

# PROYECTO:

# ONLINE CHESS



**GRUPO:** FDS-14.01/20

**CURSO:** IFCT-0609: Programación de sistemas informáticos

**REALIZADO POR:** Javier Palacios

**FECHA DE INICIO:** 30/12/2021

**ACADEMIA:** INTERNATIONAL SCHOOL



# ÍNDICE:

<b>PREFACIO Y DOCUMENTOS LEGALES .....</b>	<b>Página 4</b>
<b>INTRODUCCIÓN .....</b>	<b>Página 7</b>
Justificación del proyecto	
Funcionalidad	
<b>DISEÑO GRÁFICO DE LA APLICACIÓN.....</b>	<b>Página 8</b>
Vistas de la aplicación	
Diagrama de flujo	
<b>ESTRUCTURA DE LA BASE DE DATOS .....</b>	<b>Página 10</b>
<b>ARQUITECTURA DEL CÓDIGO.....</b>	<b>Página 11</b>
<b>PLANIFICACIÓN DEL PROYECTO .....</b>	<b>Página 13</b>
<b>DIFICULTADES ENCONTRADAS.....</b>	<b>Página 17</b>

## **PREFACIO Y DOCUMENTOS LEGALES**

*Copyright © All rights reserved. Chess Online*

Este software y la documentación relacionada se proporcionan bajo un contrato de licencia que contiene restricciones de uso y divulgación y están protegidos por las leyes de propiedad intelectual. Salvo que esté expresamente permitido en su contrato de licencia o lo permita la ley, no puede usar, copiar, reproducir, traducir, difundir, modificar, licenciar, transmitir, distribuir, exhibir, ejecutar, publicar o mostrar en cualquier parte, en cualquier forma, o por cualquier medio. Se prohíbe la ingeniería inversa, el desmontaje o la descompilación de este software, a menos que lo exija la ley para la interoperabilidad.

La información contenida en este documento está sujeta a cambios sin previo aviso y no se garantiza que esté libre de errores. Si encuentra algún error, infórmenos por escrito.

Si se trata de software o documentación relacionada que se entrega al gobierno de ESPAÑA o a cualquier persona que lo otorgue en nombre del gobierno de los ESPAÑA, Entonces se aplicará el siguiente aviso:

USUARIOS FINALES DEL GOBIERNO DE ESPAÑA: Programas de Oracle (incluido cualquier sistema operativo, software integrado, cualquier programa incrustado, instalado o activado en el hardware entregado, y modificaciones de dichos programas) y documentación informática de Oracle u otros datos de Oracle entregados a los usuarios finales del gobierno de los ESPAÑA son "software informático comercial" o "documentación de software informático comercial" de conformidad con el Reglamento Federal de Adquisiciones aplicable y los reglamentos suplementarios específicos de la agencia. Como tal, el uso, reproducción, duplicación, publicación, exhibición, divulgación, modificación, preparación de trabajos derivados y / o adaptación de i) programas de Oracle (incluido cualquier sistema operativo, software integrado, cualquier programa incrustado, instalado o activado en la entrega hardware y modificaciones de dichos programas), ii) la documentación informática de Oracle y / o iii) otros datos de Oracle, están sujetos a los derechos y limitaciones especificados en la licencia contenida en el contrato correspondiente. Los términos

que rigen el uso de los servicios en la nube de Oracle por parte del gobierno de ESPAÑA. Se definen en el contrato aplicable para dichos servicios. No se otorgan otros derechos al gobierno de los ESPAÑA.

Este software o hardware está desarrollado para uso general en una variedad de aplicaciones de administración de información. No está desarrollado ni diseñado para su uso en ninguna aplicación intrínsecamente peligrosa, incluidas las aplicaciones que pueden crear un riesgo de lesiones personales. Si usa este software o hardware en aplicaciones peligrosas, entonces será responsable de tomar todas las medidas apropiadas a prueba de fallas, respaldo, redundancia y otras para garantizar su uso seguro. Oracle Corporation y sus afiliadas renuncian a cualquier responsabilidad por los daños causados por el uso de este software o hardware en aplicaciones peligrosas.

