m-d H:i:s');
                } elseif ($usuarioId == $jugadorNegras) {
                    $ultimoMovimientoNegras = date('Y-m-d H:i:s');
                }
```

```php
                // Actualizar último movimiento en la base de datos
                $stmt = $conn->prepare("UPDATE partidas SET ultimoMovimientoBlancas = :ultimoMovimientoBlancas, ultimoMovimientoNegras = :ultimoM
                $stmt->bindParam(':partida_id', $idpartida);
                $stmt->bindParam(':ultimoMovimientoBlancas', $ultimoMovimientoBlancas);
                $stmt->bindParam(':ultimoMovimientoNegras', $ultimoMovimientoNegras);
                $stmt->execute();

                $response = array("success" => true, "message" => "Partida actualizada", "estado" => json_decode($estado));
            } else {
                $response = array("success" => false, "message" => "No se pudo actualizar la partida");
            }
        }
    } catch (PDOException $e) {
        $response = array("success" => false, "message" => "Error al actualizar la partida: " . $e->getMessage());
    }
    // Devuelve los resultados como JSON
    header('Content-Type: application/json');
    echo json_encode($response);
}
```

Esta función se ejecuta al confirmar un movimiento, almacena las fechas del movimiento y el estado de la partida.

```php
// Funcion para abandonar la partida
1 reference
function abandonarPartida($idpartida, $usuarioId, $estado)
{
    $response = array();
    try {
        $conn = get_connection();

        // Verificar si la partida tiene más de dos movimientos
        if (count($estado['position']) > 3) {
            // Actualizar tabla partidas: establecer columnas cancelada y terminada a 1
            $stmt = $conn->prepare("UPDATE partidas SET cancelada = 1, terminada = 1 WHERE idpartida = :partida_id");
            $stmt->bindParam(':partida_id', $idpartida);
            $stmt->execute();

            // Obtener información sobre si la partida es clasificada
            $esClasificada = esPartidaClasificada($idpartida);

            // Consultar los jugadores en la tabla partidas
            $stmt = $conn->prepare("SELECT jugador_blancas, jugador_negras FROM partidas WHERE idpartida = :partida_id");
            $stmt->bindParam(':partida_id', $idpartida);
            $stmt->execute();

            if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
                $jugador_blancas = $row['jugador_blancas'];
                $jugador_negras = $row['jugador_negras'];
            }
            // Determinar qué jugador ha abandonado la partida
            $jugador_abandonado = ($jugador_blancas == $usuarioId) ? 'jugador_blancas' : 'jugador_negras';
            $jugador_permanece = ($jugador_abandonado == 'jugador_blancas') ? 'jugador_negras' : 'jugador_blancas';
            // Obtener el ID del jugador que permanece
            $jugador_permanece_id = ($jugador_permanece == 'jugador_blancas') ? $jugador_blancas : $jugador_negras;
            // Si la partida es clasificada, realizar operaciones adicionales
            if ($esClasificada) {
                // OBTENER PUNTUACIONES DE LOS JUGADORES
                // Consulta para obtener las stats del usuario por ID de usuario
                $sql = "SELECT derrotas, empates, victorias, puntuacion FROM statsusuarios WHERE idUsuario = :idUsuario";
                $stmt = $conn->prepare($sql);
                $stmt->bindValue(':idUsuario', $usuarioId);
                $stmt->execute();
                $userData = $stmt->fetch(PDO::FETCH_ASSOC);

                if ($userData) {
                    $puntuacionIdUsuario = $userData['puntuacion'];
                }

                // Consulta para obtener las stats del rival por ID de rival
                $sql = "SELECT derrotas, empates, victorias, puntuacion FROM statsusuarios WHERE idUsuario = :idUsuario";
                $stmt = $conn->prepare($sql);
                $stmt->bindValue(':idUsuario', $jugador_permanece_id);
                $stmt->execute();
                $userData = $stmt->fetch(PDO::FETCH_ASSOC);

                if ($userData) {
                    $puntuacionJugadorPermanece = $userData['puntuacion'];
                }

                // SISTEMA DE ELO
                if ($puntuacionIdUsuario > $puntuacionJugadorPermanece) {
                    $puntos = calcularPuntosSuma($puntuacionIdUsuario, $puntuacionJugadorPermanece);
                } else {
                    $puntos = calcularPuntosResta($puntuacionIdUsuario, $puntuacionJugadorPermanece);
                }

                // Restar los puntos en el campo puntuación de la tabla statsUsuarios
                $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion - $puntos WHERE idUsuario = :usuario_id");
                $stmt->bindParam(':usuario_id', $usuarioId);
                $stmt->execute();

                // Sumar los puntos en el campo puntuación de la tabla statsUsuarios para el jugador que no ha abandonado
                $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion + $puntos WHERE idUsuario = :jugador_id");
                $stmt->bindParam(':jugador_id', $$jugador_permanece);
                $stmt->execute();
            }
            // Actualizar la tabla statsUsuarios: sumar +1 al campo derrota en el usuarioId
            $stmt = $conn->prepare("UPDATE statsUsuarios SET derrotas = derrotas + 1 WHERE idUsuario = :usuario_id");
            $stmt->bindParam(':usuario_id', $usuarioId);
            $stmt->execute();

