Chess Game
Software Design Specification

Group Members	Armaan Bhasin, Brandin Bulicki, Aparna Mishra, Tumaresi Yalikun, Briana Schuetz
Faculty Advisor	Dr. Filippos Vokolos, Ph. D.
Project Stakeholder	Dr. Filippos Vokolos, Ph. D.

Revision History

Name	Date	Reason for Change	Revision
Armaan Bhasin, Brandin Bulicki, Aparna Mishra, Tumaresi Yalikun, Briana Schuetz	07/23/2020	First Draft of the Design Document	0.7
Armaan Bhasin, Brandin Bulicki, Aparna Mishra, Tumaresi Yalikun, Briana Schuetz	7/26/2020	Review of First Draft	1.0
Armaan Bhasin, Brandin Bulicki, Aparna Mishra, Tumaresi Yalikun, Briana Schuetz	7/29/2020	Revision and Submission of Requirements Document	1.5

Table of Contents

1. Introduction

1.1. Purpose

This is the Software Design Specifications for the online Chess Game. The purpose of this document is to describe the implementation of the Chess Game described in the requirements document. Chess Game is designed to create an online platform that allows two people to play chess from remote locations. This document will outline software design specifications for Chess Game in addition to system architecture and system components.

1.2. Scope

This document describes the implementation detail of Chess Game as well as the system architecture and system components. The software's purpose is to provide a platform to play a real time chess game. The implementation of front-end and back-end portions are explained through UML diagrams. This document does not contain information regarding testing of the software.

1.3. Definitions

1.3.1. Technologies Definitions

React: a JavaScript library for building user interfaces. [1]

Node.js: an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside a web browser. [2]

Socket.IO: a JavaScript library for real-time web applications, which enables real-time, bi-directional communication between web clients and servers. [3]

1.3.2. Chess Game Definitions [4]

Action Chess: a timed game with only 30 minutes allowed.

Attack: when a piece is moved to a position that can capture the opponent's piece in the next move.

Capture: to remove a piece from the chess board via legal move.

Castle: to place the unmoved king next to the rook and place the king inside and rook outside.

Checkmate: an attack on the king where there is no escape.

Draw: a game that ends without a winner.

2. Design Overview

2.1. Description of Problem

This project intends to create a game of chess that implements all the rules, regulations and specifications associated with the game. The quintessential feature that this game of chess offers that differentiates it from the rest relies primarily on how it is played. This game will ensure that it takes into consideration the various moves each piece makes, synchronizes multiple players over the internet and eventually manages multiple instances of concurrent chess games.

2.2. Technologies Used

This game of online chess will use React for the front-end interface and Node.js for the back-end interface. The reason for choosing Node.js lies behind its ability to build real time high traffic apps fairly quickly, the numerous packages made available by the Node Package Manager and the increased efficiency it provides during the development process.

Socket.io will be used to enable real time event based and bidirectional communication between the front - end and back-end systems.

The game is developed for Chrome and Firefox users but has been designed with the intention to run smoothly on every standard browser available today.

2.3. System Architecture

Figure 1 shows a high-level overview of the Online Chess. The system architecture is fairly simple as the game requires minimal data management. It has to only take currently active games and players into account. Thus, data storage and modelling aren’t this project's primary concern.

The System for this game comprises of the following elements:

- Front-End Interface

  This interface will be used by the user to experience, interact, and play the game.

- Back-End Interface

  This interface will be responsible for supporting the front-end interface, specifications of back-end routes for requests, and to manage socket communication.

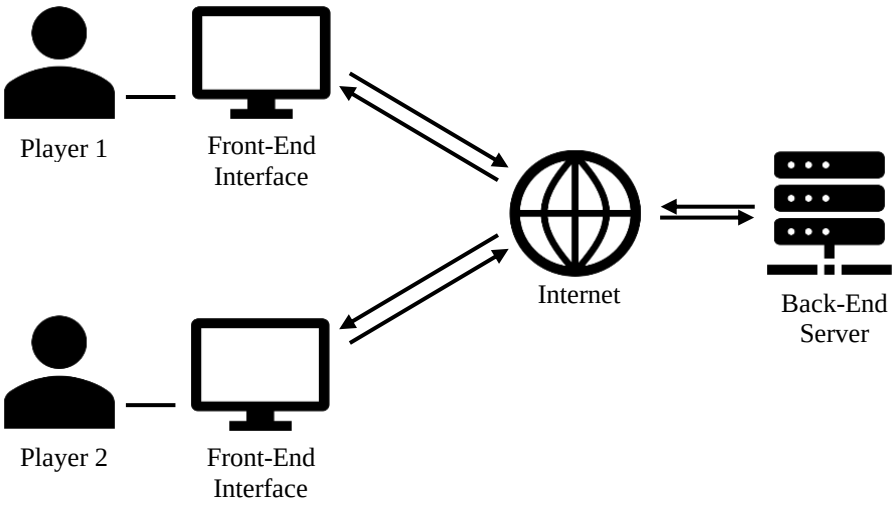

Figure 1: High-Level overview of user and server interaction for the Online Chess System Interface

2.4. System Operation

Figure 2 is the typical sequence of events that takes place when a user selects a name and initiates a game request. The communication arrows shown between the user and the back-end system below are propagated via web socket communication.

