

Confidential – Do not release
If released somewhere online please let me know at vesta06@protonmail.com

Eastland Career Center

A MULTIPLAYER CHESS GAME



Table of Contents

SECTION 1: ANALYSIS	7
Section 1.1: Background and Problem Identification	7
Client Contact Information	8
Client Interview.....	8
Prospective User(s) Identification	9
Section 1.2 Problem Modelling	9
Proposed System Objectives	9
Limitations	10
Use Case (UML)	11
Section 1.3: Preliminary Research	12
The Game of Chess	13
Special Moves	14
Game Conditions	15
Research Notes	16
Elo Rating Systems.....	17
Arpad's System	17
Glicko System	18
Conclusion	19
Research Review	19
Section 1.4: Iteration 1	19
Conclusive Objectives.....	20
Section 1.5: Iteration 2	20
End-user meeting.....	20
Research	21
Password Hashing Algorithm.....	21
Conclusive Objectives.....	23
Section 1.6: Iteration 3	24
Handling Real-time Multiplayer	24
Benefits and Drawbacks of Solution.....	26
Game Rooms	26
Conclusive Objectives.....	26
Section 1.7: Iteration 4	28
Objectives	28

Section 1.8: Iteration 5	29
Objectives	29
SECTION 2: DESIGN	31
Section 2.1: Declaration of Programming & Database Languages	31
Asynchronous vs Synchronous Approach.....	31
Section 2.2: Modelling	33
Board States	35
User Game Actions.....	36
Section 2.3: Iteration 1	38
Database Design.....	38
User Schema	38
Data Dictionary	40
Description of Algorithms	42
Database Queries	42
Section 2.4: Iteration 2	43
Sample of Client Socket Messages	43
Sample of Server Socket Messages.....	43
Section 2.5: Iteration 3	44
Sample of Client Socket Messages	44
Sample of Server Side Messages	45
Front-end Design	45
Design Objectives.....	47
Finished Design Review.....	50
Website Navigation	53
Section 2.6: Iteration 4	54
Database Design.....	54
Users Collection.....	54
Games Collection.....	55
Session Collection	57
Glicko-2 Cronjob	58
Lobby Design.....	58
PGN Download.....	58
Common Game Attributes	59
General Common Attributes	60
Regex for verifying email addresses.....	61

Description of Algorithms	61
Update Rankings.....	61
Database Queries	61
Matchmaking	62
Description of Data Structures.....	62
Data Integrity and Security.....	63

SECTION 3: TECHNICAL SOLUTION	65
Section 3.1: Requirements	65
Section 3.2: Instructions to start the program.....	65
Section 3.3: Code Tree.....	66
Section 3.4: Backend Code.....	71
app/primus.js	71
app/models/game.js.....	100
app/models/user.js	102
app/routes.js.....	104
config/database.js.....	126
config/passport.js	127
cronjob/update_rankings.js.....	138
Section 3.5: Front-end Code	146
public/assets/css/Footer-Basic.css	146
public/assets/css/Footer-Clean.css	149
public/assets/css/Navigation-Clean1.css.....	154
public/assets/css/popups.css.....	165
public/assets/css/rangeslider.css	167
public/assets/css/slider.css.....	174
public/assets/css/styles.css.....	175
public/css/game_online.css.....	186
public/css/signup.css	187
public/assets/js/custom.js	188
public/js/game_common.js	194
public/js/game_local.js.....	216
public/js/game_online.js	221
public/js/signup.js.....	231
public/js/user_list.js	236

views/common/footer.ejs	238
views/common/header.ejs	239
views/common/nav.ejs	240
views/common/scripts.ejs	242
views/common/styles.ejs	243
views/widgets/styles.ejs	243
views/game_common.ejs.....	249
views/game_local.ejs	253
views/game_online.ejs.....	254
views/index.ejs	255
views/leaderboards.ejs	258
views/login.ejs.....	260
views/search_user.ejs	261
views/signup.ejs	263
Section 3.6: Server.js & Package.json	266
server.js.....	266
package.json	270

SECTION 4: TESTING	274
Section 4.1: Iteration 1 Testing	274
Elements of system to test.....	274
Section 4.2: Iteration 2 Testing	279
Elements of the application to test.....	279
Testing Plan	280
Section 4.3 Iteration 3 Testing	287
Testing Table	287
Section 4.4 Iteration 4 Testing	291
Elements of the application to test.....	291
Testing.....	294
Section 4.5: Iteration 5 – Full System Testing	300
Chess Testing.....	301
Solo Queue.....	301
Playing versus a friend	307
Leaderboards Testing.....	309
Glicko2.....	311
Navigation Testing	314

Registration Page.....	314
Login Page	315
Leaderboard Page.....	317
Home Page.....	319
Play versus Player (Solo Page)	322
Play versus Player (Friend Page)	324
Play versus Computer page	326
SECTION 5: EVALUATION	329
Section 5.1: Iteration 1	329
Objective Analysis	329
Improvements	329
Personal Feedback	329
Section 5.2: Iteration 2	330
End-user Conference.....	330
Objective Analysis	330
Improvements	331
Section 5.3: Iteration 3	332
End-user Feedback.....	332
Objective Analysis	332
Potential Improvements.....	333
Personal Feedback.....	334
Section 5.4: Iteration 4	335
End-user Feedback.....	335
Objective Analysis	335
Potential Improvements.....	335
Personal Feedback.....	336
Personal Evaluation.....	336
Section 5.6: Appraisal	337
Analysis of end-user feedback.....	337
User Interface	337
Usability.....	338
Leaderboards.....	338
Online Chess	338
Chess vs Computer	338
Maintenace	339

Security	339
Additional Comments – Changes for the future	339
Proposed System Analysis.....	340
Section 5.6: Conclusion	342

SECTION 1: ANALYSIS

Section 1.1: Background and Problem Identification

Jesse Stamper is a student at the Eastland Career Center in Groveport Ohio. At present, he is learning Programming and Software development alongside with leading the Chess society at Eastland. Back in 2015, Jesse took leadership of the society which was comprised of only 4 members. Since his leadership, the group has grown rapidly, and there are now 27 active members of the society.

His main roles include managing fixtures between members, keeping track of scores and maintaining the general running of the group. The growth of the society has resulted in the volume of work required to maintain the group to increase. This new work load has been causing troubles for Jesse and consequently, led to a significant decline in the group's growth.

Every lunch break, the society runs a club where members come to play and increase their ranking on the group's leaderboard. Members are not required to attend any of the club's; however, this will usually lead to their ranking declining as other members of the group increase their scores. Every game is recorded and monitored by Jesse; he ensures that each game is played fairly and the scores are recorded accurately. At the end of the day, Jesse tally's up the results of the game and places the member with the highest number of wins at the top of the leaderboard.

There are several problems with the system Jesse is using, such as having to write down all the results by hand and having to monitor a game for it to be counted on the leaderboard. The process that Jesse is using is not scalable and as he has mentioned, with the group increasing in size he has been struggling to keep on top of his work. Thus, this has lead for his tasks to be completed insufficiently and could damage the reputation of his society.

One of the main problems that this has caused for the group, is a significant loss in data integrity. Besides the excessive work, Jesse records games and scores by hand meaning that there is a large chance of human error. Human error will mean that data is inaccurate and could easily aggravate members of the group if their games are recorded incorrectly. Additionally, the increase in work has meant that Jesse has not been able to fulfil his role in full, which may be the cause of the group's plateau in growth.

A simple fix for this problem system would be to increase the number of leaders or members that have a similar role to Jesse. However, this will not deter from the fact that the group's system is fundamentally flawed. The group cannot run without a moderator and will continue to suffer from inaccurate data because of human error.

Jesse has decided that he would like an application that could potentially solve or reduce the problems he is facing with his current system.

Client Contact Information

CENSORED

Client Interview

To grasp a greater understanding and description of the problem, I have arranged a Skype conference with Jesse Stamper (JS). In this meeting, we will discuss his problem further and outline the steps that we must take to solve this problem.

What are your current roles as leader of the society?

JS: *My main job is to ensure that the group runs smoothly and in an organised fashion. The prospects of the group such as organising new fixtures and attracting new members are also vital parts of my role.*

What are the benefits of your current system?

JS: *The biggest benefit of our current system is that I know exactly what has been done and how it has been done. I do not need to worry if there has been a game played incorrectly compared to using an application where, if there had been a hardware or software problem I would remain unaware.*

What are the problems with your current system?

JS: *The biggest drawback of our current system are the restrictions that it has entailed on the group. There is a limit to how many people I can manage in the group and my own skills. Although I can usually be sure I have completed a task, I can never be positive I have done it correctly. On countless occasions, games have had to be replayed because I failed to record them properly or lost my recording sheet.*

How would an application benefit you?

JS: *An application would allow me to reallocate the work and time I currently put into this group on to other tasks which will benefit the group more. It would allow for me to spend more time advertising to other pupils and be much more competitive. If people could play games against each other whenever they wanted and however many times they wanted without somebody having to watch them, it would attract a lot of people!*

What features would you like in the application?

JS: *Speaking for the group here, we would like for the application to allow our members to play and compete with one another regardless of where they are or what time it is. We would like there to be additional features such as training, match making and just playing against friends. An amazing feature would be Elo system that would be a much more accurate and fun way to have our leaderboards.*

What would you like this application to run on? Should it be for IOS, Windows, Android, etc.?

JS: *The application should be universal. A large percentage of the society are also from the software department and we all run various operating systems and have different phone types. We have agreed that it should be web based.*

Are there any specific requirements you would like to specify for this application?

JS: *Ensure that it is secure. In the past, there have been societies who have tried a similar approach for example, the History society setup a blogging website that ended up being compromised and lots of personal information about the members was released. Ever since, people have been very cautious with their information.*

What is the age range for the prospective users of the application?

JS: *The youngest students at our school are 13 and I could not give you a figure for the oldest. Sometimes teachers take part and play in our society.*

Prospective User(s) Identification

The prospective users of this application will be the members of the Chess society at Eastland. A large proportion of the members are also studying Programming and Software Development and have members as young as 13. Members in the society are both male and female.

Section 1.2 Problem Modelling

Proposed System Objectives

1. The application must be web-based for cross system compatibility
 - 1.1. The website should be user-friendly and have a minimalistic design

- 1.2. The website should store information about users. Such as games played on the system, the number of players currently online and the number of games being played.
2. The system must be able to store information about the users
 - 2.1. Users of the system can register an account with the system
 - 2.2. Users of the system can login to use the application
3. Users should be allowed to play a game of Chess against:
 - 3.1. A computer with a difficulty level of their choice
 - 3.2. A friend
 - 3.3. A random player on the system by joining a matchmaking lobby
4. The system should keep track of a player's performance
 - 4.1. Every game played by a user against another player will be used to track this performance. The system will store the total number of wins, losses and draws for each user on the system.
 - 4.2. Players will have all be assigned an Elo rating that is a mathematical representation of a player's skill level.
5. The chess game should be multiplayer and update in real-time
 - 5.1. When one player performs an action in the game, the other player should be informed of this action as it happens.
6. The system should display a leaderboard that contains a list of players along with their performance
 - 6.1. The leaderboard should place the player with the highest Elo rating at the top of the board.
7. Users in a game of Chess should be allowed to:
 - 7.1. Export information about one of their games of Chess. This information should contain all of the moves made in the game.
 - 7.2. Resign from a game of Chess
 - 7.3. Ask the other player on the board to end the match as a draw

Limitations

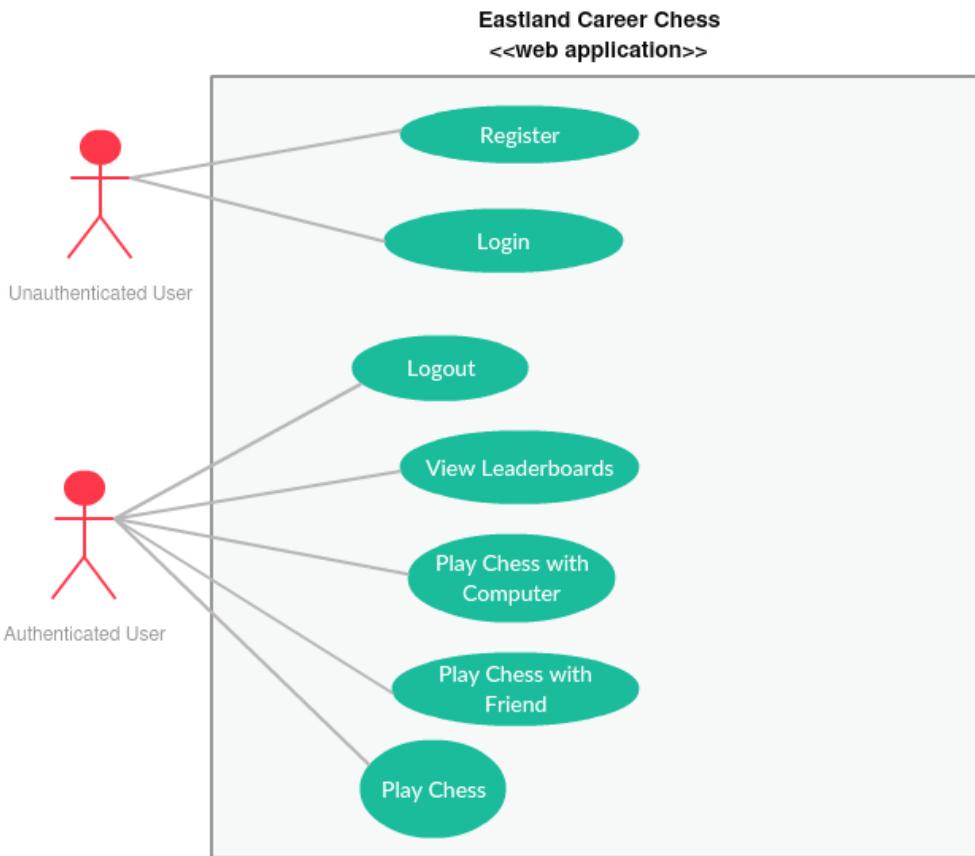
The proposed application is likely to have potential limitations. These limitations arise for various reasons for example, limited access to specific resources or they are unachievable due to the time constraints of the project. Here is a list of potential limitations for this application:

1. The program proposed is complex and will require a large amount of research before it can be tackled correctly. Due to this fact and the limitations of this project being completed solely by me, I may not be able to design certain aspects of this application as they are beyond my capabilities. These limitations can usually be overcome with additional research but there will always be a possibility that parts of the application may have to be scrapped.
2. The time provided to develop this application is restricted. The time constraint for development may mean that the application needs to be shrunk. If a part of the application ends up being limited due to a time constraint then, the problem should be evaluated from a feasibility stand point.

3. There will always be unforeseen circumstances that can cause the application to be limited in meeting its objectives. For example, one of the proposed requirements for the application was security. Although it would be more than possible to implement protection against various penetration attacks, there may still be a bug that runs on the foundations of the system. For example, there may be a problem with the operating system the application is deployed on causing for the application to not be as secure as the end-user would have liked for it to be or a security vulnerability that appears in the future which may not be solved due to the program no longer being maintained.

Use Case (UML)

I have created a visual representation for the actions of each user on the system so that Jesse and I can have a medium which we can both easily understand.



The diagram displays the two users of the system:

1. An unauthenticated user → A user who is not currently logged into the system
2. An authenticated user → A user who is currently logged into the system

The arrows pointing from each user are the actions that the user can perform, for example, a user who is authenticated can “Play Chess”.

To help grasp a better understanding of the problem for myself and the end-user, I have modelled the problem using a UML diagram. This diagram displays the users of the system and the actions that those users can perform. For example, an unauthenticated user will be allowed to register or login to the system but will the user will not be able to view the leaderboards of the system.

Section 1.3: Preliminary Research

Before embarking onto the solving the problem, I am going to undertake some research to grasp a better understanding of the project at hand. Although I am somewhat familiar with the game of Chess, there are rules and moves that I remain unsure of now. Additionally, I will be performing some research into Elo to locate the Mathematics behind the theory.

The following elements were researched preliminary meaning that all the below was completed before even attempting to solve any of the problems. This research has enabled

for myself to find better solutions to the problem and without doubt, sped up the solving process entirely.

Here are the elements researched:

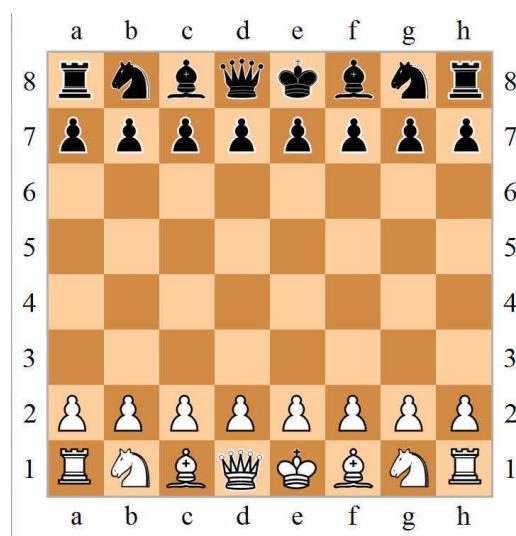
1. Chess
 - 1.1. How does the game work and what are the rules of the game?
 - 1.2. How can a game end?
 - 1.3. Are there any existing standards or techniques to implement or represent Chess?
2. Elo Systems
 - 2.1. What is an Elo system?
 - 2.2. How does an Elo system work?
 - 2.3. Are there any alternatives to using an Elo system and if so, are those alternatives a better solution to this problem?
3. Artificial Intelligence
 - 3.1. What is Artificial Intelligence?
 - 3.2. What is a Chess engine and how does it work?
 - 3.3. How do existing Chess engine's work and what are the names of current Chess engines?

The Game of Chess

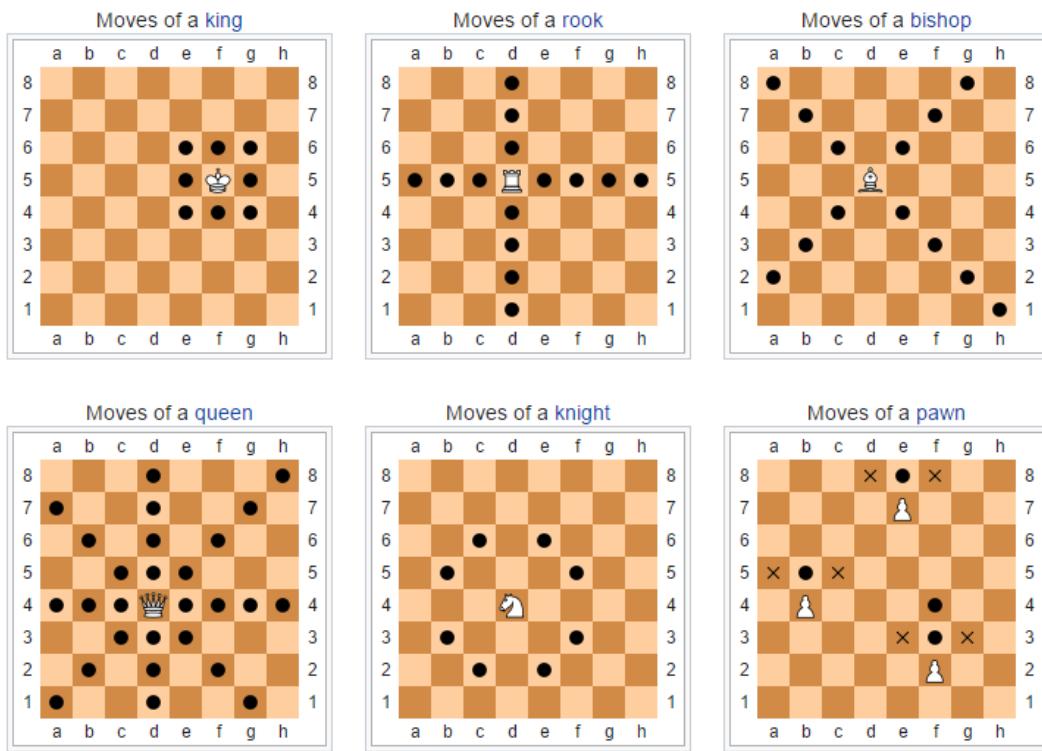
Chess is a two-player strategy board game played on a chessboard, a checkered gameboard with 64 squares arranged in an eight-by-eight grid. There are two players in a game of Chess who occupy the two teams. The first team is white and that player takes control of the white pieces and vice versa for black. Each game begins with 16 pieces and those pieces are arranged in the following order:

Both players on the board start with their pieces in the same position. The first row on the initial position shows: a rook, knight, bishop, queen, king, bishop, knight and rook.

The second row contains 8 pawns.



Movement of pieces:

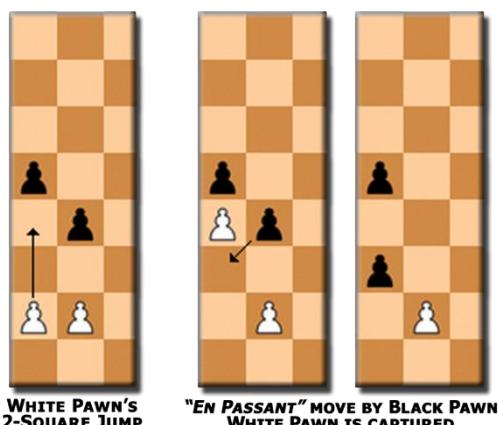


(1) <https://en.wikipedia.org/wiki/Chess>

Special Moves

En passant:

En Passant is a move in chess that allows for a special pawn capture, it can only occur immediately after a pawn moves two ranks forward from its starting position. An enemy pawn can capture that pawn had it only moved one square forward (the capturing pawn must be on its fifth rank before completing this move).

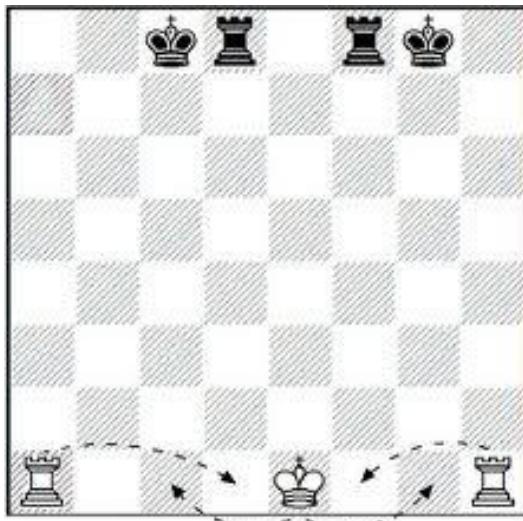


This example shows the white player's pawn moving forward by two ranks. The black player's pawn is currently sitting on the same rank on the file next to where the pawn has just been moved and uses the special pawn move En Passant to capture the opponent's pawn. The black pawn is now placed on the file of the captured piece and advances forward by one rank.

(1) <http://www.mchenryareachess.org>

Castling:

Castling is a special move for the king and it can be only completed once by each player in a game. When castling, a king and one of its rooks are moved simultaneously. The king moves two squares towards its rook and the targeted rook moves to the square at the other side of the king.



On the top of the board, the possible ends positions from black castling are shown. The king and the rook have swapped places. On the bottom of the board, the arrows indicate when castling is allowed and the final position for both pieces after castling has occurred.

However, castling is only allowed under certain conditions. These conditions are:

1. Neither the king nor the rook have previously moved during the game.
2. There cannot be any pieces between the king and the rook.
3. The king cannot be in check, nor can the king pass through squares that are under attack by enemy pieces, or move to a square where it would result in a check.

Promotion:

This is a rule that occurs when a pawn reaches its eighth rank, it is immediately changed into the player's choice of a queen, knight, rook, or bishop of the same colour. This new piece then replaces the pawn on the same square on the same move.

Game Conditions

Check:

Check is a condition in check that occurs when a player's king is under threat of capture on their opponent's next turn. A player must get out of check, if possible, by either interposing a piece between the threatening attack line from the attacking piece and the king, capturing the threatening piece or moving the king to a square where it is no longer in check.

Victory:

A game of chess can be won by a player in any of the following ways:

1. If a player cannot move out of check then he will lose the game. The opponent will be victorious as he has placed the opponent in checkmate.
2. A player can win the game if his opponent forfeits from the match.

3. Some games of chess are run under timed conditions where each side is only allocated a specific amount of time to complete the game. If one of the players runs out of time, the opponent will win the game. (**It is likely that this rule can be ignored as it has not been proposed as an objective by the end user.**)
4. A player can automatically be resigned from their games, if they are found to be rule breaking. An example of rule breaking would be the use of performance enhancing drugs or programs that find the best move for the player. (**This rule can also be ignored.**)

Draw:

A game can end as a draw which means that none of the players will lose or win the game. The following ways enable for a game to be ended as a draw.

1. Agreed Draw → This is the most common draw which occurs upon an agreement between the two players.
2. Stalemate → The player whose turn it is to move is not in check, but has no legal move.
3. Threefold Repetition → A player can claim a draw if the same position occurs three times, or will occur after their next move, with the same player to move. The repeated moves do not need to occur in succession.
4. Fivefold Repetition → This is like the three-fold rule however, there is no need to claim a draw, the game will automatically be a draw.
5. The fifty-move rule → If during the previous 50 moves, no pawn has been moved and no capture has been made then either player may claim a draw.
6. The seventy-five-move rule → Like the fifty move, players are forced to take a draw if the fifty-move rule reaches 75 moves.

The fifty-move rule, fivefold repetition and seventy-five-move rule are rule's that must be clarified with Jesse. These are special rules and are quite commonly ignored as they are very unknown.

Research Notes

1. This research has enabled me to grasp a much better understanding of the game and will benefit me when I attempt to implement certain aspects of this game. However, this research has resulted in a few questions that would be important to ask Jesse before moving forward:
 - 1.1. Would you like for there to be games that are time restricted on your application?
 - 1.2. Would you like for the website to handle cheaters?
 - 1.3. Would you like the application to include fivefold repetition?
 - 1.4. Would you like the application to include the fifty-five-move rule?
 - 1.5. Would you like the application to include the seventy-five-move rule?

Jesse and I agreed that we did not believe it would be necessary for us to arrange a conference to answer these questions. He could reply to them using Skype messenger, here was his answers to the questions:

JS: 1.1: We do not play timed games at our society as it would cause problems on the leaderboard.

1.2: Cheaters should not be a problem for us, don't worry!

1.3: No.

1.4: No.

1.5: No, I've never even heard of this rule.

Elo Rating Systems

Arpad's System

An Elo rating system is a method for calculating the relative skill levels of players in a competitor-versus-competitor game. The Elo system is named after its inventor, Arpad Elo and created to be used as an improved chess rating system. However, this technique has also been applied in multiplayer games besides chess.

Elo works by assigning each player a rating and this rating is a mathematical representation that can be used as a predictor for the outcome of a match. Two players with equal ratings who play against each other are expected to score an equal number of wins. A player whose rating is 100 points higher than their opponents would be expected to score 64% more wins than the other player.

The Elo number for a player will increase or decrease depending on the outcome of games between rated players. After each game, the difference in rating between the two players will determine the points gained or lost by the players (in addition to one point being taken from the losing side). For example, in a series of games between two differently rated players, the player with the higher rating would be expected to score more wins than the other player. However, if the player with the lower rating won then the number of Elo points gained by that player will be higher than the amount gained if the higher rated player had won. This also works the other way around, the player with the higher rating will lose more points than the lower rated player if he loses the game.

Benefits of Arpad's System

One of the reasons Arpad's system was so popular was because it was self-correcting. This means that if a player was rated incorrectly then he would either perform better or worse than the system would expect. Hence, in the long run the system would be able to readjust their Elo until it reflects their true playing strength.

Drawbacks of Arpad's System

Although this system is still widely used in competitive games till this day, the system is flawed. The problem with Arpad's Elo system is that player's ratings are not treated as if they were all estimated equally as precise. There is no distinction made between players whose ratings are precise measures of their strength versus imprecise measures of another. Systems that have continued to use Arpad's system find that player ratings vary in large magnitudes in a small space of time.

- (1) https://en.wikipedia.org/wiki/Elo_rating_system
- (2) https://en.wikipedia.org/wiki/Arpad_Elo
- (3) https://en.wikipedia.org/wiki/Chess_rating_system
- (4) <https://en.wikipedia.org/wiki/FIDE>

Glicko System

Glicko is an alternative system that is used to assess a player's strength in games of skill and is widely used on massively online games such as Counter Strike: Global Offensive and Chess.com. Instead of being a completely unique method to Arpad's system, Glicko is a more modernized approach to calculating relative skill. This system uses a much more complex formula that takes more information about a player into account before providing a ranking. The mathematics behind Glicko is extremely complicated and stretches much beyond my current knowledge and the requirements for this project. However, the formula has been tried and tested over the years and is well known to provide the most accurate rating of a player's skill. Glicko assigns each player in their system a rating, rating deviation and a rating volatility.

The volatility is a measure to indicate the degree of expected fluctuation in a player's rating and is one of the reasons why Glicko can produce more reliable ratings than Arpad's system. The volatility of a player will have a high measure when a player has had high performances after a period of stability, and the volatility of a player will be low when a player performs consistently.

The rating deviation of a user measures the uncertainty in a rating. A high rating deviation would correspond to a rating that is unreliable, a way to look at this would be that the rating deviation determines how confident the system is about their prediction. Glicko is a system that will never be completely confident in a rating that it has calculated instead, the system will work like so:

- **Glicko currently 95% sure that Jesse Stamper has a rating between 1120-1354 however, the system is estimating that his rating is 1201. This large gap between the two ratings is occurs since Jesse has a high rating deviation.**
- **If Jesse had a low rating deviation, then Glicko will have a different response. Glicko is currently 97% sure that Jesse Stamper has a rating between 1198 and 1202. The system estimates that his rating is 1199.**

Benefits of Glicko vs Arpad

1. The Glicko-2 Rating system is a much more modernized version of its previous successor, Arpad. The system uses much more complicated calculations that require computational power that may have not been around in the days of Arpad. Hence, this allows for Glicko to make more reliable calculations.
2. Glicko does not have one of the flaws that Arpad's system had, player's ratings will not vary by large magnitudes in a short period without reason.

The Drawbacks of Glicko vs Arpad

1. The Glicko system uses calculations that require more computational power than Arpad's system. This means that the time taken to calculate a rating of a player will be longer than the time it would take for Arpad's method.
2. The Glicko system uses extremely complex mathematics that can without a doubt become very confusing. Hence, the Glicko system is likely harder to implement compared to Arpad's system.

Conclusion

I have decided that I will be using the Glicko-2 rating system to create the leader board for this application. From my research, it is more than evident that Glicko will provide a more accurate result and therefore, improve the quality of the game. It is important to realise that the implementation of Glicko would be different to an implementation of Elo. According to the research paper for Glicko, the system works best when the number of games in a rating period is moderate, for example 5-10 games per player in a rating period. This means that the length of a rating period would have to be determined to optimize the system to its full potential.

- (1) <http://glicko.net/>
- (2) <http://www.glicko.net/ratings/glicko2desc.pdf>
- (3) http://www.englishchess.org.uk/wp-content/uploads/2012/04/The_Glicko_system_for_beginners1.pdf
- (4) <http://www.glicko.net/glicko/glicko2.pdf>

Research Review

To conclude my preliminary research, I believe that my research has been extremely beneficial. This has allowed me to gain a thorough understanding of my problem and identify solutions that I will research in more detail later. In addition, this has allowed for the reader of my project to gather some background on the features of my application. It is most certainly not the end of my research I will conduct for this project and it is likely there will be much more research required to solve this problem effectively.

The websites and papers used to perform this research have been listed alongside the relevant topics.

Section 1.4: Iteration 1

In this iteration, I will obtain and setup the software and the hardware required to develop this application. One of the reasons I have decided that I will be storing my application on my computer and my secure server is to prevent potential loss of data. If there was an unforeseen hardware failure or event that caused for my project to be damaged then, there I will always have access to a copy that is up to date. Additionally, the application is likely to be deployed on a Linux operating system and so, developing on a Linux server may expose bugs the application may face when deployed.

The proposed objectives for this iteration outline the steps required that must be taken to setup the developing server correctly. These steps would also be mandatory when deploying the application to the society. To complete these objectives, the following steps must have been taken before starting:

1. Ensure that the server's repositories are up to date
 - 1.1. On CentOS, this command is: **sudo yum update**
2. Install repositories:
 - 2.1. gcc-c++
 - 2.2. openssl-devel
 - 2.3. git

Conclusive Objectives

1. Obtain a local copy of NodeJS and MongoDB
 - 1.1. If supported, install the repositories via your Linux Package Manager
 - 1.2. If not supported, clone the repositories from Github and build from source
2. Using the Node Package Manager (NPM) install the following:
 - 2.1. express-generator
 - 2.2. mongodb & monk
3. Create a workspace directory for the project using Express Generator called 'Chess'
 - 3.1. Change the port of the HTTP Server to 3000
 - 3.2. Install and start the webserver using NPM
4. Using MongoDB create a new database called *Main*
 - 4.1. Inside of *Main*, create two new collections called *users* and *games*
 - 4.2. Create an authenticated user to *Main* with read and write privileges
5. Connect the workspace and the database together by using monk
 - 5.1. If hosted locally then the default address for the database will be
localhost:27017/Main

Section 1.5: Iteration 2

In this iteration of the project, I will implement the authentication system that was proposed in the application's general objectives. Before starting this problem, I had a conference with Jesse to discuss the information that he would like to store about his users.

End-user meeting

The main topic of discussion in our meeting was:

1. Application Data
 - 1.1. What information will we be storing about the prospective users?
 - 1.2. How should the prospective users register and login to the application?
 - 1.3. Would you like the users to have any additional features of authentication besides login and register. For example, should users be allowed to delete their accounts?
 - 1.4. Application Security

How would you like your users to sign up to the system? Would you like your users to can register or login using a social media account or should the application keep its own track of users?

JS: *I would like people to sign up to the application manually. It would enable for myself to have a look at the database every now and then to check who is registered on our application.*

What information would you like to be stored about your users?

JS: Their name, age and email address.

Besides login and signup, would you like your users to have any additional features? For example, can users change their email addresses or names on their accounts?

JS: *I much prefer handling any problems with accounts myself. Manually changing the data in the database or deleting accounts would be something I would like to deal with. Usually, this sort of stuff ends up causing more trouble than good*

Application security was a topic that we mentioned in our first meeting. Since we will now be storing passwords on our system so that users can login, do you have any preferred security measures?

JS: *I'm open to your decision on this one.*

I had planned to use bCrypt for the application. It's more than secure and is protected from various hash cracking attacks such as rainbow tables and is adaptive making it extremely resistant to brute-force attacks. Would you be happy with this?

JS: *Perfect.*

Any questions?

JS: *Nope, keep me updated.*

Research

Password Hashing Algorithm

This section contains information that goes under both Analysis & Design

I have chosen to use bcrypt as my hashing algorithm of choice for my project. The reason that I have chosen to use bcrypt instead of another hashing algorithm such as MD5, SHA1 or NTLM is due to the vast amount of security that is added to the system by using bcrypt. Bcrypt incorporates a salt into all its hashes which in turn protects the hash from rainbow attacks. However, a rainbow attack is only one method that people utilise to crack hashes, bcrypt is also an adaptive function meaning that the number of rounds a password is hashed with can be increased to become more bullet proof from other attacks such as brute force.

A brute force attack is the most common attack method used by hash crackers to crack a hash. Usually, this is not as simple as being pure brute force though instead, hash crackers use rule-based dictionary attacks which apply a set of rules to a list of words. These lists of words may be common password, pet names or anything of the sort that can be tested against a hash list to try and crack the hash. However, even if a cracker was using a very accurate dictionary to crack the passwords, then their results would not be promising.

The below tables display the number of hashes/second when performing a pure brute force attack to a single bcrypt hash with a 2 GHz core processor.

Number of Rounds	Number of Hashes
8	~40 hashes/sec
9	~20 hashes/sec
10	~10 hashes/sec
11	~5 hashes/sec
12	2-3 hashes/sec
13	~1 sec/hash
14	~1.5 sec/hash
15	~3 sec/hash
25	~1 hour/hash
31	2-3 days/hash

This table shows that when performing the fastest and least effective attack on a bcrypt hash, the reality is that cracking hashes is an intractable problem. For a cracker to even crack one hash, it could take days and potentially even years. On the other hand, performing this test against a more common hashing algorithm like MD5 provides results that indicate a very insecure hash. Cracking against a plain MD5 Hash using pure brute force can allow for crackers to check millions of hashes/second which in comparison to bcrypt is an enormous difference. Not to mention that most hashing algorithms are susceptible to rainbow attacks which use precomputed tables to reverse engineer hash functions. These attacks are extremely effective.

Benefits of Bcrypt

1. In the event of a database ever being leaked, hackers are very unlikely to crack any of the hashes inside of the database. It would be more than possible for a cracker to potentially crack hashes that are encryptions of common words like “dog” or “cat”. Regardless, even this could take a cracker hours to retrieve the plaintext string.
2. Bcrypt is protected against rainbow table attacks. This attack method is extremely powerful and successful when cracking hashes, protecting user’s passwords more.