            // Actualizar la tabla statsUsuarios: sumar +1 al campo victorias en el jugador que no ha abandonado
            $stmt = $conn->prepare("UPDATE statsUsuarios SET victorias = victorias + 1 WHERE idUsuario = :jugador_id");
            $stmt->bindParam(':jugador_id', $$jugador_permanece);
            $stmt->execute();
        } else {
            // Si la partida tiene menos de dos movimientos, establecer columnas cancelada y terminada a 1
            $stmt = $conn->prepare("UPDATE partidas SET cancelada = 1, terminada = 1 WHERE idpartida = :partida_id");
            $stmt->bindParam(':partida_id', $idpartida);
            $stmt->execute();
        }

        // Verificar si se actualizó correctamente
        if ($stmt->rowCount() > 0) {
            $response = array("success" => true, "message" => "La partida se abandonó correctamente");
        } else {
            $response = array("success" => false, "message" => "No se pudo abandonar la partida");
        }
    } catch (PDOException $e) {
        $response = array("success" => false, "message" => "Error al abandonar la partida: " . $e->getMessage());
    }

    // Devuelve los resultados como JSON
    header('Content-Type: application/json');
    echo json_encode($response);
}
```

Esta función se encarga de marcar una partida como cancelada o terminada y de administrar los puntos en función de los movimientos de la partida y si es clasificada o no. Esta función solo se ejecuta cuando el usuario pulsa en abandonar partida.

Al igual que existe esta función, también existe la función de abandonar partida por tiempo, que se ejecuta cuando uno de los dos usuarios se queda sin tiempo para mover.

Por último, la función finalizar partida, que se ejecuta cuando uno de los usuarios hace jaque mate dentro de una partida.

```php
// Al hacer jaque mate
1 reference
function finalizarPartida($idPartida, $colorPieza, $endGameReason)
{
    $response = array();
    try {
        $conn = get_connection();

        // Actualizar tabla partidas: establecer columnas terminada a 1
        $stmt = $conn->prepare("UPDATE partidas SET terminada = 1 WHERE idpartida = :partida_id");
        $stmt->bindParam(':partida_id', $idPartida);
        $stmt->execute();

        // Obtener información sobre si la partida es clasificada
        $esClasificada = esPartidaClasificada($idPartida);

        // Consultar los jugadores en la tabla partidas
        $stmt = $conn->prepare("SELECT jugador_blancas, jugador_negras FROM partidas WHERE idpartida = :partida_id");
        $stmt->bindParam(':partida_id', $idPartida);
        $stmt->execute();

        if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $jugador_blancas = $row['jugador_blancas'];
            $jugador_negras = $row['jugador_negras'];
        }
        // Consultar la puntuacion de los jugadores
        $stmt = $conn->prepare("SELECT puntuacion FROM statsusuarios WHERE idUsuario = :jugador_id_blancas");
        $stmt->bindParam(':jugador_id_blancas', $jugador_blancas);
        $stmt->execute();
        if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $puntuacionBlancas = $row['puntuacion'];
        }
        $stmt = $conn->prepare("SELECT puntuacion FROM statsusuarios WHERE idUsuario = :jugador_id_negras");
        $stmt->bindParam(':jugador_id_negras', $jugador_negras);
        $stmt->execute();
        if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $puntuacionNegras = $row['puntuacion'];
        }
```

```php
    // Si la partida es clasificada, realizar operaciones adicionales
    if ($esClasificada) {
        if ($endGameReason == 'insufficientMaterial' || $endGameReason == 'stalemate') {
            // SISTEMA DE ELO
            $puntos = calcularPuntosEmpate($puntuacionBlancas, $puntuacionNegras);
            if ($puntos < 0) {
                $puntosBlancas = abs($puntos);
                // Sumar los puntos en el campo puntuación de la tabla statsUsuarios para el jugador de blancas
                $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion + $puntosBlancas WHERE idUsuario = :jugador_id_blan
                $stmt->bindParam(':jugador_id_blancas', $jugador_blancas);
                $stmt->execute();

                // Sumar los puntos en el campo puntuación de la tabla statsUsuarios para el jugador de negras
                $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion + $puntos WHERE idUsuario = :jugador_id_negras");
                $stmt->bindParam(':jugador_id_negras', $jugador_negras);
                $stmt->execute();
            } else {
                $puntosNegras = abs($puntos);
                // Sumar los puntos en el campo puntuación de la tabla statsUsuarios para el jugador de blancas
                $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion - $puntos WHERE idUsuario = :jugador_id_blancas");
                $stmt->bindParam(':jugador_id_blancas', $jugador_blancas);
                $stmt->execute();