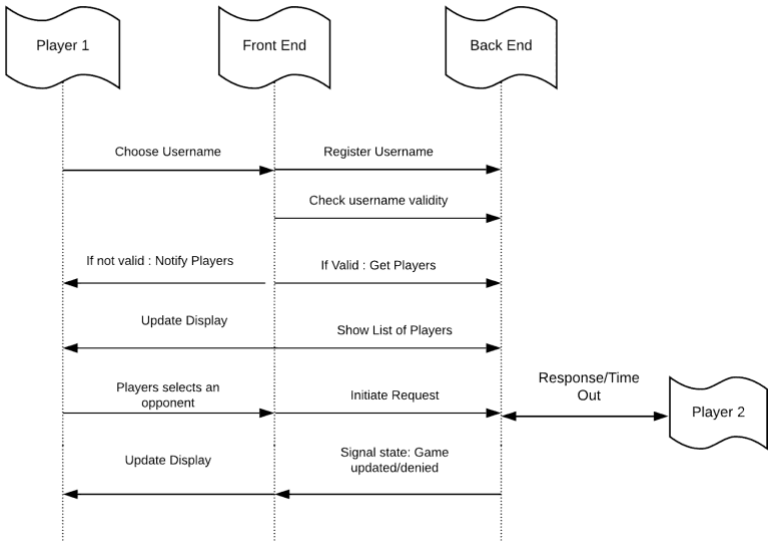

Figure 2: Sequence Diagram of game registration and requests

3. Requirements Traceability

Requirement	Description	Design Reference
R3.1.*	Board Layout	§4
R3.2.*	Moves	§4
R3.3.*	Capturing	§4
R3.4.*	Promotion	§4
R3.5.*	Legality	§4
R3.6.*	Game State	§5
R3.7.*	Server	§5
R4.1.*	Performance	§5
R4.2.*	Availability	§5
R4.3.*	Reliability	§5
R4.4.*	Usability	§4
R4.5.*	Browser & Platform Support	§5
R4.6.*	Scalability	§5
R4.7.*	Security	§5
R4.8.*	Reports & Logs	§5
R4.9.*	Development Constraints	§4, §5

4. Front-End Interface

4.1. Overview

The front-end interface of the application will communicate with the back-end interface. It will primarily handle user interactions with the game, using the MainMenu component.

4.2. Main Menu

The MainMenu component will reference and use the Singleton "WebSocketService" to create a connection to a presumed Menu "room" on the server Side. The "getConnectUsers" function in the MainMenu component will all the functions of the same name in the service, which will create an observable for an event that the back-end will emit to all connected sockets, allowing MainMenu components user lists to be updated real time.

MainMenu Components will call the WebSocketService to create and return two observables for back-end event emissions, for receiving game requests and for receiving responses to those requests. This service will subscribe to these messages for the

remainder of the component's lifespan. Upon a successful match, WebSocketService will connect users to the appropriate room on the Socket for their game as well as creating the necessary observable for receiving updates to the game board when the opponent makes their move.