Oracle y Java son marcas comerciales registradas de Oracle y / o sus afiliadas. Otros nombres pueden ser marcas comerciales de sus respectivos propietarios.

Intel e Intel Inside son marcas comerciales o marcas comerciales registradas de Intel Corporation. Todas las marcas comerciales de SPARC se utilizan bajo licencia y son marcas comerciales o marcas comerciales registradas de SPARC International, Inc. AMD, Epyc y el logotipo de AMD son marcas comerciales o marcas comerciales registradas de Advanced Micro Devices. UNIX es una marca registrada de The Open Group.

Este software o hardware y documentación pueden proporcionar acceso o información sobre contenido, productos y servicios de terceros. Oracle Corporation y sus afiliadas no son responsables y renuncian expresamente a todas las garantías de ningún tipo con respecto al contenido, productos y servicios de terceros, a menos que se establezca lo contrario en un acuerdo aplicable entre usted y Oracle. Oracle Corporation y sus afiliadas no serán responsables de ninguna pérdida, costo o daño incurrido debido a su acceso o uso de contenido, productos o servicios de terceros, excepto según lo establecido en un acuerdo aplicable entre usted y Oracle.

Esta documentación NO se distribuye bajo una licencia GPL. El uso de esta documentación está sujeto a los siguientes términos:

Puede crear una copia impresa de esta documentación únicamente para su uso personal. Se permite la conversión a otros formatos siempre que el contenido real no se altere o edite de ninguna manera. No debe publicar ni distribuir esta documentación de ninguna forma ni en ningún medio, excepto si distribuye la documentación de una manera similar a como la distribuye Oracle (es decir, electrónicamente para descargar en un sitio web con el software) o en un CD. -ROM o medio similar, siempre que la documentación se difunda junto con el software en el mismo medio. Cualquier otro uso, como la difusión de copias impresas o el uso de esta documentación, total o parcialmente, en otra publicación, requiere el consentimiento previo por escrito de un representante autorizado de Oracle. Oracle y / o sus afiliadas se reservan todos y cada uno de los derechos sobre esta documentación que no se hayan otorgado expresamente anteriormente.

Na, es coña, es una licencia MIT de toda la vida. Básicamente es libre.



# INTRODUCCIÓN

La realización del proyecto consiste en la creación de un programa de ajedrez online. Programado en Java con el IDE NetBeans (Oracle Corporation).

## Justificación del proyecto

La motivación para este proyecto consiste en el desafío que supone para mis conocimientos del lenguaje, de la optimización y de la arquitectura del código.

## Funcionalidad

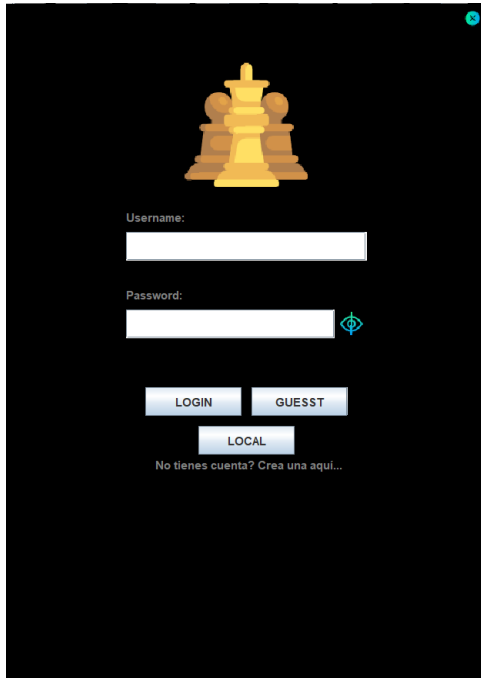
- Login y autenticación.
- Conexión por sockets.
- Comunicación mediante el envío de paquetes.
- Escaneo de red para buscar servidor
- Identificación de usuarios online.
- Implementación de chat de texto.
- Encapsulación de chats y partidas.
- Guardado de partidas en memoria.
- Asociación chat-partida.