                // Sumar los puntos en el campo puntuación de la tabla statsUsuarios para el jugador de negras
                $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion + $puntosNegras WHERE idUsuario = :jugador_id_negra
                $stmt->bindParam(':jugador_id_negras', $jugador_negras);
                $stmt->execute();
            }
            //Agregar empate en la tabla de stats para los jugadores
            $stmt = $conn->prepare("UPDATE statsUsuarios SET empates = empates + 1 WHERE idUsuario = :jugador_id_negras OR idUsuario = :juga
            $stmt->bindParam(':jugador_id_negras', $jugador_negras);
            $stmt->bindParam(':jugador_id_blancas', $jugador_blancas);
            $stmt->execute();
        } else { //$endGameReason = "CheckMate"
            // Calcular puntos y otorgarlos al jugador correspondiente
            $ganador = ($colorPieza == 'w') ? $jugador_blancas : $jugador_negras;
            $perdedor = ($ganador == $jugador_blancas) ? $jugador_negras : $jugador_blancas;
```

```php
            // SISTEMA DE ELO
            if ($puntuacionBlancas > $puntuacionNegras) {
                $puntos = calcularPuntosSuma($puntuacionBlancas, $puntuacionNegras);
            } else {
                $puntos = calcularPuntosResta($puntuacionBlancas, $puntuacionNegras);
            }

            // Restar los puntos en el campo puntuación de la tabla statsUsuarios
            $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion - $puntos WHERE idUsuario = :perdedor");
            $stmt->bindParam(':perdedor', $perdedor);
            $stmt->execute();

            // Sumar los puntos en el campo puntuación de la tabla statsUsuarios para el jugador que gana
            $stmt = $conn->prepare("UPDATE statsUsuarios SET puntuacion = puntuacion + $puntos WHERE idUsuario = :ganador");
            $stmt->bindParam(':ganador', $ganador);
            $stmt->execute();

            // Actualizar la tabla statsUsuarios: sumar +1 al campo derrota en el usuarioId
            $stmt = $conn->prepare("UPDATE statsUsuarios SET derrotas = derrotas + 1 WHERE idUsuario = :perdedor");
            $stmt->bindParam(':perdedor', $perdedor);
            $stmt->execute();

            // Actualizar la tabla statsUsuarios: sumar +1 al campo victorias en el jugador que no ha abandonado
            $stmt = $conn->prepare("UPDATE statsUsuarios SET victorias = victorias + 1 WHERE idUsuario = :ganador");
            $stmt->bindParam(':ganador', $ganador);
            $stmt->execute();
        }
    } else { //No clasificada
        if ($endGameReason == 'insufficientMaterial' || $endGameReason == 'stalemate') {
            //Agregar empate en la tabla de stats para los jugadores
            $stmt = $conn->prepare("UPDATE statsUsuarios SET empates = empates + 1 WHERE idUsuario = :jugador_id_negras OR idUsuario = :jugad
            $stmt->bindParam(':jugador_id_negras', $jugador_negras);
            $stmt->bindParam(':jugador_id_blancas', $jugador_blancas);
            $stmt->execute();
```

```php
        } else {
            // Actualizar la tabla statsUsuarios: sumar +1 al campo derrota en el usuarioId
            $stmt = $conn->prepare("UPDATE statsUsuarios SET derrotas = derrotas + 1 WHERE idUsuario = :perdedor");
            $stmt->bindParam(':perdedor', $perdedor);
            $stmt->execute();

            // Actualizar la tabla statsUsuarios: sumar +1 al campo victorias en el jugador que no ha abandonado
            $stmt = $conn->prepare("UPDATE statsUsuarios SET victorias = victorias + 1 WHERE idUsuario = :ganador");
            $stmt->bindParam(':ganador', $ganador);
            $stmt->execute();
        }
    }

    // Verificar si se actualizó correctamente
    if ($stmt->rowCount() > 0) {
        $response = array("success" => true, "message" => "La partida se termino correctamente");
    } else {
        $response = array("success" => false, "message" => "No se pudo terminar la partida");
    }
} catch (PDOException $e) {
    $response = array("success" => false, "message" => "Error al terminar la partida: " . $e->getMessage());
}

// Devuelve los resultados como JSON
header('Content-Type: application/json');
echo json_encode($response);
}
```

API Frontend (Js)

Esta API se encarga de almacenar la lógica de las funciones generadas de todos los componentes. Esta realizada en JavaScript y se comunica con el API del backend mediante peticiones axios.

En el caso de trabajar en local con la aplicación, la ruta de las solicitudes quedaría así:

```
import axios from 'axios';

const BASE_URL = 'https://localhost:443';

export async function autentificarUsuario(nombre, contrasena) {
  try {
    const response = await axios.post(`${BASE_URL}/PROYECTO/chess-main/backend/api.php/login`, { nombre, contrasena });
    return response.data;
  } catch (error) {
    console.error('Error al obtener datos:', error);
    throw error;
  }
};
```

Pero esto se debe cambiar al subirlo a un dominio, indicando en la constante BASE_URL el dominio en cuestión y modificando la ruta de las carpetas en caso necesario.

Algunas de las comprobaciones como el tamaño del avatar, los requisitos de la contraseña, el correo etc, se realizan en este apartado para no sobrecargar el servidor.