Drawbacks of Bcrypt

1. Bcrypt requires a lot of computational power and time to hash a string for example, hashing a string using 10 rounds of bcrypt can take up 0.3 seconds. Although this number can be modified other hashing algorithms like MD5 need milliseconds to perform the encryption.
 - 1.1. It can be argued that the minimal additional time required to encrypt a string using bcrypt adds security benefits that outweigh this cost.

To conclude, hashing passwords using bcrypt provides the system with vast amounts of protection from hackers. However, I would like to address one more problem that I am not going to be implementing, but would like to make the end-user aware of.

Man in the middle attacks (MITM) work by intercepting a communication between two systems. In my application's case, there will be confidential information transferred over an insecure network (in absence of an SSL certificate). Take this example of how an attacker could intercept a communication.

1. A user requests to visit the server's the application's leaderboard page so the user requests the file "leaderboards.js" from the server.
2. A MITM could intercept by replicating the actions of the server. The attacker could sniff the request packet and then swap the "leaderboards.js" file requested by the user with a malicious "leaderboards.js" file.
3. The requester will then receive the "leaderboards.js" file and load the leaderboard page, unaware of the fact that the file had been tampered.
4. The modified JS file may redirect the user to a replica phishing website used to steal personal information.

It is more than obvious that this will cause a lot of problems for the security of the system. It would be more than possible for an attacker to intercept an attack to obtain a user's plain_text password in the middle of a request. Sadly, there is no way to combat this problem without the use of an SSL certificate. An over-the-wire transmission approach could be taken where the plain text password would never be sent across the network and instead, the hash of the password would be sent across the system. Regardless, this would not stop the possibility that the request had already been tampered with and the plain text password was being sent to the attacker's database.

I would like to acknowledge the existence of this problem and ensure that the end-user will be informed of this issue. This issue is one that all Javascript developers face at least once in their careers and can be solved with a simple SSL certificate.

- (1) <https://en.wikipedia.org/wiki/Bcrypt>
- (2) <https://bcrypt.codeplex.com/>
- (3) https://www.owasp.org/index.php/Man-in-the-middle_attack

Conclusive Objectives

Here are the proposed objectives for this iteration.

1. The application must be able to differentiate between an unauthenticated user and an authenticated user
 - 1.1. An authenticated user will have an existing session with the application whereas an unauthenticated user will not have a session with the application
2. The application must allow for unauthenticated users to register to the application.
 - 2.1. For the user to submit a registration form, they must provide their first and last name name, date of birth, email address and password for the account they wish to register
 - 2.2. The first and last name provided by the user must both not be greater than 35 characters
 - 2.3. The email address provided by the user must follow the format of an email address (user@example.com) and must not be greater than 64 characters.
 - 2.4. The password provided by the user must be between 8 and 32 characters
 - 2.5. The date of birth provided by the user is required to ensure that the user is at least 13 years old

- 2.6. If the information provided by the user is valid then the email address provided must not match an email address that already exists inside of the database
 - 2.6.1. The password should be hashed using 10 iterative rounds of bcrypt and then, along with rest of the data, imported into the *Users* collection
- 2.7. If the user successfully registers to the application, they will be authenticated to the account they have created and redirected to the applications index page
- 2.8. If the user submits an unsuccessful request, they shall be redirected back to the registration page along with information related to their failed request.
- 3. The system must allow for unauthenticated users to login to the application
 - 3.1. For the user to login to the application, they must provide an email address and password
 - 3.2. The email address and password provided must match the valid data conditions located on the registration page
 - 3.3. If the email address provided by the user exists in the *Users* collection then, the corresponding hashed password for that user will be pulled
 - 3.4. The hashed password from the database and the password provided by the user shall be compared using bcrypt
 - 3.5. If the passwords match, then the user shall be authenticated to the application and redirected to the application's index page
 - 3.6. If the user submits an unsuccessful login request, they should be redirected back to the login page and informed about the problem with their request
- 4. The system must allow for authenticated users to log out of the application
 - 4.1. A user must have the ability to terminate their login session by clicking the logout button on the homepage
 - 4.2. The user who requested the logout will have their unique session terminated and will be redirected to the login page (they will also no longer be recognised as an authenticated user by the application)

Section 1.6: Iteration 3

This section contains information that goes under both Analysis & Design.

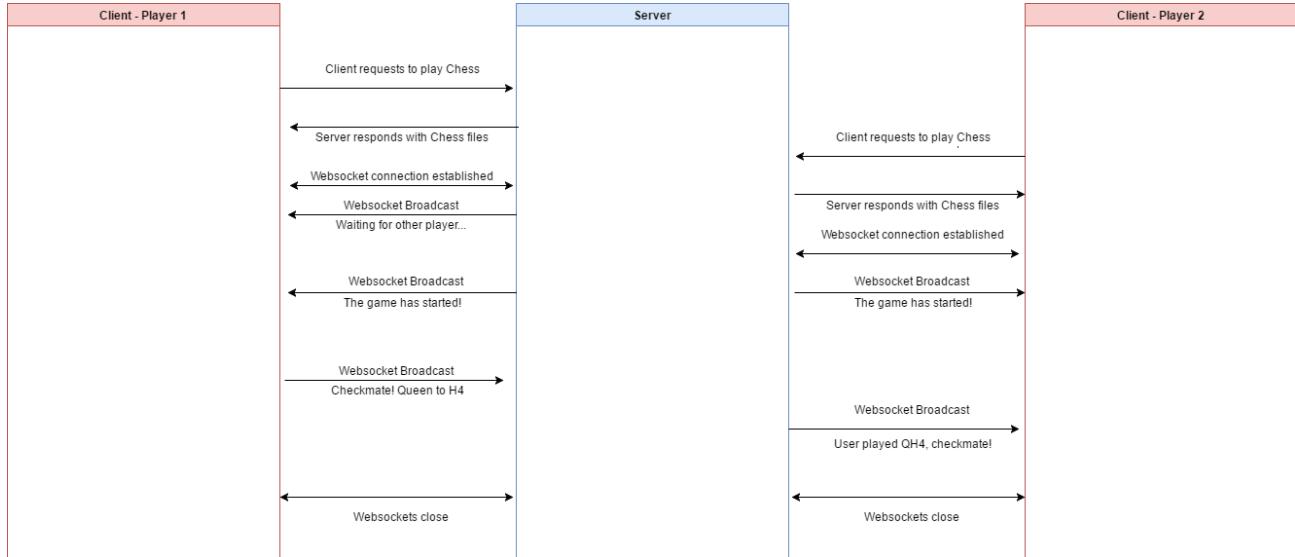
Handling Real-time Multiplayer

To handle the real-time aspect of the application, there would have to be a communication line which is always open between client(s) and the server. My proposed solution to this problem is with the use of WebSockets.

In simple terms a WebSocket is a communication protocol that allows for full-duplex communication over a TCP connection. This means that WebSockets can communicate in both directions simultaneously, a message can be sent to the other person on the socket but, at the same time the sender can receive a message from that person.

In my case, there would have to be a system that is constantly listening out for a communication from the other side. If a chess piece has been moved by a player on the board then the other player would have to be informed of this change but, it would be unsafe and unstable to have a direct communication between the client(s). This is because the server would not know the current state of a game or if the messages sent by the clients are valid. Thus, there would have to be a socket connection from both clients to the server.

There would have to be a “Client ↔ Server ↔ Client” model, where the client(s) of the system can communicate messages about the game to the server and then, the server can verify these messages and pass them onto the other user. To grasp a better understanding of this concept, I have produced a non-standard diagram of this concept:



The diagram above is replaying the following situation in a game:

1. Player A starts a game of chess
 - 1.1. A WebSocket is established with the server and a join message is emitted from the client informing the server that the player has joined game number X.
2. The server emits a message in response to inform the user about the current state of the game. If there is already another socket in this game then the game can begin but, if there is not then the user must wait for the second player.
3. Player B starts a game of chess
 - 3.1. A WebSocket is established with the server and a join message is emitted from the client.
4. The server emits a global message to sockets on the game informing them that the game is now starting
5. Player A is playing as white and moves his pawn to E4
 - 5.1. A message is emitted to the server from the client informing the server of the player's move
6. The server verifies the move received from the player and passes this move in a message to the other user.
7. Player B will receive the socket message and the board will be updated.

This process continues until the game has ended. The purpose of this diagram and explanation is to model the problem of multiplayer so that it can be easily solved when proposing objectives.

Benefits and Drawbacks of Solution

Benefits of Solution:

1. Changes can occur based on messages from the server in real-time. This means that the application would be able to update as a move is made by another player.
2. Socket connections can be serialized allowing for games and players actions to be easily tracked. For example, the application could use sockets to detect how many players are currently online.

Drawbacks of Solution:

1. WebSockets are not supported by all internet browsers. Internet Explorer 9 and earlier are not supported.
2. Potential security issue at times if there is a constant connection opened between the client and the server. Users may send data unexpectedly to try and change the outcome of a game.

Game Rooms

The application would also have to handle multiple games being played on the system simultaneously. This means that the application must be able to differentiate between different games on the system, a way this can be done is by assigning each game being played a unique identifier.

The unique identifier for each game should be a string that is both long and random. By having a long-randomized string, this will prevent unwanted parties from discovering the game id who may attempt to intervene or cause harm to the application. However, the string should remain a reasonable length for example, it would be inconvenient and unappealing for users if a game identifier was 400 characters long.

Each game identifier will then be stored in the application, inside of the game rooms store. This means that when the application is running a game, they can be addressed inside of this game room store.

Benefits of Solution:

1. The application will be able to differentiate between different game rooms. Thus, allowing for multiple games of Chess to be played at the same time.

Conclusive Objectives

1. A user must have the ability to communicate messages to and from the server
 - 1.1. A user must have the ability to establish a connection with the server by accessing the play page. On this page, a WebSocket will be established with the server
 - 1.2. A user can emit messages via the WebSocket to the server user must be able to emit messages to the server via the WebSocket
 - 1.3. The server must be able to emit messages to the user via the WebSocket
 - 1.4. The client and the server must be able to both parse and read incoming messages from the WebSocket
 - 1.5. Once the connection has been established, the user will emit a join message to the server

2. A user must have the ability to start and play a game of Chess
 - 2.1. A new game of Chess shall be created when a user accesses the application's play page without a game identifier
 - 2.2. The board creator must be seeing the board from White's perspective
 - 2.3. The user must be able to select, drag and drop Chess pieces on the board
 - 2.4. For a user to select a piece, they must click a piece using their mouse. To drag, they must hold their mouse click down and to drop they must release their mouse click while dragging a piece
 - 2.5. The user can modify the position of a piece by dropping a selected piece onto a new square on the board
 - 2.6. The user can return a selected piece to its original position by dropping it off the board
 - 2.7. The user can return a selected piece to its original position by dropping the piece returning the piece to the square it was selected from
 - 2.8. The user will be provided with a link to the playing page with a game identifier attached. The game identifier will be a random string of 20 characters and letters
3. Another user can play the same game of Chess as another user
 - 3.1. The second user can join the game of Chess by joining the unique game link provided to the other user on the board
 - 3.2. As the user access the game, they will receive the current board state in a Forsyth Edwards Notation (FEN) string in a message from the server (message delivered via their WebSocket)
 - 3.3. The FEN string received by the user will be the new board state and replace the current board state
 - 3.4. The second user can perform the same actions that the first user can perform
4. The user(s) playing a game of Chess on the same board will view their opponent's actions on the board in real-time
 - 4.1. When either of the users perform an action on the board that causes the state of the board to change (moving a piece). Then a message will be emitted through the WebSocket containing the new state of the board in FEN
 - 4.2. This string will then be emitted by the server to the other user's in the game who did not perform the action
 - 4.3. Upon receival, the user's board state will change to the new board state contained in the WebSocket
5. The clients and the server will provide data about any messages received through their WebSocket(s)
 - 5.1. The server will log any messages received from a user to the console in JSON
 - 5.2. The username, gameID and date will be logged on the server if a user emits an FEN string
 - 5.3. Users who receive an FEN string will have a copy logged to their console
 - 5.4. The server will be informed if a user terminates or destroys their WebSocket connection

Section 1.7: Iteration 4

Objectives

This iteration will focus on implementing the ability for users to play versus the Stockfish Chess AI. Users can customize the difficulty of the Chess engine by selecting a level between 1 and 20. As the level increases, the engine will become harder to beat.

1. Users can customize the difficulty of the Computer they wish to play against
 - 1.1. For a user to set the difficulty of the Computer, they will select a value on the slider on the home page after clicking the button labelled *Play versus computer*
 - 1.2. The slider will allow for the user to select a value between 1 and 20
 - 1.3. The difficulty level selected by the user will be the number of depths searched by the engine (if the user selects difficulty level 1 then the engine would search with depth 1)
 - 1.4. Users can press the button labelled *Play* to start the game
 - 1.5. The user will be redirected to a page with their selected difficulty inside of their request URL
2. Users can play a game of chess against the computer of their choice
 - 2.1. The user will start a new game of chess after landing on the play page of their choice
 - 2.2. The user will be playing on the board as white and be viewing the board from white's perspective
 - 2.3. The user can only select pieces of the game if it is not over or in a draw
 - 2.4. The user can only select a piece on the board when it is white's turn in the game
 - 2.5. The user can only select, drag and drop pieces on the board that are coloured white
 - 2.6. The user will be provided with the current state of the game in a textbox that will provide information about the game (Your turn | Opponent's turn | Game has ended)
 - 2.7. The user will be provided with the PGN string for each move of the game
 - 2.8. The user can download the PGN string of all the moves in the game to a .TXT file by clicking the button labelled Download PGN
 - 2.9. The user will be able to view the level of the engine that they have selected to the right of the board
 - 2.10. The Stockfish engine will provide the user with information about its move (the depth level, number of nodes searched and the best move the engine could find)
3. The Stockfish Engine can play a game of Chess against a user
 - 3.1. The engine will search to the depth provided by the user, the higher the depth then the engine will explore more nodes (moves)
 - 3.2. The engine will be playing as black on the board and will only be able to move the black coloured pieces on the board
 - 3.3. The engine cannot make a move when the game is over, in a draw or if the opponent is making a move.

Section 1.8: Iteration 5

Objectives

1. Users of the system will now have a game score. A user's game score will store their total number of wins, losses and draws.
 - 1.1. For a user's game score to be updated, they must finish a game of Chess. This includes a player winning by checkmate or their opponent resigning from the game
 - 1.2. If a user wins a game, their total number of wins will increment by one
 - 1.3. If a user loses a game, their total number of losses will increment by one
 - 1.4. If the game ends as a draw, then both players will have their total number of draws incremented by one
 - 1.5. When a game has ended by checkmate. The player who instantiated the checkmate will have their number of wins increases by one and their opponent will have their number of losses reduced by one.
 - 1.6. Game score should only update if the user is playing against another player. Their game score should remain unchanged if they are playing against a computer
2. Users of the system will have an Elo Rating. This rating will be calculated using Glicko-2 and will update depending on the results of games that they have played
 - 2.1. Upon registering to the application, each user will be assigned a default rating of 1000
 - 2.2. At the end of each game that a player now plays, the result will be stored on the server along with the two players who were in the match
 - 2.3. Players rankings will be updated every 24 hours on the system's Leaderboard
3. Users can now view information about other players of the system on the Leaderboard
 - 3.1. The Leaderboard will be arranged from top to bottom with the top row containing information about the player who has the highest rank
 - 3.2. The Leaderboard will display the full name, rating, age and game score of a user
 - 3.3. Players can search the database using the search button to find information about a specific user
 - 3.4. Players can change the number of fields shown or view the next page of rankings by either pressing the next page or changing the number of entries shown
 - 3.5. The default number of entries shown to a user will be 10
4. Users can now view statistics about the status of the application
 - 4.1. User's can view the total number games currently being played on the home page. This will represent all games that are being played on the system, including games against a computer
5. Users can now play a game against a random player by using the solo option on the home page
 - 5.1. After a user selects the button labelled Play Solo, a WebSocket will be established with the server that emits a message confirming there is a user in the lobby
 - 5.2. When there are two users in the lobby, the two players will be matched up to play a game with one another.
 - 5.3. Both users will be redirected to a game with a unique identifier and can play a game of chess, the same as if they were playing against a friend
6. Users will now have persistent sessions with the system. In the event of the server going offline or restarting then, users will maintain their login session.
 - 6.1. On the event of a user logging into the system, a session will be added to the database for that specific user.

- 6.2. The session will contain serialized information about that user, cookie information and most importantly an expiration date
- 6.3. The expiration date for a login session will be 24 hours which means that if a user does not terminate their session, they will remain authenticated on their device for 24 hours (even in the event of the server being turned off)
- 6.4. After 24 hours, the session will be terminated and the user will have to log in to the system again to create a new session
- 6.5. Users can also terminate their sessions earlier than the expiration date provided by logging out of the system

SECTION 2: DESIGN

Section 2.1: Declaration of Programming & Database Languages

I have decided to develop my application using a combination of languages. The application will be developed using NodeJS but incorporate hypertext languages, jQuery and AJAX. I have chosen to develop the application in Node because of its incredible performance, being built on Google's V8 engine, Node is more than capable of handling numerous simultaneous connections. In my case this is a perfect choice of language, Node will allow for me to tackle the multiplayer aspect of the application and ensure that the product is universal. In addition to can solve this problem, Node has the largest ecosystem of open source libraries in the world. This library is known as the Node Package Manager (NPM) enabling me to access various modules to perform certain tasks.

These libraries can be very time-saving and useful in situations for example, in Node you must create your own HTTP server meaning all the routes must be organised and created by yourself. However, this task is far from difficult and there are modules such as Express which are capable of handling basic routing and the creation of a HTTP server. Alongside Node, I will be using MongoDB.

MongoDB is a database type that goes under the term NOSQL banner, instead of using tables and rows like a relational database, Mongo is built on an architecture of collections and documents. Documents comprise key-value pairs (unique identifiers that point to the location of data) and are the basic units of data in Mongo. This means that unlike SQL, Mongo does not require for types of values to be specified such as VARCHAR (255) or BOOLEAN()

The front end of the application will be designed using HTML5, CSS & JS while the back-end will be running on Node to handle all the routes for these requests, piping of files etc. Additionally, I will be using the E-JS templating engine in my HTML files.

Asynchronous vs Synchronous Approach

One of the most important topics to note about this application being developed in Node.js is the underlying execution model of Node compared to other languages such as PHP, Python or Java. For example, a node application that pulls a list of all the usernames from a table and then prints "Hello World" to the console will work as follows:

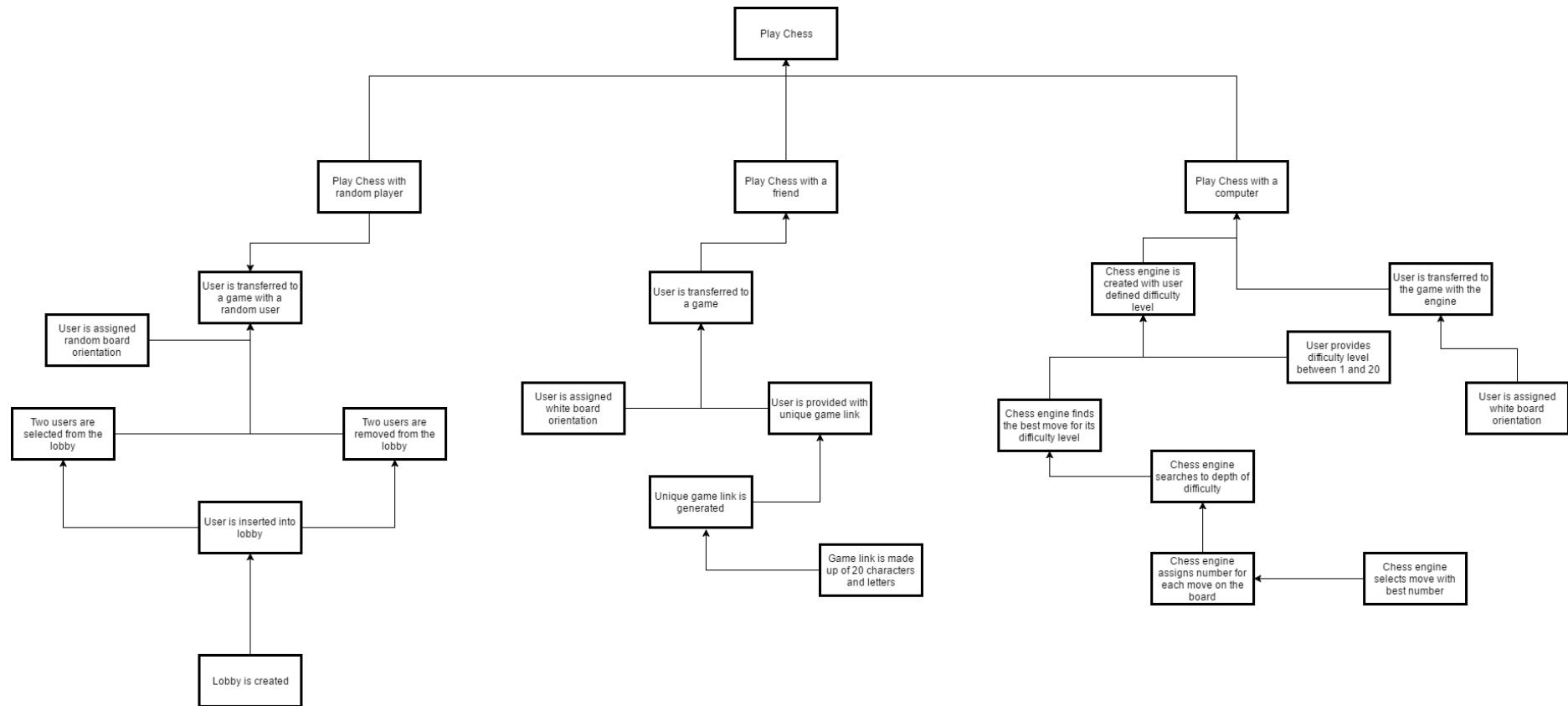
1. All the usernames will be fetched from the database.
2. After the usernames, have been successfully fetched from the database then Hello World will be printed to the console.

At first glance, this seems to execute similarly to PHP. However, the difference in Node is that if the database query was extremely slow then the Hello World would not be printed to the console until that line of code has been successfully executed whereas in PHP there

would be a unique process for every HTTP request. Therefore, in PHP, if a request results in the execution of a piece of code being slow, it would result in a slow loading page for this user, but other users requesting other pages would not be affected. NodeJS only uses one process and so if there is a slow database query inside of this process, this creates a global problem effecting all users on the site.

For this reason, NodeJS uses the concept of event-driven, a-synchronous call backs by utilizing events loops. In situations like the one mentioned above, we would pass a call-back function as a second parameter which enables for the code execution to be asynchronous. Simply put, this allows for the code to not have to wait for the database query to be completed. Instead Node will make a note after executing the database query to execute the anonymous function that was passed and then immediately executes the “Hello World” line of code to the console. Node will continue to cycle through what is known as an event loop until there is nothing else left for it to do.

Section 2.2: Modelling



Confidential – Do not release

If released somewhere online please let me know at vesta06@protonmail.com

This diagram breaks down the three game modes of the application into smaller components.

The three game modes are:

1. Play a game of chess with a random player
2. Play a game of chess with a friend
3. Play a game of chess with a computer

Each game modes will be sharing some attributes and these attributes are:

1. The chessboard and rules of chess.
2. The ability for users to be able to select, drag and drop pieces on the board.

The game modes that involve playing against another player will also have the common modules:

1. Inserting the results of a game into the database
 - 1.1. Users will have their game score updated in the database accordingly. If they win, their total number of wins increase by one etc.
 - 1.2. Using the results of those games to later update the rankings of players

Board States

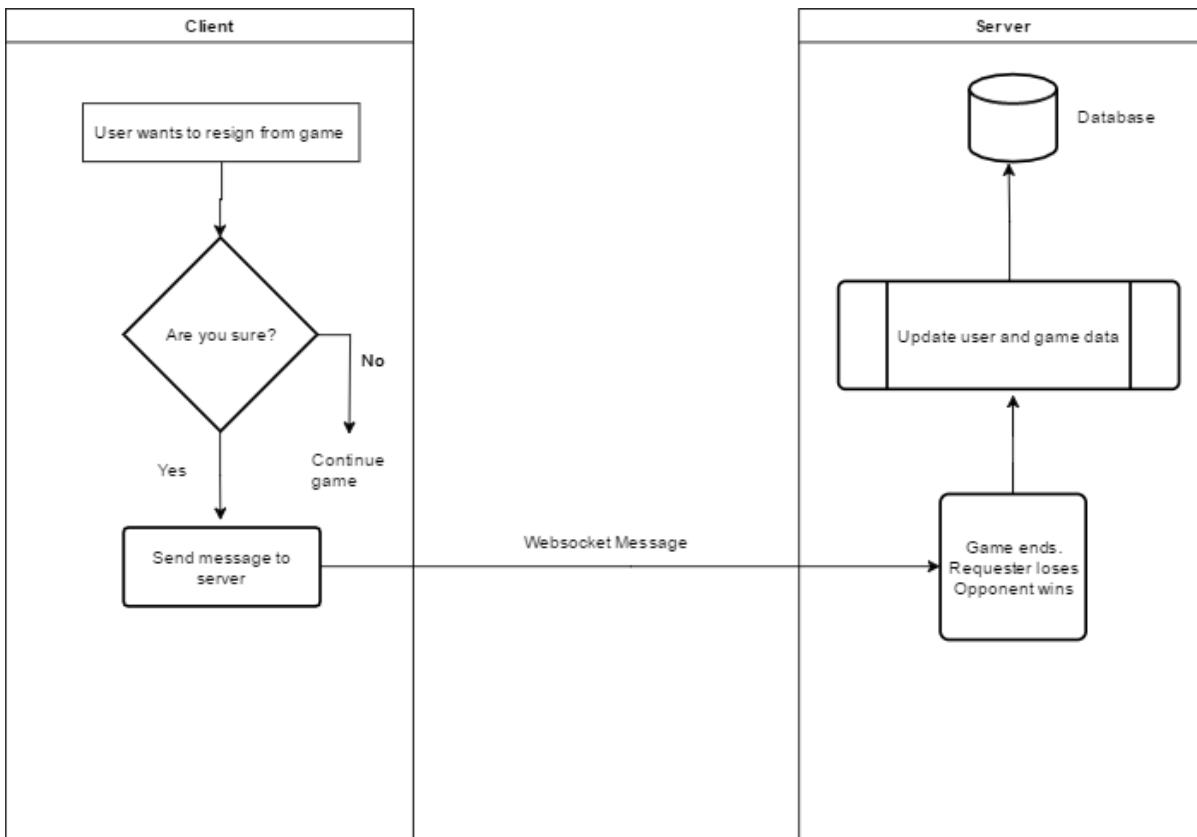
Each game mode has a board and the board for each chess game can be in various states. Below is a table that models these potential states and explains the purpose of each state.

State Name	Purpose	Explanation
WAITING_JOIN	<i>To determine whether there are enough players on the board for the game to begin.</i>	In a player versus player game, the waiting join state will be triggered when there is only one player on the board. In a player versus computer game, the waiting join state will be triggered during the time it takes for the Stockfish chess engine to start.
YOUR_TURN	<i>To inform a player if it is their move on the board.</i>	The board state will be set for a client depending on if it is his move. If it is the client's move, then appropriate messages can be displayed to that user.
WAITING_MOVE	<i>To inform a player if it is their move on the board.</i>	The board state will be set if it is not currently the user's move on the board. Appropriate messages can be displayed and the client can be restricted from trying to move pieces on the board.
GAME_OVER	<i>To detect if a game of chess has ended.</i>	The board state will be set to GAME_OVER if a game has ended (the ways a game can end have been explained elsewhere in this document) This state can be used to prevent games that are not currently live from being used on the active games counter. This state can also be used to prevent users from moving pieces on the board.
IN_CHECK	<i>To detect if a player has been placed into check.</i>	The board state will be triggered if the client has been placed into check in a game. This can be used to display appropriate messages and prevent the user from making invalid moves.
CHECK	<i>To detect if a player has placed his opponent into check.</i>	The board state will be triggered if the client places his opponent into check. This can be used to display appropriate messages to the client.

User Game Actions

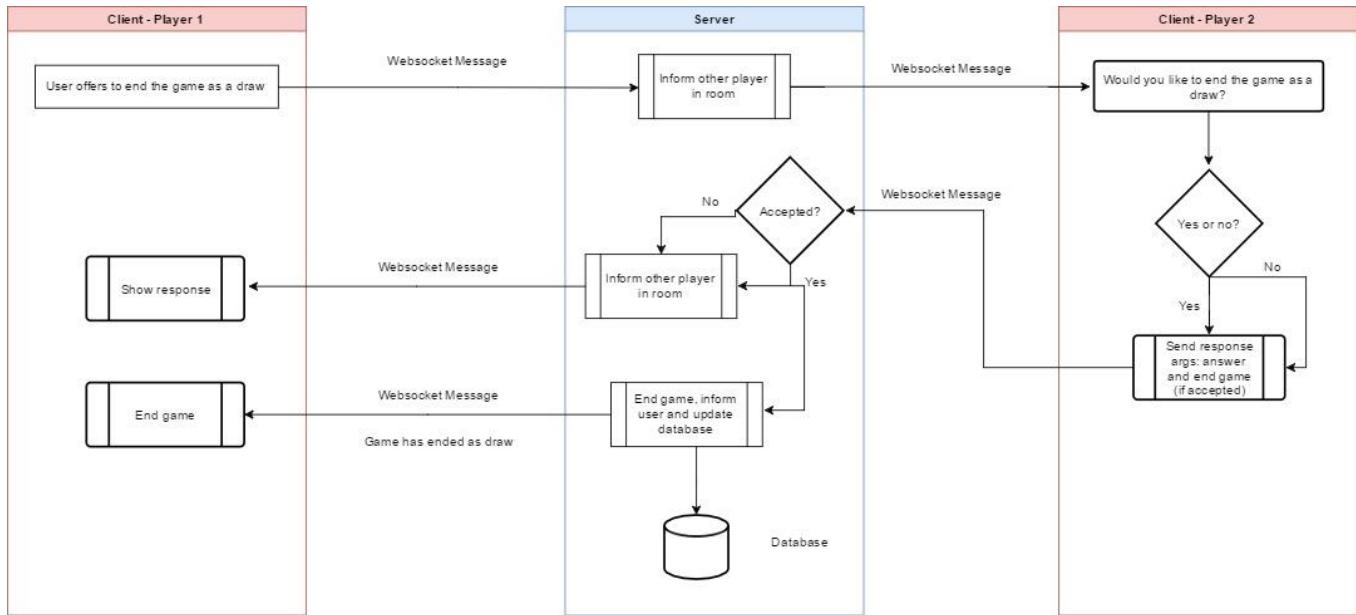
In a game, users can perform certain actions which will change the status of a board. These actions will not occur as part of the chess game and instead, these actions will trigger an event in the game.

User resigns from a game:



1. User selects the resign button in his game:
 - 1.1. User is asked for confirmation, if yes → continue with request otherwise → cancel request and continue with the game.
2. A message is sent via the WebSocket to the server should contain the following information:
 - 2.1. UID for the user who would like to resign from the game.
 - 2.2. Unique game identifier
3. Server receives the request and:
 - 3.1. Changes the status of the board to GAME_OVER.
 - 3.2. Informs the other player on the board that he has won the game.
 - 3.3. Updates the game score for both players in the database (The requester will get +1 to his number of losses and his opponent will get +1 to his number of wins).
 - 3.4. Data about the game and the results of the game will be inserted into the games collection for storage and usage in ranking updates.

User offers to end the game as a draw:



1. Player 1 offers to end the game as a draw
2. A WebSocket message is emitted containing:
 - 2.1. UID of the player who requested to end the game as a draw
 - 2.2. Unique game identifier
3. Server receives the request and then informs the other player in the room.
4. Player 2, the other player in the room, is then asked if he would also like to end the game as a draw.
 - 4.1. A WebSocket message will be emitted to the server containing the answer from the user.
 - 4.2. If the user said yes then the game will also end for the client, disabling the user from making a move on the board and changing the game state to GAME_OVER
5. The server will parse the message and check if Player 2 accepted and then inform the other player in the room
 - 5.1. If the player said yes then, the game will end and the results will be inserted into the database.
 - 5.2. The players game scores will be updated and both players will have their number of draws incremented by one
 - 5.3. The game will also be recorded in the games collection for storage and so that it can be used in a ranking update
6. The other player on the board will be informed that the other user has agreed to his request and the game will end

Section 2.3: Iteration 1

Database Design

As stated in the declaration, my data basing system is built on an architecture of collections and documents. Consequently, Mongo can store data types that normal SQL databases were not able to hold. Additionally, most of the time NOSQL databases are schema-less or at least have a flexible database scheme.

Now, there are no standard methods to visually represent a NOSQL database; however, I have discovered that developers have been using JSON & various other non-standard diagrams to represent their databases. Since Mongo has such a flexible schema, the way in which the database was modelled for this application was by using a series of questions. Compared to SQL where data is very specific and you already have the answers to the question.

I modelled the schema for the user based off the following questions:

What do we want to store about the user?

1. *First Name*
2. *Last Name*
3. *Email Address*
4. *Date of Birth*
5. *Hashed Password*

What are the components of all the fields required?

1. *A string*
2. *A string*
3. *A string*
4. *An object*
5. *String*

We can store the date of birth of a user inside of an object because it is supported by MongoDB. Inside of the object we will store:

1. **Day → Integer**
2. **Month → Integer**
3. **Year → Integer**

User Schema

Since there are no standardised diagrams for a NOSQL database and there are none specified in the NEA guide, I have decided to follow other developers in representing my databases and collections using JSON.

```
userSchema      : {  
    firstName   : String,  
    lastName   : String,  
    email      : String,  
    password   : String,  
    birthday   : {day: Number, month: Number, year: Number},  
,
```

Benefits of User Schema:

The user schema above represents the data structure that will be stored in MongoDB for a user. As shown in the birthday, Mongo can store objects inside of the database meaning that storing data about a user is extremely flexible. Schema is not even required for a NOSQL database however, a schema is always useful when passing on to other developers and ensuring that the database follows the same structure throughout.

Confidential – Do not release

If released somewhere online please let me know at vesta06@protonmail.com

Data Dictionary

Below is a data dictionary for the local object inside of the user's collection:

Field Name	Data Type	Validation	Validation Explanation	Example of Correct Data
Email	String	Does the email address contain an @ sign? Is the email address greater than 64 characters?	All email addresses must contain an @ sign before their domain. If an email address does not contain an @ sign then the email address would be invalid. The maximum length of an email address is 254 characters however, 64 is just a well-rounded number for email addresses. Most people do not have email addresses larger than 64 characters and increasing the size would just require additional bytes for storage.	example@example.com
Password	String	Is the password less than 8 characters or greater than 32 characters?	A password that is less than 8 characters is insecure. In the event of the database being compromised, regardless of the password encryption, the password could be easily cracked by a hash cracker. Passwords that are greater than 32 characters are just excessively long and there would be no reason for a user to have a password if that. Also, the increased string length would mean that the system would have to use more computational power to encrypt the password.	djiaksalz

firstName	String	Is the first name greater than 35 characters?	A UK Government Data study recommends that a first name is not greater than 35 characters.	CENSOR
lastName	String	Is the last name greater than 35 characters?	Same as firstName .	BLUR
Month	Number (int) In JSON, integers are known as “numbers”	N/A	Users are restricted to the values they can input. (Numbers 1 → 12)	4
Day	Number (int)	N/A	Users are restricted to the values they can input. (Numbers 1 → 31)	28
Year	Number (int)	Is the year greater than the current year – 13?	Ensure that the user is old enough to be registering to the application.	1999

Confidential – Do not release

If released somewhere online please let me know at vesta06@protonmail.com

Description of Algorithms

Registration Algorithm:

USER INPUT First name

IF > 35 characters → Registration failed

USER INPUT Last name

IF > 35 characters → Registration failed

USER INPUT Email address

IF > 65 characters → Registration failed

USER INPUT Password

IF < 8 OR > 32 characters → Registration failed

USER INPUT Date of birth

IF birthyear > (Current Year – 13) → Registration failed

CONVERT email address to lower case

SEARCH email address in users collection

IF already exists → Registration failed

OTHERWISE:

HASH password using 10 rounds of brypt

INSERT user object into users collection

Login Algorithm:

USER INPUT Email Address

USER INPUT Password

SEARCH email address in user's collection

IF does not already exist → Login failed

OTHERWISE:

CONVERT email address to lower case

RETRIEVE and STORE corresponding hash for user

COMPARE password and hash

IF match → Login successful

OTHERWISE:

Login failed

Convert email address to lower case

Search email address inside of user's collection

IF the email address does not exist

Login failed

Database Queries

User.findOne({ 'local.email': email })

Search the entire email filed in the user's collection to check if there is a string that contains the request email inside of it. If so, a document will be returned.