4.3. Front-End UML Diagram

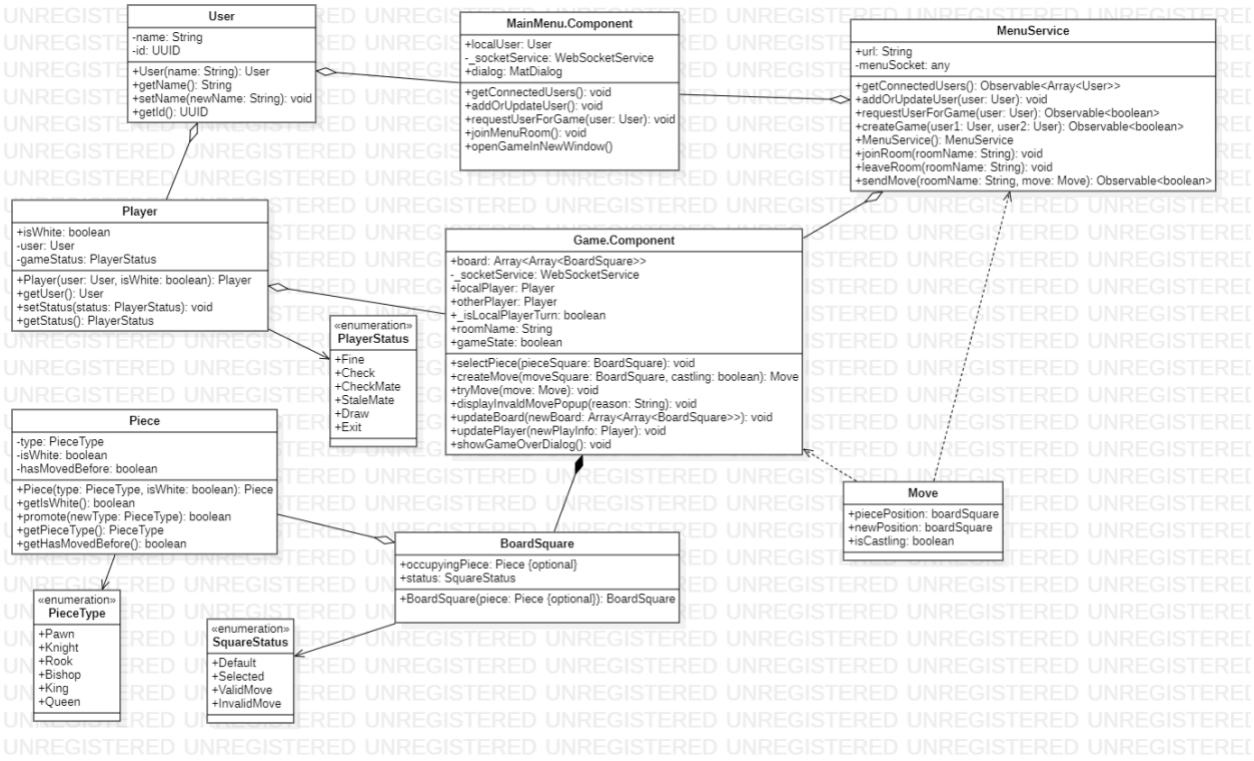

Figure 3: UML Diagram of the Front-End Interface for the online Chess Game

5. Back-End Interface

5.1. Overview

The back-end interface of the application will support the front-end interface. It will primarily handle games with the GameManager component and process moves with the MoveValidation service.

5.2. Game Management

The back-end interface will handle players and games. It will maintain a list of players waiting to start a game and players playing a current game. The GameManager

will handle matching players and starting a game with two players. It will call upon the MoveValidation to process all moves. After, it will relay the valid or invalid move to the front-end to handle notifying the players.

5.3. Move Validation

The back-end interface will validate all moves attempted by players. MoveValidation will be a service called upon by the GameManager component of the application. Move validation will depend on the current state of the board, the player attempting the move, the piece to be moved, and the intended location. There are eight distinct pieces that have its own method of movement. When a move is attempted, move validation will verify the piece can move to the intended location by comparing it to its current location. If the movement is valid and the square at the location is vacant, the move will be granted. If the square is not vacant, move validation will verify the moving piece's ability to capture the opponent's piece. If a move is invalid, false will be passed to the GameManager which will relay the result to the player who will be given another chance to make a valid move. If a move is valid, true will be passed to the GameManager which will relay it to the players and update both boards. The new game state will also be determined if the move is valid.

5.4. Back-End UML Diagram

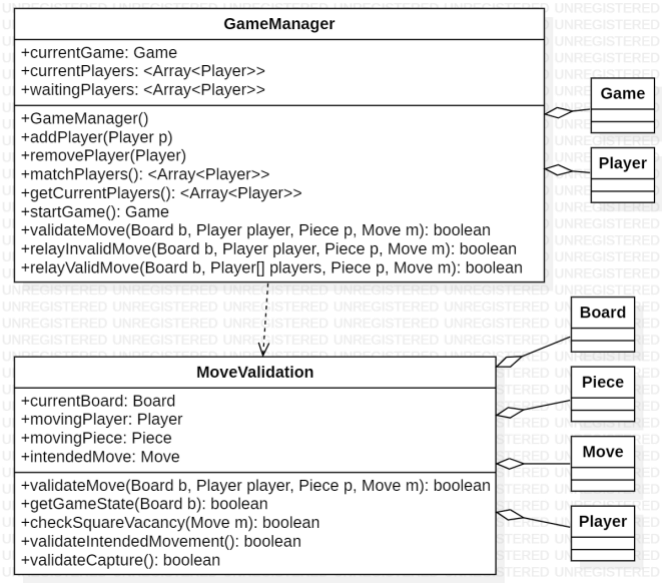

Figure 4: UML Diagram of the Back-End Interface for the online Chess Game

6. References

[1] https://reactjs.org/

[2] https://nodejs.org/en/

[3] https://socket.io/

[4] "Chess Terms and Definitions - ChessCentral." https://www.chesscentral.com/pages/learn-chess-play-chess-better/chess-terminology.html. Accessed 24 Jul. 2020.