```
export async function subirAvatar(avatar, idUsuario) {
  try {
    // Validar el tamaño del avatar (entre 0 y 10 MB)
    const maxSize = 10 * 1024 * 1024; // 10 MB en bytes
    if (avatar.size > maxSize) {
      throw new Error('El tamaño del avatar no puede superar los 10 MB');
    }

    // Validar el tipo de archivo del avatar (jpg, jpeg o png)
    const allowedTypes = ['image/jpeg', 'image/png', 'image/jpg'];
    if (!allowedTypes.includes(avatar.type)) {
      throw new Error('El avatar debe ser un archivo de tipo JPG, JPEG o PNG');
    }

    const avatarData = await convertFileToBase64(avatar);
    const response = await axios.post(`${BASE_URL}/PROYECTO/chess-main/backend/api.php/subirAvatar`, { avatarData, idUsuario });
    return response.data;
  } catch (error) {
    console.error('Error al subir avatar:', error);
    throw error;
  }
}

function convertFileToBase64(file) {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => resolve(reader.result);
    reader.onerror = error => reject(error);
  });
}
```

```
export async function agregarAmigo(amigoId, usuarioId) {
  try {
    const response = await axios.post(`${BASE_URL}/PROYECTO/chess-main/backend/api.php/agregarAmigo`, { amigoId, usuarioId });
    return response.data;
  } catch (error) {
    console.error('Error al buscar amigos por nombre:', error);
    throw error;
  }
};

export async function borrarAmigo(amigoId, usuarioId) {
  try {
    const response = await axios.post(`${BASE_URL}/PROYECTO/chess-main/backend/api.php/borrarAmigo`, { amigoId, usuarioId });
    return response.data;
  } catch (error) {
    console.error('Error al encontrar amigos por id:', error);
    throw error;
  }
};

export async function obtenerAmigosUsuario(usuarioId) {
  try {
    const response = await axios.post(`${BASE_URL}/PROYECTO/chess-main/backend/api.php/obtenerAmigos`, { usuarioId });
    return response.data;
  } catch (error) {
    console.error('Error al encontrar amigos por id:', error);
    throw error;
  }
};
```

Pero en general es una forma de administrar las funciones en un mismo fichero, sin necesidad de sobrecargar los componentes con la lógica de las funciones.

Componentes (React)

En este apartado mostraré algunos de los componentes de la aplicación, como puede ser el componente Home o el componente Board.

```jsx
import React, { useState, useEffect } from 'react';
import SideMenu from '../SideMenu/SideMenu';
import './home.css';
import { useNavigate } from 'react-router-dom';
import { obtenerDatos } from '../../services/apiService';
import { obtenerPartidas } from '../../services/apiService';


const Home = () => {
  const [loading, setLoading] = useState(true);
  const navigate = useNavigate();
  const idUsuario = localStorage.getItem('userId');
  const [errorPartidas, setErrorPartidas] = useState('');
  const [partidasUsuario, setPartidasUsuario] = useState([]);

  useEffect(() => {
    if (!idUsuario) {
      navigate('/');
    } else {
      const fetchDatos = async () => {
        try {
          const response = await obtenerDatos(idUsuario);
          console.log(response);
          if (response) {
            setLoading(false);
          }
        } catch (error) {
          console.error('Error fetching datos:', error);
        }
      };

      const obtenerPartidasUsuarios = async () => {
        try {
          const response = await obtenerPartidas(idUsuario);
          console.log('RESPUESTA DE PARTIDAS');
          console.log(response.partidas);
          if (response.success) {
            setPartidasUsuario(response.partidas);
```

```jsx
          } else {
            setErrorPartidas(response.message);
          }
        } catch (error) {
          setErrorPartidas(error.message);
        }

      };
      fetchDatos();
      obtenerPartidasUsuarios();
    }
  }, [idUsuario, navigate]);

  const handleBoardClick = (estado, idPartida) => {
    navigate('/board', { state: { estado, idPartida } });
  };

  return (
    <div className="root">
      <SideMenu loading={loading} />
      <main className="content">
        <div className="column">
          <div className="listTitle">Listado de Partidas Activas</div>
          {errorPartidas && <p className="errorMessage">{errorPartidas}</p>}
          <ul className="resultList gameList">
            {partidasUsuario && partidasUsuario.length > 0 && partidasUsuario.map((partida, index) => {
              if (partida.dias_restantes !== "0 horas" || partida.dias_restantes !== "0 días") {
                return (
                  <li key={index} className="resultadoItem">
                    {partida.estado && (
                      <div className="row align-items-center"  onClick={() => handleBoardClick(partida.estado, partida.idpartida)}>
                        <div className="col-md-3 d-flex align-items-center justify-content-center">
                          <div className="player player-white d-flex align-items-center">
                            <div className="div-imagen">
                              <img src={`${partida.jugador_blancas.avatar}`} alt={partida.jugador_blancas.nombre} classN
```