Section 2.4: Iteration 2

Sample of Client Socket Messages

Joining a game:

```
OBJECT  
ACTION: 'JOIN'  
DATA: OBJECT  
    GameID: Room number  
    Username: Email address  
    Side: Orientation
```

Moving a chess piece:

```
OBJECT  
ACTION: 'MOVE'  
DATA OBJECT  
    GameID: Room number  
    PGN: Portable Game Notation string  
    Username: Email address
```

Sample of Server Socket Messages

When a user joins a game:

```
FIND game room  
FIND players inside of game room  
  
WRITE "PLAYERS_UPDATE" to players in room  
DATA: OBJECT  
    Username: Name of player who has joined  
    Side: Orientation of joined player  
    Players: Players in game room
```

When a user makes a move:

```
FIND game room  
WRITE "UPDATE" message to other player in room  
DATA OBJECT  
    PGN: Portable Game Notation string of move  
    User: Name of player who made the move
```

When a user leaves a game:

```
WRITE "leave" message to other user in room  
    "USER" has left room
```

Section 2.5: Iteration 3

Sample of Client Socket Messages

Joining a game:

```
OBJECT  
ACTION: 'JOIN'  
DATA: OBJECT  
    GameID: Room number  
    Username: Email address  
    Side: Orientation
```

Moving a chess piece:

```
OBJECT  
ACTION: 'MOVE'  
DATA OBJECT  
    GameID: Room number  
    PGN: Portable Game Notation string  
    Username: Email address
```

Sample of Server Side Messages

Moving a chess piece:

OBJECT

ACTION: 'MOVE'

DATA OBJECT

GameID: Room number

PGN: Portable Game Notation string

Username: Email address

Front-end Design

Jesse and I had a meeting where we discussed and created a design for the front end of the application. The process was interactive, I would put his ideas to paper and add some of my own ideas until we were both happy with the end design.

Interactive Meeting:

I was thinking that we should try to create a quick hand drawn design for the user interface. Judging from our previous meetings, you are wanting the application to be easy to use and pleasing to the eye.

JS: *That's correct, I know that website design is not your speciality so just try to keep it simple.*

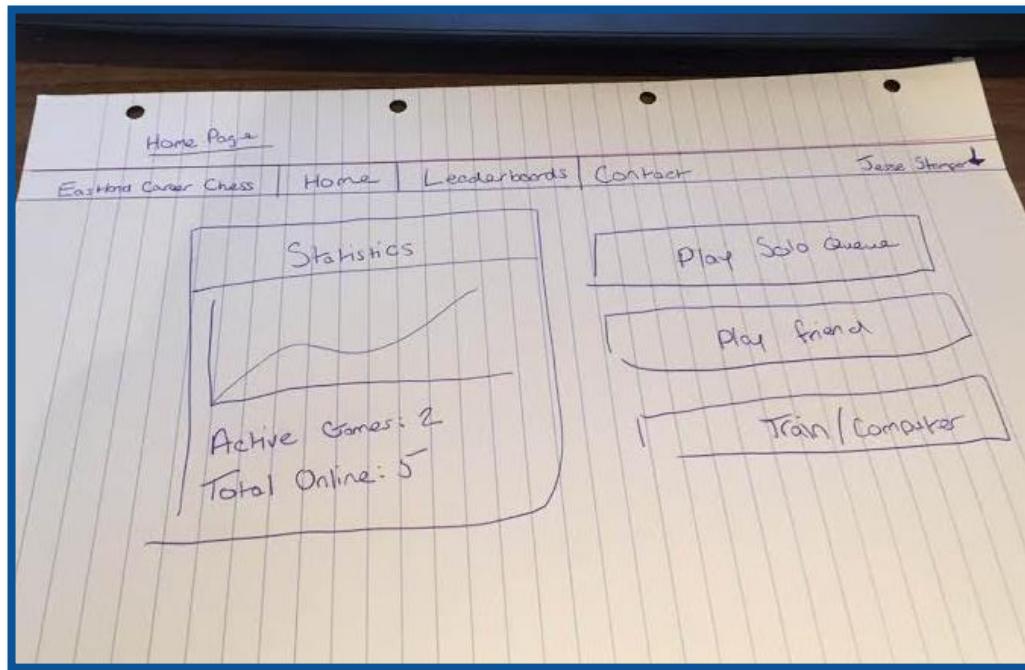
I thought that for the home page, a navigation bar at the top allowing for users to navigate through the website followed by some containers and buttons so that users can start playing games would be a nice idea.

JS: *I am honestly not too fussed, if you could place the school's name or logo in the bar that would be great. Also, a footer would be good.*

Let me just quickly draw this, one second. I was thinking that we could have some graphs on the home page to display some information about the games being played and the players who are using the system.

JS: *Not quite following.*

Here we are:

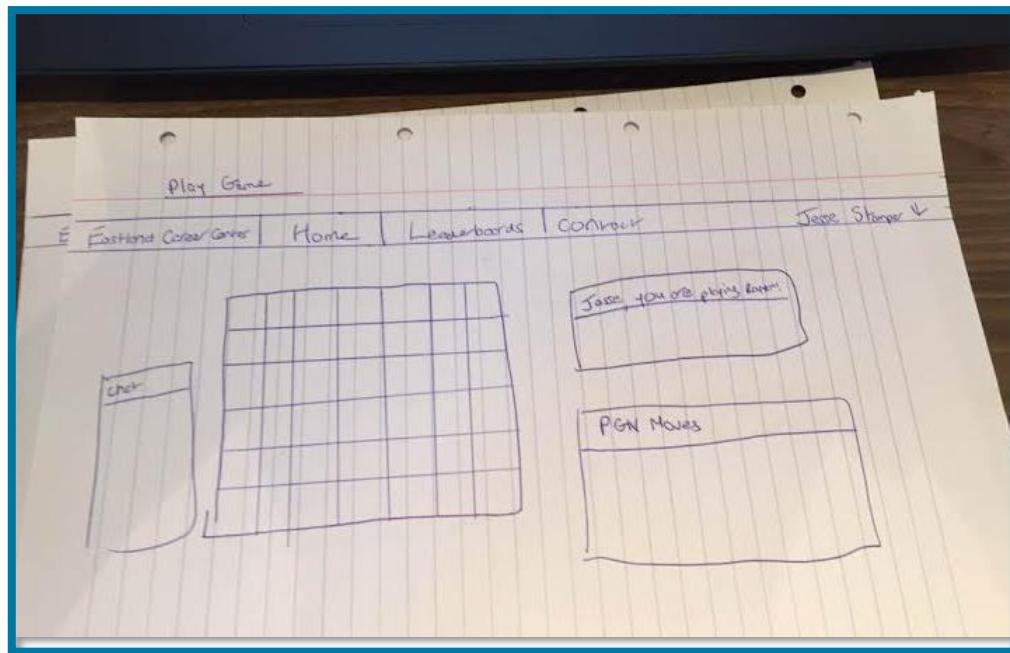


JS: Ahh, I see now. This is like what I also had in mind but, I am not a fan of the graph. Could we change this so I can place a message on the home page?

Sure thing. Also, remember that this is obviously not drawn to scale and I'll be able to throw it together for you before the end of today. Onto the playing page, should we keep the navigation bar on that page?

JS: We may as well use the navigation bar as the header for every page. How about the playing page? A box for information about your opponent and the PGN of the game?

Maybe something on the lines of this?



JS: I assume the square box is representing the Chess board?

That's correct.

JS: Looks good to me, would you be able to throw in a footer too? I would prefer for the contact buttons and information about our school to just be on the footer. There is not really much need to be making additional pages and what not.

Sure, that's good with me. As for the leaderboard, login and register pages would you just like these as normal? By that I mean, would you just like for there to be a table with a search button for the leaderboard and your basic login & register forms?

JS: Should be fine.

Final question before I hop off to create this for you, do you have any specific colours in mind that you would like the website designed with?

JS: Not really, just stay away from using the regular Bootstrap if you do end up using it.

Sure, I'll keep you updated.

JS: Thanks.

Additional Skype Messages:

Hey Jesse, any chance you could potentially tell me what you would like in your footer for the website? Right now, I've gone with a link to the home page, contact page for your school and a copyright symbol for your school. Anything else you would like on there?

JS: Please can you include the courses page for our school and a link the about page on our schools site.

Sure thing.

JS: Also, I asked our group and we do not need a chat box on our site.

Okay, I'll remove it now.

Design Objectives

This objective outlines the design for the front-end of the application. The specifications for the website have been created using the designs specified in a meeting with Jesse and I will be using these so that I can create the website.

The website will contain the following pages:

1. Registration Page

- 1.1. There will be a textbox for the user's first name which will display input as a string
- 1.2. There will be a textbox for the user's last name which will display input as a string
- 1.3. There will be a textbox for the user's email address which will display input as a string
- 1.4. There will be a textbox for the user's password which will be displayed as HTML's password format (an asterisk: *)

- 1.5. There will be a date selection form where the user can select their day, month and year for their date of birth

- 1.6. There will be a register button labelled *Signup* at the end of the form

2. Login Page

- 2.1. There will be a textbox for the user's email address and password which have the same attributes as those on the registration form

- 2.2. There will be a login button labelled *Login* at the end of the form

3. Header

- 3.1. There will be a navigation bar at the top of the page

- 3.2. On the top left of the navigation bar there will be large and bold text which reads *Eastland Career Center*

- 3.3. On the right side of the navigation bar, there will be (from left to right) the options labelled *Home* and *Leaderboard*

- 3.4. After Leaderboard, there will be a dropdown menu labelled with the user's full name

- 3.4.1. The first selection will be labelled *My Account*

- 3.4.2. The second selection will be labelled *logout*

4. Footer

- 4.1. There will be a box at the bottom of the page

- 4.2. The box will contain labels which are centered reading from left to right (*Home*, *About*, *Contact*, *Courses*)

- 4.3. Underneath the labels, there will be a copyright symbol labelled *Eastland Career Center 2016 → 2017*

5. Home Page

- 5.1. The home page will contain the header

- 5.2. Under the navigation bar from the header, there will be a rectangular box that is positioned slightly left from the center of the page

- 5.3. The box will be labelled *Information* and contain a message written by Jesse and information about the status of the application (the number of active games and players online)

- 5.4. To the right of the rectangular box, there will be another rectangular box which is much smaller in width than the other box

- 5.5. The box will be labelled *Game Modes* and contain three rounded buttons labelled *Play Solo*, *play with a friend* and *Play vs Computer*

- 5.6. Upon clicking the *Play Solo* button, a box would appear with a message that informs the user they have been placed into the queue

- 5.7. The *Play with Friend* button will redirect the user to the play page for playing with friends

- 5.8. The *Play vs Computer* button will cause for a box to appear with a slider from *1* → *20* and a Play Button labelled *Play*

- 5.8.1. When the play button is pressed, the player is redirected to the play page against a computer with the difficulty level of their choice

- 5.9. The home page will contain the footer

6. Leaderboard Page

- 6.1. The Leaderboard page will contain the header

- 6.2. There will be a large rectangular box that fills nearly half of the page (excluding the header and the footer) which contains a table

6.3. The table will have the columns (from left to right) labelled *Full Name* | *Age* | *Rating*
| (*Wins/Losses/Draws*)

6.4. The Leaderboard page will contain the footer

7. Play Page (Friend / Solo)

7.1. The play page will contain the header

7.2. In the centre of the page, there will be a Chessboard with two rounded buttons underneath labelled *Offer Draw* and *Resign*

7.3. To the left of the board, there will be a small rectangular box indicating the status of the game *Your Turn* | *Waiting for Opponent*

7.4. The contents of this box will update depending on the current state of the game (if the game has ended then this will change to *You win!*)

7.5. To the right of the board be a tall rectangular box labelled *Opponent*

7.6. This box will contain the *name*, *ranking* and *score* of the opponent if the opponent has connected

7.7. Underneath the opponent box, there will be another box titled *PGN* of the game and a button to export the game labelled *Export* which would export the contents to a .TXT file

7.8. The play page will contain the footer

8. Play Page (Computer)

8.1. The page will contain the header

8.2. The page will also contain the Chessboard, Game Status and PGN boxes from the Play (Friend or Solo) page

8.3. The Opponent Box will also be on this page however, this will contain information about the Computer

8.4. The box will contain the *Difficulty Level* selected by the user and information outputted by the Stockfish Engine (time taken to take move / best move / nodes explored)

Finished Design Review

I sent the finished design to Jesse:

Hey Jesse, I've just finished on the design!

Here are the pictures of the pages that unauthenticated users can view:

Login:



EASTLAND CAREER CENTER

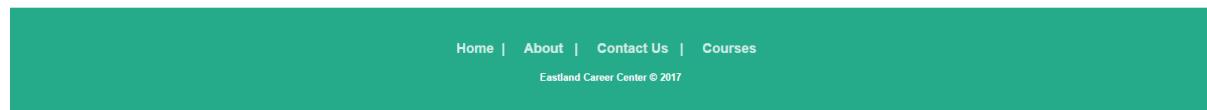
LOGIN REGISTER

➡ Login

Email

Password

Need an account? [Signup](#)



Home | About | Contact Us | Courses

Eastland Career Center © 2017

Register:



EASTLAND CAREER CENTER

LOGIN REGISTER

👤 Register

Email *

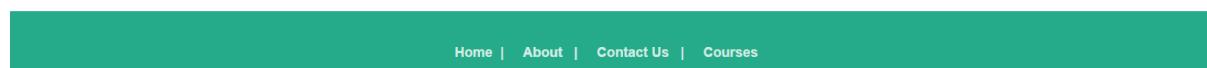
Password *

First Name *

Last Name *

Birthday *

Already have an account? [Login](#)



Home | About | Contact Us | Courses

And, here are the pages that authenticated users can view:

1. Home
2. Leaderboard
3. Play:
 - 3.1. Solo
 - 3.2. With friend
 - 3.3. Computer

The screenshot shows the homepage of 'EASTLAND CAREER CENTER'. The top navigation bar has links for 'HOME' and 'LEADERBOARDS'. Below the navigation is a large grey area containing two boxes: 'Information' and 'Game Modes'. The 'Information' box contains placeholder text about a lorem ipsum dolor sit amet, consectetur adipiscing elit. It also shows 'Active Games: 0' and 'Total Online: 0'. The 'Game Modes' box contains three buttons: 'Play solo', 'Play with friend', and 'Train/Computer'. At the bottom of the page is a green footer bar with links for 'Home', 'About', 'Contact Us', and 'Courses', followed by the text 'Eastland Career Center © 2017'.

**You must provide me with some text if you can to fill in the lorem ipsum.
I have thrown in some example accounts in the leaderboard page.**

The screenshot shows the 'Leaderboards' page. The top navigation bar has a link for 'LEADERBOARDS'. The main content area features a table titled 'Leaderboards' with columns for Position, Name, Rating, Age, and Wins/Losses/Draws. The table data is as follows:

Position	Name	Rating	Age	Wins/Losses/Draws
1	Example User	1127	19	4 / 1 / 0
2	Loka Poa	1068	99	1 / 0 / 0
3	Jesse Stamper	1063	28	3 / 1 / 0
4	Rayyan Iqbal	1000	18	0 / 0 / 0
5	John Baw	1000	27	0 / 0 / 0
6	Emily King	1000	55	1 / 1 / 1

Showing 1 to 6 of 6 entries

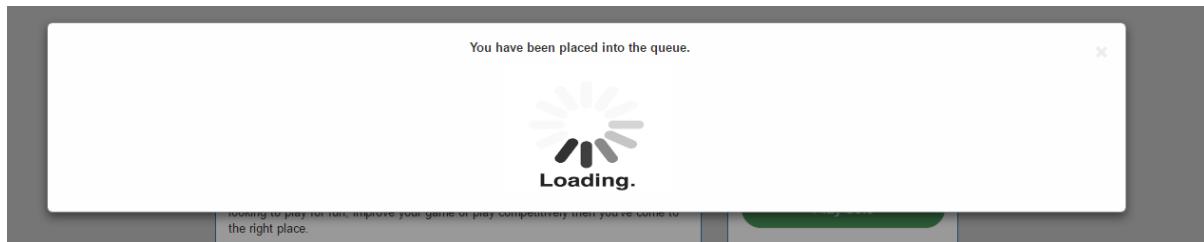
Previous 1 Next

Play Page:

CENSORED

Button Functionality:

I have also added some features for the buttons on the website. When a user presses the play solo button, they will get a popup with a buffering symbol to indicate they are now waiting for another player.



Finally, I also added a slider for the training option.



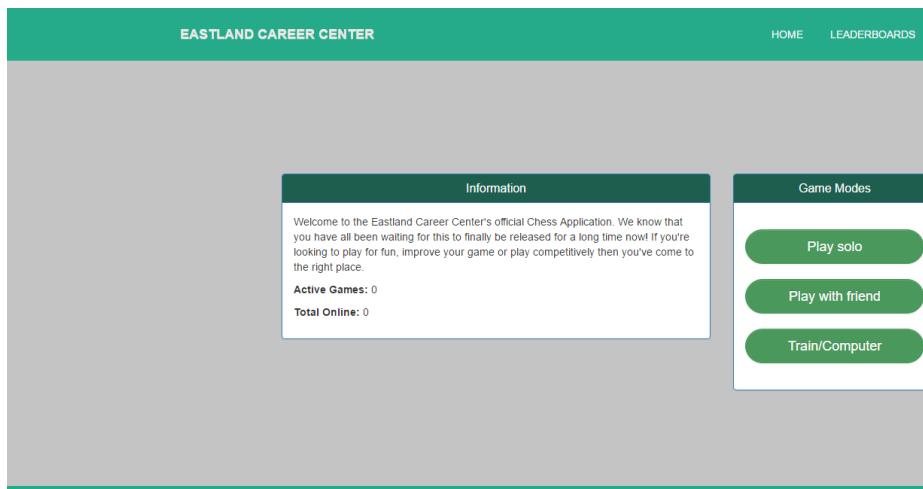
Let me know what you think when you can!

Jesse responded a couple of days later:

JS: Hey CENSOR, this is incredible work. I really was not expecting you to make so much progress, super happy. Here is the text I would like you to place on the home page:

“Welcome to the Eastland Career Center's official Chess Application. We know that you have all been waiting for this to finally be released for a long time now! If you're looking to play for fun, improve your game or play competitively then you've come to the right place.”

I've made the change, Jesse, can you confirm if this is okay?



JS: *Perfect.*

Website Navigation

Unauthenticated Users:

- Home Page → Login
 - Signup
- Navigation Bar (common to all pages)
 - Signup
 - Login

Authenticated Users:

- Home Page → Index
 - Play Solo
 - Play with friends
 - Play computer
- Navigation Bar (common to all pages)
 - Home (Index)
 - Leaderboards
 - Logout

Footer, common to all users:

1. Home
 - 1.1. Index relative to the user state. If the user is not authenticated, this will redirect them to the login page and if they are logged in to the index page
 - 1.2. About → Eastland's about me section on their school's website
 - 1.3. Contact → Eastland's contact section on their school's website
 - 1.4. Courses → Courses section found on Eastland's website

Section 2.6: Iteration 4

This section will contain the bulk of the design for the project. As the foundations for the application have been built, the remaining elements will be added to the application.

Database Design

Users Collection

The schema for the user's collection must be modified so that the application can now handle its new tasks. Users will now be assigned rankings and game scores meaning that the server would have to keep track of a players ranking in addition to the data previously stored about a user.

A Glicko-2 rank can be stored using three different values:

- 1. A volatility → How stable is a user's rating? If a rating is unstable then it is likely to vary by large orders of magnitude**
- 2. A rating deviation → How confident is the system about a user's rating? A new player who just signs up to the application will make the system less certain about whether that user's rating is accurate**
- 3. A rating → This will be the number that the system believes to be a user's rating**

The server would also have to store information about a user's game score. A game score is comprised of the following:

- 1. The total number of games that a user has won → Games only count if they are played as solo or against a friend**
- 2. The total number of games that have been lost by a user → Games only count for the same reason as above**
- 3. The total number of games that have been drawn by a user → Games only count for the same reason**

How can this data be stored about a user?

Both the game score for a user and their ranking information can be stored inside of separate objects. Thus, the new JSON Schema for the Users collection is as follows:

Users:

```
firstName :String,  
lastName :String,  
email :String,  
password :String,  
rank : Object, // {vol, rd, rating} For Glicko-2  
score : Object, // {wins, draws, losses}  
birthday : Object, // {day, month, year}
```

Games Collection

The end-user has requested for there to be data about games being stored on the application. To proceed, Jesse and I exchanged information via Skype messenger.

What information would you like to be stored about games that are played on the system?

JS: *Information about the game such as all the moves that were made in a game of Chess, who played in the game and when the game was played. There should also be some information about the player who won the game and how the player won the game.*

What information would the application need to store about a game to save the information Jesse requested?

- 1. Game Status → What is the status of a game?
Has the game ended or is the game still being played?**

2. **Players → Who played in the game?**
What are the Object ID's of the players who were playing in the game?

3. **Winner → Who won the game?**

4. **PGN String → What were the moves that were made in the game?**

5. **Game Start → When did the game start?**

6. **Game End → When did the game end?**

7. **Ending Reason → How did a game end?**
Did a player resign or win by checkmate?

However, there should be more information stored about games in the collection. Since Glicko recommends that rankings should be updated on a timely basis for example, at the end of every day. There should be an indicator to identify whether a game has caused for the rankings of players to change.

1. **Settled → Has Glicko-2 updated the rankings based on the result of a game?**

How can we store this information?

1. **Game Status → String, a game can be in many different states. For example, a game could be expired, live, ended etc.**
2. **Players → An array that contains the Object ID's of the players who played in the game.**
3. **Winner → String, what is the Object ID of the player who won the game?**
4. **PGN → String, what is the portable game notation string that represents the moves made a game?**
5. **End Reason → String, a game can end in multiple ways. A game may end by a player resigning, losing connection etc.**
6. **Game Start → Timestamp**
7. **Game End → Timestamp**

JSON Schema for the Games Collection

```

status:String, // Is the game still live or has it finished?
players: Array, // Object ID's of players
winner:String, // Object ID of winner
pgn:String,
endReason:String, // How did a game end? Checkmate, resignation
settled: Boolean, // Has the game been used to update rankings yet?
updateTime: Date, // Timestamps
createTime: Date,

```

Session Collection

Users will now also have persistent logins with the changes made in this iteration. This means that a user will now require a session to be stored inside of the database.

What information does the application need to store about a user for them to have a persistent session?

- 1. A session**
- 2. Object ID for the corresponding user**
- 3. An expiration**

What should a session be made of?

A session should store the cookie information for a user so that if the server must restart, users will not have to reauthenticate themselves to the system.

How can this data be stored?

- 1. A session → Object**
- 2. Passport → Object → Object ID of the user and any other information about the user's current status. For example, if the user should be redirected to a specific page**
- 3. Expiration Date → Date**

Since the data being stored in the sessions collection is unique, I will not be designing a schema for the collection. The benefit of a NOSQL approach in this case is that Mongo will be able to handle all queries to the database as normal.

Full Name: BLUR, CENSOR

Candidate Number: N/A

Centre Number: N/A

Glicko-2 Cronjob

In Professor Mark E. Glickman's document regarding an example Glicko-2 system, he recommends that for a Glicko-2 system to perform optimally then rankings should only be updated on a timely basis. (<http://www.glicko.net/glicko/glicko2.pdf>)

To achieve this in my system, a cronjob would be a smart way to update the rankings for the system. The Glicko-2 ranking updater should be a file that is technically separate from the whole project, it would not necessarily be required for the application to function. However, the rank updater should be running in the background and run over a certain time.

On Linux, this can be done with the command *crontab -e with the parameters to update rankings every 24 hours*. Glickman recommended that rankings should update after every tournament but, my application will not have any tournaments or competitions and therefore, a 24-hour time is a large enough sample space for ranking updates. If Jesse, then discovers that rankings could be updated better using a longer or a shorter time-period then, he can adjust the cronjob accordingly during deployment.

Lobby Design

The lobby element of the application will be implemented as a FIFO structure, the first player to join the queue will be the first person to leave the queue. Therefore, a queue structure would be a great way to implement the matchmaking for the application.

- 1. Player 1 joins the lobby, there is now one player in the lobby.**
- 2. Player 2 joins the lobby, there are now two players in the lobby.**
- 3. The application notices that there is now a pair of players in the lobby and the two players will be selected using the slice command in JavaScript.**
- 4. After the players, have been selected, the players will then be removed from the lobby.**
- 5. The two players will then be transferred to a game playing a random orientation. A dice will be thrown so both sides have a 50% chance of playing a specific colour.**

The lobby queue will be stored in an array, the priority goes to the first user to join the queue. There will be no need to store a pointer to the front and end of the queue since the elements will be popped and pushed onto the array and there is no requirement to store the total length of an array in Javascript. Also, we can not specify the number of players in a queue because we do not know how many players could join the match making queue at once.

PGN Download

Jesse requested that users should be allowed to export their games after they have been completed, since the PGN information for each move will already be displayed on the board it, allowing for the users of the system to download the PGN would be very useful.

The file will be downloaded after the PGN button has been pressed on the application and since the data for the client is already in string format, saving the file to a .TXT file would be

the best way to store this data. Text files are universally accessible and almost every operating system can read and write to a text file.

The data will be stored to the text file line by line so that it will be easy for users to read and import into a PGN rendering website.

Common Game Attributes

There are three different game modes that a user can play on the application:

- 1. Play versus a random player**
- 2. Play versus a friend**
- 3. Play versus a computer**

Since each of these games all share some common attributes, it would only be appropriate to create an API for the application which holds these common features of a game. It would be inefficient for the code to be developed in a way where each game mode had the code to create the Chess game or for all the socket messages and commands to be stored in both the player versus player modes.

Identifying the attributes that are shared by all the games

1. Each game mode has a board and so the board states should be applied for all the games:
 - 1.1. WAITING_JOIN**
 - 1.2. WAITING_MOVE**
 - 1.3. YOUR_TURN**
 - 1.4. GAME_OVER**
 - 1.5. CHECK**
 - 1.6. IN_CHECK**
2. The Chess logic
3. The Chess board
4. PGN String for the game and the downloadable PGN
5. User Information
 - 5.1. The username of the player**
 - 5.2. Orientation of the player**
6. Message boxes indicating the status of a game
 - 6.1. Waiting for opponent → Waiting for another player or for the Stockfish engine to connect**
 - 6.2. Opponent to move → It is not your turn, please wait before trying to make a move!**
 - 6.3. Your turn → Your turn, make a move!**
 - 6.4. Check → You have placed your opponent into check.**
 - 6.5. In check → You have been placed into check**
 - 6.6. Ended → The game has ended**

Identifying the attributes that are shared by online games

The player versus player game modes also share common attributes which are not shared by the common game. It would still be redundant for these to be separated from the features that are shared by all three games and instead, we should only release certain subroutines to the online games and vice versa for the player versus computer games:

1. The buttons
 - 1.1. To end a game as a draw**
 - 1.2. To resign from a game**
2. The link to a game containing the unique game identifier
3. The WebSocket messages that are emitted for each player for example, when a player makes a move on the board or resigns from a game.
4. Information about your opponent
 - 4.1. Full name**
 - 4.2. Rank**
 - 4.3. Wins**
 - 4.4. Losses**
 - 4.5. Draws**

General Common Attributes

Besides the Chess game, files are going to share common attributes which should be stored separately on the server. If the scripts and files were going to be embedded, it would mean that there would be more code on the server and since these will be commonly used scripts for the user, they will be stored into browser cache. Also, the code would simply be redundant if it was used in multiple files, compared to storing common lines of code in separate files. This also helps with code readability and usability.

- 1. Each webpage on the application will contain a navigation bar so the header should be separated.**
- 2. Each webpage on the application will contain a footer so the footer should be separated.**
- 3. Each webpage will require the custom CSS files used for the design of the application.**
- 4. Each webpage will require the custom scripts used in the application.**
- 5. Each webpage will have a title that ends with | Eastland Career Center and use the UTF-8-character set, so this should also be separated.**

Regex for verifying email addresses

This Regex is an improvement on the previous Regex I had implemented for this application to allow for users to sign up only using valid email addresses. The new regex now supports a much wider range of valid email addresses.

```
^(([^<>0\[\]\.,;:\s@"]+(\.[^<>0\[\]\.,;:\s@"]+)*|(.+))@((\[[0-9]{1,3}\].[0-9]{1,3}\).[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(([a-zA-Z\-\_0-9]+\.)+[a-zA-Z]{2,}))$
```

Examples:

The screenshot shows a regex testing interface. At the top, it says "REGULAR EXPRESSION" and "1 match, 24 steps (~0ms)". Below that is the regex pattern: `^(([^<>0\[\]\.,;:\s@"]+(\.[^<>0\[\]\.,;:\s@"]+)*|(.+))@((\[[0-9]{1,3}\].[0-9]{1,3}\).[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(([a-zA-Z\-_0-9]+\.)+[a-zA-Z]{2,}))$`. In the "TEST STRING" field, the input is `local@example.com`. To the right of the test string is a link "SWITCH TO UNIT TESTS".

2 IMAGE CENSOR

Description of Algorithms

Update Rankings

1. Connect to the database
2. Find all the games inside of the games collection which have settled set as false
3. Using the object identifiers embedded in each game, find the two users who were playing in the games
4. Fetch the rating, rating deviation and volailtities for the players in a game and create a local instance of a Glicko player for each user
5. Create an array to hold all the match results. This should contain the local identitiers for the users who played in the game and the results of the game.
 - 5.1. ([1] = player 1, [2] = player 2, [3] Game Result) → arguments for storing the game
 - 5.2. A lost game for player one can be represented with a 0
 - 5.3. A draw can be represented by 0.5
 - 5.4. A game won by player 1 can be represented with a 1
6. After all games have been appeneded to the game store, use Glicko 2 to update the local rankings of each individual player.
7. Once this has been completed, update all the rankings found in the user collection with the new rankings collected locally.
8. Once this has been completed then change all of the games used from {settled: false} to {settled true}

Database Queries

```
{settled: false}, {$set: {settled: true}}, {multi: true}
```

Change **all** games inside of the collection that have the value of settled from false to true

Matchmaking

All of this will occur after a user has pressed the play solo button on the home page to join the matchmaking queue.

1. Find the email of the address of the user inside of the user collection
2. Then, search the lobby for the user's unique identifier to check if the user is already inside of the lobby.
 - 2.1. IF the user is already inside of the lobby then send a websocket message informing the user he has been refused.
3. OTHERWISE: Push the users websocket identifier onto the active player counter and the lobby queue
4. If the number of players is less than 2 in the lobby then wait
5. OTHERWISE: Select the first two players in the lobby
6. Generate a random number between 0 and 1 to give the players a random orientation on the board. If the number is less than 0.5 then the first user will be playing as white on the board and vice versa for black
7. Create the instance of the new game and then save the game to the games collection
8. Once the game has been saved then:
 - 8.1. Fetch the websocket identifiers of both users
 - 8.2. Use the websocket identifiers to remove both players from the lobby
 - 8.3. Send a websocket message containing the unique identifier to the two paired players. The two paired players will then redirect to the game page.

Description of Data Structures

Below are some of the data structures I will be using to store data in the application. Although there will be many more data stores, here are some of the main elements of the system.

Identifier	Type	Explanation
AllRooms	Object	Each room on the board will contain information about the board. For example, we may want to access the users who are playing on a specific board. We can access this by finding the user identifiers stored inside of the object. E.g. AllRooms[gameId][UserId]
AllPlayers	Object	We store all the players on the system as an object so that we can store the unique identifier

		of a user along with their websocket identifier. This allows for much easier data management.
Lobby	Array	The matchmaking lobby is stored as an array since we only need to store a reference to one unique identifier for a user. We can then use this identifier to find more information about the user in the allPlayers object.
Glicko-2 Settings	Object	We store the default settings for Glicko-2 inside of an object so that we can access the attributes of the settings. For example, inside of the settings we may have the Rating: 200, this would store the default rating assigned to each user of the system.
Board Constants	Object	We store the states the board in an object which is constant. The object can remain as a constant since the potential states of the boards will not change. Board Status: WAITING_JOIN: 0

Data Integrity and Security

The application should maintain a high level of data integrity as security measures have already been put in place to ensure that the system is secure. For example, although users will not technically be allowed to export a game that previously occurred, data about each game will always be stored in the games collection. This means that if a user believes a bug has occurred on the system that caused his game score or ranking to change incorrectly then this can be verified using the data stored on the system.

There is a login system in place to ensure that when users provide data to the server they are who they say they are, I have mentioned a potential security vulnerability however, with the lack of an SSL certificate. In addition, data integrity could be lost if the server is penetrated by a hacker, there are no measures in place to ensure that the data inside of the database is not being modified by an external user. A simple fix to prevent against unauthorized users gaining access to the database would be by whitelisting the local ip addresss of the server to ensure that hackers are more restricted from gaining access to the server.

The server should always be kept secure as hackers are constantly running scripts to both scan and attempt to bruteforce servers. Just to note the pure severity of this problem, here is the message displayed to me when I last logged into the server:

There were 29032 failed login attempts since the last successful login.

And so, it would be extremely important for the server that the application will be running on to also be secure. If the server is a linux box then I recommend using puttyGen to generate private and public session keys between the server administrators and the server. This ensures that no unauthorized personal can connect to the server.

Besides unauthorized access, data integrity is hindered as there will be no system in place to ensure that users signing up are who they say they are and thus, this must be managed by the applications administrator (Jesse). If a user knew the link to the application then they could easily impersonate another user on the system. In order, to manage this problem in future sufficiently a verification system should be implemented to ensure that users are verified before they can have access to the system.

The final problem causing data to be unintegral on the system is in the event of user's accounts being compromised. At present, there are well over 8 billion accounts released publicly that contain personal information about individuals such as emails, passwords and more. These are known as SQL Dumps and since there is no captcha system or 2 factor authentication system in place for logging into the application then, there will always be the possibility that users may have their accounts compromised and be unaware that they are in fact compromised.

Besides these general problems, generally meaning that these apply to quite a lot of systems in the world, the system has solid integrity. The server will continue to consistently verify incoming traffic from users to ensure they are who they say they are before performing any data changing or destructive actions.

