# 1 System Interactions

## 1.1 Register

When the user opens the application they will have the option to create a new account. In this case they will supply a username and password to the client. The client will relay this information to the server which will check to see if the username is unique and create a new account in the user data database. The user will then be able to login.

## 1.2 Login

When the user opens the application they will be able to provide their username and password to login to their account. The client will relay this information to the server which will validate the user's password against the account details in the user data database. If they passwords match the server will send a success signal to the client, and they main screen will be displayed to the user. If authentication fails, the user will receive a failure message and be able to attempt login again.

## 1.3 Game Creation

The user has the option to either join an existing game or create a new one. If the user chooses to create a new game then the server will instantiate a new instance of the Game class, create a Player, and log the game information in the game database. Then a new game will be displayed to the user

## 1.4 Game Play

When the user clicks on a piece, the client will relay this information to the server. The Game instance will access the getMoves method of the ChessPiece on that square and return a list of Moves to the client, which will highlight those possible destination squares on the chessboard. The user will then drop the piece on a destination square. The server validate this move against the possible destination squares. If it is valid the move will be executed in the Game and the server will send a refreshBoard call to both Players, which will update the game state on the client side.

# 2 Benefits, Assumptions, and Risks

## 2.1 Benefits

* Loose coupling. The design calls for a dumb GUI that has no knowledge of the game being played. The benefit of this design choice is that the client and server sides will have little reliance on each other. It would also be possible to reuse the GUI for other games.
* High cohesion. The Game class will encompass two Players, a Board (8x8 matrix of ChessPieces), and individual Pieces. This improves the robustness, reusability, and reliability of the application.

## 2.2 Assumptions

* Our users are generally aware of the rules of chess and how to play chess. These will not be made explicit in the application.

## 2.3 Risks

* The feature to highlight possible destination squares might slow down the game and take away from the user experience.
* We may need to refactor the code in a way to delegate certain responsibilities to more suitable classes.