```
                                </div>
                                <span className="nombreAmigo">{partida.jugador_blancas.nombre}</span>
                              </div>
                            </div>
                            <div className="col-md-2 d-flex align-items-center justify-content-center">
                              <div className="partidaImagen" />
                            </div>
                            <div className="col-md-3 d-flex align-items-center justify-content-center">
                              <div className="player player-black d-flex align-items-center">
                                <div className="div-imagen">
                                  <img src={`${partida.jugador_negras.avatar}`} alt={partida.jugador_negras.nombre} classNam
                                </div>
                                <span className="nombreAmigo">{partida.jugador_negras.nombre}</span>
                              </div>
                            </div>
                            <div className="col-md-2 d-flex align-items-center justify-content-center">
                              <div className="infoPartida" style={{ fontSize: '12px' }}>
                                Turno: <br /> {partida.estado.turn === 'w' ? 'Blancas' : 'Negras'}
                                {((partida.estado.turn === 'w' && partida.jugador_blancas.id === parseInt(idUsuario)) ||
                                (partida.estado.turn === 'b' && partida.jugador_negras.id === parseInt(idUsuario))) && (
                                <div style={{ color: 'red', fontSize: '12px' }}>
                                  ¡Tu turno!
                                </div>
                              )}
                              </div>
                            </div>
                            <div className="col-md-2 d-flex align-items-center justify-content-center">
                              <div className="infoPartida" style={{ fontSize: '12px' }}>
                                Tiempo: <br /> {partida.dias_restantes}
                              </div>
                            </div>
                          </div>
                        )}

                      </li>
                    );
                  } else {
                    return null; // No renderizar la partida
                  }
                })}
              </ul>
            </div>
          </main>
        </div>
      );



};

export default Home;
```

Este componente es el encargado de mostrar la información de las partidas que el usuario tiene en curso. Permitiendo acceder al componente Board con la partida en cuestion mediante un click.

El componente board es uno de los mas complejos de la aplicación. En el se encuentra el estado de la partida y las piezas del tablero.

```
Click here to ask Blackbox to help you code faster
import React, { useState, useEffect } from 'react';
import { useAppContext } from '../../contexts/Context'
import './Board.css';
import Ranks from './bits/Ranks';
import Files from './bits/Files';
import Pieces from '../Pieces/Pieces';
import PromotionBox from '../Popup/PromotionBox/PromotionBox';
import Popup from '../Popup/Popup';
import GameEnds from '../Popup/GameEnds/GameEnds';
import arbiter from '../../arbiter/arbiter';
import { getKingPosition } from '../../arbiter/getMoves';
import SideMenu from '../SideMenu/SideMenu';
import { obtenerPartida, actualizarPartida, abandonarPartida } from '../../services/apiService';
import { useLocation, useNavigate } from 'react-router-dom';
import actionTypes from '../../reducer/actionTypes';
import Control from '../Control/Control';
import MovesList from '../Control/bits/MovesList';

const Board = () => {
    const navigate = useNavigate();
    const ranks = Array(8).fill().map((x, i) => 8 - i);
    const files = Array(8).fill().map((x, i) => i + 1);
    const location = useLocation();
    const { estado: locationEstado, idPartida } = location.state || {};

    // Inicializamos appStateNuevoFormato y otros estados
    const [appStateNuevoFormato, setAppStateNuevoFormato] = useState(null);
    const appState = appStateNuevoFormato || locationEstado || originalAppState;
    let { appState: originalAppState, dispatch } = useAppContext(); // ESTADO ORIGINAL DE LA APLICACION
    const position = appState.position?.[appState.position.length - 1] || []; // Manejo del estado inicial
    console.log(position);

    const [loading, setLoading] = useState(true);
    const [mostrarMenu, setMostrarMenu] = useState(false);
    const [mostrarModalAbandonar, setMostrarModalAbandonar] = useState(false);
    const idUsuario = localStorage.getItem('userId');
    const [error, setError] = useState('');
    const [success, setSuccess] = useState('');
```

```
    useEffect(() => {
        if (!idUsuario) {
            navigate('/');
        } else {
            const fetchPartida = async () => {
                try {
                    const response = await obtenerPartida(idPartida);
                    console.log('ESTADO PARTIDA');
                    console.log(response);
                    if (response.success) {
                        if (parseInt(idUsuario) === parseInt(response.partidas.jugador_blancas)) {
                            localStorage.setItem('playerColor', 'w');
                        } else {
                            localStorage.setItem('playerColor', 'b');
                        }

                        const nuevoEstado = response.partidas.estado;
                        const appStateNuevoFormato = {
                            position: nuevoEstado.position || originalAppState.position,
                            turn: nuevoEstado.turn || originalAppState.turn,
                            candidateMoves: nuevoEstado.candidateMoves || originalAppState.candidateMoves,
                            movesList: nuevoEstado.movesList || originalAppState.movesList,
                            promotionSquare: nuevoEstado.promotionSquare || originalAppState.promotionSquare,
                            status: nuevoEstado.stauts || originalAppState.status,
                            castleDirection: nuevoEstado.castleDirection || originalAppState.castleDirection
                        };
                        setAppStateNuevoFormato(appStateNuevoFormato);
                        console.log('APPSTATE NUEVO FORMATO')
                        console.log(appStateNuevoFormato)
                        dispatch({ type: actionTypes.REC_STATE, payload: {...appStateNuevoFormato,...nuevoEstado } });
                        originalAppState = appStateNuevoFormato
                        console.log(originalAppState);
                        setLoading(false);
                        setMostrarMenu(true);
                    }
                } catch (error) {
                    setMostrarMenu(true);
                    console.error('Error al cargar la partida:', error);
                }
            };
            fetchPartida();
        }
    }, [idPartida, idUsuario, navigate, dispatch]);
```