# DISEÑO GRÁFICO LA APLICACIÓN

El diseño consta de tres vistas

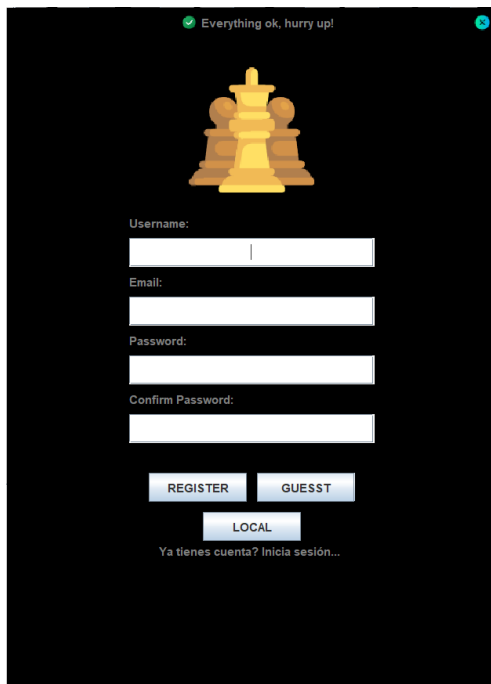
## Vistas de la aplicación

### ·Login



The login screen features a dark blue background with a golden chess king piece at the top center. Below the piece are two input fields: 'Username:' and 'Password:'. The 'Password:' field includes a toggle icon for visibility. At the bottom, there are three buttons: 'LOGIN', 'GUEST', and 'LOCAL'. A link 'No tienes cuenta? Crea una aquí...' is positioned below the 'LOCAL' button. A green status message 'Everything ok, hurry up!' is visible in the top right corner.