Full Name: BLUR, CENSOR
Candidate Number: N/A
Centre Number: N/A

SECTION 3: TECHNICAL SOLUTION

Section 3.1: Requirements

1. NodeJS
2. MongoDB

Section 3.2: Instructions to start the program

1. After NodeJS and MongoDB has been installed then go to the config/database.js and change the database to the link to your database.
2. Go into the main directory of the project and run the command *npm install*
2.1. You are required to have connection to the internet to run this command and please note that this command may take some time to run.
3. After you have installed all of the packages, run *npm install* one more time to double check that everything has installed correctly. If all of the packages have installed correctly, this process should only take a couple of seconds.
4. Run the command *npm start*
4.1. This will start the application, to view the application go to → <http://localhost:3000>
5. Using a cronjob program of your choice run:
5.1. *node cronjob/update_ranking.js* every 24 hours (I recommend using crontab -e)

Section 3.3: Code Tree

Below is an explanation of the contents of each file in the application.

```
## Code Tree
+ Back End
  + `app/primus.js`
    + Handles all of the websockets for the server
    + Two methods exposed to api for getting number of active players & games

  + Add `app/models/game.js`
    + schema for game collection, which has following fields:
      + status: String
        + enums of 'created', 'playing', 'gameover', 'terminated'
      + players: Array
        + with user id string as element
          + should contain exactly 2 elements. If a game is just 'created', one of them should be null
      + winner: String
        + object id of the winning player.
        + null if it's a draw game or the game is not ended yet
      + pgn: String
        + pgn data of the game
```

- + endReason: **String**
 - + reserved, to describe how is the game ended.
 - + 'by move','resign','close_resign','offer_draw'
- + settled: Boolean
 - + whether a game has been accounted into rank update.
 - + **default:** false
- + updateTime/createTime: Date
 - + **track** the update **time**
- + **Add** `cronjob/update_ranking.js`
 - + update ranking **on** timely basis
 - + should be added to cronjob by command `crontab -e` **in** CentOS
 - + should be run every 24-48 hours
- + Modify `server.js`
 - + calls websockets using `primusHost.**init**`
 - + **session** stored **in** mongodb **for** persistent logins
- + Modify `app/routes.js`
 - + routes
 - + GET /game/**local**
 - + play **local** game with **defaultlevel** 10
 - + GET /game/**local**/**:level**

- + play **local** game with AI at \${level}
- + GET /game/:id
 - + play online game with friend
- + GET /leaderboards/:perPage/:pageNum
 - + leaderboards with pagination
- + GET /search/**user**/:query/:perPage/:pageNum
 - + search result with pagination
- + POST /game/**create**
 - + **create** a game with creator playing white
- + POST /game/**create**/:gametype
 - + **create** a game, and you can specify creator's side
- + POST /checkemail
 - + check duplicate email **address**.
 - + return 'true' if not duplicate found, otherwise 'false'

- + Front **End**
 - + Templates
 - + `views/common`
 - + common templates used by every pages
 - + `views/widget`
 - + small widget that included by at least 2 pages
 - + `views/game_common.ejs`
 - + both online and **local** game **include** this as structure

- + `views/game_local.ejs`
 - + **include** game_common.ejs, and **add** its own script
- + `views/game_online.ejs`
 - + **include** game_common.ejs, and **add** its own script
- + `views/search_user.ejs`
 - + search result page, utilizing user_list widget

- + Javascript
 - + `public/js/game_common.js`
 - + common logic **for** chess game, including updating board status, providing pgn **link**, etc.
 - + `public/js/game_local.js`
 - + based **on** game_common.js, communicate with stockfish to play a game vs computer
 - + `public/js/game_online.js`
 - + based **on** game_common.js, allows **for** multiplayer game by using primus
 - + `public/js/signup.js`
 - + for realtime signup form validation
 - + `public/js/user_list.js`
 - + binding **events** for search **box** and entries #
 - + `public/assets/js/jquery.validate.min.js`
 - + library of jquery validate
 - + `public/assets/js/stockfish.js`
 - + library of stockfish

+ CSS

- + `public/assets/css/game_online.css`
 - + CSS **for** online games
- + `public/assets/css/signup.css`
 - + CSS **for** signup form
- + `public/assets/css/Footer-Basic.css`
 - + one variation of footer
- + `public/assets/css/Footer-Clean.css`
 - + second variation of footer
- + `public/assets/css/popups.css`
 - + popup messages **for** home page
- + `public/assets/css/rangeslider.css`
 - + slider **for** home page when selecting computer **level**
- + `public/assets/css/slider.css`
- + `public/assets/css/styles.css`
 - + global styles

*/

Section 3.4: Backend Code

app/primus.js

```
// Purpose of Primus.JS  
  
// Setup and configure the websockets for the application  
  
// Store information about the game which can be shared  
  
// with routes.js, eg. allRooms & allPlayers  
  
  
  
  
// Import modules local to Node  
  
var path = require('path');  
  
  
  
// Import NPM modules  
  
var Primus = require('primus');  
  
var Rooms = require('primus-rooms');  
  
  
  
// Import modules local to project
```

```

var Game = require('./models/game');

var User = require('./models/user');

// Configure lobby and initialize variables

var LOBBY_ROOM = 'lobby_ybbol'; // Set a lobby name (just a random name..)

var primus; // Websockets

var allRooms; // All game rooms

var allPlayers; // All players on game

var lobby;

// Update User Score for both players (user.local.score)

// the trick here is to tell which field to update (wins, draws, or losses)

// which is based on the gameResult 1 = first player wins, 0 = draw & -1 = first player loses

var updateScore = function (playerIds, gameResult) {
    var fields = ['wins', 'draws', 'losses'];

    return Promise.all(

```

```

playerIds.map(function (uid, i) {
  var obj = {};
  // Modify field with depending on game result
  var type = fields[1 + gameResult * Math.pow(-1, i + 1)];
  obj['local.score.' + type] = 1;
  // Push results to mongo
  return User.findByIdAndUpdate(uid, {
    $inc: obj
  });
}

);
// Set a game to 'gameover', and inform users of game about new status
// Ways a game can end (normal rules of chess + custom endings)
// 1. gameover by checkmate, stalemate etc
// 2. gameover by resign (resign button)
// 3. gameover by offering draw (offer draw button)

```

```

// 4. gameover by closing tabs (client uses connection etc)

var endGame = function (game, winner, reason) {

    // Configure new board data

    game.status      = 'gameover';
    game.winner      = winner;
    game.endReason   = reason;
    game.updateTime = new Date();

    // Store unique id to use later

    var room = game._id.toString();

    // if winner is not specified (game has ended as a draw) - result = 0
    // if the first user is the winner, his score will increment by one
    // vice versa -> his opponent's score will decrement by one

    var gameResult = !winner ? 0 : (game.players[0] === winner ? 1 : -1);
}

```

```

// We wait when until all the data is saved

return Promise.all([
    game.save(),
    // Pass the players & results so they can be updated in Mongo
    updateScore(game.players, gameResult)
])

.then(function () {
    // remove room from the room list
    delete allRooms[room];
})

.then(function () {

    // If the reason is a 'resign' type (resign, close_resign etc)
    if (/resign/.test(reason)) {
        primus.room(room)
    }
})

```

```

        .write ({
          action: 'resign',
          data: {
            gameId: room,
            side: game.players[0] === winner ? 'black' : 'white'
          }
        });

      }

primus.room(room)
  .write ({
    action : 'gameover',
    data : {
      isDraw: winner === null, // if there is no winner then the game has ended
      as a draw
      winSide: winner === null ? null : (game.players[0] === winner ? 'white' :
      'black')
    }
  })

```

```

        } ) ;
    }

    .catch(function (e) {
        // Output error caught (if function failed to save data)
        console.log(e.stack);
    });
}

// Add a new room (game) into active rooms

var addActiveRoom = function (gameId, userId, sparkId) {
    allRooms[gameId] = allRooms[gameId] || {};
    allRooms[gameId][userId] = allRooms[gameId][userId] || [];
    // Push websocket id (uid for user) into game
    allRooms[gameId][userId].push(sparkId);
};

// Update User score for both players (user.local.score) depending on the game result
// Search the active rooms for player and remove the player's session according to sparkId.

```

```

var removeSparkIdFromRoom = function (sparkId) {
    // Rotate all the rooms

    Object.keys(allRooms).forEach(function (gameId) {
        var room = allRooms[gameId];

        // Rotate inside of a room

        Object.keys(room).forEach(function (userId) {

            // Check if there is an opponent of selected sparkId - sparkId

            var opponentId = Object.keys(room).find(function (uid) {
                return uid !== userId;
            });

            // List of all connections of current room -

            var userSparkIds = room[userId];
        });
    });
}

```



```

        .then(function (game) {
            // pass to function: (game, winner, reason)
            return endGame(game, opponentId, 'close_resign');
        })
    }
}

}) ;

} ;

// Add a new player into active players

var addActivePlayer = function (userId, sparkId) {
    allPlayers[userId] = allPlayers[userId] || [];
    allPlayers[userId].push(sparkId);
};

```

```

// Remove the player with specified sparkId from active players.

var removeSparkIdFromPlayer = function (sparkId) {

    var toRemove = [ ];

    // Rotate through players

    Object.keys(allPlayers).forEach(function (username) {

        allPlayers[username] = allPlayers[username].filter(function (sid) {

            return sid !== sparkId;

        })

        // defensive check...

        if (allPlayers[username].length === 0) {

            toRemove.push(username);

        }

    });

    // remove the player

    toRemove.forEach(function (username) {

        delete allPlayers[username];
    })
}

```

```
    } ) ;  
};  
  
// If a user exists the website, remove them from Active Games & Rooms (if they're in one)  
  
var removeActiveBySparkId = function (sparkId) {  
    removeSparkIdFromPlayer(sparkId);  
    removeSparkIdFromRoom(sparkId);  
};  
  
// Lobby Functions  
  
// Push a new player into the lobby queue (FIFO structure)  
  
var queueInLobby = function (userId, sparkId) {  
    lobby.push({  
        userId: userId,  
        sparkId: sparkId  
    }) ;  
};
```

```
// Remove a player from the lobby

var removeFromLobby = function (sparkId) {
    lobby = lobby.filter(function (obj) {
        return obj.sparkId !== sparkId;
    });
}

// Multiplayer Matchmaking

// Pick the first two players in the lobby & then remove them from the lobby

var pairInLobby = function () {
    // Pick the players
    var ret = lobby.slice(0, 2);
    // Remove them from the lobby
    lobby = lobby.slice(2);
    return ret;
}
```

```

};

// Prime (Websocket) Initializer

var initialize = function(server) {
    // Setup variables
    allRooms = {};
    allPlayers = {};
    lobby = [];

    // Configure Websockets
    primus = new Primus(server, { transformer: "engine.io" }) // use engine io templating engine
    primus.plugin('rooms', Rooms); // rooms plugin..
    primus.save(path.join(__dirname, '.', 'public', 'js', 'primus.js')) // show directories for
    sockets

    // Websocket Rules
    primus.on("connection", function (spark) {
        // When a websocket has been opened, start listening for data.

```

```

spark.on('data', function(data) {
    data = data || {};
    // Setup data we are expecting to receive
    var action = data.action;
    var room = data.room;
    var data = data.data;
    var joinUser = data.user;
    var player = data.player;

    // Webserver Actions (needs some more commenting)
    // When a user joins a game
    if(action === 'join') {
        Game.findById(room) // sync code
            .then(function(game) {
                return Promise.all(
                    game.players.map(function(uid) {

```

```

        if (!uid) return Promise.resolve(null);

        return User.findById(uid).then(function (user) {
            delete user.local.password;

            return user;
        });
    }

).then(function (players) {
    var thePlayer = players.find(function (p) {
        return p.local.email === data.username;
    });

    if (!thePlayer) {
        // defensive exception

        throw new Error(data.username + ' is not a player in game: ' + room);
    }
})

```

```

// Add to active rooms & players so we can log on homepage

addActivePlayer(thePlayer._id.toString(), spark.id);

addActiveRoom(room, thePlayer._id.toString(), spark.id);

// Put the player into a game room

spark.join(room, function () {

    // emit message to the room through websocket

    process.nextTick(function () {

        spark.room(room)

            .write({

                action: 'players_update',
                data: {

                    username: data.username,
                    side: data.side,
                    players: players
                }
            })
    });
});

```

```

        }
    }

    .catch(function (e) {
        // catch & log errors
        console.log(e.stack);
    });
}

// When a user leaves a game

else if (action === 'leave') {
    spark.leave(room, function () {
        // emit message to client
        spark.write('You left game: ' + room);
    });
}

// Create Lobby Functionality

```

```

// (1) Place users into lobby (2) Lobby rules (3) Matchmaking

else if (action === 'lobby') {

    User.findOne({ 'local.email': data.username }) // find user in database

    .then(function (user) {

        spark.join(LOBBY_ROOM, function () {

            // User should not be allowed in the queue multiple times

            if (lobby.find(function (obj) { return obj.userId ===
user._id.toString() })) {

                spark.write({

                    action: 'lobby_refuse',

                    data: { msg: 'You are already waiting in lobby' }

                });

            }

            return;

        })

        // Push data to functions again..

        addActivePlayer(user._id.toString() , spark.id);

        queueInLobby(user._id.toString() , spark.id);

    })
}

```

```

// Easy Matchmaking system, any 2 players will be paired.

if (lobby.length < 2)    return;

var candidates = pairInLobby();
var uids        = candidates
                    .map(function (obj) {
                        return obj.userId;
                    });
// fetch Object ID's of paired users from database,
then store them in an array

uids.sort(function () {
    return Math.random() > 0.5; // throw dice to randomly select sides
});

// Instance a new game of Chess

var newGame = new Game({
    status: 'playing',

```

```

    players: uids,
    winner: null,
    settled: false,
    createTime: new Date(),
    updateTime: new Date(),
  } );
}

return newGame.save() // add to database
// Once saved then:
.then(function (game) {
  candidates.forEach(function (obj) {
    var sp = primus.spark(obj.sparkId); // fetch websocket ids
    // swapping ID for ObjID from Mongo
    removeSparkIdFromPlayer(obj.sparkId);
    // make player leave lobby
    sp.leave(LOBBY_ROOM);
    // Emit message about new game beginning
  });
}

```

```

        sp.write ({

            action: 'paired',

            data: {

                gameId: game._id.toString(),

                gameUrl: '/game/' + game._id.toString()

            }

        });

    });

}

.catch(function (e) {

    // catch and log exceptions

    console.log(e.stack);

}) ;

};

}

.catch(function (e) {

    console.log(e.stack);

})

```

```

        } );
    }

    // If a player makes a move in a game

    else if (action == 'move') {

        // Is there any data in the move message

        if (!data || !data.gameId || !data.username || !data.pgn) return;

        Promise.all([
            Game.findById(data.gameId),
            User.findOne({ 'local.email': data.username })
        ]
        // find the game & user who made the move
    )

    // once data is found:

    .then(function (tuple) {
        // create data in local scope (defensive)

        var game = tuple[0];
        var user = tuple[1];
    }
}

```

```

    // Both gameId and username should be valid

    if (!game || !user) return;

    if (game.status !== 'playing') throw new Error('You can only move in a playing
game!');

    // Update values for game and then save in database

    game.pgn = data.pgn;
    game.updateTime = new Date();
    return game.save();
}

.then(function () {
    // Send a message to other player in the room with his move

    spark.room(room)
        .except(spark.id)
        .write({
            action: 'update',
            data: { pgn: data.pgn, user: data.user }
        });
}
);

```

```

        }

        .catch(function (e) {
            // Catch any errors & log
            console.log(e.stack);
        });
    }

    // If a game has ended

    else if (action === 'gameover') {
        if (!data || !data.gameId || !data.winSide) return; // handle bad data

        // Find the game in Database to check status & make sure it's a live game
        // && Discover winning side of game
        Game.findById(data.gameId)
            .then(function (game) {
                if (!game) return;

                if (game.status !== 'playing') throw new Error('You can only set a playing game to game over!');

                // Find the winning side

```

```

        var winner = data.isDraw ? null : game.players[data.winSide === 'white' ? 0 :
1];

        return endGame(game, winner, 'by checkmate'); // game, winner, reason
    }

    .catch(function (e) {
        console.log(e.stack);
    });
}

// If a player offers for a draw (offer draw button)

else if (action === 'offerdraw') {
    spark.room(room)
        // Send a message to the other player
        .except(spark.id)
        .write({
            action : 'offerdraw',
            data : data
        });
}

```

```

}

// If the other player accepts the draw

else if (action === 'acceptdraw') {

  Game.findById(data.gameId)

    .then(function (game) {

      if (!game) return;

      if (game.status !== 'playing') throw new Error('You can only set a playing
game to game over!');

      return endGame(game, null, 'negotiate');

    })

    .catch(function (e) {

      console.log(e.stack);

    });

}

// If a player resigns from the game (pressing button)

else if (action === 'resign') {

  Game.findById(data.gameId)

```

```

        .then(function (game) {
            if (!game) return;

            if (game.status !== 'playing') throw new Error('You can only resign a playing game');

            // Other player is now the winning of the game
            var winner = game.players[data.side === 'white' ? 1 : 0];

            return endGame(game, winner, 'resign') // pass to function so can be updated in db
                .then(function () {
                    return data.side === 'white' ? 'black' : 'white'; // find colour of winner
                });
        })
        .catch(function (e) {
            // log exceptions...
            console.log(e.stack);
        });
    }
);

```

```

} ) ;

// If a player disconnects from the game, remove them from lobby & active players

primus.on("disconnection", function (spark) {
    removeActiveBySparkId(spark.id);
    removeFromLobby(spark.id);
}) ;

console.log('Primus Websockets have started successfully');

}

// Local functions to export to project (allows to be used in other files..)

module.exports = {
    init: initialize,
    activeGameCount: function () {
        return Object.keys(allRooms).filter(function (gameId) {
            // Only count games with two players in them (active games)
    }
}

```

```
        return Object.keys(allRooms[gameId]).length == 2;  
    }) .length;  
},  
activePlayerCount: function () {  
    return Object.keys(allPlayers).length;  
}  
};
```

app/models/game.js

```
// Import NPM modules  
  
var mongoose = require('mongoose');  
  
// Generate a unique Object ID so it can be used in database  
  
var ObjectId = mongoose.Types.ObjectId;  
  
// Define the schema for a game of Chess  
  
var gameSchema = mongoose.Schema({  
    status: String, // Is the game still live or has it finished?
```

```
players: Array, // Object ID's of players
winner: String, // Object ID of winner
pgn: String,
endReason: String, // How did a game end? Checkmate, resignation
settled: Boolean, // Has the game been used to update rankings yet?
updateTime: Date, // Timestamps
createTime: Date,
};

// export the model to the rest of the application
module.exports = mongoose.model('Game', gameSchema);
```

app/models/user.js

```
// Import NPM Modules

var mongoose = require('mongoose');

var mongoosePaginate = require('mongoose-paginate');

var bcrypt = require('bcrypt-nodejs');

// Create user schema

var userSchema = mongoose.Schema({  
    local : {  
        firstName : String,  
        lastName : String,  
        email : String,  
        password : String,  
        rank : Object, // {vol, rd, rating} For Glicko-2  
        score : Object, // {wins, draws, losses}  
        birthday : Object, // {day, month, year}  
    },  
});
```

```
userSchema.plugin(mongoosePaginate);

// User Methods

// (1) Hash a password using bcrypt (10 rounds)

userSchema.methods.generateHash = function(password) {
    return bcrypt.hashSync(password, bcrypt.genSaltSync(10), null);
};

// (2) Compare a plain text password to a bcrypt hash (Defensive Check)

userSchema.methods.validPassword = function(password) {
    return bcrypt.compareSync(password, this.local.password);
};

// Export the User model so it can be used elsewhere in the project

module.exports = mongoose.model('User', userSchema);
```

app/routes.js

```
// Import & setup NPM modules

var btoa = require('btoa');

var ensureLoggedIn = require('connect-ensure-login').ensureLoggedIn;

// Import modules local to project

var Game = require('../app/models/game');

var User = require('../app/models/user');

var primusHost = require('../app/primus');

// Setup board constants..

var GAME_ERROR = {

  ROOM_FULL: 'Room is FULL!',

  PLAYING_WITH_SELF: 'You are not supposed to play with yourself in this room!'
}
```

```

GAME_TERMINATED: 'Game is already terminated!',  

GAME_OVER: 'Game Over!',  

};  
  

// Return an array of integers within the specified range,  

// _step is an optional argument which defaults to 1 when not provided  

var range = function (start, end, _step) {  

    var step = _step || 1;  

    var ret = [];  

    // Set the exit condition on the loop based on the value of step  

    for (var i = start; step >= 0 ? (i < end) : (i > end); i += step) {  

        ret.push(i);  

    }  
  

    return ret;  

};  


```

```
// Output data to response as JSON

var outputJson = function (res, error_code, data, msg) {
  var obj = {
    error_code: error_code,
    msg: msg,
    data: data
  };

  res.send(JSON.stringify(obj));
  res.end();
};

// Create a new game of Chess & store in MongoDB

var createGame = function (userId, isWhite) {
```

```

var game = new Game ( {
    status: 'created',
    players: isWhite ? [userId, null] : [null, userId],
    winner: null,
    settled: false,
    createTime: new Date(),
    updateTime: new Date(),
} );
}

return game.save(); // Store in MongoDB
};

// Get user collections, paginate & accept custom queries

var getUsers = function (query, options, perPage, pageNum) {
    return User.paginate(query, Object.assign({
        limit: perPage,
        page: pageNum
    }, options));
};

```

```
}, options))

.then(function (result) {
    // Parse incoming data for query

    var users = result.docs;

    var totalCount = result.total;

    var totalPage = result.pages;

    var perPage = result.limit

    var pageNum = result.page;

    var pages = [];

    var startPosition = perPage * (pageNum - 1);

    var endPosition = startPosition + users.length - 1;

    for (var i = 1; i <= totalPage; i += 1) {
        pages.push(i)
    }

    return {
        users: users,
```

```

        totalCount: totalCount,
        totalPage: totalPage,
        perPage: perPage,
        pageNum: pageNum,
        pages: pages,
        startPosition: startPosition,
        endPosition: endPosition
    } ;
} ) ;
} ;

// Grab users from database for Leaderboard

var getLeaders = function (perPage, pageNum) {
    // Sort & place player with highest rating as rank #1
    return getUsers({}, {sort: {'local.rank.rating': -1}}, perPage, pageNum);
};

```

```

// Search function, allow for users to be searched on the Leaderboard

// Search using first name or last name

var searchUserByName = function (query, perPage, pageNum) {

  var normalizeRegStr = function(str) {

    return query.replace(/(\.|\?|\/)/g, '\\$1'); // Setup query w/regex
  };

  var reg = new RegExp(normalizeRegStr(query), 'i');

  var query = {

    $or: [
      {'local.firstName': reg},
      {'local.lastname': reg},
    ]
  };

  // Return found player

  return getUsers(query, {sort: {'local.rank.rating': -1}}, perPage, pageNum);
};

```

```

// Export Route Modules to be used in other parts of project

// (1) Create game (2) Create specific games (3) Email Check for registration

module.exports = function(app, passport) {

    // Allow for users to create games if they are authenticated

    app.post('/game/create', ensureLoggedIn('/login'), function(req, res) {

        var userId = req.user._id.toString();

        createGame(userId, true)

        .then(function (game) {

            outputJson(res, 0, { gameId: game._id.toString() });

        })

        .catch(function (e) {

            console.log(e.stack);

        });

    });

    // Allow for users to create specific games

    app.post('/game/create/:gametype', ensureLoggedIn('/login'), function(req, res) {

```

```

var userId = req.user._id.toString();
var isWhite = req.params.gametype === 'white';

createGame(userId, isWhite)
.then(function (gameId) {
    outputJson(res, 0, { gameId: gameId });
}) ;
} );

// Only allow for users to sign up with an email that is not already registered
app.post('/checkemail', function(req, res) {
    User.findOne({ 'local.email': req.body.email })
    .then(function (user) {
        res.send(user ? 'false' : 'true');
        res.end();
    })
}) ;
}

```

```

// Routing

// Defensive route for playing against a computer. Set Stockfish to level 10
app.get('/game/local', ensureLoggedIn('/login'), function (req, res) {
  res.redirect('/game/local/10');

}

// Allow for users to choose the level of the computer
app.get('/game/local/:level', ensureLoggedIn('/login'), function (req, res) {
  // Parse level in request

  var level = parseInt(req.params.level, 10);

  // Maximum level players can choose is 10

  if (parseInt(req.params.level, 10) === NaN) {

    return res.redirect('/game/local/10');

  }

  // Start game

```

```

res.render('game_local.ejs', {
  user: req.user,
  level: level
}) ;
}

// Play a game with a friend

app.get('/game/:id', ensureLoggedIn('/login'), function(req, res) {
  // First check if the game provided is actually a game

  Game.findById(req.params.id)
    // then:
    .then(function (game) {
      var userId = req.user._id.toString();
      var players = game.players;
      var p = Promise.resolve(game);
      var error;
    })
    .catch(function (err) {
      res.status(500).send(err.message);
    });
});

```

```
// Find out the status of the game that the user has requested

switch (game.status) {

    // Has the game expired?

    case 'terminated': {

        error = GAME_ERROR.GAME_TERMINATED;

        break;

    }

    // Has the game already ended?

    case 'gameover': {

        error = GAME_ERROR.GAME_OVER;

        break;

    }

    // Has the game just been created?

    case 'created': {

        if (players.indexOf(userId) !== -1) {

            // Waiting for second player now...

    }
}
```

```

    } else {

        game.players = !!players[0] ? [players[0], userId] : [userId, players[1]];

        game.status = 'playing';

        game.updateTime = new Date();

        p = game.save();

    }

break;

}

// Is the game already full?

case 'playing': {

    if (players.indexOf(userId) === -1) {

        error = GAME_ERROR.ROOM_FULL;

    }

break;

}

}

// Start the game

```

```

p.then(function (game) {
    game.pgn = game.pgn ? btoa(game.pgn) : ''; // BTOA for multi browser functionality
    game.orientation = game.players.indexOf(userId) === 1 ? 'black' : 'white'; // First
player = white
    // Start game...
    res.render('game_online.ejs', {
        error: error,
        game: game,
        user: req.user
    });
})
.catch(function (e) {
    console.log(e.stack);
})
);

// Index page for Authenticted Users (If they are signed in)

```

```

app.get('/', ensureLoggedIn('/login'), function(req, res) {
  res.render('index.ejs', {
    user: req.user,
    activePlayerCount: primusHost.activePlayerCount(),
    activeGameCount: primusHost.activeGameCount(),
    // Pass user object & functions for application statistics
  });
}

// Routing for Leaderboard page
// Leaderboard Home Page

app.get('/leaderboards', ensureLoggedIn('/login'), function(req, res) {
  getLeaders(25, 1) // Show top 25 players
  .then(function (data) {
    res.render('leaderboards.ejs', Object.assign(data, {
      user : req.user,
    }));
  });
}

```

```

        searchText: '',
        showPosition: true
    } ) );
}

.catch(function (e) {
    console.log(e.stack);
})
;

// Show user defined page (2nd page will show top 25 --> 50) (depending on user defined
entries/page)

app.get('/leaderboards/:perPage/:pageNum', ensureLoggedIn('/login'), function(req, res) {
    // Parse request data

    var perPage = parseInt(req.params.perPage, 10);
    var pageNum = parseInt(req.params.pageNum, 10);

    // Defensive Checks

    if ([10, 25, 50, 100].indexOf(perPage) ) {

```

```

return res.redirect('/leaderboards');

}

if (pageNum <= 0) {
  pageNum = 1;

}

// If we're all good then pull the users form the database
getLeaders(perPage, pageNum)
  .then(function (data) {
    // Re-render new page
    res.render('leaderboards.ejs', Object.assign(data, {
      user : req.user,
      searchText: '',
      showPosition: true
    }));
  })
  .catch(function (e) {

```

```

// catch & output any errors
    console.log(e.stack);
}
}

// Route for searching a user on the leaderboard
app.get('/search/user/:query/:perPage/:pageNum', ensureLoggedIn('/login'), function(req, res) {
    // Parse the data from the request
    var perPage = parseInt(req.params.perPage, 10);
    var pageNum = parseInt(req.params.pageNum, 10);
    // Defensive Chceking
    if ([10, 25, 50, 100].indexOf(perPage) ) {
        return res.redirect('/search/user/' + req.params.query + '/10/1');
    }

    if (pageNum <= 0) {

```

```
pageNum = 1;

}

// If we're all good then search the username in the database

searchUserByName(req.params.query, perPage, pageNum)
.then(function (data) {
  res.render('search_user.ejs', Object.assign(data, {
    user : req.user,
    searchText: req.params.query,
    showPosition: false
  )) ;
}

.catch(function (e) {
  // output & log exceptions
  console.log(e.stack);
}) ;
```

```
    } ) ;  
  
    // Terminating a user session  
  
    app.get( '/logout' , function(req, res) {  
        req.logout();  
        res.redirect( '/' );  
    } );  
  
    // Routing for users that are not signed in  
    // unauthenticated user routes  
  
    // Index page  
  
    app.get( '/login' , function(req, res) {  
        res.render( 'login.ejs' , { message: req.flash( 'loginMessage' ) } );  
    } );  
  
    // Processing a login request
```

```

app.post('/login', passport.authenticate('local-login', {
  successReturnToOrRedirect: '/',
  failureRedirect : '/login',
  failureFlash : true
}) );

```

// Registration

// Display the registration form

```

app.get('/signup', function(req, res) {
  var d = new Date(); // timeStamp for MongoDB
  var y = d.getFullYear(); // Check the current year for check later on

  res.render('signup.ejs', {
    message: req.flash('signupMessage'),
    years: range(y, y - 100, -1), // Ages shown, max age 100
  });
});

```

```

months: 'Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec'.split(',') ,
dates: range(1, 32) // 1 --> 31
} );
} );

// Process signup requests

app.post('/signup', passport.authenticate('local-signup', {
  successRedirect : '/',
  failureRedirect : '/signup',
  failureFlash : true
}) );
}

// Routing requests to middleware to ensure that a user is authenticated

function isLoggedIn(req, res, next) {
  if (req.isAuthenticated())
    return next();
}

```

```
    res.redirect('/'); // When logged in redirect to home page
}

function isLoggedInProfile(req, res, next) {
  if (req.isAuthenticated())
    return next();

  res.redirect('/login'); // Not logged in
}
};
```

config/database.js

```
// Database Configuration
```

```
module.exports = {  
  'url' : 'mongodb://dbUser:dbPassword@IP/dbName' // Your database link here  
};
```

config/passport.js

```
// Load and setup NPM modules  
  
var LocalStrategy = require('passport-local').Strategy;
```

```
var glicko2 = require('glicko2');

// Load up modules local to the project

var User = require('../app/models/user');

// Setup Glicko-2 (Modified Default Settings)

var settings = {

  tau: 0.5, // How accurate should Glicko be? (Higher = more processing required)

  rating: 1000, // Default Rating

  rd: 200, // Rating Deviation (Default confidence argument)

  vol: 0.06 // How stable or unstable is this players rating?

};

var ranking = new glicko2.Glicko2(settings);

// Export functions for use in other parts of the project
```

```
module.exports = function(passport) {  
  
  // serialize users for persistent logins  
  
  passport.serializeUser(function(user, done) {  
  
    done(null, user.id); // save user ID in session  
  });  
  
  // deserialize user  
  
  passport.deserializeUser(function(id, done) {  
  
    User.findById(id, function(err, user) {  
  
      done(err, user); // user object attaches to the request  
    });  
  });  
  
  // Register
```

```

passport.use('local-login', new LocalStrategy({
    // by default, local strategy uses username and password, we will override with email
    usernameField : 'email',
    passwordField : 'password',
    passReqToCallback : true // allows us to pass in the req from our route (lets us check if a
    user is logged in or not)
},
function(req, email, password, done) {
    if (email)
        email = email.toLowerCase(); // Use lower-case e-mails to avoid case-sensitive e-mail
        matching

    // asynchronous
    process.nextTick(function() {
        User.findOne({ 'local.email' : email }, function(err, user) {
            // if there are any errors, return the error
            if (err)
                return done(err);

            // if there is no user with the given email
            if (!user)
                return done(null, false, { message: 'Incorrect email' });

            // if user is found, but password is wrong
            if (!user.checkPassword(password))
                return done(null, false, { message: 'Incorrect password' });

            // all good, create token
            var token = user.generateToken();
            done(null, user, token);
        });
    });
});

```

```

        // if no user is found, return the message

        if (!user)

            return done(null, false, req.flash('loginMessage', 'No user found.'));

        if (!user.validPassword(password))

            return done(null, false, req.flash('loginMessage', 'Oops! Wrong password.'));

        // all is well, return user

        else

            return done(null, user);

        } );
    } );

}) ;

passport.use('local-signup', new LocalStrategy(


    usernameField: 'email',

```

```

passwordField: 'password',
passReqToCallback: true // allows us to pass in req from our route
},
// Registration functionality
function(req, email, password, done) {
  var error = function(msg) {
    return done(null, false, req.flash('signupMessage', msg));
  };

// Validate an email address using regex (local@example.com)
function validateEmail(email) {
  // Example: local@example.com
  var re = /^[(([^<>()\\[]\\\\.\\,;:\\s@"]+(\\\.[^<>()\\[]\\\\.\\,;:\\s@"]+)*|("."+"))@((\\[\\[0-
9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}])|(([a-zA-Z\\-0-9]+\\.)+[a-zA-Z]{2,}))$/;
  return re.test(email);
}

// Registration Conditions

```

```

if (!validateEmail(email)) // Regex for validating email
    return error('Please enter a valid email address');

if (email.length > 64)
    return error('Your email cannot be longer than 64 characters.');

if (password.length < 8 || password.length > 32)
    return error('Please use a password that is between 8 & 32 characters.');

if (req.body.firstName.length <= 0 || req.body.firstName.length > 35)
    return error('Your first name cannot be longer than 35 characters!');

if (req.body.lastName.length <= 0 || req.body.lastName.length > 35)
    return error('Your last name cannot be longer than 35 characters!');

if (req.body.birthYear > (new Date()).getFullYear() - 13)
    return error('You must be older than 13 to use our application.');

// Defensive conversion

if (email)
    email = email.toLowerCase(); // Use lower-case e-mails to avoid case-sensitive e-mail
matching (in db)

```

```

// asynchronous

process.nextTick(function() {
    // if the user is not already logged in:

    if (!req.user) {
        User.findOne({ 'local.email' : email }, function(err, user) {
            // Return any errors

            if (err)
                return done(err);

            // Check if email already exists in DB

            if (user) {
                return error('Someone has already registered with that email!');
            } else {
                // Create the new user object for database

                var newUser = new User();
                var rPlayer = ranking.makePlayer(); // Glicko-2

                // User Object

```

```
newUser.local = {
    email: email,
    password: newUser.generateHash(password), // Hash password with bcrypt
    firstName: req.body.firstName,
    lastName: req.body.lastName,
    birthday: {
        year: req.body.birthYear,
        month: req.body.birthMonth,
        date: req.body.birthDate,
    },
    rank: { // Glicko-2
        rating: rPlayer.getRating(),
        rd: rPlayer.getRd(),
        vol: rPlayer.getVol()
    },
    score: { // Gamescore
        wins: 0,
```

```

        losses: 0,
        draws: 0
    }

};

// When this is done then save the user

newUser.save()
    .then(function () { done(null, newUser); })
    .catch(done); // catch exceptions

}

});

// Defensive Check

// If the user is signed in but does not have an account.

} else if (!req.user.local.email) {

    // ...presumably they're trying to connect a local account

    // but let's check if the email used to connect a local account is being used by
another user

    User.findOne({ 'local.email' : email }, function(err, user) {

```

```

    if (err)
        return done(err);

    if (user) { // signupMessage is already being used by /connect/local, so use
loginMessage

        return done(null, false, req.flash('loginMessage', 'Someone has already
registered with that email!'));

    } else {

        // Create the account

        var user = req.user;

        user.local.email = email;

        user.local.password = user.generateHash(password);

        user.save(function (err) {

            if (err)
                return done(err);

            return done(null, user);
        });
    }
}

```

```

        }

    } );

} else {

    // user is logged in and already has a local account. Ignore signup.

    // users cannot create a new account if they are already logged in

    return done(null, req.user);

}

}

}

);
}
;
}
;
```

cronjob/update_rankings.js

```

/* How rankings are updated

(1) Find all of the 'unsettled' games in the games collection

(2) Find all of the users who were playing in these games

(3) Create Glicko Rankings for players

(4) Create the games

(5) Calculate new rankings using Glicko-2

```

(6) Get the new rankings from the calculation and update ranks in database

(7) Change the fetched games to {settled: true}

*/

```
// Import NPM modules
```

```
var glicko2 = require('glicko2');
```

```
var mongoose = require('mongoose');
```

```
// Import modules local to project
```

```
var User = require('../app/models/user');
```

```
var Game = require('../app/models/game');
```

```
var configDB = require('../config/database.js');
```

```
// Setup DB
```

```
mongoose.Promise = global.Promise;
```

```
mongoose.connect(configDB.url);
```

```

// Glicko-2 Settings

var settings = {
    tau: 0.5,
    rating: 1000,
    rd: 200,
    vol: 0.06
};

// Concatenate an array

// example I/O: a = [1, 2] & b = ["x", "y"] --> c = [1, 2, "x", "y"]

var concat = function (listOfList) {
    return [] .concat .apply ([] , listOfList);
};

// Convert string back to Object ID

var toObjectId = function (str) {

```

```
        return mongoose.Types.ObjectId(str);

    };

var update = function (users, games) {
    var ranking = new glicko2.Glicko2(settings);

    // Pull data from users for Glicko
    var rankPlayers = users.reduce(function (obj, user) {
        var rating = user.local.rank.rating;
        var rd = user.local.rank.rd;
        var vol = user.local.rank.vol;

        // Create player
        obj[user._id.toString()] = ranking.makePlayer(rating, rd, vol);
    });

    return obj;
}
```

```

}, { } );
}

// Each element of matches should have 3 items
// ([1] = player1, [2] = player2, [3] = Game result )

var matches = games.map(function (game) {
    // Get the two cached players.

    var ret = game.players.map(function (uid) {
        return rankPlayers[uid];
    });
}

// [0 = player 1 loses] && [0.5 = draw] && [1 = player 1 wins]
ret.push(!game.winner ? 0.5 : (game.players[0] === game.winner ? 1 : 0));
return ret;
});

// Now update the rankings locally for players
ranking.updateRatings(matches);

```

```

// Now we update the ranks inside of the database

return Promise.all(
  Object.keys(rankPlayers).map(function (uid) {
    var rankPlayer = rankPlayers[uid];
    var newRank = {
      rating: Math.round(rankPlayer.getRating()),
      rd: Math.round(rankPlayer.getRd()),
      vol: Math.round(rankPlayer.getVol() * 10000) / 10000 // Scaling factors
    };
    return User.findByIdAndUpdate(uid, {
      $set: { 'local.rank': newRank }
    });
  })
);

// After we have updated the ranks, log the changes made

.then(function () {
  console.log(Object.keys(rankPlayers).length + ' players\' have had their ranks updated!');
});

```

```

        } ) ;
    } ;

// Change the status of the games updated to settled
// This is so in the next ranking update, these will not be used again (self explanatory)

Game.find({
    status: 'gameover',
    settled: false
})
.exec()
.then(function (games) {
    var playerIds = concat(games.map(function (game) {
        return game.players;
    }) );
    return User.find({

```

```

    _id: {
      $in: playerIds.map(toObjectId)
    }
  })
  .exec()
  .then(function (users) {
    return update(users, games);
  })
  .then(function () {
    return Game.update({settled: false}, {$set: {settled: true}}, {multi: true});
  })
  .then(function () {
    console.log('done!');
  });
}
.catch(function (e) {
  console.log(e.stack);
})

```

```
    }
    .then(function () {
      mongoose.connection.close();
    });
  }
}
```

