# 3 Specific Requirements

Since some requirements are more critical than others, we assign them different priority levels. These levels are defined as:

|  |  |
| --- | --- |
| **Priority Level** | **Description** |
| **1** | This is the highest priority level. Requirements of this level are absolutely essential to the product's functionality and must be fully satisfied in order for the software to be released. |
| **2** | Requirements of this level are not critical to the software's underlying functionality but are highly desirable in order to provide users with a complete experience. These requirements should be satisfied. |
| **3** | Requirements of this level are additions that will make the software competitive among other chess applications. They will be fulfilled after priority 1 and 2 requirements. |

## 3.1 Functional Requirements

### 3.1.1 Login Screen

R1.1.1 The application will allow the user to create an account. **Priority 1**

R1.1.2 The application will display a prompt for the user to input their username and password. **Priority 1**

R1.1.3 The application will authenticate the user's credentials with the server. Upon success the user will be directed to the home screen. If verification fails, the application will display a failure message and allow the user to try again. **Priority 1**

R1.1.4 The login screen will feature branding that distinguishes the software. **Priority 3**

3.1.2 Home Screen

R1.2.1 The home screen will have a menu that allows the user to:

R1.2.1.1 Create a new game. **Priority 1**

R1.2.1.2 Join an existing game. **Priority 1**

R1.2.1.3 Join an existing game by game id number. **Priority 2**

R1.2.1.4 View their game history and replay through old games. **Priority 3**

R1.2.1.5 View performance statistics. **Priority 3**

### 3.1.3 Server

R1.3.1 The server will be implemented using Java sockets. **Priority 1**

R1.3.2 The server will maintain a collection of games, complete with game id and user id numbers. **Priority 1**

R1.3.3 The server will store moves for individual games so if the connection is lost the user will be able to resume play where they left off once reconnected. **Priority 1**

R1.3.4 The server will receive and transmit moves between players. **Priority 1**

### 3.1.4 Game Design

R1.4.1 Each game will have a game id number and user id numbers for the players. **Priority 2**

R1.4.2 There will be a ChessBoard class which has the following properties:

R1.4.2.1 Board:ChessPiece[8][8] - An 8x8 array of ChessPieces. **Priority 1**

R1.4.2.2 state:ENUM - An enumerated state (white to move, black to move, white win, black win, draw). **Priority 1**

R1.4.2.3 validateMove(ChessMove):boolean - Checks that the user's move is legal. If the user enters an illegal move the game should display a message to the user, but otherwise do nothing. **Priority 1**

R1.4.2.4 update(ChessMove):void - Send the move to the server. **Priority 1**

R1.4.2.5 history:List<ChessMove> - The game history. **Priority 3**

R.1.4.3 There will be a ChessPiece abstract class which has the following properties:

R1.4.3.1 x:int, y:int - The piece's location on the board. **Priority 1**

R1.4.3.2 create():void - Instantiates a new piece on the board. **Priority 1**

R1.4.3.3 destroy():void - Removes a chess piece from the board. This occurs when an enemy piece moves to occupy the square this piece occupies. **Priority 1**

R1.4.3.4 generateMoves(ChessBoard):List<ChessMove> - For move validation. Each piece will be able to return a list of legal moves. **Priority 1**

R1.4.3.5 There will be individual chess piece classes (Rook, Bishop, Pawn, etc.) that extend the ChessPiece class. These classes will override the above functions where necessary (move generation) and provide additional functionality if required (promotion). **Priority 1**

R1.4.4 There will be a ChessMove class which has the following properties:

R1.4.4.1 id:int - The move number and id (ply). **Priority 1**

R1.4.4.2 fromX:int, fromY:int - The starting location on the board. **Priority 1**

R1.4.4.3 toX:int, toY:int - The ending location on the board. **Priority 1**

### 3.1.5 Game Play

R1.5.1 The chess board will be displayed to each user with their respective pieces oriented at the bottom of the board. **Priority 2**

R1.5.2 The user will be able to:

R1.5.2.1 Offer a draw. **Priority 2**

R1.5.2.2 Resign. **Priority 2**

R1.5.2.3 Request to abort the game. **Priority 3**

R1.5.2.4 View the game notation. **Priority 3**

R1.5.3 The game will end automatically in the case of:

R1.5.3.1 The players agree to a draw. **Priority 2**

R1.5.3.2 Checkmate. **Priority 1**

R1.5.3.3 Stalemate. **Priority 1**

R1.5.3.4 Insufficient mating material. **Priority 2**

R1.5.3.5 Three-fold repetition. **Priority 3**

R1.5.3.6 A user runs out of time. **Priority 3**

R1.5.4 The game will implement a clock accurate to 0.1 seconds. **Priority 3**

## 3.2 Non-Functional Requirements

### 3.2.1 Extensibility

R2.1.1 The application will be built in a way that allows it to be easily modified and expanded. Additional functionality will be able to be added to the existing framework and will not require a redesign of the software. **Priority 2**

R2.1.2 The design will allow us to implement a chat window and Artificial Intelligence players if we choose to do so in the future. **Priority 3**

### 3.2.2 Maintainability

R2.2.1 The application will be developed using Github as our source control tool so that it will be easy to enforce version control, view code history and revisions, file bugs, and evaluate the software against the requirements. **Priority 1**

## 3.3 GUI Mockups