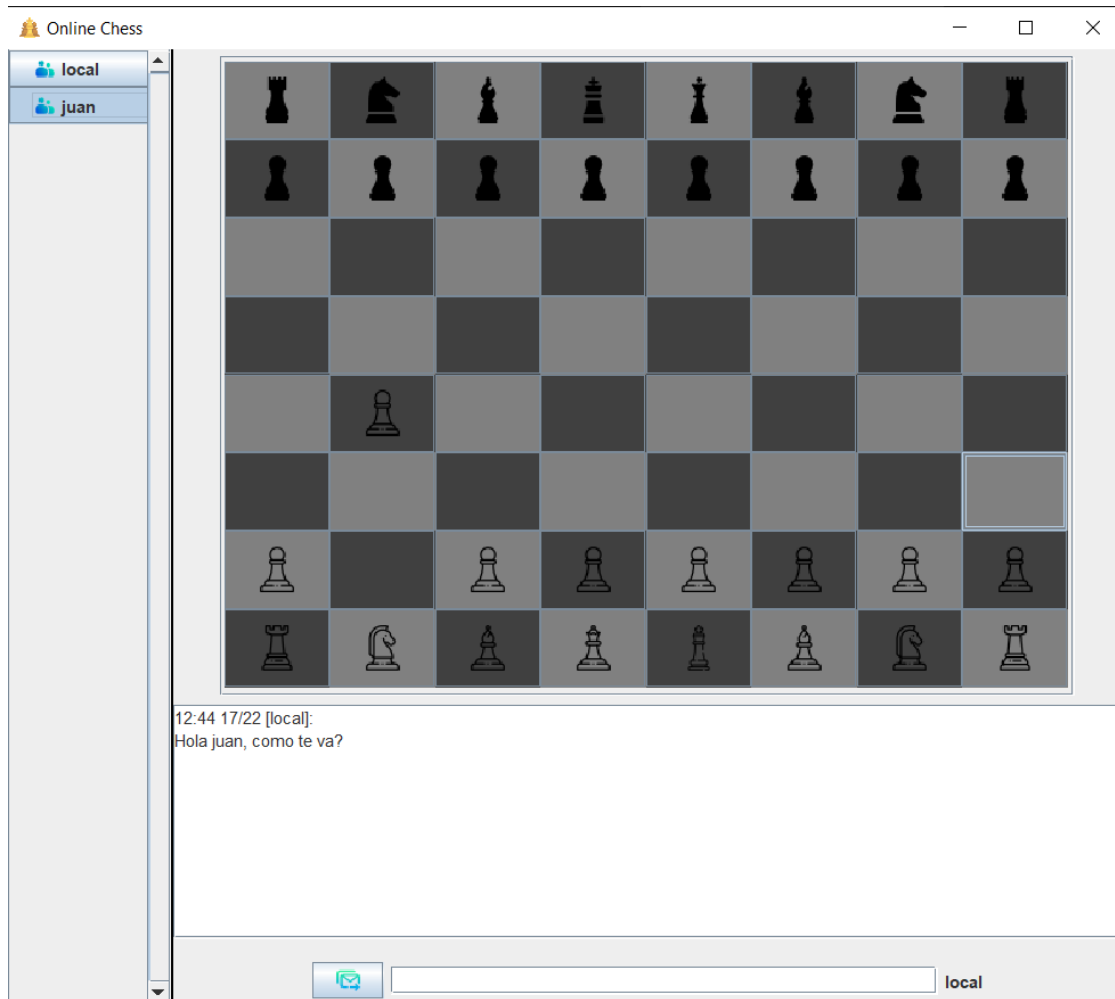
### ·Registro



The registration screen has a dark blue background with a golden chess king piece at the top center. Below the piece are four input fields: 'Username:', 'Email:', 'Password:', and 'Confirm Password:'. At the bottom, there are three buttons: 'REGISTER', 'GUEST', and 'LOCAL'. A link 'Ya tienes cuenta? Inicia sesión...' is positioned below the 'LOCAL' button. A green status message 'Everything ok, hurry up!' is visible in the top right corner.



·Buscar y jugar partida.



### Flujo de la aplicación

Al iniciar la aplicación el usuario deberá elegir entre iniciar sesión, registrarse, jugar como invitado o jugar en local (sin necesidad de conexión a la red).