Section 3.5: Front-end Code

public/assets/css/Footer-Basic.css

```
.footer-basic {
  padding: 40px 0;
  background-color: #ffffff;
  color: #4b4c4d;
}

.footer-basic ul {
  padding: 0;
  list-style: none;
  text-align: center;
  font-size: 18px;
```

```
line-height:1.6;  
margin-bottom:0;  
}  
  
.footer-basic li {  
padding:0 10px;  
}  
  
.footer-basic ul a {  
color:inherit;  
text-decoration:none;  
opacity:0.8;  
}  
  
.footer-basic ul a:hover {  
opacity:1;  
}
```

```
.footer-basic .social {  
    text-align:center;  
    padding-bottom:25px;  
}  
  
.footer-basic .social > a {  
    font-size:24px;  
    width:40px;  
    height:40px;  
    line-height:40px;  
    display:inline-block;  
    text-align:center;  
    border-radius:50%;  
    border:1px solid #ccc;  
    margin:0 8px;  
    color:inherit;
```

```
    opacity:0.75;  
}  
  
.footer-basic .social > a:hover {  
    opacity:0.9;  
}  
  
.footer-basic .copyright {  
    margin-top:15px;  
    text-align:center;  
    font-size:13px;  
    color:#aaa;  
    margin-bottom:0;  
}
```

public/assets/css/Footer-Clean.css

```
.footer-clean {  
    padding:50px 0;
```

```
background-color:#fff;  
color:#4b4c4d;  
}
```

```
.footer-clean h3 {  
margin-top:0;  
margin-bottom:12px;  
font-weight:bold;  
font-size:16px;  
}
```

```
.footer-clean ul {  
padding:0;  
list-style:none;  
line-height:1.6;  
font-size:14px;  
margin-bottom:0;
```

```
}

.footer-clean ul a {

    color: inherit;

    text-decoration: none;

    opacity: 0.8;

}

.footer-clean ul a:hover {

    opacity: 1;

}

.footer-clean .item.social {

    text-align: right;

}

@media (max-width: 767px) {
```

```
.footer-clean .item {  
    text-align:center;  
    padding-bottom:20px;  
}  
  
}  
  
@media (max-width:991px) {  
    .footer-clean .item.social {  
        text-align:center;  
    }  
}  
  
.footer-clean .item.social > a {  
    font-size:24px;  
    width:40px;  
    height:40px;  
    line-height:40px;
```

```
display:inline-block;  
text-align:center;  
border-radius:50%;  
border:1px solid #ccc;  
margin-left:10px;  
margin-top:22px;  
color:inherit;  
opacity:0.75;  
}
```

```
.footer-clean .item.social > a:hover {  
    opacity:0.9;  
}
```

```
@media (max-width:991px) {  
    .footer-clean .item.social > a {  
        margin-top:40px;
```

```
        }

    }

@media (max-width:767px) {
    .footer-clean .item.social > a {
        margin-top:10px;
    }
}

.footer-clean .copyright {
    margin-top:14px;
    margin-bottom:0;
    font-size:13px;
    opacity:0.6;
}

public/assets/css/Navigation-Clean1.css
.navigation-clean {
    background:#25aa8a;
```

```
padding-top:10px;  
padding-bottom:10px;  
color:#ebe8e8;  
border-radius:0;  
box-shadow:none;  
border:none;  
margin-bottom:0;  
}  
  
@media (max-width:767px) {  
    .navigation-clean {  
        padding-top:0;  
        padding-bottom:0;  
    }  
}  
  
@media (max-width:767px) {
```

```
.navigation-clean .navbar-header {  
    padding-top:10px;  
    padding-bottom:10px;  
}  
  
.  
.navigation-clean .navbar-brand {  
    font-weight:bold;  
    color:inherit;  
}  
  
.navigation-clean .navbar-brand:hover {  
    color:#222222;  
}  
  
.navigation-clean.navbar-inverse .navbar-brand:hover {  
    color:#f0f0f0;
```

```
}

.navigation-clean .navbar-brand img {
    height:100%;
    display:inline-block;
    margin-right:10px;
    width:auto;
}

.navigation-clean .navbar-toggle {
    border-color:#ddd;
}

.navigation-clean .navbar-toggle:hover, .navigation-clean .navbar-toggle:focus {
    background:none;
}
```

```
.navigation-clean.navbar-inverse .navbar-toggle {  
    border-color:#555;  
}  
  
.navigation-clean .navbar-toggle .icon-bar {  
    background-color:#888;  
}  
  
.navigation-clean.navbar-inverse .navbar-toggle .icon-bar {  
    background-color:#eee;  
}  
  
.navigation-clean .navbar-collapse, .navigation-clean .navbar-form {  
    border-top-color:#ddd;  
}  
  
.navigation-clean.navbar-inverse .navbar-collapse, .navigation-clean.navbar-inverse .navbar-form {
```

```
border-top-color:#333;  
}  
  
.navigation-clean .navbar-nav > .active > a, .navigation-clean .navbar-nav > .open > a {  
background:none;  
box-shadow:none;  
}  
  
.navigation-clean.navbar-default .navbar-nav > .active > a, .navigation-clean.navbar-default  
.navbar-nav > .active > a:focus, .navigation-clean.navbar-default .navbar-nav > .active >  
a:hover {  
color:#fffffe;  
box-shadow:none;  
background:none;  
pointer-events:none;  
}  
  
.navigation-clean.navbar .navbar-nav > li > a {
```

```
padding-left:18px;  
padding-right:18px;  
}  
  
.navigation-clean.navbar-default .navbar-nav > li > a {  
color:#f7f7f7;  
}  
  
.navigation-clean.navbar-default .navbar-nav > li > a:focus, .navigation-clean.navbar-default  
.navbar-nav > li > a:hover {  
color:#f9fcff !important;  
background-color:transparent;  
}  
  
.navigation-clean .navbar-nav > li > .dropdown-menu {  
margin-top:-5px;  
box-shadow:0 4px 8px rgba(0,0,0,.1);  
background-color:#fff;
```

```
border-radius:2px;  
}  
  
@media (max-width:767px) {  
    .navigation-clean .navbar-nav .open .dropdown-menu {  
        box-shadow:none;  
    }  
}  
  
@media (max-width:767px) {  
    .navigation-clean .navbar-nav .open .dropdown-menu > li > a {  
        color:#37434d;  
        padding-top:12px;  
        padding-bottom:12px;  
        line-height:1;  
    }  
}
```

```
.navigation-clean .dropdown-menu > li > a:focus, .navigation-clean .dropdown-menu > li > a {  
    line-height:2;  
    font-size:14px;  
    color:#37434d;  
}  
  
.navigation-clean .dropdown-menu > li > a:focus, .navigation-clean .dropdown-menu > li >  
a:hover {  
    background:#eee;  
    color:inherit;  
}  
  
.navigation-clean.navbar-inverse {  
    background-color:#1f2021;  
    color:#fff;  
}
```

```
.navigation-clean.navbar-inverse .navbar-nav > .active > a, .navigation-clean.navbar-inverse
.navbar-nav > .active > a:focus, .navigation-clean.navbar-inverse .navbar-nav > .active >
a:hover {

    color:#8f8f8f;

    box-shadow:none;

    background:none;

    pointer-events:none;

}

.navigation-clean.navbar-inverse .navbar-nav > li > a {

    color:#dfe8ee;

}

.navigation-clean.navbar-inverse .navbar-nav > li > a:focus, .navigation-clean.navbar-inverse
.navbar-nav > li > a:hover {

    color:#fff !important;

    background-color:transparent;

}
```

```
.navigation-clean.navbar-inverse .navbar-nav > li > .dropdown-menu {  
background-color:#1f2021;  
}  
  
.navigation-clean.navbar-inverse .dropdown-menu > li > a:focus, .navigation-clean.navbar-inverse .dropdown-menu > li > a {  
color:#f2f5f8;  
}  
  
.navigation-clean.navbar-inverse .dropdown-menu > li > a:focus, .navigation-clean.navbar-inverse .dropdown-menu > li > a:hover {  
background:#363739;  
}  
  
@media (max-width:767px) {  
.navigation-clean.navbar-inverse .navbar-nav .open .dropdown-menu > li > a {  
color:#fff;  
}
```

```
}
```

public/assets/css/popups.css

```
.modal {  
    display:none;  
    position:fixed;  
    z-index:1;  
    padding:100px;  
    left:0;  
    top:0;  
    width:100%;  
    height:100%;  
    overflow:auto;  
    background-color:rgb(0,0,0);  
    background-color:rgba(0,0,0,0.4);  
}
```

```
.modal-content {  
background-color:#fefefe;  
margin:auto;  
padding:20px;  
border:1px solid #888;  
width:80%;  
}  
  
}
```

```
.close {  
color:#aaaaaa;  
float:right;  
font-size:28px;  
font-weight:bold;  
}
```

```
.close:hover, .close:focus {  
color:#000;
```

```
text-decoration:none;  
cursor:pointer;  
}  
  
public/assets/css/rangeslider.css
```

```
.rangeslider,  
.rangeslider__fill {  
    display: block;  
    -moz-box-shadow: inset 0px 1px 3px rgba(0, 0, 0, 0.3);  
    -webkit-box-shadow: inset 0px 1px 3px rgba(0, 0, 0, 0.3);  
    box-shadow: inset 0px 1px 3px rgba(0, 0, 0, 0.3);  
    -moz-border-radius: 10px;  
    -webkit-border-radius: 10px;  
    border-radius: 10px;  
}  
  
.rangeslider {  
    background: #e6e6e6;
```

```
position: relative;  
}  
  
.rangeslider--horizontal {  
    height: 20px;  
    width: 100%;  
}  
  
.rangeslider--vertical {  
    width: 20px;  
    min-height: 150px;  
    max-height: 100%;  
}  
  
.rangeslider--disabled {  
    filter: progid:DXImageTransform.Microsoft.Alpha(Opacity=40);  
    opacity: 0.4;
```

```
}

.rangeslider__fill {
  background: #00ff00;
  position: absolute;
}

.rangeslider--horizontal .rangeslider__fill {
  top: 0;
  height: 100%;
}

.rangeslider--vertical .rangeslider__fill {
  bottom: 0;
  width: 100%;
}

.rangeslider__handle {
  background: white;
```

```
border: 1px solid #ccc;
cursor: pointer;
display: inline-block;
width: 40px;
height: 40px;
position: absolute;

background-image:
url('data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4gPHN2ZyB2ZXJzaW9uPSIxLjEiIHtbG5zPSJodHRwOi8vd3d3LnczM9yZy8yMDAwL3N2ZyI+PGRlZnM+PGxpbmVhckdyYWRpZW50IGlkPSJncmFkIiBncmFkaWVudFVuaxRzPSJvYmp1Y3RCb3VuZGluz0JveCIgeDE9IjAuNSIgeTE9IjAuMCiGeDI9IjAuNSIgeTI9IjEuMCi+PHN0b3Agb2Zmc2V0PSIwJSIgc3RvcC1jb2xvcj0iI2ZmZmZmZiIgc3RvcC1vcGFjaXR5PSIwLjAiLz48c3RvcCBzmZZZXQ9IjEwMCUiIHN0b3AtY29sb3I9IiMwMDAwMDAiIHN0b3Atb3BhY210eT0iMC4xIi8+PC9saW51YXJHcmFkaWVudD48L2RlZnM+PHJ1Y3QgeD0iMCiGeT0iMCigd21kdGg9IjEwMCUiIGHlaWdodD0iMTAwJSIgZmlsbD0idXJsKCNncmFkKSIgLz48L3N2Zz4g');

background-size: 100%;

background-image: -webkit-gradient(linear, 50% 0%, 50% 100%, color-stop(0%, rgba(255, 255, 255, 0)), color-stop(100%, rgba(0, 0, 0, 0.1)));
background-image: -moz-linear-gradient(rgba(255, 255, 255, 0), rgba(0, 0, 0, 0.1));
background-image: -webkit-linear-gradient(rgba(255, 255, 255, 0), rgba(0, 0, 0, 0.1));
background-image: linear-gradient(rgba(255, 255, 255, 0), rgba(0, 0, 0, 0.1));

-moz-box-shadow: 0 0 8px rgba(0, 0, 0, 0.3);
```

```
-webkit-box-shadow: 0 0 8px rgba(0, 0, 0, 0.3);  
box-shadow: 0 0 8px rgba(0, 0, 0, 0.3);  
-moz-border-radius: 50%;  
-webkit-border-radius: 50%;  
border-radius: 50%;  
}  
.rangeslider__handle:after {  
content: "";  
display: block;  
width: 18px;  
height: 18px;  
margin: auto;  
position: absolute;  
top: 0;  
right: 0;  
bottom: 0;  
left: 0;
```

```

background-image: url ('data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiB1bmNvZGluzz0idXRmLTgiPz4gPHN2ZyB2ZXJz
aW9uPSIxLjEiIHtbG5zPSJodHRwOi8vd3d3LnczMm9yZy8yMDAwL3N2ZyI+PGRlZnM+PGxpbmVhckdyYWRpZW50IGlkPSJ
ncmFkIiBncmFkaWVudFVuaxRzPSJvYmplY3RCb3VuZGluz0JveCIgeDE9IjAuNSIgeTE9IjAuMCIGeDI9IjAuNSIgeTI9Ij
EuMCIGePHN0b3Agb2Zmc2V0PSIwJSIGc3RvcC1jb2xvcj0iIzAwMDAwMCIGc3RvcC1vcGFjaXR5PSIwLjEZi8+PHN0b3Agb
2Zmc2V0PSIxMDAIIiBzdG9wLWNvbG9yPSIjZmZmZmZmIiBzdG9wLW9wYWNpdHk9IjAuMCIVPjwvbGluZWFFyR3JhZGllbnQ+
PC9kZWZzPjxyZWN0IHg9IjAiIHk9IjAiIHdpZHRoPSIxMDAIIiBoZWlnaHQ9IjEwMCUiIGZpbGw9InVybcgjZ3JhZCkiIC8
+PC9zdmc+IA==') ;

background-size: 100%;

background-image: -webkit-gradient(linear, 50% 0%, 50% 100%, color-stop(0%, rgba(0, 0, 0, 0.13)), color-stop(100%, rgba(255, 255, 255, 0)));

background-image: -moz-linear-gradient(rgba(0, 0, 0, 0.13), rgba(255, 255, 255, 0));

background-image: -webkit-linear-gradient(rgba(0, 0, 0, 0.13), rgba(255, 255, 255, 0));

background-image: linear-gradient(rgba(0, 0, 0, 0.13), rgba(255, 255, 255, 0));

-moz-border-radius: 50%;

-webkit-border-radius: 50%;

border-radius: 50%;

}

.rangeslider__handle:active, .rangeslider--active .rangeslider__handle {

background-image: url ('data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiB1bmNvZGluzz0idXRmLTgiPz4gPHN2ZyB2ZXJz

```

```
aW9uPSIxLjEiIHtbG5zPSJodHRwOi8vd3d3LnczMm9yZy8yMDAwL3N2ZyI+PGRlZnM+PGxpbmVhckdyYWRpZW50IGlkPSJ
ncmFkIiBncmFkaWVudFVuaxRzPSJvYmplY3RCb3VuZGluz0JveCIgeDE9IjAuNSIgeTE9IjAuMCIGeDI9IjAuNSIgeTI9Ij
EuMCIGePHN0b3Agb2Zmc2V0PSIwJSIGc3RvcC1jb2xvcj0iIzAwMDAwMCIGc3RvcC1vcGFjaXR5PSIwLjEiLz48c3RvcCBvZ
mZZZXQ9IjEwMCUiIHN0b3AtY29sb3I9IiMwMDAwMDAiIHN0b3Atb3BhY210eT0iMC4xMiIvPjwvbGluzWfyR3JhZGllbnQ+
PC9kZWZzPjxyZWN0Ihg9IjAiIHk9IjAiIHdpZHRoPSIxMDAlIiBoZWlnaHQ9IjEwMCUiIGZpbGw9InVybCgjZ3JhZCkiIC8
+PC9zdmc+IA==') ;

background-size: 100%;

background-image: -webkit-gradient(linear, 50% 0%, 50% 100%, color-stop(0%, rgba(0, 0, 0, 0.1)), color-stop(100%, rgba(0, 0, 0, 0.12)));

background-image: -moz-linear-gradient(rgba(0, 0, 0, 0.1), rgba(0, 0, 0, 0.12));

background-image: -webkit-linear-gradient(rgba(0, 0, 0, 0.1), rgba(0, 0, 0, 0.12));

background-image: linear-gradient(rgba(0, 0, 0, 0.1), rgba(0, 0, 0, 0.12));

}

.rangeslider--horizontal .rangeslider__handle {

top: -10px;

touch-action: pan-y;

-ms-touch-action: pan-y;

}

.rangeslider--vertical .rangeslider__handle {

left: -10px;
```

```
touch-action: pan-x;  
-ms-touch-action: pan-x;  
}  
  
-moz-box-shadow: 0 0 8px rgba(255, 0, 255, 0.9);  
-webkit-box-shadow: 0 0 8px rgba(255, 0, 255, 0.9);  
box-shadow: 0 0 8px rgba(255, 0, 255, 0.9);  
}
```

public/assets/css/slider.css

```
label {  
display: block;  
margin-bottom: 2.5em;  
font-size: 13px;  
font-weight: bold;  
}
```

public/assets/css/styles.css

```
.main {  
    padding: 150px 150px 0 !important;  
}
```

```
.panel-heading {  
    background:#1e5e4e !important;  
    color:#ffffff;  
    font-weight:bold !important;  
    text-align:center !important;  
}
```

```
a.btn {  
    background:#4a985b;  
    border-color:#4a985b;
```

```
border-radius:25px;  
}  
  
a.btn:hover {  
background:#00851d;  
border-color:#4a985b;  
border-radius:25px;  
}  
  
div.footer-basic.footer {  
background-color:#25AA8A;  
color:#FFF;  
font-weight:bolder;  
}  
  
p.copyright {  
color:#FFF !important;
```

```
}

div.container.play {
    padding: 80px 80px 0 !important;
}

h3.panel-title {
    color:#fff;
}

div.play-background {
    background:#c5c4c4;
}

input#startBtn.btn.btn-success {
    border-radius:25px;
}
```

```
input#clearBtn.btn.btn-danger {  
    border-radius:25px;  
}  
  
div.main-background {  
    background:#c5c4c4;  
}  
  
div.col-md-12.leadertable {  
    padding:200px;  
}  
  
output {  
    display:block;  
    font-size:30px;  
    font-weight:bold;  
}
```

```
text-align:center;  
margin:30px 0;  
width:100%;  
}  
  
input[type=range] {  
-webkit-appearance:none;  
width:100%;  
margin:13.8px 0;  
}  
  
input[type=range]:focus {  
outline:none;  
}  
  
input[type=range]::-webkit-slider-runnable-track {  
width:100%;
```

```
height:8.4px;  
cursor:pointer;  
box-shadow:1px 1px 1px #000000, 0px 0px 1px #0d0d0d;  
background:#4a985b;  
border-radius:1.3px;  
border:0.2px solid #010101;  
}  
  
input[type=range]::-webkit-slider-thumb {  
box-shadow:1px 1px 1px #000000, 0px 0px 1px #0d0d0d;  
border:1px solid #000000;  
height:36px;  
width:16px;  
border-radius:3px;  
background:#deffff;  
cursor:pointer;  
-webkit-appearance:none;
```

```
margin-top:-14px;  
}  
  
input[type=range]:focus::-webkit-slider-runnable-track {  
background:#4a985b;  
}  
  
input[type=range]::-moz-range-track {  
width:100%;  
height:8.4px;  
cursor:pointer;  
box-shadow:1px 1px 1px #000000, 0px 0px 1px #0d0d0d;  
background:#4a985b;  
border-radius:1.3px;  
border:0.2px solid #010101;  
}
```

```
input[type=range]::-moz-range-thumb {  
    box-shadow: 1px 1px 1px #000000, 0px 0px 1px #0d0d0d;  
    border: 1px solid #000000;  
    height: 36px;  
    width: 16px;  
    border-radius: 3px;  
    background: #deffff;  
    cursor: pointer;  
}  
  
input[type=range]::-ms-track {  
    width: 100%;  
    height: 8.4px;  
    cursor: pointer;  
    background: transparent;  
    border-color: transparent;  
    color: transparent;
```

```
}
```

```
input[type=range]::-ms-fill-lower {  
background:#4a985b;  
border:0.2px solid #010101;  
border-radius:2.6px;  
box-shadow:1px 1px 1px #000000, 0px 0px 1px #0d0d0d;  
}
```

```
input[type=range]::-ms-fill-upper {  
background:#4a985b;  
border:0.2px solid #010101;  
border-radius:2.6px;  
box-shadow:1px 1px 1px #000000, 0px 0px 1px #0d0d0d;  
}
```

```
input[type=range]::-ms-thumb {
```

```
    box-shadow:1px 1px 1px #000000, 0px 0px 1px #0d0d0d;  
    border:1px solid #000000;  
    height:36px;  
    width:16px;  
    border-radius:3px;  
    background:#deffff;  
    cursor:pointer;  
    height:8.4px;  
}
```

```
input[type=range]:focus::-ms-fill-lower {  
    background:#4a985b;  
}
```

```
input[type=range]:focus::-ms-fill-upper {  
    background:#4a985b;  
}
```

```
#myBtn4 {  
    width:100px;  
    height:33px;  
}  
  
.pgn-container {  
    max-height: 200px;  
    overflow-y: scroll;  
}  
  
.pgn-download {  
    margin-top: 15px;  
    width: 100%;  
}  
  
.wrapper {
```

```
position: relative;  
  
min-height: 100%;  
  
}  
  
.main-content {  
  
padding-bottom: 250px !important;  
  
}  
  
.footer-basic {  
  
position: absolute;  
  
bottom: 0;  
  
left: 0;  
  
width: 100%;  
  
}
```

public/css/game_online.css

```
.opponent-info {  
  
list-style: none;  
  
padding-left: 0;
```

```
}
```

```
.opponent-info li {  
    margin-bottom: 10px;  
}
```

```
.opponent-info label {  
    display: inline-block;  
    margin: 0;  
    padding-right: 15px;  
    min-width: 85px;  
    color: #666;  
}
```

public/css/signup.css

```
.form-control.valid {  
    border-color: #3c763d;
```

```
}

.form-control.error {
    border-color: #a94442;
}

label.error {
    margin-top: 5px;
    color: #a94442;
}

#birthYear-error {
    display: block;
}


```

public/assets/js/custom.js

```
$ (document) .ready (function () {
```

```
var modal1 = document.getElementById('myModal1');

var modal2 = document.getElementById('myModal2');

var modal3 = document.getElementById('myModal3');

// Get the button that opens the modal

var btn1 = document.getElementById("myBtn1");

var btn2 = document.getElementById("myBtn2");

var btn3 = document.getElementById("myBtn3");

var btn4 = document.getElementById("myBtn4");

// Get the <span> element that closes the modal

var span1 = document.getElementById("close1");

var span2 = document.getElementById("close2");

var span3 = document.getElementById("close3");

// When the user clicks the button, open the modal
```

```

btn1.onclick = function() {
    modal1.style.display = "block";

    var $main = $( '#main_container' );
    var username = $main.data('username');
    var primus = new Primus();

    primus.on('data', function (msg) {
        if (!msg || !msg.action) return;

        switch (msg.action) {
            case 'paired':
                if (!msg.data || !msg.data.gameUrl) break;
                window.location.href = msg.data.gameUrl;
                break;
        }
    });
}

```

```

primus.write({
  action: 'lobby',
  data: {
    username: username
  }
}) ;

}

btn2.onclick = function(e) {
  $.ajax({
    type: 'post',
    dataType: 'json',
    url: '/game/create'
  })
  .then(function (result) {
    if (result.error_code != 0) {
      console.log('Error: code ' + result.error_code + ', msg: ' + result.msg);
    }
  })
}

```

```
        return;

    }

    var gameId = result.data && result.data.gameId;
    window.location.href = '/game/' + gameId;
} );

e.preventDefault();
return false;
}

btn3.onclick = function() {
    modal3.style.display = "block";
}

btn4.onclick = function () {
    var level = $('#ai_level').val();
    window.location.href = '/game/local/' + level;
}
```

```
};

// When the user clicks on <span> (x), close the modal
span1.onclick = function() {
    modal1.style.display = "none";
}

span2.onclick = function() {
    modal2.style.display = "none";
}

span3.onclick = function() {
    modal3.style.display = "none";
}

// When the user clicks anywhere outside of the modal, close it
```

```
window.onclick = function(event) {  
    if (event.target == modal1) {  
        modal1.style.display = "none";  
    }  
  
    if (event.target == modal2) {  
        modal2.style.display = "none";  
    }  
  
    if (event.target == modal3) {  
        modal3.style.display = "none";  
    }  
}  
};
```

public/js/game_common.js

```
'use strict'; // global scope
```

```
// Used to prevent redundant code, code is shared by _online & _local
(function (_global, _module, _define) {

    // Board constants

    var BOARD_STATUS = {
        WAITING_JOIN: 0,
        WAITING_MOVE: 1,
        YOUR_TURN: 2,
        GAME_OVER: 3,
        CHECK: 4,
        IN_CHECK: 5,
    };

    // Alert user of game status

    var inform = function (text) {
```

```

setTimeout(function() {
    alert(text);
}, 200);
};

// Initialize the whole chess game and board

// opts could have following fields (hooks), which have the same pattern of arguments
(data, api):

// 1. onLoad

// 2. onMove

// 3. onYouWin

// 4. onGameOver

// 5. onOfferDrawClicked
//      + if not provided, offer draw button will be hidden;

// 6. onResignClicked
//      + if not provided, resign button will be hidden;

// Initialize the Chess game & the board for the game

var initializeChess = function (_opts) {

```

```

var opts = _opts || {};
var gameId = opts.gameId || 'local_game';
var game = new Chess();
var $url = $( '#gameurl' );
var $urlWrapper = $( '#gameurl_wrapper' );
var $board = $( '#board' );
var $status = $( '#game_status' );
var $pgn = $( '#game_pgn' );
var $offerDraw = $( '#startBtn' );
var $resign = $( '#clearBtn' );
var $downPgn = $( '#pgn_download' );

// Board not in position, means not in playing status
if (!$board || !$board.length) return;

var pgn = $board.data('pgn');
var orientation = $board.data('orientation');

```

```

var username = $board.data('username');

var serverStatus = $board.data('status');

var currentStatus = null;

// PGN.length acts as additional defensive check

if (pgn && pgn.length) {

    game.load_pgn(window.atob(pgn));
}

var position = pgn && pgn.length > 0 ? game.fen() : 'start';

// Check the game status and then

// 1. update board status accordingly

// 2. call the hooks (onGameOver, onYouWin)

var checkGame = function checkGame() {

    // Assign appropriate strings for the depending on colour

    var turnColour = (game.turn() === 'w' ? 'white' : 'black'); // extracting from game
logic

    var lastColour = (game.turn() === 'w' ? 'black' : 'white');

```

```

// Who's turn is it on the board? What colour is waiting?

var statusToSet = turnColour === orientation ?
    BOARD_STATUS.YOUR_TURN :
    BOARD_STATUS.WAITING_MOVE;

// Initialize

var isDraw;
var winSide;

// Who has been placed in check?

if (game.in_check()) {
    statusToSet = turnColour === orientation ?
        BOARD_STATUS.IN_CHECK :
        BOARD_STATUS.CHECK;
}

```

```

// Has the game ended? If so, how has it ended?

if (game.game_over()) {

    statusToSet = BOARD_STATUS.GAME_OVER;

    // Send GAME_OVER to server only if you win

    if (turnColour !== orientation) {

        if (game.in_checkmate()) {

            // If the game has ended by checkmate then the player who made the last
            move has won.

            isDraw = false;

            winSide = orientation;

        } else if ( game.in_draw() || // Or has the game ended by a type of draw?
                    game.in_stalemate() ||
                    game.in_threelfold_repetition() ||
                    game.insufficient_material()) {

            isDraw = true;

            winSide = null;

        }
    }
}

```

```

// Emit a message via websocket with winner of game

if (opts.onYouWin) {
    opts.onYouWin({
        gameId: gameId,
        username: username,
        winSide: turnColour === 'white' ? 'black' : 'white',
        pgn: game.pgn(),
    }, api);
}

}

// Pass data so message can be displayed to user (You win or you lose)

if (opts.onGameOver) {
    opts.onGameOver({
        gameId: gameId,
        winSide: turnColour === 'white' ? 'black' : 'white',
        isLocalPlayerWin: turnColour !== orientation,
    });
}

```

```

        username: username,
        pgn: game.pgn(),
    }, api);
}

// Alerts for game endings

if (game.in_checkmate()) inform(lastColour + " has won the
game by checkmate!");

else if (game.in_draw()) inform("It's a draw!");

else if (game.in_stalemate()) inform("Stalemate!");

else if (game.in_threefold_repetition()) inform("Three move repetition,
the game is over.")

else if (game.insufficient_material()) inform("There is not enough
material. Game has ended as a draw.");
}

updateStatus(statusToSet); // Change the message displayed to user (waiting, your
turn etc)
};

// Board Rules (When a user is picking up a piece)

```

```

var onDragStart = function onDragStart(source, piece, position, orientation) {
  if (
    [
      BOARD_STATUS.YOUR_TURN,
      BOARD_STATUS.IN_CHECK
    ].indexOf(currentStatus) === -1 ||
    game.game_over() || // Cannot move pieces if the game has ended, draw, not your
    turn etc
    game.in_draw() ||
    // If the player is white then do not let the player move any of the black
    pieces.
    (orientation === 'white' && /b/.test(piece)) ||
    // If the player is black then do not let the player move any of the white
    pieces.
    (orientation === 'black' && /w/.test(piece))
  ) {
    return false;
  }
};

```

```
// When a player drops a piece:  
  
var onPieceDrop = function(source, target) {  
  
    // Check if that the player is moving is legal.  
  
    var move = game.move({  
  
        from: source,  
  
        to: target,  
  
        promotion: 'q' // NOTE: The game will always promote a pawn to a Queen for  
simplicity.  
  
    });
  
  
    // If the move is illegal then return the piece back to its original position.  
  
    if(move === null) {  
  
        return 'snapback';  
  
    }
  
  
};  
  
var onSnapEnd = function onSnapEnd(target) {
```

```
// Update the move made by the user on the Chess on the board itself.  
console.log('pgn updated', game.pgn());  
  
// Emit message to websocket about the move made by the user  
  
if (opts.onMove) {  
    opts.onMove ({  
        gameId: gameId,  
        username: username,  
        pgn: game.pgn(),  
        fen: game.fen()  
    }, api);  
}  
  
updateByPgn(); // Update the board  
};  
  
// Update board status with text and background color
```

```
var updateStatus = function (status) {  
  var bgColor;  
  var text;  
  
  currentStatus = status;  
  
  // Set board statuses for players  
  
  switch (status) {  
  
    // When the player is waiting for his oppoent to join  
  
    case BOARD_STATUS.WAITING_JOIN:  
  
      bgColor = 'rgb(250, 76, 156)';  
      text = 'Waiting for opponent';  
  
      break;  
  
    case BOARD_STATUS.WAITING_MOVE:  
  
      bgColor = 'rgb(47, 157, 227)';  
      text = 'Opponent To Move';  
  
      break;  
  
    case BOARD_STATUS.YOUR_TURN:  
  }  
};
```

```
        bgColor = 'rgb(245, 109, 43)';

        text     = 'Your Turn';

        break;

case BOARD_STATUS.CHECK:

        bgColor = 'rgb(4, 198, 0)';

        text     = 'Check!';

        break;

case BOARD_STATUS.IN_CHECK:

        bgColor = 'rgb(251, 55, 55)';

        text     = 'You are in check!';

        break;

default:

        bgColor = '#666';

        text     = 'Gameover';

        break;

}

// Display board status to user
```

```

$status.attr('style', 'background: ' + bgColor + ' !important;');
$status.find('h3').text(text);

};

// Update 3 parts of UI related with pgn
// 1. board
// 2. pgn box
// Update the Board UI using PGN

var updateByPgn = function (pgn) {
    if (pgn && pgn.length) {
        game.load_pgn(pgn); // load PGN string into box
    }

    // update board ui to new position
    board.position(game.fen());
    var pgnText = game.pgn({
        newline_char: '\n',

```

```

    max_width: 5

} ) ;

// pgn should be reversed before showed in PGN box due to string order

var pgnHtml = pgnText.split('\\n')
    .reverse()
    .join('<br />');

$pgn.html(pgnHtml); // update pgn box

// If PGN exists then display the download box

if (pgnText.length > 0) {
    $downPgn.attr('href', 'data:text/plain;base64,' + window.btoa(pgnText))
        .show();
} else {
    $downPgn.hide(); // or just do not show the box
}

```

```

    checkGame(); // check status of the game
};

// update game link (which is just the request url)

var updateLink = function (url) {
    $urlWrapper.show();
    $url.html('<a href="' + url + '">' + url + '</a>');
};

// Buttons to resign from a game & ask for a draw

var initButtons = function () {
    if (opts.onOfferDrawClicked) {
        // If the user clicks the button emit message via websocket
        $offerDraw.click(function () {
            opts.onOfferDrawClicked({
                gameId: gameId,
                username: username
            });
        });
    }
};

```

```

        }, api);

    });

} else {

    $offerDraw.hide(); // do not show for games vs computer

}

// If the user resigns from a game

if (opts.onResignClicked) {

    $resign.click(function () {

        opts.onResignClicked({

            gameId: gameId,
            username: username,
            side: orientation

        }, api);

    });

} else {

    $resign.hide(); // hide button

}

```

```

};

// Setup the board config

// Assign functions for user interaction

var boardConfig = {

  draggable: true, // pieces can be dragged

  position: position,

  orientation: ['black', 'white'].indexOf(orientation) !== -1 ? orientation :
'white',

  onDragStart: onDragStart,

  onDrop: onPieceDrop,

  onSnapEnd: onSnapEnd,

  pieceTheme: '/img/chesspieces/wikipedia/{piece}.png' // pictures of pieces

};

var board = ChessBoard('board', boardConfig); // create board

// Create API so hooks can be called

var api = {

  BOARD_STATUS: BOARD_STATUS,

```

```
updateLink: updateLink,  
updateStatus: updateStatus,  
updateByPgn: updateByPgn,  
checkGame: checkGame,  
inform: inform,  
moveChess: function (m) {  
    game.move(m);  
    updateByPgn();  
},  
getCurrentStatus: function () {  
    return currentStatus;  
}  
;  
if (opts.showGameLink) {  
    updateLink(window.location.href);  
}  
initButtons();
```

```
updateByPgn () ;  
  
// When a user joins, emit message via websocket  
  
if (opts.onLoad) {  
  
    opts.onLoad ({  
  
        gameId: gameId,  
  
        username: username,  
  
        orientation: orientation  
  
    }, api);  
  
}  
  
switch (serverStatus) {  
  
    case 'created':  
  
        updateStatus (BOARD_STATUS.WAITING_JOIN);  
  
        break;  
  
    case 'playing':  
  
        checkGame ();
```

```
        break;

    }

};

// export the function so it can be used elsewhere in the project

if (_module) {

    module.exports = initializeChess;

} else if (_define) {

    define(initializeChess);

}

if (_global) {

    window.initializeChess = initializeChess;

}

}) (typeof window !== 'undefined', typeof module !== 'undefined', typeof define !== 'undefined');
```

public/js/game_local.js

```
'use strict'; // execution scope

(function () {
    // Extract parameters from the request URL
    var level = (function () {
        var url = window.location.href.replace(/(.*?)#.*$/);
        var parts = url.split('/');
        return parts[parts.length - 1];
    });
    var id = level();
    if (id === '') {
        id = 'level1';
    }
    var game = document.getElementById(id);
    if (!game) {
        game = document.createElement('div');
        game.id = id;
        document.body.appendChild(game);
    }
    var title = document.createElement('h1');
    title.textContent = 'Level ' + id;
    game.appendChild(title);
});
```

```

    var lvl = parseInt(parts[parts.length - 1], 10); // find what level of computer the
user is playing against

    return Math.min(20, Math.max(0, lvl));
}();

// Create chess engine

var stockfish = new Worker('/assets/js/stockfish.js');

var openFishData = function (arg) {
    return arg.data;
};

// Setup jqeuery alias

var $output = $('#ai_output');

// Functions

initializeChess({
    onMove: function (data, api) { // inform client of move
        stockfish.postMessage('position fen ' + data.fen);
        stockfish.postMessage('go depth ' + level);
    }
});

```

```

    },
onGameOver: function (data, api) {
    api.updateStatus(api.BOARD_STATUS.GAME_OVER);
    api.inform('You ' + (data.isLocalPlayerWin ? 'Win' : 'Lose') + '!');
},
onLoad: function (data, api) {
    // initial req/res pairs:
    // 1. uci / uciok
    // 2. isready / readyok
    // 3. ucinewgame / [some options]
    //
    // in the progress of a normal chess game, we send stockfish
    //      + the new position ('position fen ...'),
    //      + ask it to calculate ('go depth 10')
    // and it will respond with some steps and finally a bestmove. that's all we need.
    // Consider stockfish as a server. Given a request, it will respond accordingly
}

```

```

stockfish.onmessage = function (event) {
    var data = openFishData(event);
    var move;

    if (!/^	option|id)/.test(data) && !^\\s*/.test(data)) {
        $output.prepend($('<li>' + data + '</li>'));
    }

    // Data is the request
    if (data === 'uciok') {
        stockfish.postMessage('isready'); // stockfish response
    }

    else if (data === 'readyok') {
        stockfish.postMessage('ucinewgame');
        api.updateStatus(api.BOARD_STATUS.YOUR_TURN);
    }

    else if (/^bestmove/.test(data)) {