```
//Funcion para aceptar el movimiento
const handleAcceptMove = async () => {
    try {
        console.log('ESTADO AL MANDAR MOVIMIENTO ORIGINAL!!!');
        console.log(originalAppState);
        const response = await actualizarPartida(idPartida, idUsuario, originalAppState);
        console.log('ACTUALIZAR PARTIDA');
        console.log(response);
        if (response.success) {
            console.log('ESTADO DESPUES DE ACTUALIZAR')
            console.log(response.estado)

            console.log('PARTIDA ACTUALIZADA');
            setSuccess(response.message);
            setError('');
        }else{
            setSuccess('');
            setError(response.message);
        }
    } catch (error) {
        setError('Error al aceptar el movimiento:', error);
        setSuccess('');
    }
};
```

```
// Función para abandonar la partida
const abandonarPartidaModal = async () => {
    try {
        const response = await abandonarPartida(idPartida, idUsuario, originalAppState);
        if (response.success) {
            console.log('PARTIDA ABANDONADA');
            navigate('/home');
        }
    } catch (error) {
        console.error('Error al abandonar la partida:', error);
    }
};

// Función para mostrar el modal de abandonar partida
const mostrarModalAbandonarPartida = () => {
    setMostrarModalAbandonar(true);
};

// Función para ocultar el modal de abandonar partida
const ocultarModalAbandonarPartida = () => {
    setMostrarModalAbandonar(false);
};
```

Aquí se encontrarian las llamadas al API Services y las funciones para mostrar y ocultar el modal de abadonar partida.

```
    return (
        <div className="root">
            {mostrarMenu && <SideMenu loading={loading} />}
            <main className="content content-chess" style={{ marginBottom: "60px" }}>
                <div className="panelChess">
                    <div className="row chess">
                        <div className="col">
                            <div className='board-container'>
                                <div className='board' >
                                    <Ranks ranks={ranks} />
                                    <div className='tiles'>
                                        {ranks.map((rank, i) =>
                                            files.map((file, j) =>
                                                <div
                                                    key={file + '' + rank}
                                                    i={i}
                                                    j={j}
                                                    className={`${getClassName(7 - i, j)}`}>
                                                </div>
                                            )
                                        )}
                                    </div>
                                    <Pieces idPartida={idPartida}/>
                                    <Popup>
                                        <PromotionBox />
                                        <GameEnds />
                                    </Popup>
                                    <Files files={files} />

                                </div>
                                </div>
                            </div>
                            <div className="col">
                                <Control>
                                    <MovesList />
                                    {error &&  <p className="error">{error}</p>}
                                    {success &&  <p className="success">{success}</p>}
                                </Control>
                            </div>
                        </div>
                    </div>

                    {mostrarMenu && appStateNuevoFormato && (
                        <div className="row">
                            <div className="col">
                                <button onClick={handleAcceptMove} className="btn btn-success mr-2">Confirmar Movimiento</button>
                            </div>
                            <div className="col">
                                <button onClick={mostrarModalAbandonarPartida} className="btn btn-danger">Abandonar Partida</button>
                            </div>
                        </div>
                    )}
                    {/* Modal de abandonar partida */}
                    {mostrarModalAbandonar && (
                        <div className="modal fade show" tabIndex="-1" role="dialog" style={{ display: "block", backgroundColor: "rgba(0,0,0,0.5)"
                            <div className="modal-dialog modal-dialog-centered" role="document">
                                <div className="modal-content">
                                    <div className="modal-body">
                                        <p className="text-white text-center">¿Estás seguro de abandonar la partida? Contará como derrota si la pa
                                    </div>
                                    <div className="modal-footer">
                                        <div className="row">
                                            <div className="col">
                                                <button type="button" className="btn btn-success" onClick={abandonarPartidaModal}>Aceptar</button>
                                            <div className="col">
                                                <button type="button" className="btn btn-danger" onClick={ocultarModalAbandonarPartida}>Cancelar</
                                            </div>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        )}
            </main>
        </div>
    );
}

export default Board;
```

Finalmente, esto sería lo que devuelve el componente Board, un tablero y varios componentes hijos que se renderizan en funcion de lo que ocurra en el componente padre. Por ejemplo, MoveList es el listado de movimientos de la partida y PromotionBox o GameEnds, son popups que se renderizan al promocionar un peon en la ultima casilla o al finalizar una partida.

Arbiter, helper y reducer

Por otro lado, se encuentran diferentes carpetas relacionadas con la logica del ajedrez.

En la carpeta Arbiter encontraremos tres ficheros, en move.js encontramos los movimientos “especiales” que contiene el ajedrez, como es la captura al paso de un peon o el enroque largo y corto.