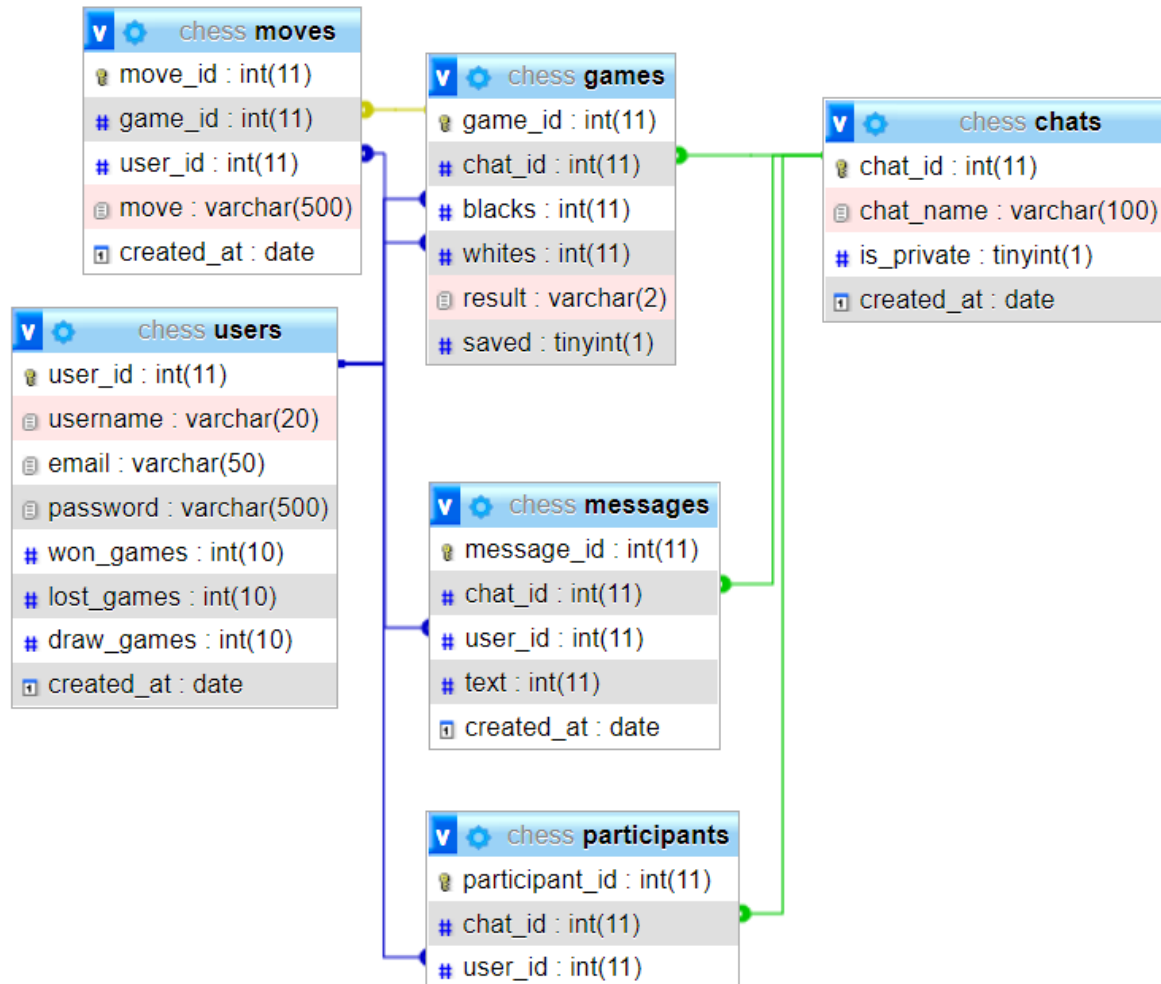
Una vez elegido el tipo de sesión, el usuario tendrá acceso a un tablero y (en caso de estar conectado a la red) a un número de chats equivalente al número de usuarios conectados al servidor.

El usuario deberá elegir un chat e iniciar la partida moviendo pieza, el primero que mueve queda asignado como blancas.

La partida continuará hasta que el rey adversario sea consumido.

## ESTRUCTURA DE LA BASE DE DATOS

Consta de seis tablas:



La tabla **users** guarda los datos de cada usuario.

La tabla **chats** guarda los datos generales de cada chat

La tabla **participants** guarda las relaciones entre **users** y **chats** (**M to M**)

La tabla **messages** guarda la información de cada mensaje individual, también podría usarse como la tabla relación entre **users** y **chats** salvando una tabla.

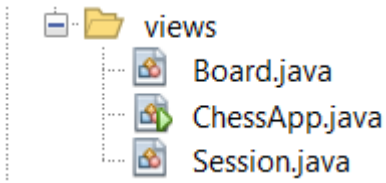
La tabla **moves** es similar a la de **messages** guardando la información individual de cada movimiento de forma individual.

Por último la tabla **games** guarda información general de un juego englobando una colección específica de **moves**.

## ARQUITECTURA DEL CÓDIGO

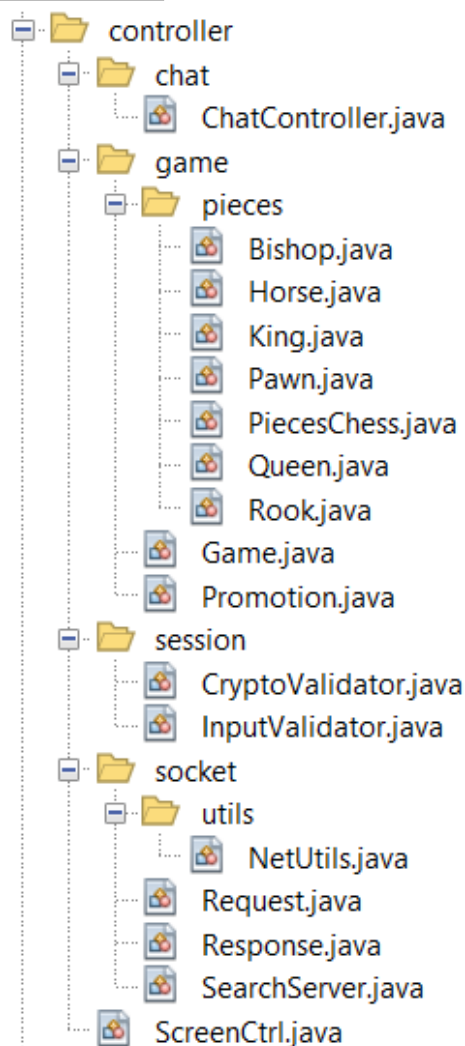
Se seguirá la arquitectura de Modelo-Vista-Controlador (MVC). Este proyecto consta de que consta de seis clases:

### Vista



- class ChessApp: es la clase principal que contiene el frame y es el punto de entrada de la aplicación
- class Board: se encarga de la representación visual del tablero.
- class Session: Vista de login-registro

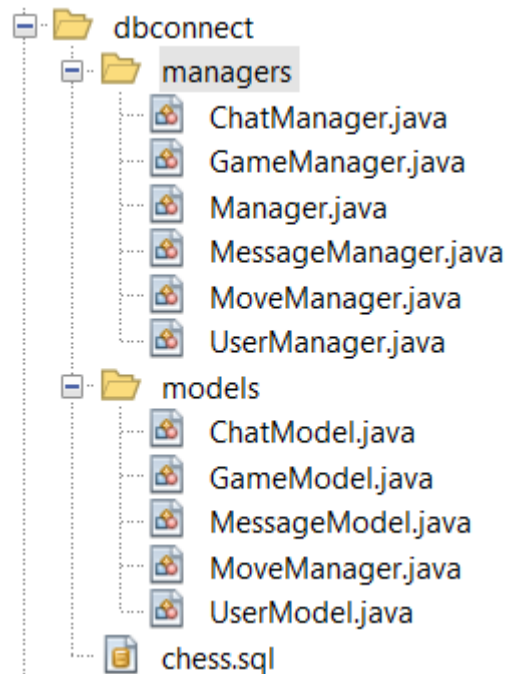
### Controlador



Destaca la carpeta chat, donde se gestiona los mensajes de texto y los movimientos. La carpeta game, donde se encuentra toda la lógica del juego. La de session, donde

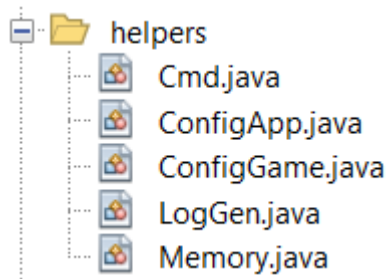
se valida el inicio de sesión. La carpeta socket, donde se encuentra toda la lógica de comunicación con el servidor (gestión request-response). Por último la clase ScreenCtrl que controla la visualización de los chats en su conjunto, el almacenado de memoria y el paso de un chat a otro.

### Modelo



Los managers establecen la conexión con la base de datos y manejan las operaciones de inserción, deletación, actualización y lectura de los datos. Los modelos representan las tablas de la base de datos y constituyen el empaquetado y transmisión de los datos entre el manager y el controlador. Estableciendo de esta forma un contrato o protocolo que rige el intercambio de esta información otorgando al controlador total agnosticismo sobre la base de datos y viceversa.

### Helpers



Por otra parte se incluyen dos clases dentro del paquete “helpers” que se encargan del registro de logs de la aplicación, configuración y otras tareas misceláneas.

# PLANIFICACIÓN DEL PROYECTO

La estructuración de las tareas a realizar en la duración del proyecto es la siguiente:

## Búsqueda de información y análisis

Se han tenido en cuenta diversas fuentes de información. Gran parte del diseño ha sido inspirado en la aplicación web “chess.com”. En cuanto al programa se han seguido varias directrices especificadas en “Toledo Nanochess source code”.