```

```
move = data.split(/\s+/g) [1];  
  
api.moveChess ({  
    from: move.substr(0, 2),  
    to: move.substr(2)  
});  
}  
};  
  
stockfish.postMessage('uci');  
  
api.updateStatus(api.BOARD_STATUS.WAITING_JOIN);  
}  
});  
});()
```

public/js/game_online.js

```
'use strict'; // es5 && global scope

(function () {
    // Configs

    var primus = new Primus();

    var urlParts = window.location.href.replace(/(\\?|#).*$/, '').split('/');
    // Find split
    // request URL parameters

    var gameId = urlParts[urlParts.length - 1];

    var updateOpponentInfo = function (opponent) {
```

```
// Setup jQuery Aliases

var $container = $( '#component_info' );
var $opFirstName = $( '#player2_name' );
var $opFullName = $( '#opponent_fullname' );
var $opRank = $( '#opponent_rank' );
var $opWins = $( '#opponent_wins' );
var $opLosses = $( '#opponent_losses' );
var $opDraws = $( '#opponent_draws' );

if (!opponent || !opponent.local) {
    $container.hide();
    return;
}

var info = opponent.local;
```

```

$container.show();

$opFirstName.text(info.firstName);

$opFullName.text(info.firstName + ' ' + info.lastName);

$opRank.text(info.rank.rating);

$opWins.text(info.score.wins);

$opLosses.text(info.score.losses);

$opDraws.text(info.score.draws);

};

// setup socket messages with game_common functions

initializeChess({
    gameId: gameId,
    showGameLink: true,
    onMove: function (data, api) {
        primus.write({
            action: 'move',
            room: data.gameId,

```

```
        data: {
            gameId: data.gameId,
            pgn: data.pgn,
            username: data.username
        }
    ) );
},
onYouWin: function (data, api) {
    primus.write({
        action: 'gameover',
        room: data.gameId,
        data: {
            gameId: data.gameId,
            pgn: data.pgn,
            username: data.username,
            winSide: data.winSide
        }
    })
}
```

```
    } ) ;  
},  
  
onOfferDrawClicked: function (data, api) {  
  
    if (!confirm('Do you really want to call for a draw?')) return;  
  
    primus.write({  
  
        action: 'offerdraw',  
  
        room: data.gameId,  
  
        data: {  
  
            gameId: data.gameId  
  
        }  
    }) ;  
},  
  
onResignClicked: function (data, api) {  
  
    if (!confirm('Do you really want to resign?')) return;  
  
    primus.write({  
  
        action: 'resign',  
  
        room: data.gameId,  
    }) ;  
},
```

```

        data: {
            gameId: data.gameId,
            side: data.side,
            username: data.username
        }
    ) );
}

onLoad: function (data, api) {
    var orientation = data.orientation;
    var username = data.username;
    var setupPrimus = function () {
        primus.on('data', function (msg) {
            if (!msg || !msg.action) return;

            switch (msg.action) {
                case 'players_update':
                    if (!msg.data.players) {

```

```
        break;

    }

var _opponent = msg.data.players.find(function (p) {
    return p && p.local.email !== username;
}) ;

if (!_opponent) {
    break;
}

api.checkGame();
updateOpponentInfo(_opponent);
break;

case 'update':
    api.updateByPgn(msg.data.pgn);
```

```

break;

case 'gameover':
    api.updateStatus(api.BOARD_STATUS.GAME_OVER);

    if (msg.data.isDraw) {
        api.inform("It's a draw!");
    } else if (msg.data.winSide === orientation) {
        api.inform('Congratulations, you win!');
    } else {
        api.inform('Unlucky, you have lost the game.');
    }

break;

case 'offerdraw':
    if (confirm('Your opponent is offering you a draw, would you like
to accept?')) {

```

```

primus.write ({
    action: "acceptdraw",
    room: data.gameId,
    data: {
        gameId: data.gameId,
        username: data.username
    }
}) ;

}

break;

case 'resign':
    if (msg.data.side !== orientation) {
        api.inform('Your opponent has resigned!');
    }
    break;
}

```

```
    } ) ;  
  
    primus.write( {  
        action: "join",  
        room: data.gameId,  
        data: {  
            gameId: data.gameId,  
            username: data.username,  
            side: orientation,  
        }  
    } ) ;  
}  
  
setupPrimus() ;  
  
// Players should confirm when they are going to close a page where a game is still  
being played  
  
window.addEventListener("beforeunload", function (e) {
```

```

var isGameOn = [
    api.BOARD_STATUS.WAITING_JOIN,
    api.BOARD_STATUS.GAME_OVER
] .indexOf(api.getCurrentStatus()) === -1;

if (isGameOn) {
    e.returnValue = "\o/";
}

} );
}
} ) ();
}

```

public/js/signup.js

```

'use strict';

// Live validation for registration form

```

```
$ (function () {  
    // a hot fix for too many invokes of jquery-validation remote  
    // refer: https://github.com/jquery-validation/jquery-validation/issues/466#issuecomment-  
77819801  
  
    $.validator.methods._remote = $.validator.methods.remote;  
  
  
    var timer = 0;  
  
    $.validator.methods.remote = function () {  
        clearTimeout(timer);  
  
        var args = arguments;  
  
        timer = setTimeout(function () {  
            $.validator.methods._remote.apply(this, args);  
        }.bind(this), 300);  
  
        return "pending";  
    };
```

```
// Rules of registration form
$( '#signup_form' ).validate ({
    rules: {
        email: {
            required: true,
            email: true,
            maxlength: 64,
            remote: {
                url: '/checkemail',
                method: 'post'
            }
        },
        password: {
            required: true,
            minlength: 7,
            maxlength: 32
        }
    }
});
```

```
        } ,  
  
        firstName: {  
  
            required: true,  
  
            maxlength: 35  
        } ,  
  
        lastName: {  
  
            required: true,  
  
            maxlength: 35  
        } ,  
  
        birthYear: {  
  
            required: true,  
  
            max: (new Date()).getFullYear() - 13  
        } ,  
    } ,  
  
    // Responses  
  
    messages: {  
  
        email: {
```

```
        required: 'Please enter your email address',
        email: 'Please enter a valid email address',
        maxlength: 'Your email address should not be longer than 64 characters',
        remote: 'Another user has already signed up with that email!',
    },
    password: 'Please enter a password that is between 8 and 32 characters',
    firstName: 'Your first name can not be longer than 35 characters!',
    lastName: 'Your last name can not be longer than 35 characters!',
    birthYear: 'You must be at least 13 to use our application',
}
}

});
```

public/js/user_list.js

```
'use strict';

(function () {
    // Configuration for jquery alias & queries
    var $search = $('#search_user');
    var $perPage = $('#per_page');
    var oldValue = $perPage.val();
    var path = window.location.pathname;
    var isSearch = /^\/search\/user//.test(path);
    var isLeader = /^\/leaderboards//.test(path);
```

```

// When value changes
$perPage.on('change', function () {
  var p = $perPage.val();
  var parts;
  if (p === oldValue) return; // no change

  if (isSearch) {
    parts = window.location.href.split('/');
    parts.splice(parts.length - 2, 2, [p, 1]);
    window.location.href = parts.join('/');
  } else if (isLeader) {
    window.location.href = '/leaderboards/' + p + '/1';
  }
});

// Once the user presses a key, we can start to search.
$search.on('keydown', function (e) {
  var query;

```

```

    if (e.keyCode === 13) {

        query = $search.val().replace(/^\s+|\s+$/g, '');
    }

    if (query.length === 0) {
        window.location.href = '/leaderboards';
    } else {
        window.location.href = '/search/user/' + query + '/' + oldValue + '/1';
    }
}

})();

})();

```

views/common/footer.ejs

```

<div class="footer-basic footer">
    <footer>
        <ul class="list-inline">

```

```
<li><a href="/">Home &nbsp; | </a></li>

<li><a href="http://www.eastlandfairfield.com/About.aspx">About &nbsp; | </a></li>

<li><a href="http://www.eastlandfairfield.com/ContactUs.aspx">Contact Us &nbsp; | </a></li>

<li><a href="http://www.eastlandfairfield.com/ProgramsCourses.aspx"> Courses &nbsp; </a></li>

</ul>

<p class="copyright">Eastland Career Center © 2017</p>

</footer>

</div>
```

views/common/header.ejs

```
<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title><%= page.title %> | Eastland Career Center</title>

<%- include('styles'); %>
```

```

<% (css || []).forEach(function (uri) { %>
  <link rel="stylesheet" href="<%= uri %>"%>
<% }) %>

<%- include('scripts'); %>
</head>

views/common/nav.ejs
<div>

  <nav class="navbar navbar-default navigation-clean">
    <div class="container">
      <div class="navbar-header"><a class="navbar-brand text-uppercase navbar-link"
        href="/">Eastland Career Center </a>
        <button class="navbar-toggle collapsed" data-toggle="collapse" data-
        target="#navcol-1"><span class="sr-only">Toggle navigation</span><span class="icon-
        bar"></span><span class="icon-bar"></span><span class="icon-bar"></span></button>
    </div>
    <div class="collapse navbar-collapse" id="navcol-1">
      <ul class="nav navbar-nav navbar-right">

```

```

<% if (user) { %>

    <li role="presentation"><a href="/">HOME </a></li>

    <li role="presentation"><a href="/leaderboards">LEADERBOARDS </a></li>

        <li class="dropdown"><a class="dropdown-toggle" data-toggle="dropdown"
aria-expanded="false" href="#"><%= (user.local.firstName + ' ' +
user.local.lastName).toUpperCase() %><span class="caret"></span></a>

            <ul class="dropdown-menu" role="menu">

                <li role="presentation"><a href="/logout">Logout</a></li>

            </ul>
        </li>

    </li>

<% } else { %>

    <li role="presentation"><a href="/login">LOGIN</a></li>

    <li role="presentation"><a href="/signup">REGISTER</a></li>

<% } %>

</ul>
</div>
</div>

```

```
</div>
```

views/common/scripts.ejs

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/chess.js/0.10.2/chess.js"></script>

<script src="/assets/js/jquery.min.js"></script>
<script src="/assets/bootstrap/js/bootstrap.min.js"></script>
<script src="/assets/js/rangeslider.min.js"></script>
<script src="/assets/js/revcustom.js"></script>

<script src="/js/chessboard-0.3.0.min.js"></script>
<script src="/js/primus.js"></script>
```

views/common/styles.ejs

```
<link rel="stylesheet" href="/assets/bootstrap/css/bootstrap.min.css">

<link rel="stylesheet" href="/css/chessboard-0.3.0.min.css">

<link rel="stylesheet" href="/assets/css/Footer-Basic.css">

<link rel="stylesheet" href="/assets/css/Footer-Clean.css">

<link rel="stylesheet" href="/assets/css/Navigation-Clean1.css">

<link rel="stylesheet" href="/assets/css/popups.css">

<link rel="stylesheet" href="/assets/css/rangeSlider.css">

<link rel="stylesheet" href="/assets/css/styles.css">

<link rel="stylesheet" href="/assets/css/styles.css">
```

views/widgets/styles.ejs

```
<div>

  <div class="row">

    <div class="col-sm-6">

      <div class="dataTables_length form-group" id="dataTables-example_length">

        <label>Show</label>
```

```

        <select id="per_page" class="form-control" style="display: inline-block; width: 80px;" name="dataTables-example_length" aria-controls="dataTables-example">

            <option value="10" <%= perPage === 10 ? 'selected' : ''%> >10</option>
            <option value="25" <%= perPage === 25 ? 'selected' : ''%> >25</option>
            <option value="50" <%= perPage === 50 ? 'selected' : ''%> >50</option>
            <option value="100" <%= perPage === 100 ? 'selected' : ''%> >100</option>

        </select>

        <label>entries</label>

    </div>

</div>

<div class="col-lg-12" style="margin: 0px 0px 0px 0px;">

    <div id="dataTables-example_filter form-inline" class="dataTables_filter">

        <div class="form-group">

            <label for="search_user">Search:</label>

            <input

                id="search_user" class="form-control" style="display: inline-block; max-width: 200px"

```

```

        type="search" placeholder="" aria-controls="dataTables-example"
        value="<% = searchText %>">
    </div>
</div>
</div>
</div>

<div class="row">
    <div class="col-sm-12">
        <table width="100%" class="table table-striped table-bordered table-hover dataTable
no-footer dtr-inline" id="dataTables-example" role="grid" aria-describedby="dataTables-
example_info" style="width: 100%;">
            <thead>
                <tr role="row">
                    <% if (showPosition) { %>
                        <th style="text-align: center; width: 161px;" class="sorting_asc"
tabindex="0" aria-controls="dataTables-example" rowspan="1" colspan="1" aria-sort="ascending"
aria-label="Position: activate to sort column descending">Position</th>
                    <% } %>

```

```

        <th style="text-align: center; width: 433px;" class="sorting"
tabindex="0" aria-controls="dataTables-example" rowspan="1" colspan="1" aria-label="Name:
activate to sort column ascending">Name</th>

        <th style="text-align: center; width: 252px;" class="sorting"
tabindex="0" aria-controls="dataTables-example" rowspan="1" colspan="1" aria-label="Rating:
activate to sort column ascending">Rating</th>

        <th style="text-align: center; width: 262px;" class="sorting"
tabindex="0" aria-controls="dataTables-example" rowspan="1" colspan="1" aria-label="Age:
activate to sort column ascending">Age</th>

        <th style="text-align: center; width: 268px;" class="sorting"
tabindex="0" aria-controls="dataTables-example" rowspan="1" colspan="1" aria-
label="Wins/Losses/Draws: activate to sort column ascending">Wins/Losses/Draws</th>

    </tr>

</thead>

<tbody>

<% users.forEach(function (user, i) { %>

    <tr style="text-align: center;" class="gradeX odd" role="row">

        <% if (showPosition) { %>

            <td class="sorting_1"><%= startPosition + i + 1 %></td>

        <% } %>

        <td><%= user.local.firstName + ' ' + user.local.lastName %></td>

```

```

<td><%= user.local.rank.rating %></td>

<td>
  <% if (user.local.birthday) { %>
    <%= (new Date()).getFullYear() - user.local.birthday.year
  %>
  <% } else { %>
    N/A
  <% } %>
</td>

<td>
  <%= user.local.score.wins %> /
  <%= user.local.score.losses %> /
  <%= user.local.score.draws %>
</td>

</tr>
<% } ) %>
</tbody>
</table>

```

```

</div>

</div>

<div class="row">

    <div class="col-sm-6">

        <div class="dataTables_info" id="dataTables-example_info" role="status" aria-
live="polite">Showing <%= startPosition + 1 %> to <%= endPosition + 1 %> of <%= totalCount %>
entries</div>

    </div>

</div>

<div class="row">

    <div class="dataTables_paginate paging_simple_numbers" id="dataTables-example_paginate"
style="margin-left: 50%">

        <ul class="pagination" style="transform: translateX(-50%)">

            <li class="paginate_button previous <%= pageNum === 1 ? 'disabled' : '' %>" aria-
controls="dataTables-example" tabindex="0" id="dataTables-example_previous"><a
href="#">Previous</a></li>

            <% pages.forEach(function (n) { %>

                <li class="paginate_button <%= pageNum === n ? 'active' : '' %>" aria-
controls="dataTables-example" tabindex="0">

                    <a href="/leaderboards/<%= perPage %>/<%= n %>"><%= n %></a>

```

```

        </li>

        <% } ) %>

        <li class="paginate_button next <%= pageNum === totalPages ? 'disabled' : '' %>"  

aria-controls="dataTables-example" tabindex="0" id="dataTables-example_next"><a  

href="#">Next</a></li>

</ul>

</div>

</div>

</div>

```

views/game_common.ejs

```

1. <div class="play-background main-content">
2.   <% if (error) { %>
3.     <h2><%= error %></h2>
4.   <% } else { %>
5.     <div class="container play">
6.       <div class="row">
7.         <div class="col-md-3">
8.           <div class="panel panel-default">
9.             <div class="panel-heading" id="game_status">
10.               <h3 class="panel-title">Game Status </h3></div>
11.           </div>
12.
13.           <div class="panel panel-default" id="gameurl_wrapper" style="display:  

none">

```

```

14.          <div class="panel-heading">
15.              <h3 class="panel-title">Game Link </h3></div>
16.              <div class="panel-body"><span id="gameurl" style="margin-left:
17.      10px; word-break: break-all; word-wrap: break-word;"></span></div>
18.          </div>
19.
20.          <div class="col-md-6">
21.              <div
22.                  id="board"
23.                  data-pgn="<% game.pgn %>"
24.                  data-orientation="<% game.orientation %>"
25.                  data-status="<% game.status %>"
26.                  data-username="<% user.local.email %>"
27.              ></div>
28.              <br>
29.              <br>
30.              <span style="margin: 0px 170px 0px 170px;">
31.                  <input class="btn btn-
    success" type="button" id="startBtn" value="Offer Draw" />
32.                  <input class="btn btn-
    danger" type="button" id="clearBtn" value="Resign" />
33.              </span>
34.          </div>
35.
36.          <div class="col-md-3">
37.              <% if (isOnline) { %>
38.                  <div id='component_info' class="panel panel-
    default" style='display: none'>
39.                      <div class="panel-heading">
40.                          <h3 class="panel-title">
41.                              <span id='player1_name'><%= user.local.firstName ||
    user.local.email %></span>,

```

```

42.          <span>you are playing</span>
43.          <span id='player2_name'></span>
44.      </h3>
45.  </div>
46.  <div class="panel-body">
47.    <ul class="opponent-info">
48.      <li>
49.        <label>Opponent:</label>
50.        <span id='opponent_fullname'></span>
51.      </li>
52.      <li>
53.        <label>Rank:</label>
54.        <span id='opponent_rank'></span>
55.      </li>
56.      <li>
57.        <label>Wins:</label>
58.        <span id='opponent_wins'></span>
59.      </li>
60.      <li>
61.        <label>Losses:</label>
62.        <span id='opponent_losses'></span>
63.      </li>
64.      <li>
65.        <label>Draws:</label>
66.        <span id='opponent_draws'></span>
67.      </li>
68.    </ul>
69.  </div>
70. </div>
71. <% } else { %>
72.  <div id='component_info' class="panel panel-default">
73.    <div class="panel-heading">

```

```

74.          <h3 class="panel-title" style="font-size: 15px; line-
    height: 17px;">
75.              <%= user.local.firstName || user.local.email %>,
76.              you're playing a level <%= level %> computer!
77.          </h3>
78.      </div>
79.      <div class="panel-body">
80.          <h5>AI is talking:</h5>
81.          <ul id="ai_output" style="padding-left: 15px; max-height:
    120px; overflow-y: scroll;">
82.              <li>Hello Ya!</li>
83.          </ul>
84.      </div>
85.  </div>
86. <% } %>
87.
88.      <div class="panel panel-default">
89.          <div class="panel-heading">
90.              <h3 class="panel-title">PGN Moves</h3></div>
91.          <div class="panel-body">
92.              <div class="pgn-container">
93.                  <span id="game_pgn"></span>
94.              </div>
95.              <a download="pgn.txt" href="#" id="pgn_download" class="btn
    btn-success pgn-download">Download PGN</a>
96.          </div>
97.      </div>
98.  </div>
99.      </div>
100.     </div>
101. <% } %>
102. </div>

```

views/game_local.ejs

```
1. <!DOCTYPE html>
2. <html>
3.
4. <%- include('common/header', {
5.   page: {
6.     title: 'Training'
7.   },
8.   css: []
9. }); %>
10.
11.  <body>
12.    <div class="wrapper">
13.      <%- include('common/nav'); %>
14.      <%- include('game_common', {
15.        game: {
16.          orientation: 'white',
17.          status: 'created'
18.        },
19.        error: null,
20.        isOnline: false,
21.        level: level
22.      }); %>
23.      <%- include('common/footer'); %>
24.    </div>
25.    <script src="/js/game_common.js"></script>
26.    <script src="/js/game_local.js"></script>
27.  </body>
28.
29. </html>
```

views/game_online.ejs

```
1. <!DOCTYPE html>
2. <html>
3.
4. <%- include('common/header', {
5.   page: {
6.     title: 'Ranked Game'
7.   },
8.   css: [
9.     '/css/game_online.css'
10.    ]
11. }) ; %>
12.
13. <body>
14.   <div class="wrapper">
15.     <%- include('common/nav') ; %>
16.     <%- include('game_common', {
17.       game: game,
18.       error: error,
19.       isOnline: true,
20.     }) ; %>
21.     <%- include('common/footer') ; %>
22.   </div>
23.
24.
25.   <script src="/js/game_common.js"></script>
26.   <script src="/js/game_online.js"></script>
27. </body>
28.
29. </html>
```

views/index.ejs

```
1. <!DOCTYPE html>
2. <html>
3.
4. <%- include('common/header', {
5.   page: {
6.     title: 'Index'
7.   },
8.   css: ['/assets/css/slider.css']
9. }); %>
10.
11. <body>
12.   <div class="wrapper">
13.     <%- include('common/nav'); %>
14.
15.     <div class="main-background main-content">
16.       <div class="container main" id="main_container" data-username="<% user.local.email %>">
17.         <div class="row">
18.           <div class="col-md-8">
19.             <div class="panel panel-primary">
20.               <div class="panel-heading">
21.                 <h3 class="panel-title">Information</h3></div>
22.               <div class="panel-body">
23.                 <p>Welcome to the Eastland Career Center's official Chess Application. We know that you have all been waiting for this to finally be released for a long time now! If you're looking to play for fun, improve your game or play competitively then you've come to the right place.</p>
24.                 <p><strong>Active Games: </strong><%= activeGameCount %></p>
25.                 <p><strong>Total Online: </strong><%= activePlayerCount %></p>
26.               </div>
27.             </div>
```

```

28.          </div>
29.          <div class="col-md-4">
30.              <!-- Main Panel -->
31.      <div class="panel panel-primary">
32.          <!-- Panel Heading-->
33.          <div class="panel-heading">
34.              <h3 class="panel-title">Game Modes</h3>
35.          </div>
36.          <!--/ Panel Heading-->
37.          <div class="panel-body" > <!-- Panel Body-->
38.
39.              <br /><a id="myBtn1" class="btn btn-danger btn-lg center-block" href="#">Play
    solo</a>
40.              <br />
41.
42.          <!-- Modall content -->
43.          <div id="myModal1" class="modal">
44.
45.
46.              <div class="modal-content" style="margin: 0 auto; text-align: center;">
47.                  <span id="close1" class="close">&times;</span>
48.                  <p><strong>You have been placed into the queue.</strong></p>
49.                  
50.              </div>
51.
52.          </div>
53.          <!-- / Modall content -->
54.
55.          <a id="myBtn2" class="btn btn-danger btn-lg center-block" href="#">Play with
    friend</a>
56.          <br />
57.          <!-- Modal2 content -->

```

```

58.          <div id="myModal2" class="modal">
59.
60.
61.            <div class="modal-content" style="margin: 0 auto; text-align: center;">
62.              <span id="close2" class="close">&times;</span>
63.              <h2>Game Link: </h2>
64.              <p> <a href="http://localhost:3000/profile?gameId=e2d0466c7aca0f47dc0c">http://localhost:3000/profile?gameId=e2d0466c7aca0f47dc0c </a></p>
65.
66.            </div>
67.
68.        </div>
69.        <!-- / Modal2 content -->
70.        <a id="myBtn3" class="btn btn-danger btn-lg center-block" href="#">Train/Computer</a>
71.          <br />
72.        <!-- Modal3 content -->
73.        <div id="myModal3" class="modal">
74.
75.
76.          <div class="modal-content" style="margin: 0 auto; text-align: center;">
77.            <span id="close3" class="close">&times;</span>
78.            <div id="dlevel">
79.              <h2>Difficulty Level</h2>
80.              <output></output>
81.              <input type="range" min="1" max="20" value="1" data-rangeSlider id="ai_level"><br>
82.              <a id="myBtn4" class="btn btn-success center-block" href="#">Play</a>
83.
84.            </div>
85.          </div>
86.
87.        </div> <!-- / Modal3 content -->
88.

```

```

89.      </div> <!-- Main Panel Body -->
90.    </div> <!-- Main Panel -->
91.      </div>
92.        </div>
93.        </div>
94.      </div>
95.
96.
97.      <%- include('common/footer'); %>
98.    </div>
99.    <script src="/assets/js/custom.js"></script>
100.   </body>
101.
102.  </html>

```

views/leaderboards.ejs

```

1. <!DOCTYPE html>
2. <html>
3.
4. <%- include('common/header', {page: {title: 'Leaderboard'}, css: []}); %>
5.
6. <body>
7.   <div class="wrapper">
8.     <%- include('common/nav'); %>
9.
10.    <div class="row main-background main-content" style="padding: 100px;">
11.      <div class="col-md-12">
12.        <div class="panel panel-default">
13.          <div class="panel-heading">
14.            <span style="color: #FFF; font-weight: bold;">Leaderboards</span>
15.          </div>

```

```

16.          <!-- /.panel-heading -->
17.          <div class="panel-body">
18.            <div id="dataTables-example_wrapper" class="dataTables_wrapper dt-bootstrap
no-footer">
19.              <%- include('widgets/user_list', {
20.                users: users,
21.                totalCount: totalCount,
22.                totalPage: totalPage,
23.                perPage: perPage,
24.                pageNum: pageNum,
25.                pages: pages,
26.                startPosition: startPosition,
27.                endPosition: endPosition,
28.                searchText: searchText,
29.                showPosition: showPosition
30.              }) %>
31.            </div>
32.            <!-- /.table-responsive -->
33.          </div>
34.          <!-- /.panel-body -->
35.        </div>
36.        <!-- /.panel -->
37.      </div>
38.      <!-- /.col-lg-12 -->
39.    </div>
40.
41.
42.    <%- include('common/footer') ; %>
43.  </div>
44.
45.  <script src="/js/user_list.js"></script>
46.
47. </body>

```

```
48.  
49.    </html>
```

views/login.ejs

```
1. <!doctype html>  
2. <html>  
3.  
4. <%- include('common/header', {  
5.   page: {  
6.     title: 'Login'  
7.   },  
8.   css: [  
9.     '//netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css'  
10.    ]  
11.  }) ; %>  
12.  
13. <body>  
14.   <div class="wrapper">  
15.     <%- include('common/nav', {user: null}); %>  
16.  
17.     <div class="container main-content" style="min-height: 500px; padding-top: 50px;">  
18.       <div class="col-sm-6 col-sm-offset-3">  
19.         <h1><span class="fa fa-sign-in"></span> Login</h1>  
20.  
21.         <% if (message.length > 0) { %>  
22.           <div class="alert alert-danger"><%= message %></div>  
23.         <% } %>  
24.  
25.         <!-- LOGIN FORM -->  
26.         <form action="/login" method="post">
```

```

27.          <div class="form-group">
28.            <label>Email</label>
29.            <input type="text" class="form-control" name="email">
30.          </div>
31.          <div class="form-group">
32.            <label>Password</label>
33.            <input type="password" class="form-control" name="password">
34.          </div>
35.
36.          <button type="submit" class="btn btn-warning btn-lg">Login</button>
37.        </form>
38.
39.        <hr>
40.
41.        <p>Need an account? <a href="/signup">Signup</a><br /><br /></p>
42.      </div>
43.    </div>
44.
45.    <%- include('common/footer'); %>
46.  </div>
47.  <script src="/assets/js/jquery.validate.min.js"></script>
48. </body>
49. </html>

```

views/search_user.ejs

1. <!DOCTYPE html>
2. <html>

```
3.  
4. <%- include('common/header', {page: {title: 'Leaderboard'}, css: []}); %>  
5.  
6. <b>body</b>  
7. <div class="wrapper">  
8. <%- include('common/nav'); %>  
9.  
10. <div class="row main-background main-content" style="padding: 100px;">  
11. <div class="col-md-12">  
12.   <div class="panel panel-default">  
13.     <div class="panel-heading">  
14.       <span style="color: #FFF; font-weight: bold;">Search Result</span>  
15.     </div>  
16.     <!-- /.panel-heading -->  
17.     <div class="panel-body">  
18.       <div id="dataTables-example_wrapper" class="dataTables_wrapper dt-bootstrap no-footer">  
19.         <%- include('widgets/user_list', {  
20.           users: users,  
21.           totalCount: totalCount,  
22.           totalPage: totalPage,  
23.           perPage: perPage,  
24.           pageNum: pageNum,  
25.           pages: pages,  
26.           startPosition: startPosition,  
27.           endPosition: endPosition,  
28.         })>
```

```
28.      searchText: searchText,
29.      showPosition: showPosition
30.    }) %>
31.    </div>
32.    <!-- /.table-responsive -->
33.  </div>
34.  <!-- /.panel-body -->
35. </div>
36. <!-- /.panel -->
37. </div>
38. <!-- /.col-lg-12 -->
39.</div>
40.
41.
42.<%- include('common/footer'); %>
43.</div>
44.<script src="/js/user_list.js"></script>
45.
46.</body>
47.
48.</html>
```

views/signup.ejs

```
1. <!doctype html>
```

```

2. <html>
3.
4. <%- include('common/header', {
5.   page: {
6.     title: 'Register'
7.   },
8.   css: [
9.     '/css/signup.css',
10.    '//netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css'
11.   ]
12. }) ; %>
13.
14. <body>
15. <div class="wrapper">
16.
17. <%- include('common/nav', {user: null}); %>
18. <div class="container main-content" style="min-height: 500px; padding-top: 50px;">
19.   <div class="col-sm-6 col-sm-offset-3">
20.
21.     <h1><span class="fa fa-user"></span> Register</h1>
22.
23.     <% if (message.length > 0) { %>
24.       <div class="alert alert-danger"><%= message %></div>
25.     <% } %>
26.
27.     <!-- SIGNUP FORM -->
28.     <form action="/signup" method="post" id="signup_form">
29.       <div class="form-group">
30.         <label>Email *</label>
31.         <input type="text" maxlength="64" class="form-
control" name="email" placeholder="name@example.com" required>
32.       </div>
33.       <div class="form-group">

```

```

34.          <label>Password *</label>
35.          <input type="password" minlength="8" maxlength="32" class="form-
control" name="password" placeholder="At least 8 characters" required>
36.        </div>
37.        <div class="form-group">
38.          <label>First Name *</label>
39.          <input type="text" maxlength="35" class="form-
control" name="firstName" required>
40.        </div>
41.        <div class="form-group">
42.          <label>Last Name *</label>
43.          <input type="text" maxlength="35" class="form-
control" name="lastName" required>
44.        </div>
45.        <div class="form-group">
46.          <label>Birthday *</label>
47.          <div class="form-group form-inline">
48.            <select name="birthDate" class="form-control">
49.              <% dates.forEach(function (date) { %>
50.                <option value="<%= date %>"><%= date %></option>
51.              <% }) %>
52.            </select>
53.
54.            <select name="birthMonth" class="form-control">
55.              <% months.forEach(function (month, i) { %>
56.                <option value="<%= i + 1 %>"><%= month %></option>
57.              <% }) %>
58.            </select>
59.
60.            <select name="birthYear" class="form-control">
61.              <% years.forEach(function (year) { %>
62.                <option value="<%= year %>"><%= year %></option>
63.              <% }) %>

```

```

64.          </select>
65.      </div>
66.  </div>
67.
68.      <button type="submit" class="btn btn-warning btn-lg">Signup</button>
69.  </form>
70.
71.  <hr>
72.
73.      <p>Already have an account? <a href="/login">Login</a><br /><br /></p>
74.  </div>
75. </div>
76.
77.      <%- include('common/footer'); %>
78.  </div>
79.
80.  <script src="/assets/js/jquery.validate.min.js"></script>
81.  <script src="/js/signup.js"></script>
82. </body>
83. </html>

```

Section 3.6: Server.js & Package.json

server.js

```

// Import modules local to Node

var path = require('path');

```

```
// Import NPM Modules

var express = require('express');

var mongoose = require('mongoose');

var passport = require('passport');

var flash = require('connect-flash');

var morgan = require('morgan');

var cookieParser = require('cookie-parser');

var bodyParser = require('body-parser');

var session = require('express-session');

var MongoStore = require('connect-mongo')(session);

var glicko2 = require('glicko2');

// Import modules local to project

var configDB = require('./config/database.js');

var Game = require('./app/models/game');

var User = require('./app/models/user');

var primusHost = require('./app/primus');
```

```
// Configure Webserver

var app = express();
var port = process.env.PORT || 3000;

// Connect to database & setup user information

mongoose.connect(configDB.url);
require('./config/passport')(passport);

// Setup Express

app.use(morgan('dev')) // log requests to console
app.use(cookieParser()); // read cookies for authentication
app.use(bodyParser.json()); // get information from HTML forms
app.use(bodyParser.urlencoded({extended: true})); // allow for urls to be parsed with qs
libraries

// Setup view engine
```

```
app.set('view engine', 'ejs');

// Persistent Login (user's remain logged in even if server is restarted)
app.use(session({
  secret: 'eastlandsecretssessionkey', // Secret key for session
  resave: true,
  saveUninitialized: true,
  cookie: {},
  store: new MongoStore({
    url: configDB.url
  })
}) );

app.use(passport.initialize()); // Start passport
app.use(passport.session()); // Allow for persistent login sessions
app.use(flash()); // Allow for messages to be stored in sessions
```

```

// Setup routes

require('./app/routes.js')(app, passport); // Load routes into app & passport

// Set public folder to as default web route

app.use(express.static(path.join(__dirname, 'public')));

// Start server

var server = app.listen(port);

console.log("Eastland Chess is running on port: " + port);

// Setup websockets with webserver

primusHost.init(server);

```

package.json

```
{
  "name": "eastland-chess",
  "version": "1.0.0",
  "description": "A real-time multiplayer chess game",
}
```

```
"main": "server.js",
"scripts": {
  "start": "node server.js",
  "test": "echo \"Error: no test specified\" && exit 1"
},
"author": "CENSOR BLUR",
"license": "ISC",
"dependencies": {
  "bcrypt-nodejs": "latest",
  "body-parser": "~1.15.2",
  "btoa": "^1.1.2",
  "connect-ensure-login": "^0.1.1",
  "connect-flash": "~0.1.1",
  "connect-mongo": "^1.3.2",
  "cookie-parser": "~1.4.3",
  "ejs": "~2.5.2",
  "engine.io": "^2.0.2",
```