```
import { copyPosition } from "../helper"

export const movePiece = ({position,piece,rank,file,x,y}) => {

    const newPosition = copyPosition(position)

    if(piece.endsWith('k') && Math.abs(y - file) > 1){ // Enroques
        if (y === 2){ // Enroque largo
            newPosition[rank][0] = ''
            newPosition[rank][3] = piece.startsWith('w') ? 'wr' : 'br'
        }
        if (y === 6){ // Enroque corto
            newPosition[rank][7] = ''
            newPosition[rank][5] = piece.startsWith('w') ? 'wr' : 'br'
        }
    }

    newPosition[rank][file] = ''
    newPosition[x][y] = piece
    return newPosition
}
```

```
export const movePawn = ({position,piece,rank,file,x,y}) => {
    const newPosition = copyPosition(position)

    // Comprobar que comer al paso captura una celda vacia
    // Detectar y borrar el peon comido
    if (!newPosition[x][y] && x !== rank && y !== file)
        newPosition[rank][y] = ''

    newPosition[rank][file] = ''
    newPosition[x][y] = piece
    return newPosition
}
```

En el fichero getMoves.js se encuentran las funciones para calcular los movimientos según la pieza y la posicion.

```
import arbiter from "./arbiter"

export const getRookMoves = ({position,piece,rank,file}) => {
    const moves = []
    const us = piece[0]
    const enemy = us === 'w' ? 'b' : 'w'

    const direction = [
        [-1,0],
        [1,0],
        [0,-1],
        [0,1],
    ]

    direction.forEach(dir => {
        for (let i = 1; i <= 8; i++) {
            const x = rank+(i*dir[0])
            const y = file+(i*dir[1])
            if(position?.[x]?.[y] === undefined)
                break
            if(position[x][y].startsWith(enemy)){
                moves.push ([x,y])
                break;
            }
            if(position[x][y].startsWith(us)){
                break
            }
            moves.push ([x,y])
        }
    })
```

```
export const getKingMoves = ({position,piece,rank,file}) => {
    let moves = []
    const us = piece[0]
    const direction = [
        [1,-1], [1,0],  [1,1],
        [0,-1],         [0,1],
        [-1,-1],[-1,0], [-1,1],
    ]

    direction.forEach(dir => {
        const x = rank+dir[0]
        const y = file+dir[1]
        if(position?.[x]?.[y] !== undefined && !position[x][y].startsWith(us))
        moves.push ([x,y])
    })
    return moves
}
```

En este ejemplo, tenemos la funcion que determina el movimiento de las torres y del rey.

Por ultimo, se encuentra el fichero arbiter, que es el encargado de gestionar que los movimientos sean validos y de contener la logica del ajedrez.

```
isPlayerInCheck : function ({positionAfterMove, position, player}) {
    const enemy = player.startsWith('w') ? 'b' : 'w'
    let kingPos = getKingPosition(positionAfterMove,player)
    const enemyPieces = getPieces(positionAfterMove,enemy)

    const enemyMoves = enemyPieces.reduce((acc,p) => acc = [
        ...acc,
        ...(p.piece.endsWith('p')
        ?   getPawnCaptures({
                position: positionAfterMove,
                prevPosition:  position,
                ...p
            })
        :   this.getRegularMoves({
                position: positionAfterMove,
                ...p
            })
        )
    ], [])

    if (enemyMoves.some (([x,y]) => kingPos[0] === x && kingPos[1] === y))
    return true

    else
    return false
},
```

En este ejemplo, se calcula si el rey se encuentra en jaque.

```
isCheckMate : function(position,player,castleDirection) {
    const isInCheck = this.isPlayerInCheck({positionAfterMove: position, player})

    if (!isInCheck)
        return false

    const pieces = getPieces(position,player)
    const moves = pieces.reduce((acc,p) => acc = [
        ...acc,
        ...(this.getValidMoves({
                position,
                castleDirection,
                ...p
            })
        )
    ], [])

    return (isInCheck && moves.length === 0)
},
```

Y en este otro, partiendo de si el rey se encuentra en jaque, se calcula si es jaque mate.

Una vez visto el fichero arbiter y todo lo que contiene, paso a explicar el fichero helper, que es el que contiene las funciones relacionadas con la notación de los movimientos y la posicion de las piezas.

```
export const getNewMoveNotation = ({piece,rank,file,x,y,position,promotesTo}) => {
    let note = ''
    rank = Number(rank)
    file = Number(file)
    if (piece[1] === 'k' && Math.abs(file-y) === 2){
        if (file < y)
            return 'O-O'
        else
            return 'O-O-O'
    }

    if(piece[1] !== 'p'){
        note+=piece[1].toUpperCase()
        if(position[x][y]){
            note+='x'
        }
    }
    else if (rank !==x && file !== y ){
        note+=getCharacter(file+1)+'x'
    }

    note+=getCharacter(y+1)+(x+1)

    if(promotesTo)
        note += '=' + promotesTo.toUpperCase()

    return note
}
```

Esta es la funcion para anotar el movimiento de las piezas. Esta notacion esta basada en la notacion algebraica, donde se calcula la fila y la columna, la pieza que se mueve, si es una captura o un enroque, etc.

Para mas información sobre las notaciones en ajedrez, puedes consultar el siguiente enlace: https://es.wikipedia.org/wiki/Notaci%C3%B3n_(ajedrez)

En este fichero, tambien se encuentra la funcion para crear una posicion inicial.