## Elaboración del diseño

Se desarrollan las vistas con el objetivo de que sean intuitivas y funcionales. Para evitar abrumar al usuario las opciones solo se le presentan en caso necesario. Por ejemplo el botón que permite reintentar la conexión con el servidor solo aparece al agotar el tiempo de espera tras 5 escaneos consecutivos de la red (teniendo en cuenta que haya conexión a internet) y desaparece al ser pulsado.

También es esencial que el usuario esté informado tanto léxica como visualmente del estado de la aplicación para evitar la incertidumbre. En esa línea, al cerrar la ventana de sesión sin pulsar ninguna opción se informa al usuario que la sesión se ha iniciado en modo local y que tendrá ciertas limitaciones.

La pantalla inicial debe ser sencilla y el tablero debe indicar las posiciones posibles para cada pieza siguiendo los estilos o filosofía propuestos.

## Desarrollo de la aplicación

### **Juego**

Se desarrolla en el IDE NetBeans versión 12.6 utilizando la base de datos MySQL a través de Xampp que también despliega el servidor Apache.

Se ha optado por un desarrollo modular, primero desarrollar el núcleo de la aplicación o la parte más importante, lo que constituye el producto mínimo viable. En el caso de esta aplicación es la lógica del juego junto con el tablero y las piezas.

Para permitir un mejor mantenimiento y escalabilidad gran cantidad de variables (desde el tamaño, forma y color del tablero hasta la disposición de las piezas) quedan supeditadas a un archivo de configuración.

```

public class ConfigGame implements Serializable {

    public static Dimension context;
    private final int rows = 0; //extra-minus rows
    private final int cols = 0; //extra-minus columns;
    //tile color
    public static final Color WTILE = Color.gray;
    public static final Color BTILE = Color.darkGray;
    public static final Color SLCT = Color.red;
    public static final Color ALLW = Color.cyan;
    public static final Color CHECK = Color.magenta;

    //piece team reference
    public static final String WHITES = "pqkbhr";
    public static final String BLACKS = "PQKBHR";

    protected Map<String, Integer> board;
    protected Map<String, String> img;
    protected Map<String, String> start;

    //Types of plays:
    private int choose = 0;
    String std = "rhbkqbhrppppppp-----PPPPPPPRHBQKBHR";
    String p = "rhbkqbh-pppppppk-----r-----R-----KPPPPPP-HBQKBHR";
    String pawn = "pppkpppppppppppppppppppp-----PPPPPPPPPPPPPPPPPKPPP";
    String cstl = "r--kk--rk--rr--kR--KK--RK--RR--Kr--kk--rk--rr--kR--KK--RK--RR--K";
    String out = "ppppppprhbkqbhrppppppp-----PPPPPPPRHBQKBHRPPPPPPPP";

    public ConfigGame (int chooseBoard){
        this.choose = chooseBoard;

        context = new Dimension(600,450); //board size

        img = new HashMap<>();
        img.put("R", "img/T.png");
        img.put("H", "img/H.png");
        img.put("B", "img/B.png");
        img.put("Q", "img/Q.png");
        img.put("K", "img/K.png");
        img.put("P", "img/P.png");
        img.put("r", "img/Y.png");
        img.put("h", "img/J.png");
        img.put("b", "img/V.png");
        img.put("q", "img/W.png");
        img.put("k", "img/L.png");
        img.put("p", "img/O.png");
        img.put("n", "null");

        start = new HashMap<>();
        start.put(std, "8,8");
        start.put(p, "8,8");
        start.put(pawn, "8,8");
        start.put(cstl, "8,8");
        start.put(out, "10,8");

        int defaultH=8, defaultW=8;
        if(choose > 0){
            String[] s = start.values().toArray()[choose].toString().split(",");
            defaultH = Integer.parseInt(s[0]);
            defaultW = Integer.parseInt(s[1]);
        }

        board = new HashMap<>();
        board.put("h", defaultH+rows);
    }
}

```

Siguiendo con esta filosofía las piezas tienen lógica propia siendo los primeros componentes en ser desarrollados con independencia unas de otras y del tamaño del tablero. La lógica general del juego (como enroques o jacques) queda supeditada a la clase que se encarga de juntar todas las clases para formar una unidad funcional.