```
"express": "^4.14.1",
"express-session": "~1.14.1",
"express-static": "^1.0.3",
"glicko2": "^0.8.4",
"method-override": "~2.3.6",
"mongoose": "~4.6.1",
"mongoose-paginate": "^5.0.3",
"morgan": "~1.7.0",
"passport": "~0.3.2",
"passport-local": "~1.0.0",
"primus": "^6.1.0",
"primus-rooms": "^3.4.1"
}
```


Full Name: BLUR, CENSOR

Candidate Number: N/A

Centre Number: N/A

SECTION 4: TESTING

Section 4.1: Iteration 1 Testing

The first iteration of my project designed the preliminary elements of my application. These elements must be installed and tested properly because they are vital development tools for the future of my application. Therefore, I believe that it is only rational to various perform tests on my system to ensure that it is running correctly.

Although there is nothing on the server that I have developed personally, incorrectly setup software may impair performance, cause memory leaks or worst of all cause unexpected bugs. For this iteration and the remaining iterations, I will complete a series of tests on parts of my system to ensure that they are robust. I will also be aiming to perform three different types of testing on aspects of my application:

1. **Black box testing → Input & Output testing**
2. **White box testing → Analysing subroutines by checking all the pathways through the code**
3. **Unit testing → Ensuring that each unit carries out the function that it has been designed for and incorporates both black box and white box testing**

Finally, to ensure that the testing stage of my project is done properly, I will be testing elements of my application using:

1. **Normal data → Normal data is data the system is expecting and should be more than able to work with.**
2. **Boundary data → Boundary values include maximum, minimum, just inside/outside boundaries, typical values and error values.**
3. **Erroneous data → Erroneous data is data that should cause the system to tell the user that there is a problem with data entered the system.**

By testing all three types of data on my system, it will become easier to discover bugs in the code such as conditional errors or just incorrect code. It is important to note that there will be various tests that will occur throughout the development of my project which will not be documented. Usually, these tests will occur when writing code, especially white box testing, to ensure that the code I use for my technical solution is correct. These tests will occur but are likely to not be very extensive.

Elements of system to test

1. Are the repositories on the system up to date?
2. NodeJS and MongoDB were installed on the system, have they been installed correctly?
3. Is the webserver running?
 - 3.1. Is the webserver running on the correct port?
4. Has the database user been created?
 - 4.1. Have the collections been created inside of the database?

4.2. Does this user have the correct privilege?

Test	Full Name, BLUR, CENSOR Candidate Number: N/A	Purpose of Test? Centre Number: N/A	How can we test it?	Expected Outcome?	Actual Outcome?	Evidence
1	Does the server have all repositories up-to-date with to their latest long-term-support versions?	To ensure that the server is not running or using any modules that could cause potential performance problems to the server.	CentOS Command: yum update	All the packages should be up-to-date, there should not be any other repositories left to update or install.	As expected.	[root@developer ~]# yum update Loaded plugins: fastestmirror Loading mirror speeds from cached hostfile * base: linux.cs.uu.nl * extras: mirror.cj2.nl * updates: mirror.amsiohosting.net No packages marked for update [root@developer ~]#
2	Has the server successfully installed NodeJS and MongoDB onto the server?	To ensure that the application's code will execute and if the database system has been setup correctly.	CentOS Commands: node -v mongodb --version	A console response from the node and mongo to indicate their versions showing they are installed on the system.	As expected.	 root@developer:~ [root@developer ~]# node -v v6.9.5 [root@developer ~]# mongod --version db version v2.6.12 2017-02-22T07:22:04.833+0000 git version: d73c92b1c85703828b55c2916a5dd4ad46535f 6a
3	Is the webserver running Express on port 3000?	To ensure that the web-server is running correctly and listening on the port instructed	While running NPM access the server's IPV4 address via a HTTP GET request	A response confirming the GET request from Node & the express landing page being displayed on the	As expected.	> notchess@0.0.0 start /root/Chess > node ./bin/www GET / 200 468.910 ms - 170 GET /stylesheets/style.css 200 13.401 ms - 111 GET /favicon.ico 404 48.648 ms - 925 [redacted]

			requesting browser.		
Has the 'main' database with the collections 'users' & 'games' been successfully created? 4.1	To check if the database system is working and if the correct collections have been created.	Mongo Commands in order: use Main show collections	The database main will show collections 'games' and 'users'	Expected output.	<pre>> use Main switched to db Main > show collections games system.indexes users > █</pre>

Can the database user be authenticated to the 'Main' database with read and write privileges? 4.2	To check if the database user can connect and be authenticated to the correct database with the correct permissions	<p>Connect to the database 'main' using Mongo Management Studio and attempt to read and write into one of the collections inside of the database.</p> <p>Create a new document and then see the document can be viewed.</p>	A new document will be created and when refreshed that document can be viewed.	Expected.	HIDDEN <p>Main.games Main.users</p> <p>Run ► F S C Add document</p> <p>1</p> <p>1 of 1 documents successfully deleted.</p>
--	---	---	--	-----------	---

Full Name: BLUR, CENSOR

Candidate Number: N/A

Centre Number: N/A

Section 4.2: Iteration 2 Testing

The second iteration of my development designed the applications registration and authentication systems. Since there is a vast amount of data handling and process in this iteration, testing normal, boundary and erroneous data is vital to discover potential bugs with the system.

Although I am more than aware that there are some genuine problems that I could reveal with my system due to the environment I am currently developing on, i.e. the application could potentially be susceptible to a man-in-the-middle attack this is far beyond the scope of A Level and I will not be testing this element of the application. In addition, I have provided a solution for when the application is deployed.

The testing in this iteration will mainly be black box focused testing. The registration and login system is designed to perform a specific task and the outputs are more than predictable so I have chosen to direct my testing around black box.

Elements of the application to test

1. Real-time error messages on the registration form
2. User boundary data
 - 2.1. Can users register with a first name > 35 characters?**
 - 2.2. Can users register with a last name > 35 characters?**
 - 2.3. Can users register with an invalid email address?**
 - 2.4. Can users register with an email address that is > 64 characters?**
 - 2.5. Can users register with a password that is between 8 and 32 characters?**
 - 2.6. Can users register to the application if they are younger than 13?**
3. NoSQL Queries
 - 3.1. Are the users being inserted into the users' collection after signing up?**
 - 3.2. Are the users having persistent session keys being stored in the sessions collection?**
 - 3.3. Are passwords being hashed using bcrypt before they are imported into the database?**
 - 3.4. Are users allowed to register with an email address that is already being used by another user?**
4. Do users have persistent login sessions even if the webserver is restarted?

Full Name: BLUR, CENSOR

Candidate Number: N/A

Centre Number: N/A

Testing Plan

Test Number	What needs to be tested?	Why does it need to be tested?	How can we test it?	Should we complete multiple tests with (normal, boundary and erroneous data) Is there an expected output from the system?
1	When registering to the application, do users get the appropriate real-time error messages when entering boundary data to the fields?	To check if the application handles error messages in real-time with the correct conditional arguments.	<ol style="list-style-type: none">Enter an email address that is invalid and one that is > 64 characters.Enter a first and last name that are both > 35 charactersEnter a password that is not between 8 and 32 charactersEnter a date of birth that makes the user younger than 13	Whitebox Boundary Test: The system should output the appropriate error messages if the user tries to enter any of the boundary test values stated. These error messages will appear as soon as the user types an invalid value e.g. If a user types: ol as an email address, the error would pop: <i>"You have entered an invalid email address"</i>
2	Can a user submit a registration form even if their data is invalid?	To check if the backend of the application handles incorrect data correctly. Although users are restricted with text entries, users could still try to submit invalid	Using the same test data from number one, modify the HTML of the web page using inspect element to modify the fields to send an invalid request.	Unit / Erroneous: The request should be sent to the server but appropriate error messages relating to the request should be returned to the user.

		registration requests to the server.		
3	After a registration request is submitted, is the user object inserted into the user's collection?	To check if the user schema and database queries are correct.	Submit a registration request with appropriate data	Blackbox-Based Test: We should expect for the object passed to the passport user routine to be inserted into the user's collection
4	Do users have persistent login sessions after they login to the application?	To check if the authentication system works and the persistent login feature.	<ol style="list-style-type: none"> 1. Login to the application using a registered user's information. 2. Restart the server (CTRL + C, npm start) 3. Return to the home page to check if the user is still logged into the application 	Unit Test: We should expect for the user to be authenticated to the application even after the server has been restarted alongside with his session object being stored in the sessions collection.
5	Can the system differentiate between an unauthenticated user and an authenticated user?	To check if user privileges are being assigned correctly if a user is logged or not logged into the application.	When logged in, try to navigate to the register page and try to navigate to the home page when unauthenticated.	Unit Test: The authenticated user should be redirected from the register page and the unauthenticated user should be redirected to the login page when trying to navigate to the home page
6	Can a user log out of the application?	To check if a user can kill their local and server side sessions.	When logged in navigate to the route /logout and check if the user becomes unauthenticated. Then, check the database to see if the session object remains in the sessions collection	Unit Test: The session should be destroyed from the client's web browser and the server should remove the session in the sessions collection.

--	--	--	--	--

Full Name: BLUR, CENSOR

Candidate Number: N/A

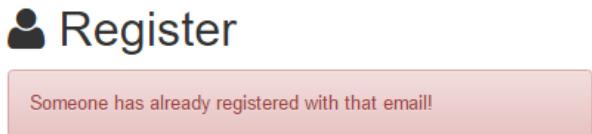
Centre Number: N/A

Testing

Test Number	Test Data	Output	Passed / Failed?
1.0	Password: b Remaining values found in picture	<p> Register</p> <p>Email *</p> <p><input type="text" value="emailAddress"/></p> <p>Please enter a valid email address</p> <p>Password *</p> <p><input type="password" value="*"/></p> <p>Please enter a password that is between 8 and 32 characters</p> <p>First Name *</p> <p><input type="text" value="superlongstringsuperlongstringsuper"/></p> <p>Last Name *</p> <p><input type="text" value="superlongstringsuperlongstringsuper"/></p> <p>Birthday *</p> <p><input type="text" value="1"/> <input type="text" value="Jan"/> <input type="text" value="2017"/></p> <p>You must be at least 13 to use our application</p> <p><input type="button" value="Signup"/></p>	Passed
1.1	Modifying the maxlength of the text fields to allow for erroneous data input. The letter "J" repeated.	<p> Register</p> <p>Email *</p> <p><input type="text" value="aaaaaaaaaaaa"/></p> <p>Please enter a valid email address</p> <p>Password *</p> <p><input type="password" value="....."/></p> <p>Please enter a password that is between 8 and 32 characters</p> <p>First Name *</p> <p><input type="text" value="oooooooooooo"/></p> <p>Your first name can not be longer than 35 characters!</p> <p>Last Name *</p> <p><input type="text" value="oooooooooooo"/></p> <p>Your last name can not be longer than 35 characters!</p> <p>Birthday *</p> <p><input type="text" value="1"/> <input type="text" value="Jan"/> <input type="text" value="2017"/></p> <p>You must be at least 13 to use our application</p> <p><input type="button" value="Signup"/></p>	Passed
1.2	An email address that already exists in the user's collection.	<p> Register</p> <p>Email *</p> <p><input type="text" value="test@test.com"/></p> <p>Another user has already signed up with that email!</p>	Passed

Test Number 1

Test Number 2

Test Number	Test data	Output	Passed/Failed?
2.0	Trying to submit a request with one of the text fields containing invalid data.	The request is rejected.	Passed
2.1	Submitting a field with an invalid email address after modifying the code to allow for the request to be sent to the server.	 <p>Someone has already registered with that email!</p> <p>Email *</p>	Passed

Test Number 3

Test number	Test data	Output	Passed / Failed?
3.0	Email: Test@test.com Password: test First & last name: test DOB: 28/04/1990	<pre>{ "_id": ObjectId('58cda8b12e93f16808c19a9e'), "local": { "email": "test@test.com", "password": "\$2a\$10\$Ol3GuKBN7nrz7538TEkIDOHaWX9bi2GzpBdpM4jDvjn/749tqceFi", "firstName": "test", "lastName": "test", "birthday": { "date": "28", "month": "4", "year": "1990" } } }</pre>	Passed

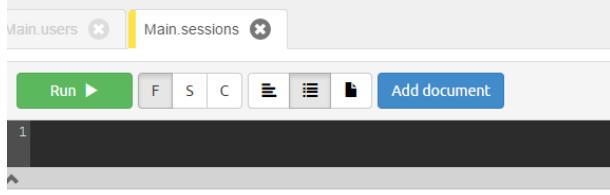
Test Number 4

Test number	Test data	Output	Passed / Failed?
4.0	Login using the test account	User is now logged into the application and the session has been created in the database.	Passed

Test Number 5

Test number	Test data	Output	Passed / Failed?
5.0	Navigate to the home page when unauthenticated. Navigate to the registration page when authenticated.	The unauthenticated user is redirected to the home page. The authenticated user can view the register page but cannot submit a registration form because he is currently signed into an account.	Passed with good handling to erroneous request

Test Number 6

Test number	Test data	Output	Passed / Failed?
6.0	Logout after signing into the test account	The user was logged out and the session was destroyed locally and from the database. 	Passed

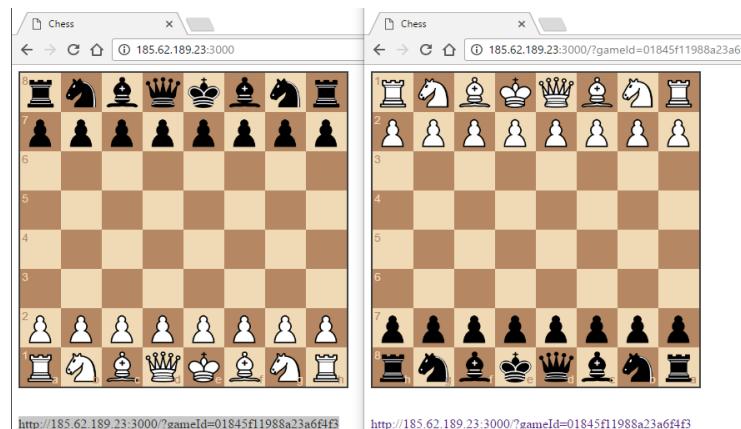
		<p> Login</p> <p>Email <input type="text"/></p> <p>Password <input type="password"/></p> <p><input type="button" value="Login"/></p>	
--	--	---	--

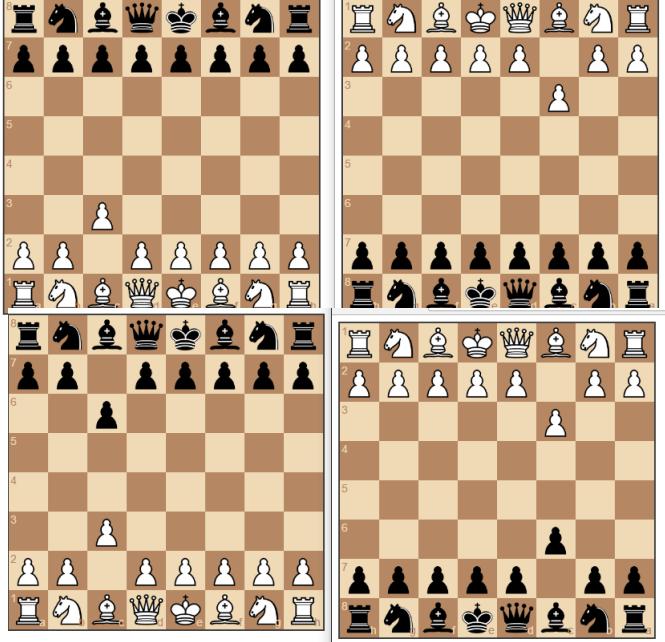
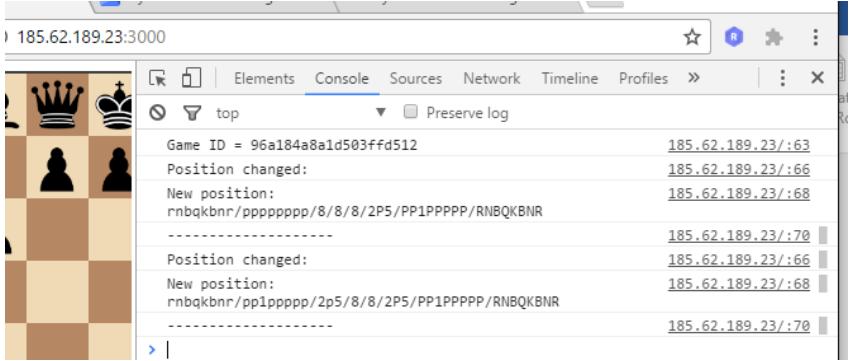
Full Name: BLUR, CENSOR
Candidate Number: N/A
Centre Number: N/A

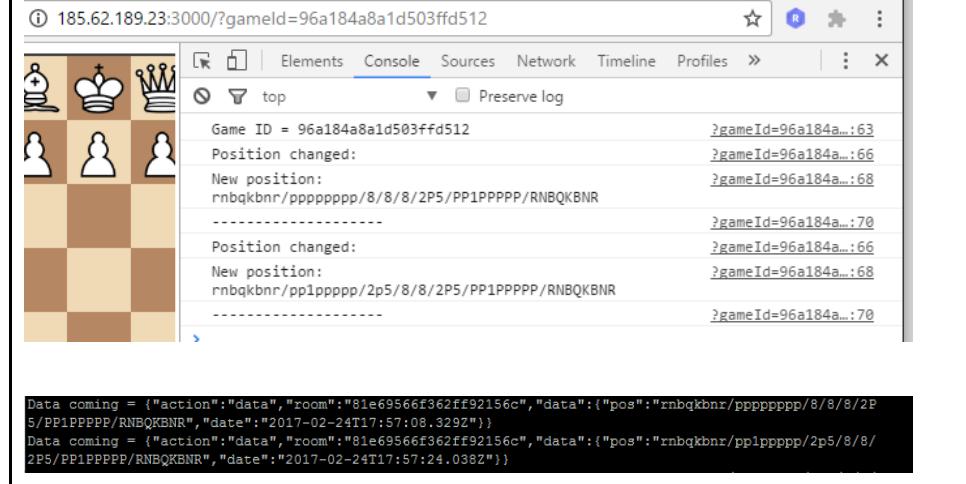
Section 4.3 Iteration 3 Testing

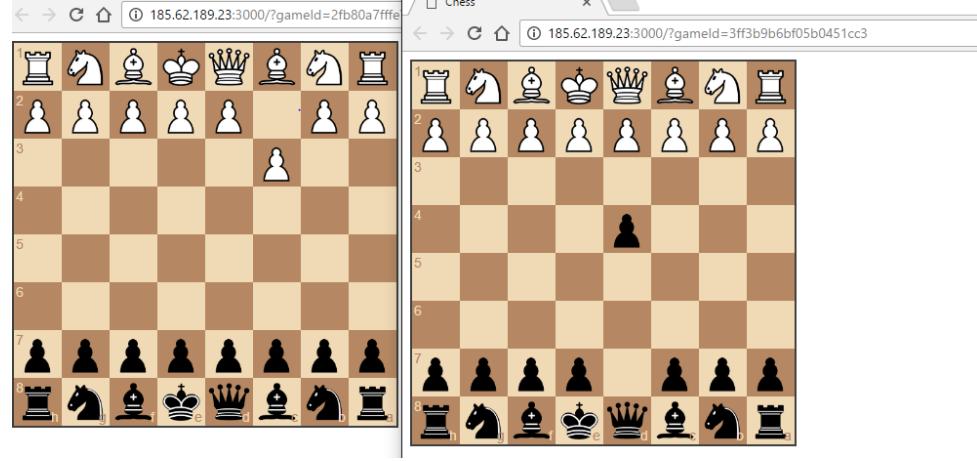
This iteration was designed to implement the real-time multiplayer methodology into the application. It was designed as its own unique project so that it can later be implemented with Chess logic and into the game itself. The model for handling real-time multiplayer works by opening a WebSocket from both clients to the server so that on the event of a recognized action being performed by the user on the board, the server will be notified and update the opponent's game appropriately.

Testing Table

Test	Purpose of Test?	How can we test it?	Expected Outcome?	Actual Outcome?	Evidence
Are both players of the board given their correct perspective of the board when joining a new game?	To check whether the two players are being identified uniquely and are on the correct side of the board.	Instantiate a new game by visiting the static landing page and open the uniquely generated game URL in another tab and compare the board's perspective.	The first player should see the board from white's perspective whereas the second player should see the board from black's perspective.	As expected. The left board represents the first player and the right board represents the second player.	 http://185.62.189.23:3000/?gameId=01845f11988a23a6f4f3 http://185.62.189.23:3000/?gameId=01845f11988a23a6f4f3

<p>Can both players that are in the same game make moves that update on both of their boards in real time?</p>	<p>Has the web-socket been successfully opened between the client and the server and if so, are the actions that a player makes being recognised and rendered correctly.</p>	<p>Create a new game and open the game on two different tabs in an Internet Browser. Move a chess piece on both boards.</p>	<p>When white moves a piece on the board then the move will be rendered across both boards in the same game. Vice versa for the black side.</p>	<p>As expected. The first picture represents the white player moving a piece and the second picture represents the black player moving a piece.</p>	
<p>Are both players and the server provided with the Forsyth Edwards Notation of the board after a change to the move has occurred?</p>	<p>To check whether the server and both clients are being provided with the correct FEN strings to represent the board and if the moves are unique to the game itself.</p>	<p>Same as before.</p>	<p>After a move, has been made by either of the two players, the FEN string should appear in the console and on the server.</p>	<p>The FEN strings of the current board state will be displayed in both user's consoles and logged to the server.</p>	 <pre> 185.62.189.23:3000 Game ID = 96a184a8a1d503ffd512 Position changed: New position: rnbqkbnr/pppppppp/8/8/2P5/PP1PPPPP/RNBQKBNR ----- Position changed: New position: rnbqkbnr/pp1ppppp/2p5/8/8/2P5/PP1PPPPP/RNBQKBNR -----</pre>

					 <p>The screenshot shows a browser window with developer tools open. The console tab is selected, displaying a log of chess moves. The log includes messages like 'Game ID = 96a184a8a1d503ffd512', 'Position changed:', 'New position: rnbqbnr/ppppppp/8/8/2P5/PP1PPPPP/RNBQBNR', and 'Data coming' with JSON objects representing game data. A chessboard icon is also visible.</p>
Do the sockets reconnect if there is an interruption to the connection between the client and the server?	Will connections be established if there is a drop-in connection between the two devices or will the game be able to be continued?	In a game, close one of the tabs during the game to close the WebSocket and then reopen the same game on another tab.	The player who closed his connection will be able to continue playing the game as normal when the re-joins the same join. This will be recorded in the server receiving a connection.	The black player can leave the game and reconnect as normal however, the white player cannot because there is no condition that specifies the colour of the player. (This can be fixed with the implementation of '/white' and '/black' to	<pre data-bbox="1190 714 2156 921">Data coming = {"action": "join", "room": "ff4bad31c963595db634"} You joined room : ff4bad31c963595db634 {"551e1a0904bf3b57d3c8": "rnbqbnr/pp1pppp/3p4/8/8/2PP4/PP2PPPP/RNB PPPPP/RNBQBNR"} Conn disconnected. Creating Connections Data coming = {"action": "join", "room": "d945ac5af1d633e35a9d"}</pre>

				specify the colour of the player.	
Can multiple games be played and still work correctly in real-time?	To check whether the system is scalable to handle games simultaneously	Create two games and join both games and make moves on both boards to see if they update in real time	The two games should function as the first example.	As expected. Two different games being played simultaneously from black's perspective.	

Section 4.4 Iteration 4 Testing

In this Iteration, I developed, designed, and implemented the option for users to play against a computer. This was quite a large implementation since alongside implementing the option for users to play against a computer, the Chess logic was also implemented. Thus, the only moves that can now occur on a board are the legal moves. I decided that implementing the legality of the game would work well by implementing it first with a computer (the computer will be able to easily put me into different game states such as checkmate, check and stalemate).

Elements of the application to test

1. Computer customization
 - 1.1. Can users select the difficulty of the Computer they wish to play against?**
 - 1.2. Will the system set the difficulty of the Computer to 10 if the user tries to play against an invalid Computer level?**
2. Can the user only make legal moves on the board?
3. Can the user perform special moves such as castling?
4. Does the board display the correct error messages depending on the status of the game?
5. Does the Computer provide information to the user about its thinking?
6. Can the user lose, win, or draw the game by being placed into the correct states?

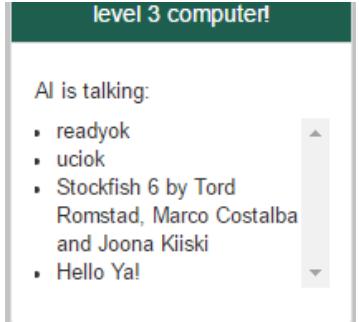
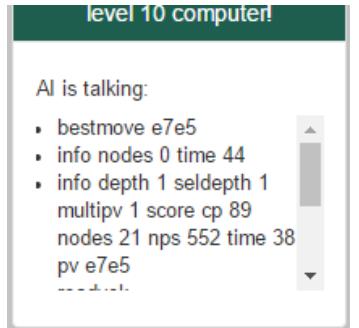
It's important to note that the game of chess is vast and the number of potential moves that can occur in a game is very large. Thus, I will not be testing all the elements of the application and will be testing general application functionality to conclude if the application is working correctly, i.e. I will not be testing every single legal move in every single position on the board.

Test Number	What needs to be tested?	Why does it need to be tested?	How can we test it?	Normal / Boundary / Erroneous Data? Expected output?
1.1	Can the users select the difficulty of the Computer they wish to play against?	Users should be allowed to select the difficulty of their artificial opponent so that they can train against a computer levelled to their standards.	On the home page, select a difficulty level on the slider between 1 & 20.	Normal / Boundary / Erroneous Data should all be used here. Expected output: depends on the input to the subroutine, the computer level should be assigned as provided by the user unless the input is a boundary.
1.2	Will the system set the difficulty of the Computer to 10 if the user tries to play against a Computer level greater than 20?	Error handling	Enter a number that is greater than 20 into a game request.	Boundary Data Expected output: The game should automatically set the level of the Computer to level 10.
2	Can the user only make legal moves on the board?	Users should only be allowed to make legal Chess moves on the board and should only be allowed to move pieces if they are his pieces.	<ol style="list-style-type: none"> 1. Try to make an illegal move on the board. 2. Try to move a piece on the board that is not the orientation of the player. 3. Try to move a piece on the board when the game has ended 	Normal Data Expected output: To work as the game of chess

3	Can the user perform special moves on the board?	Users should be allowed to perform special moves on the board such as Castling, Promotion & En Passant.	Try to make a special move on the board.	Normal Data
4	Does the board status display the correct messages depending on the status of the game?	The system should be able to identify different statuses of the game to inform the users of the current state of the game.	Play a game of chess against the Computer and record the error messages	Normal Data Expected messages: Your turn Opponent's turn Check You are in check!
5	Does the Computer provide information about its thinking?	To check if the chess engine is working correctly, exploring the correct nodes and difficulty levels are working correctly.	Play a game of chess against a computer and record the messages outputted by the engine	Normal Data
6	Can a user lose, draw, or win the game depending on the current position of the game?	To check if the board logic and the board are integrating together correctly and if the system can recognise a game that has ended	Lose a game of chess to the Computer	Normal Data Expected: Alert informing the user that he has lost the game.

Testing

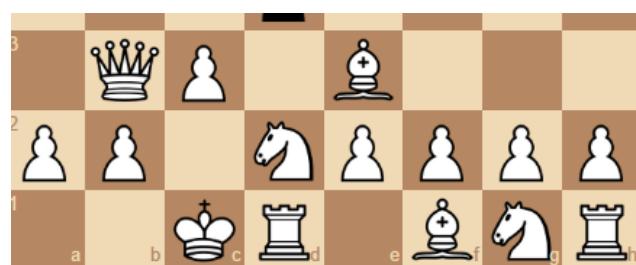
Test Number 1

Test Number	Test Data	Output	Passed/Failed?
1.1	Computer Level 3 (Normal Data)	 level 3 computer! AI is talking: <ul style="list-style-type: none">▪ readyok▪ uciok▪ Stockfish 6 by Tord Romstad, Marco Costalba and Joona Kiiski▪ Hello Ya!	Passed
1.1	String: testRandom (Erroneous Data)	 level 10 computer! AI is talking: <ul style="list-style-type: none">▪ bestmove e7e5▪ info nodes 0 time 44▪ info depth 1 seldepth 1 multipv 1 score cp 89 nodes 21 nps 552 time 38 pv e7e5	Passed
1.2	Boundary Data > 20 &< 1	User is forced to play against a level 10 Computer again.	Passed

Test Number 2

Test Number	Test Data	Output	Passed / Failed?
1.1 Try to make a legal move with a pawn on the board (moving two squares forward)	Move white pawn to E4		Passed
1.2 Try to make an illegal move with the horse	Move the horse to B3	Piece was snapped back to its original location on the board.	Passed
1.3 Try to take the pawn on D5 with the pawn on E4	Take the pawn on D5		Passed (Pawn was captured by opponents' queen)

Test Number 3

Test Number	Test Data	Output	Passed/Failed?
3.1 Can the user perform the special move castling on the board?	Try to Castle with the rook on H1.		Passed
3.2 Can the user castle with the rook on A1?	Try to castle with the rook on A1.		Passed

Test Number 4

Test Number	Data	Output	Passed / Failed?
4.1 Does the game know when it is a player turn?	Before making a move on the board take a screenshot and after making a move on the board, take a screenshot.		Passed
4.2 Does the game know when a player is in check?	Let the Computer place me into check		Passed
4.3 Does the game know when a player has lost?	Let the Computer checkmate me via fool's mate.	Yes, an alert was popped up indicating I had lost the game.	Passed

4.4

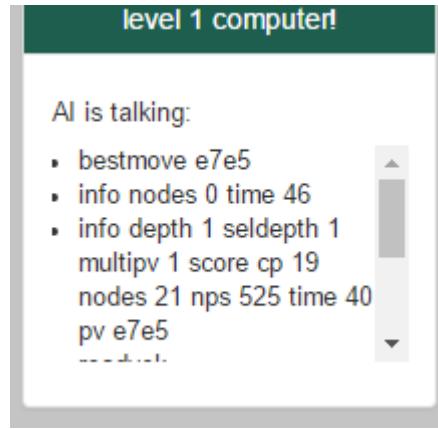
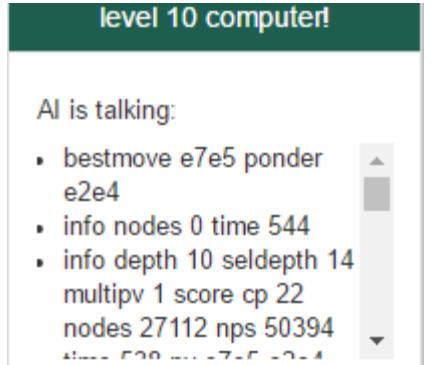
Does the game know
when a game has
ended?

Let the Computer checkmate me via
fool's mate.



Passed

Test Number 5

Test Number	Test Data	Output	Passed / Failed?
5.1 Does the engine provide details about its thinking in coming up with a move on the board?	Let the engine make a move.	 <pre> level 1 computer! AI is talking: • bestmove e7e5 • info nodes 0 time 46 • info depth 1 seldepth 1 multipv 1 score cp 19 nodes 21 nps 525 time 40 pv e7e5 ... </pre>	Passed
5.2 Does the chess engine explore more possible moves if the difficulty level is increased?	Level 1 Computer make a move. Level 10 computer make a move.	<p>Level 1: See picture above</p>  <pre> level 10 computer! AI is talking: • bestmove e7e5 ponder e2e4 • info nodes 0 time 544 • info depth 10 seldepth 14 multipv 1 score cp 22 nodes 27112 nps 50394 ... </pre>	Passed (depth of 10 explored)

Section 4.5: Iteration 5 – Full System Testing

The fifth and final iteration of my project was by far the largest part of my agile development process. In this iteration, all the previously designed elements were merged together along with additional aspects of the application being implemented, such as the ranking system, game scores, leaderboard and more. Since the proposed application has now been developed, the testing at this stage will be the most rigorous. To solve this vast amount of testing, I will break down the components of my project so that it is not only more manageable for myself but so that I can test specific modules in my application and the full system.

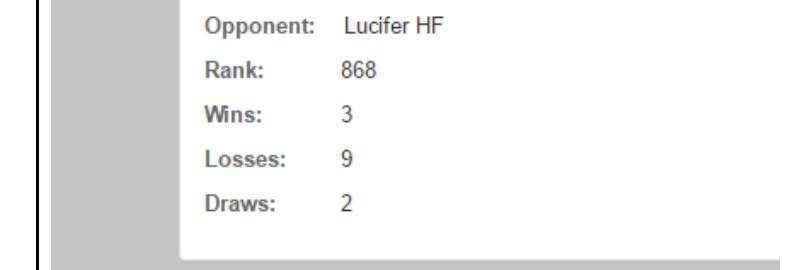
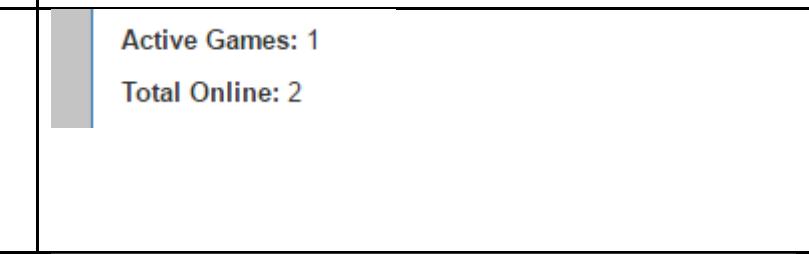
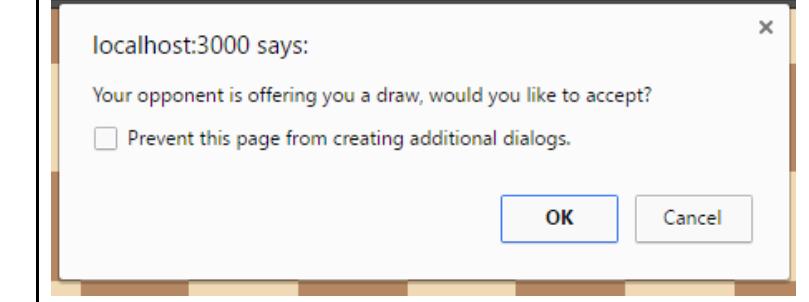
I will break down my testing into the following components:

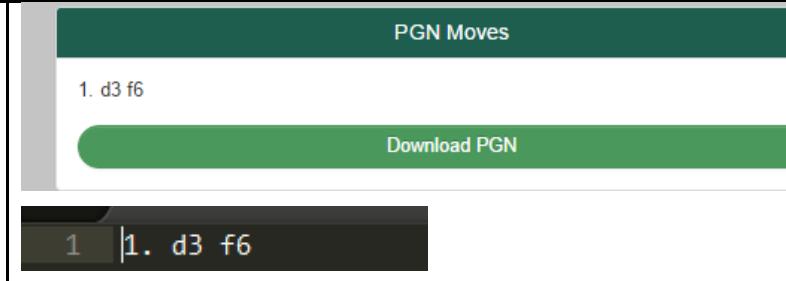
1. Chess
 - 1.1. Solo queue (matchmaking vs a random player)**
 - 1.2. Playing versus a friend**
 - 1.3. Playing versus a computer**
2. Leaderboard
 - 2.1. Glicko2**
3. Navigation
 - 3.1. Form navigation**

Chess Testing

Solo Queue

Test	Purpose of Test	How can it be tested?	Expected Outcome	Actual Outcome	Evidence?
When a user joins the lobby queue, does the number of total player's online increase by one?	To check if the application can identify when a user is in queue.	Pressing the "play solo" button when a user is authenticated. Normal data	The number of total players displayed on the homepage should increment by one.	As expected.	 Active Games: 0 Total Online: 1
If a user tries to join the lobby queue twice on the same account, does the number of total players online increase again?	To check if the system stops users from joining a lobby more than once on the same account.	While already in queue, join the queue again by pressing the play solo button on the home page.	The number of total players online should remain the same	As expected.	As before.

If two different players join the lobby, do they get matched up and redirected into a game against each other?	To check if match making works correctly.	Join the lobby queue on two different accounts.	Players should be matched up playing a random orientation on the board. Details of their opponent should be displayed.	As expected.	
When a game has begun between two players, does the active game count number increase by one?	To check if the counter is working correctly.	Same as before	Total number of games online should increase by one	As expected.	
Decline and accept a draw request	To see if the function is working correctly and if the draw is accepted, is a user's game score for draws increased correctly?	Ask to end a player versus player game as a drawer. First decline the offer then accept the offer	Game should end. Active games should decrease by one, total online should decrease one the players leave the page. Both players draw scores increment by one. If declined, nothing happens. Game is continued.	As expected.	 localhost:3000 says: Your opponent is offering you a draw, would you like to accept? <input type="checkbox"/> Prevent this page from creating additional dialogs. OK Cancel Active Games - 1 Number of draws for each player +1

Resigning from a game	Does the resign function work and are game scores being updated correctly for the players in the database?	In a player versus player game, select the resign from game button.	User who requested to resign from the game should lose the game. Active games counter should decrease by one, the opponent will gain +1 wins & requester will get -1 losses	As expected.	
After a user makes a move on the board, does the PGN moves box appear with the PGN moves of the board?	To check if the user can download the PGN moves of the board and if they are displayed correctly on the website.	In a player versus player game, make a move and try to download the PGN for the game.	.TXT file of the PGN will be downloaded. TXT file should be formatted with each move line by line.	As expected.	
Does a user get resigned from a game if they try to exit?	To check if games that are abandoned by users still get ended.	In a player versus player game, close the website for the game by closing the internet browser.	A pop up message should appear to the user confirming that they would like to leave the website. If they continue, they will be resigned from the game.	As expected.	<p>User who closes the game loses, his number of losses get -1.</p> <p>Opponent wins the game and his number of wins get +1.</p>

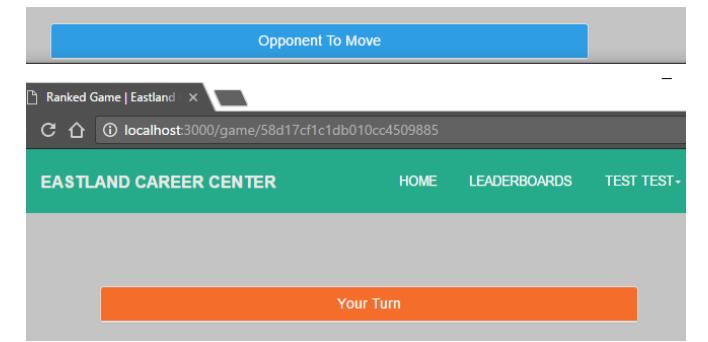
When a game ends, does the PGN string for the game get stored inside of the `games` collection	To check if the database query is working correctly and if the appropriate data for each game is being stored.	Play and finish a game of Chess.	The database should contain information about the game such as the creation time, end time, end reason, PGN string and the winners of the game.	As expected.	<pre>{ "_id": ObjectId("58d15a1f1239d60eb93d3c21"), "status": "gameover", "winner": "58d15a5d1239d60eb93d3c22", "settled": false, "createTime": ISODate('2017-03-21T16:51:43.802Z'), "updateTime": ISODate('2017-03-21T16:58:17.327Z'), "players": ["58d0176e0c8de905d064697a", "58d15a5d1239d60eb93d3c22"], "__v": 1, "pgn": "1. e4 c6", "endReason": "close_resign" }</pre>
Can a user join the matchmaking queue if he is already in a game?	To check if users can play multiple matchmaking games at the same time.	Join the queue whilst the user is already in a game, having another player in queue.	Player should not be allowed in game and active players remains the same.	As expected.	
When a game ends, can the user re-join the game?	To check if games are destroyed once they end	Try to join a game using the game identifier of a game which has already ended.	User should be redirected to the game has ended page	As expected.	Game Over!

<p>Can a user place his opponent into check?</p>	<p>To check if the appropriate error messages are being displayed and the chess logic is working correctly</p>	<p>In a match making game, play the following moves:</p> <ol style="list-style-type: none"> 1. C3 D6 2. QA4+ 	<p>White player should be notified he has placed his opponent into check.</p>	<p>As expected.</p>		
<p>Can a user win a game by checkmate?</p>	<p>To check if the game logic is working correctly and if game scores are updating when users win by logic.</p>	<p>In a player versus player match making game, play:</p> <ol style="list-style-type: none"> 1. C3 C5 2. D3 B5 3. C4 F5 4. E4 G5 5. QH5# 	<p>White player should get +1 wins and his opponent +1 losses.</p>	<p>As expected.</p>		

Can a user draw a game by stalemate?	To check if special moves work in player versus player games.	In a player versus player match making game, play: 1. D4 E5 2. QD3 G5 3. B4 A5 4. BxA5 RA6 5. D5 RB6 6. A6 C6 7. A7 C5 8. A8=Q	The pawn should be promoted to a queen. As expected.	
--------------------------------------	---	--	---	---

Playing versus a friend

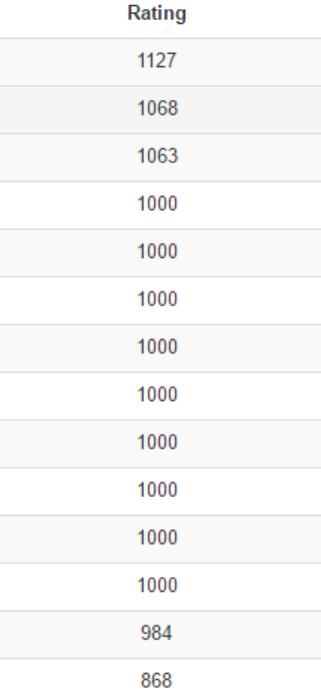
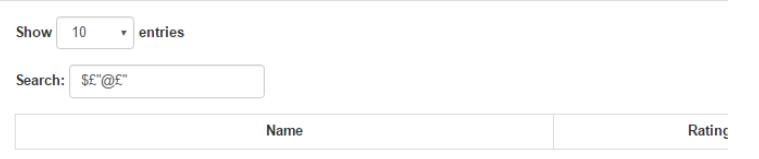
Although the playing with a random player on the system runs the Chess system using the same logic as playing with a friend, tests should still be performed to ensure that the system has been setup correctly.

Test	Purpose of test	How can it be tested?	Expected Outcome	Actual Outcome	Evidence?
Can a player join his own unique game link to start the game?	To check if the system can properly identify the users connecting on the system. This user should not be allowed to join his own game.	Create a play versus friend game by selecting the option on the home page.	Nothing on the page should change, the user should not be allowed to join the game.	As expected.	
Does the game begin when a different user joins the game?	To check if the play versus friend game mode is working correctly.	Join the link provided to the first user in the game on a different user account.	The game should begin and the appropriate response messages should be shown. Now, the white player should be able to move pieces on the board.	As expected.	

Does the Chess logic still work in the play vs friend game mode?	To check if the Chess logic is still working correctly in the player versus friend game mode	<p>Try to get the board into the following states in a player versus friend game:</p> <ol style="list-style-type: none"> 1. Check 2. Checkmate 3. Draw 4. Resign 5. Draw 	The game should output the same results that were found in the player versus random player match making system.	As expected.	<p>Screenshots are not provided for this as it would be a waste of space. System performed the same as found in the player versus random player game mode.</p> <p>Games are also inserted into the database.</p>
Does a user get redirected to the game page he was trying to access if he was unauthenticated?	To check if unauthenticated users get redirected to the correct page while not being logged in.	<p>Join a friend's game being unauthenticated. Login to the application and see if redirected to the game</p>	<p>The user should be redirected to the login page.</p> <p>After the user logs in, they should be redirected to the game that they were trying to join previously.</p>	As expected.	<ol style="list-style-type: none"> 1. Redirected to /login 2. Logged in 3. Redirected to game previously requested