```
export const createPosition = () => {

    const position = new Array(8).fill('').map(x => new Array(8).fill(''))

    for (let i = 0; i < 8; i++) {
        position[6][i] = 'bp'
        position[1][i] = 'wp'
    }

    position[0][0] = 'wr'
    position[0][1] = 'wn'
    position[0][2] = 'wb'
    position[0][3] = 'wq'
    position[0][4] = 'wk'
    position[0][5] = 'wb'
    position[0][6] = 'wn'
    position[0][7] = 'wr'

    position[7][0] = 'br'
    position[7][1] = 'bn'
    position[7][2] = 'bb'
    position[7][3] = 'bq'
    position[7][4] = 'bk'
    position[7][5] = 'bb'
    position[7][6] = 'bn'
    position[7][7] = 'br'

    return position
}
```

Por ultimo, el fichero reducer que contiene las ordenes que se envian según lo que ocurra en la partida.

En este fichero se encuentran las acciones que se pueden enviar a la partida.

```
        case actionTypes.NEW_GAME : {
            return {
                ...action.payload,
            }
        }

        case actionTypes.TAKE_BACK : {
            let {position,movesList,turn} = state
            if (position.length > 1){
                position = position.slice(0,position.length-1)
                movesList = movesList.slice(0,movesList.length-1)
                turn = turn === 'w' ? 'b' : 'w'
            }

            return {
                ...state,
                position,
                movesList,
                turn,
            }
        }
```

En el ejemplo, se puede crear una nueva partida o volver a un movimiento anterior.

```
case actionTypes.NEW_MOVE : {
    let {position,movesList,turn} = state
    position = [
        ...position,
        action.payload.newPosition
    ]
    movesList = [
        ...movesList,
        action.payload.newMove
    ]
    turn = turn === 'w' ? 'b' : 'w'

    return {
        ...state,
        position,
        movesList,
        turn,
    }
}
```

También es el encargado de ejecutar los movimientos en el tablero.

Para llamar a estas acciones se hace uso de la funcion dispatch(orden()), un ejemplo claro de esto, lo podemos obtener en el componente pieces, al realizar un movimiento.

```
const move = async (e) => {
    const {x,y} = calculateCoords(e)
    const [piece,rank,file] = e.dataTransfer.getData("text").split(',')


    if(appState.candidateMoves.find(m => m[0] === x && m[1] === y)){
        const opponent = piece.startsWith('b') ? 'w' : 'b'
        const castleDirection = appState.castleDirection[`${piece.startsWith('b') ? 'white' : 'black'}`]


        if ((piece==='wp' && x === 7) || (piece==='bp' && x === 0)){
            openPromotionBox({rank,file,x,y})
            return
        }
        if (piece.endsWith('r') || piece.endsWith('k')){
            updateCastlingState({piece,file,rank})
        }
        const newPosition = arbiter.performMove({
            position:currentPosition,
            piece,rank,file,
            x,y
        })
        const newMove = getNewMoveNotation({
            piece,
            rank,
            file,
            x,
            y,
            position:currentPosition,
        })
        // Llamar a reducer pasando la orden con los parametros
        dispatch(makeNewMove({newPosition,newMove}))
        console.log('AL HACER UN MOVIMIENTO');
        console.log(appState);

        let endGame = false;
        let endGameReason = "";
            if (arbiter.insufficientMaterial(newPosition)){
                endGame = true;
                endGameReason = "insufficientMaterial";
                dispatch(detectInsufficientMaterial())
            }
            else if (arbiter.isStalemate(newPosition,opponent,castleDirection)){
                endGame = true;
                endGameReason = "stalemate";
                dispatch(detectStalemate())
            }
            else if (arbiter.isCheckMate(newPosition,opponent,castleDirection)){
                endGame = true;
                endGameReason = "checkmate";
                dispatch(detectCheckmate(piece[0]))
            }
         if(endGame && !isPracticeRoute){
            try{
            const response = await finalizarPartida(idPartida, piece[0], endGameReason);
            if(response.success){
                if(endGameReason === "insufficientMaterial"){
                    dispatch(detectInsufficientMaterial());
                }else if(endGameReason === "stalemate"){
                    dispatch(detectStalemate())
                }else{
                    dispatch(detectCheckmate(piece[0]))
                }

            }
         }catch (error) {
            console.error('Error al finalizar la partida:', error);
         }
         }
    }
    dispatch(clearCandidates())
}
```

En este ejemplo, se ejecutaria todo lo relacionado al hacer un movimiento. Primero, se calcularia si un peon esta coronando en la ultima casilla, después se calcularia si es un enroque, actualizando el estado del enroque en caso de que se diese el enroque, ya que en el ajedrez solo se puede hacer un enroque por partida.

Después pasariamos al arbiter el movimiento que queremos hacer partiendo de la posicion actual y anotamos el movimiento en MoveList con la funcion

getNewMoveNotation(), una vez hecho esto, damos la orden de ejecutar el movimiento haciendo uso del dispatch y calculamos posteriormente si la posicion después del movimiento se encuentra en estado de “material insuficiente”, “ahogado” o “jaque mate”.

Por ultimo, si la partida se encuentra en alguno de estos estados y no se encuentra en /Practice, se ejecutara finalizarPartida y se anotará en la base de datos el reparto de puntos ademas de darse la partida como finalizada.