### **Escalabilidad**

Una vez comprobado que las clases forman una unidad funcional en si mismas y se han corregido los posibles errores, el siguiente paso es crear múltiples juegos al mismo tiempo y asegurarse que tienen una correcta encapsulación.

### **Implementación**

Llegados a este punto la aplicación debe integrarse con el chat para formar un diseño en conjunto.

### **Comunicación**

En esta fase se desarrolla el servidor haciendo uso de la librería Sockets y haciendo que acepte peticiones de forma continua con un bucle dentro de un hilo de ejecución paralelo.

En este proyecto el servidor consta de dos hilos que aceptan peticiones: uno para responder a las conexiones de los equipos y otro para gestionar las peticiones una vez conectados. De esta forma ante conexiones masivas el tráfico dentro de la aplicación se vería reducido pero no detenido.

Para la localización del servidor el cliente analiza la red en la que se encuentra y realiza peticiones a cada ip (1-255) por el puerto designado.

Una vez comprobadas las comunicaciones la gestión de las peticiones se realiza transmitiendo una clase serializable llamada packager donde la variable "status" indica que procedimiento aplicar en cada caso.

En la parte del cliente la información relevante se guarda en la variable "storage" que consta de un array de "Memories" (clase que agrupa la información relevante de cada chat). Esta es la clase que condiciona la gestión y procesamiento de la información.

## **Autorización**

Por último la autorización del usuario se realiza mediante login o registro, conectando con la base de datos ya sea para insertar un nuevo registro o para comprobar las credenciales guardadas. La contraseña viaja encriptada desde el servidor y se comprueba en la parte del cliente si coincide o no con la introducida por el usuario.



## DIFICULTADES ENCONTRADAS

### Bordes

La mayor dificultad del proyecto ha sido el desarrollo de la lógica de las piezas.

Especialmente la creación de bordes o barreras para evitar movimientos no permitidos. Esta dificultad es fruto de la distribución del tablero como celdas numeradas consecutivamente y la necesidad de que las piezas tuvieran una gestión intrínseca de sus movimientos permitidos. Una alternativa mucho más sencilla sería adoptar el modelo de bordes invisibles donde al tablero inicial se le añaden dos columnas (una a cada lado) que indican que se ha llegado al final del tablero. Este modelo complicaría la generación del tablero pero facilitaría la gestión de los movimientos de las seis piezas.

En última instancia este modelo fue rechazado porque dificultaba la escalabilidad a tableros de diferentes tamaños y modos de juego donde se eliminan los bordes.

### Búsqueda de servidor

Otra gran dificultad fue la búsqueda de servidor por parte del cliente sin que el usuario tuviera que introducir manualmente la dirección ip y permitiendo que el servidor se ejecutara desde cualquier terminal sin tener que cambiar la dirección en el código. La primera solución consistía en el envío de un paquete de datos a un puerto concreto a todas las ips de la misma red que la parte de cliente (suponiendo una máscara de subred de 255.255.255.0).

Esta solución era altamente ineficiente por tener que esperar al fallo de cada respuesta antes de enviar la siguiente.

El siguiente modelo ejecutaba cada una de las 255 peticiones en un hilo paralelo. El siguiente problema a resolver fue la posibilidad de tener el servidor apagado o que el paquete de datos no llegara de forma correcta y que por tanto nunca se recibiera respuesta por parte del servidor.

Para ello cada socket se creo en un hilo aparte con un timer para ejecutar la petición cada cinco segundos. Esto supuso una gran carga para el dispositivo y al cabo de un minuto la cantidad de procesos en ejecución ralentiza el dispositivo de forma notable.

La última versión solo realiza 5 peticiones o intentos de conexión por ip, transcurridos esos 5 intentos el proceso es destruido y se le da la opción al usuario de iniciar otra tanda de intentos.