Leaderboards Testing

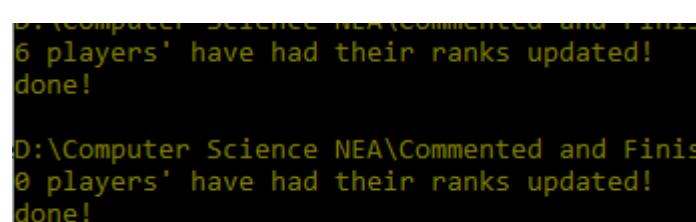
Test	Purpose of test	How can it be tested?	Expected Outcome	Actual Outcome	Evidence?
Does the leaderboard display all the players in the database?	To check if the database query is working correctly.	Load the leaderboard page and compare the users with the users inside of the database.	All the users in the database should be displayed on the leaderboard.	As expected, passed.	 <p>Showing 1 to 14 of 14 entries</p>
Does the leaderboard allow for users to be searched?	Is the search algorithm and database query working correctly when searching a user in the database	Search a user in the database.	Information about the searched user should appear	As expected, passed.	CENSOR
Are the leaderboard updating in real-time?	To check if the database is updating properly when scores about users change	Resign from a player versus player game.	The resigning player's losses should increase by one when vesting the leaderboard.	As expected	CENSOR BLUR: Losses + 1

Are the leaderboards displaying user's rankings in the correct order?	To check if the leaderboard is displaying information in the correct order	Check if player ratings are decreasing from #1 to the last entry	The #1 player should have the highest rating while the last player should have the lowest rating	As expected	 <table border="1"> <thead> <tr> <th>Rating</th> </tr> </thead> <tbody> <tr><td>1127</td></tr> <tr><td>1068</td></tr> <tr><td>1063</td></tr> <tr><td>1000</td></tr> <tr><td>984</td></tr> <tr><td>868</td></tr> </tbody> </table>	Rating	1127	1068	1063	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	984	868						
Rating																											
1127																											
1068																											
1063																											
1000																											
1000																											
1000																											
1000																											
1000																											
1000																											
1000																											
1000																											
1000																											
1000																											
984																											
868																											
Can users search up names that do not exist in the database or special characters?	To check if the system can handle special characters, invalid requests and paginate correctly.	Search up a random string of special characters such as \$£"@£"	The request should be successful however, there should be no users returned from the database. The leaderboard should also paginate the number of entries to 10 as there were less than 10 entries discovered.	As expected.	 <p>Show 10 entries</p> <p>Search: \$£"@£"</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Rating</th> </tr> </thead> <tbody> <tr><td>Player 1</td><td>1000</td></tr> <tr><td>Player 2</td><td>1000</td></tr> <tr><td>Player 3</td><td>1000</td></tr> <tr><td>Player 4</td><td>1000</td></tr> <tr><td>Player 5</td><td>1000</td></tr> <tr><td>Player 6</td><td>1000</td></tr> <tr><td>Player 7</td><td>1000</td></tr> <tr><td>Player 8</td><td>1000</td></tr> <tr><td>Player 9</td><td>1000</td></tr> <tr><td>Player 10</td><td>1000</td></tr> </tbody> </table>	Name	Rating	Player 1	1000	Player 2	1000	Player 3	1000	Player 4	1000	Player 5	1000	Player 6	1000	Player 7	1000	Player 8	1000	Player 9	1000	Player 10	1000
Name	Rating																										
Player 1	1000																										
Player 2	1000																										
Player 3	1000																										
Player 4	1000																										
Player 5	1000																										
Player 6	1000																										
Player 7	1000																										
Player 8	1000																										
Player 9	1000																										
Player 10	1000																										

Can an unauthenticated user try to make a request to the user page?	To see if the leaderboard page is properly handling unregistered users	Try to access the leaderboard and search user page while unauthenticated	The user should be redirected to the login page	As expected.	Redirected to the login page after trying to access /search & /leaderboards
--	--	--	---	--------------	---

Glicko2

Test	Purpose of Test	How can it be tested?	Expected Outcome?	Actual Outcome?	Evidence?												
Are user's rankings updated when the script is run?	To check if user's rankings are being updated the database when the script is run	After completing a player versus player game, run the script called update_rankings with node inside of the cronjob folder	The players should have their rankings updated.	The players had their rankings updated.	<pre>D:\Computer Science NEA\Commented and Finished\cronjob>node update_rankings.js 6 players' have had their ranks updated! done!</pre> <pre>D:\Computer Science NEA\Commented and Finished\cronjob></pre>												
Are games that have been using to update rankings modified in the database to unsettled?	To check if games are not going to be used more than once that have already	Check the games collection after running the update_rankings script to see if the games that were previously unsettled have been set to settled.	The previous games used in a ranking update should be set to settled.	As expected.	<table border="1"> <thead> <tr> <th>_id</th> <th>status</th> <th>winner</th> <th>settled</th> </tr> </thead> <tbody> <tr> <td>ObjectId("58c8699f5f1169375cf3a69f")</td> <td>"created"</td> <td>NULL</td> <td>true 03/14/2</td> </tr> <tr> <td>ObjectId("58c872a062812d21b0bb78e2")</td> <td>"created"</td> <td>NULL</td> <td>true 03/14/2</td> </tr> </tbody> </table>	_id	status	winner	settled	ObjectId("58c8699f5f1169375cf3a69f")	"created"	NULL	true 03/14/2	ObjectId("58c872a062812d21b0bb78e2")	"created"	NULL	true 03/14/2
_id	status	winner	settled														
ObjectId("58c8699f5f1169375cf3a69f")	"created"	NULL	true 03/14/2														
ObjectId("58c872a062812d21b0bb78e2")	"created"	NULL	true 03/14/2														

	been used in a ranking update	The update_rankings.js script can also be run again to check defensively.	The script should not update any player rankings if ran again.		
Are the rankings that are pushed to the database then pushed onto the leaderboard?	To check if the leaderboards is correctly updating after a ranking update.	<p>Check the players on the leaderboards and note them down.</p> <p>Perform a ranking update after completing a game and see if the leaderboard update correctly.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. A player should have his ranking increased if he wins a game 2. A player ranking should remain the same if he draws a game 3. A player ranking should decrease if he loses a game 	Players rankings should update accordingly to games played.	Player 1 had a ranking increase after beating player 2, who had a loss in his ranking.	Passed test.
Are the game results used by	Are the game results	Run the update ranking after a player:	The winner of a game	As expected.	Would require a lot of screenshots. System was also tested against players who had lower ratings than a player with a higher rating.

Glicko-2 accurate?	being passed to Glicko correctly? If a game is won, are rankings updated depending on this fact? (Vice versa for losses & draws)	1. Wins a game 2. Loses a game 3. Draws a game In the player versus player mode	should have his ranking increased. The loses of a game should have his ranking decreased. Games that end as a draw should remain the same.		The player with the lower rating gained a larger increase to their rating by beating the higher rated player than the player who was higher beating the lower rated player.
Can Glikco-2 update multiple player's rankings at the same time?	To check if the system can handle updating multiple users (more than 2) at the same time.	Update the rankings after multiple games have been played on multiple accounts.	Multiple accounts rankings should be updated.	As expected.	<pre>D:\Computer Science NEA\Commented and Finished\cronjob>node update_rankings.js 6 players' have had their ranks updated! done! D:\Computer Science NEA\Commented and Finished\cronjob></pre>

Navigation Testing

In the navigation testing, I will be testing whether the navigation throughout the site is working correctly on all pages of the website. This is useful in identifying potential routing errors with the system.

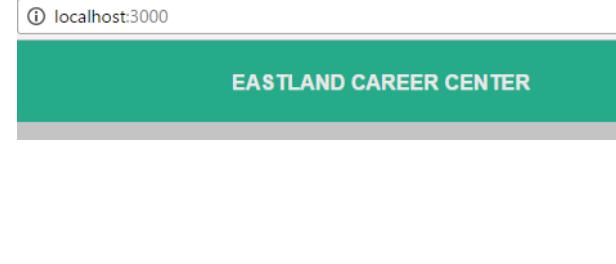
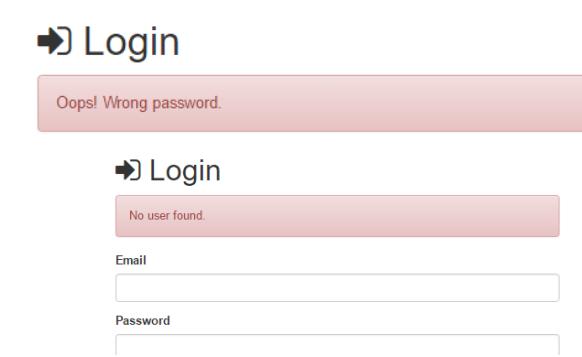
Registration Page

Test	Purpose of test	How can it be tested?	Expected Outcome?	Actual Outcome?	Additional notes
Do the “Register” and “Login” menu links redirect the user to the right routes?	Does the header for unauthenticated users work correctly?	Click the “Register” item on the navigation bar for an unauthenticated user. Click the “Login” item on the navigation bar for an unauthenticated user.	1. After clicking the register item, the user should be redirected to the /register route 2. After clicking the login item, the user should be redirected to the /login route	As expected.	1. User was redirected to the /login & /register page when selecting the elements of the navigation bar on the registration page. 2. User was also redirected to the /login & /register page when selecting the elements of the navigation bar on the login page. 3. Selecting the Eastland Career Center logo redirects the user to the /login page
Are the hyperlinks on the registration page working correctly?	Does the hyperlink named “login” redirect to the correct route?	Click on the “login” hyperlink on the signup page.	User should be redirected to the /login route	As expected.	After selecting the login hyperlink, the user was redirected to the /login page

Are users being redirected to the home page after they register?	To check if a user is authenticated and redirected after signup.	Register an account on the website and note down the redirects	After registering, the user should be redirect to the route "/" (index page) and be authenticated to the system.	As expected.	HOME LEADERBOARDS REGISTRATION TEST ▾
---	--	--	--	---------------------	---------------------------------------

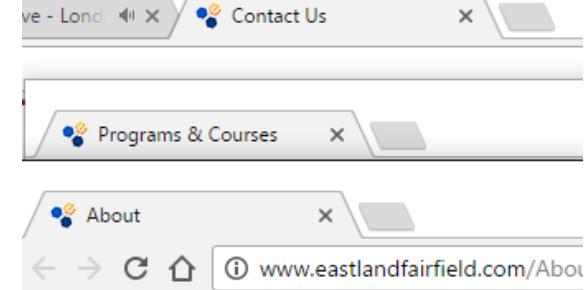
Login Page

Test	Purpose of test	How can it be tested?	Expected Outcome?	Actual Outcome?	Evidence
Are unauthenticated users redirected to the /login route if they try to access a page they are not authorized to view?	To check if unauthorised users are redirected to the /login page if they attempt to access a locked page.	Try to access the route /index, /leaderboards and other pages to check if redirected to the /login page	Unauthenticated users will be directed to the /login page when trying to access protected pages such as /leaderboards		<ul style="list-style-type: none"> 1. All pages redirect to /login with authentication <ul style="list-style-type: none"> 1.1. /game/local 1.2. /game 1.3. /leaderboards 1.4. /search/user 1.5. /sjsj (random)
Are the hyperlinks on the login page working correctly?	To check if the signup hyperlink is routing the user correctly	Select the hyperlink labelled signup on the /login page	After selecting the button, the user should be redirected to the /signup page		<ul style="list-style-type: none"> 1. Redirect to the /signup page

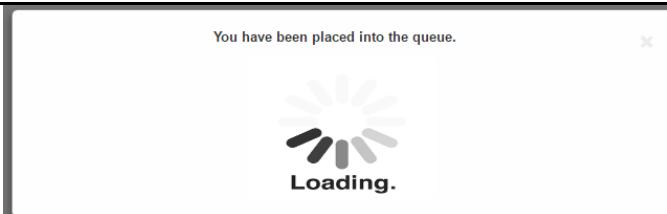
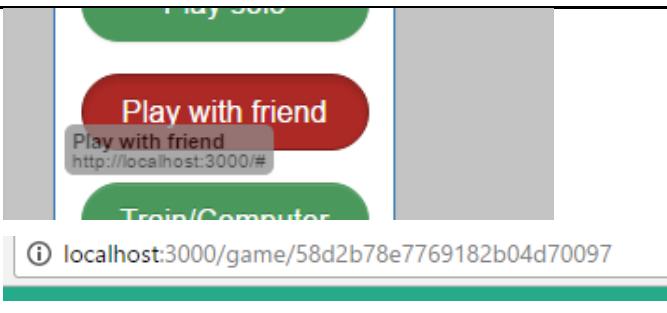
Are users redirected to the / (index page) after logging in?	To check if users are directed to the home page after authenticating themselves to the application	Login to the application with an email address and password found in the database	The user should be redirected to the route / and be authenticated to the system.		
Are users being provided with error messages and redirections to the home page if they make an invalid login request?	To ensure that incorrect login requests are being handled routing invalid login requests.	Send an invalid login request: 1. Using a password that does not match for a corresponding user in the database. 2. Using an email address that does not exist in the database. 3. With fake information.	Appropriate error message and re-routing to /login depending on invalid request.		

Leaderboard Page

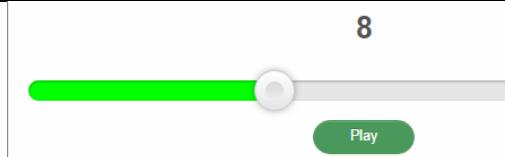
Test	Purpose of test	How can it be tested?	Expected Outcome?	Actual Outcome?	Evidence
Is the correct header displayed on the leaderboard page for authenticated users?	To check if the appropriate header is being displayed on the page.	View the leaderboard page while authenticated	The header which includes [Home Leaderboards Account Name] should be displayed		CENSOR
Are the links in the header working correctly?	To check if the authenticated header navigates correctly	Select the all the elements in the header and check if they redirect correctly.	<ol style="list-style-type: none"> 1. Home → (/) the index page 2. Leaderboards → (/leaderboards) 3. CENSOR BLUR → Logout (/logout) (user logged out) 		<ol style="list-style-type: none"> 1. Home redirection successful 2. Leaderboard redirection successful 3. Successfully logged out after selecting logout button
Can unauthenticated users access the page?	Are the page permissions working correctly?	Try to access the page as an unauthenticated user and an authenticated user.	<p>The unauthenticated user should be redirected to the /login route.</p> <p>The authenticated user should have his request as normal and view the /leaderboard page.</p>		Leaderboards

Do the footer menu links redirect to the correct sites?	Are the links in the footer correct and redirecting the user appropriately?	Select all the menu links on the footer	<ol style="list-style-type: none"> 1. Home → route to (/) 2. About → Eastland Fairfield's about page on their website 3. Contact Us → Eastland Fairfield Page 4. Courses → Eastland Fairfield Page 		
Does the search field work with entries?	Are users being redirected to the search page with the number of entries they requested?	Select 10 entries and search a user then select 25 entries and search a user.	<p>Minimum entries will be shown for the search unless there are a great number of entries.</p> <p>User should be redirected to /search/user/search/numberOfEntries</p>		<ol style="list-style-type: none"> 1. Searching for 25 users in a search that returns 5 users will set the entries to 10. 2. Search for a user with entry 10 that returns 9 users sets entries to 10. 3. Redirected to /search/user/entry/numberOfEntries 4. Number of entries changed to 10 if not enough results to paginate correctly.

Home Page

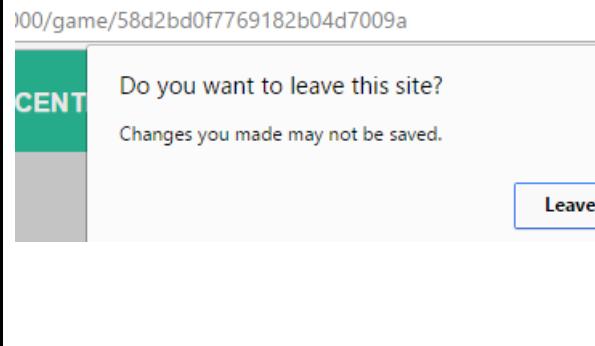
Test	Purpose of test	How can it be tested?	Expected Outcome?	Actual Outcome?	Evidence
What header is being displayed on the home page?	To check if the header for authenticated users is being displayed on the home page.	Go to the home page and view the header	The header: [Home Leaderboards My Name (dropdown)]		N/A
Does the play solo button trigger the popup with the loading screen?	To check if the play solo button is working correctly	On the home page, press the button labelled play solo	A popup should appear to the user with a .GIF of a buffering bar		
Does the play friend button redirect the user to a play friend game?	To check if the play versus friend button is working correctly	On the home page, press the button labelled Play with friend.	The button should turn red as it is clicked and then the user should be redirected to /game/ID with ID being a unique game identifier.		

Does the play computer button trigger the popup with a slider?	To check if the play / train versus computer button is working correctly	On the home page, press the button that is labelled “Train/Computer”	The button should turn red and a popup should appear with a slider to set the difficulty level of the Computer		
Does the slider allow for the user to change the position of the slider?	To check if the drag feature of the slider is working correctly	When the slider has popped up by pressing the “Train / Computer” button try to drag the slider left and right.	As the button is moved to the right, the slider should increase and the number should increase and vice versa for the other way.		
Does the slider allow for the user to user to extend beyond the limits of 1 and 20?	To check if the slider is preventing the user from going past the boundaries	Try to slide the slider to the left when it is already on number 1 and try to slide the slide to the left when it is on 20.	The user will not be able to move the slider and the number will not change.		

Does the slider button allow for the user to create a friend game match?	To check if the button on the slider popup is working correctly and if the user is being redirected to the correct location	Press the button on the slider with a difficulty level selected labelled play	The user will be redirected to the /game/local/LEVEL page with the level being the number passed into the slider.		 
When a user is paired up with another user in queue, are they redirected to the game page?	To check if the redirection is working correctly during match making	Join the lobby queue on two different accounts	Both users will be redirected to a game page where they will be playing against each other		N/A

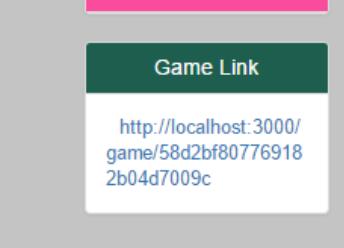
Play versus Player (Solo Page)

Test	Purpose of test	How can it be tested?	Expected Outcome?	Actual Outcome?	Evidence
Is the correct header displayed on the leaderboard page for authenticated users?	Is the correct header being displayed on the page?	In a player versus player (solo game) check the header at the top of the page.	The authenticated header should be displayed.	As expected	N/A
Has the user been assigned a random orientation on the board?	Is the correct orientation being assigned to the user?	Start multiple player versus player games by adding two users to the matchmaking queue and then note down the orientation assigned.	Randomized orientation, 50% chance of black and white.	<ol style="list-style-type: none"> 1. Game 1 → Assigned black orientation 2. Game 2 → Assigned white orientation 3. Game 3 → Assigned white orientation 4. Game 4 → Assigned black orientation 5. Game 5 → Assigned white orientation 	<p>Game 1:</p>  <p>Game 2:</p> 

Press one of the navigations on the page	To check if the defensive popup to inform the user they will leave the game is working correctly.	In a player versus player game, try to press a button on the website that would result in the user leaving the game page.	A popup should appear to the user asking them to confirm if they would like to leave the website.		
---	---	---	---	---	---

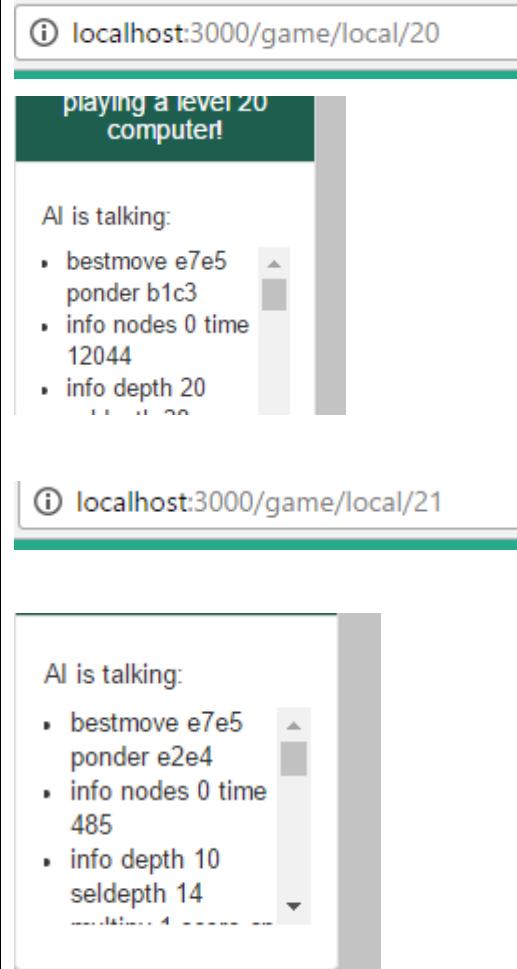
Play versus Player (Friend Page)

Test	Purpose of test	How can it be tested?	Expected Outcome?	Actual Outcome?	Evidence
Is the header for authenticated users being displayed on the page?	Is the correct header being displayed on the page?	On the player versus friend page check if the header at the top is for authenticated users	The authenticated header (Home Leaderboards Full name).		N/A
Has the user been assigned the white orientation on the board?	Is the correct orientation being assigned to the user?	Start a player versus friend game by selecting the play versus friend button on the home page	The creator should be assigned the white orientation on the board.	<p>Game 1 → White orientation</p> <p>Game 2 → White orientation (Check twice for defensive check)</p>	<p>Game 1:</p>  <p>Game 2:</p> 

Has the user been provided with a game link he can send to a friend?	To check if a uniquely generated link is being provided to the requesting user	When creating a player versus friend game, copy the link and paste it into another browser.	The user should be provided with a unique link that can be sent to the user's friend. The joiner of the link should now be in the game with his opponent.		 <p>Game Link</p> <p>http://localhost:3000/ game/58d2bf80776918 2b04d7009c</p>	Game Picture N/A
Is the joiner of the link assigned the black board orientation?	Is the correct orientation being assigned to the joining user of the game?	View the orientation assigned after joining a provided link	The joiner should be playing the black orientation on the board		N/A	

Play versus Computer page

Test	Purpose of test	How can it be tested?	Expected Outcome?	Actual Outcome?	Evidence
Is the header for authenticated users being displayed on the page?	Is the correct header being displayed on the page?	On the play versus computer page, check if the header at the top is the header for authenticated users	The authenticated header (Home Leaderboards etc) should be displayed		N/A
Is the white orientation being assigned to the user?	Is the correct orientation being assigned to the user?	Start a play versus computer game and check the orientation that has been assigned to the user	The user should be assigned the white orientation on the board		N/A

Has the user been assigned the correct computer level?	To check if the POST request for setting a computers level is working correctly	Change the number in the request for the route /local/game/number	The user be playing against the Computer level requested. If the number is > 20 then the user should be assigned to play against the level 10 computer.	
				 <pre>localhost:3000/game/local/20 playing a level 20 computer! AI is talking: • bestmove e7e5 ponder b1c3 • info nodes 0 time 12044 • info depth 20 ... 11 20 localhost:3000/game/local/21 AI is talking: • bestmove e7e5 ponder e2e4 • info nodes 0 time 485 • info depth 10 seldepth 14 ... 14 14 4 ----- 4</pre>

Full Name: BLUR, CENSOR

Candidate Number: N/A

Centre Number: N/A

SECTION 5: EVALUATION

Section 5.1: Iteration 1

Objective Analysis

The first stage of this project has been extremely successful and I am personally quite happy with the time that it has taken for me to complete this stage. All the objectives have been reached without any limitation and have all been tested successfully in the testing section. Due to this being an extremely preliminary stage of my project, I will not be commenting much on this; however, here are the results and will not be discussing this with my end-user.

The following objectives have been completed:

1. NodeJS and MongoDB have been installed successfully
2. Express, MongoDB & Monk have been installed successfully
 - 2.1. The packages have also been included in the package.json file making it easier for the system to be worked on by another developer. If another developer would like to work on this project, then the command “npm install” will install the modules required to run the application.
3. The database has been setup correctly with the two collections.
 - 3.1. The authenticated has read and write privileges to the database.

Improvements

I have little to comment on this section however, one improvement that could have been made to this stage of the project would be to implement a backup system. The backup system should upload a copy of the project daily to a secure cloud such as Dropbox, Amazon or any other cloud hosting services. However, I still have a backup on the server, my computer and my other server but additional security is never a bad thing.

Personal Feedback

Overall, I have been happy with the development of my application. In this small-time frame, I have been able to enhance my understanding of the languages and systems which I am planning to work with which hopefully, will allow for me to tackle this project in a much more efficient matter.

Section 5.2: Iteration 2

End-user Conference

This conference occurred after the end-user was shown the progress made on the application.

How did you find the application so far, Jesse?

JS: *Good stuff, would like for the error messages to be slightly changed if possible but besides that, good stuff.*

Sure thing, what would you like changed?

JS: *Please can you change the errors from saying “Your email is not valid” to “Please enter a valid email address”. Similarly, with the other error messages, make them say “Your” so they are directed at the user.*

No problem, I'll make this change now. Did you find anything else you would like changed?

JS: *I'm happy so far.*

Any comments?

JS: *Thank you for adding the real-time errors and persistence!*

Objective Analysis

The following objectives have been completed for this Iteration without limitation:

1. The application can now differentiate between an unauthenticated user and authenticated user
2. An unauthenticated user can now register to the application
 - 2.1. All the conditional arguments for input values from first name to date of birth have been implemented.
3. An unauthenticated user can now login to the application
4. An authenticated user can now log out of the exception

Although the objectives have been successfully achieved, of the objectives have been achieved exceeding the desired requirements:

1. There are now real-time error messages for unauthenticated users.
 - 1.1. If a user tries to enter invalid information to any of the fields then error messages will be triggered as they occur. Instead of having to wait for the server to respond with an error message.
2. Users now have persistent logins meaning that even if the server is restarted while a user is authenticated to the system. They will remain authenticated.
3. Strong error handling in place
 - 3.1. Users who are authenticated cannot submit additional registration requests, good for security.

Improvements

I am quite happy with the current success of my project. However, I would like to address some potential improvements that can be made to the application. Since I am storing passwords and transferring passwords over an insecure network (due to the lack of an SSL certificate) this is a mega security hole.

- 1. Passwords being stored on the database, regardless of whether they are encrypted, is unsafe. It is likely there may be exploit on my system which a hacker could use to penetrate my system, i.e. a zero day, although the user requested it. Storing passwords on the system is not really the best practice.**
- 2. The application is susceptible to man-in-the-middle attacks and should be deployed using an SSL certificate when released.**

The system could be improved with the implementation of better error handling on the application. Although the system has some boundary conditions in place for email addresses and date of births, they can still provide incorrect responses. Due to the ever-changing list of valid email addresses, the regex should be maintained and kept up to date with new domains. Identifying valid domains would be a way to prevent users signing up with invalid email addresses.

Additionally, the maximum length of an email address is where the “local-part” in local-part@domain.com can be up to 64 characters in length and 254 for the domain. This could be a nice feature to add in the future, along with some verification emails before a user can be registered to the system to prevent spam or a captcha.

Section 5.3: Iteration 3

This iteration implemented the client server model for my application. It allowed for me to gain experience in using Sockets and provide me with a better understanding of a way in which I can solve this problem for my end-user. I did find that there were some problems with this iteration, in all honesty I found this implementation quite difficult. But, I am more than glad to have broken down my project into smaller problems so that I can tackle problems which are realistic.

End-user Feedback

How do you find the Chess application so far, Jesse? Do you like the design for the board?

JS: *It is great! I showed it to some of the members in our group, we like how you have used a similar system to Lichess (which loads of us use from time to time) making it much easier for us to familiarize ourselves with the system.*

How have you found the process for this iteration?

JS: *An annoyance was the fact that this seems to have taken quite a long time to be completed. In future, we should try to shorten the amount of time these iterations take.*

I agree, I spent a lot of time trying to figure out the complexities of the system for future development. However, I believe that the additional time will allow for future aspects to be solved in a quicker.

JS: *Fair enough.*

Did you find any bugs or problems with the system?

JS: *Only one bug with the application. I showed it to our members and apparently if you move a bunch of pieces on the board simultaneously the pieces start flying around.*

I also noticed this bug, it will be fix in the next iteration. The bug is caused by the lack of game logic. Once the logic for the board is implemented, when each user can only make one move at a time then this bug will be fixed.

Any other comments on the program?

JS: *I'm happy, you seem to have understood my request very well. Keep up the great work and take care.*

Objective Analysis

Taking Jesse's feedback on the current iteration, he seems to be quite happy with the current progress as he stated that he and the prospective users have used a similar design on another Chess application called lichess.

All the objectives in this iteration have been completed without limitation:

1. **A user can communicate message to and from the server** → Users can communicate messages to and from the server using WebSockets. These messages can be parsed and understood by both members in this duplex communication line.

2. **A user can start and play a game of chess** → Users of the system can now play games of chess, move pieces on the board and use a link to add another player to the game. This link is sent to another player who can then use this link to join the game.

3. **Another user can play the same game of Chess as another user** → A second user can join the same game as another user and will be viewing the board from black's perspective. In addition to being able to move the pieces on the board, multiple games can now occur on the system which is exceeding an initial requirement set by the end user.

4. **The user(s) playing a game of Chess on the same board can view their opponent's actions on the board in real-time** → Actions on a specific board will update in real-time. The data about the board will also be sent to the users console and logged on the server.

5. **The clients and the server will provide data about any messages received through their WebSocket(s)** → The server logs information about any change that happens on a specific board and the board that this change has occurred on. In addition, any user changes on the board will also be logged.

The objectives set out for this iteration have been iteration were quite informative and useful. I am very happy with the specifications that were provided out to complete this aspect of the application. However, I do believe that they could have been improved by providing more information about simultaneous games and how the system not only needs to be able to handle an individual game but multiple games. Handling multiple games on the system simultaneously is an essential part of this system and so specifying it in the objectives would have been much more helpful when it came to the design and implementation sections.

Potential Improvements

There could have been some improvements made to this iteration which would have created a more bug-free system or made it easier for myself to develop this aspect of the application. The first thing to note is that as pointed out by Jesse and myself, there is a bug in the system when multiple users try to move multiple pieces on the board at the same time.

A simple fix for this could have been implemented to prevent this from happening by just preventing users from moving more than one piece on the board at the same time or only allowing for one user to make a move on the board at the same time. Inevitably this bug is going to be fixed with the implementation of board logic but, it would have been much better to deploy a prototype to the end user's which did not have that bug.

Time management could also be improved. Punctuality is an element of this project that is extremely important and should not be ignored, the end user should constantly be up to date with the progress of the application. In future, I will keep this in mind and ensure that communication between myself and the end user starts to increase.

Personal Feedback

As the amount of time spent on this project starts to increase, I have started to become less satisfied with the progress I have been making on the application. I seem to have found parts of this application a struggle and had to perform a vast amount of research into the problem itself and into the theoretical side to solve it. I have gained a much better understanding of the programming language I am using, my programming paradigm and approaching complicated problems.

Since I have received a small amount of criticism in this iteration, I will try my best to utilise this in a good way to motivate myself to improve my time management skills and communication to the end user.

Section 5.4: Iteration 4

The fourth iteration of my project has so far been the most exciting element of my project. I was happy to be considering artificial intelligence, neural networks and finding a suitable solution to a problem that seems slightly out of this world. I am extremely happy that I could solve this problem as some believed that it may have been out of my scope of knowledge.

In this iteration, the mode for users to play versus a computer system was added to the game. This engine was customizable by the user meaning that they could set the difficulty of the computer that they would like to play against. The higher the difficulty level set by the user then the harder the computer will be for the user to beat.

End-user Feedback

Due to Jesse being busy with his personal life and me wanting for this part of the application to be shown to the prospective users of the application, I attached the files and instructions to run the system to Jesse so that he could show it to the members of the society in their next meeting. He was not able to get back to me via voice and so he left me the following messages:

Hello Jesse, I have been able to put together and complete the training game mode for your application. I have attached the files and instructions for you to be able to run the system to show to your society members.

JS: *Thank you.*

JS: *Here are some comments from our members:*

1. **Cory Sparks:** Easy to use and appealing.
2. **Michael Raynsford:** Lost in 4 moves!

We are very happy, sorry for the lack of feedback. I am very busy.

Objective Analysis

From the small feedback provided, the prospective users seem happy with the program. It would have been nice if Jesse and the users could have given some reviews or ideas for improvements on the application but it seems like everything is in check.

All the objectives in this iteration have been completed without iteration. All the objectives inside of this iteration have been completed without limitation (1 → 3)

The following additional features have been added to this iteration:

Potential Improvements

One of the ways this iteration could have been improved is if the computers were provided with elo ratings. It would be very interesting if each difficulty level of a Computer could be assigned their own user rating so that when users play against them they are more able to know just how good a Computer is because currently, it is a process of trial and error.

Another nicer implementation would be if the users had the ability to choose which side of the board they would like to play on so that they could try their luck on different orientations of the board. Some players may prefer to start the game while others do not.

Personal Feedback

In the last iteration, I was quite disappointed with the amount of progress being made compared to the amount of time being used on the project. However, it seems like I have been able to overcome this plateau and my development skills and time management have dramatically improved. Although this seemed like a complex problem, the preliminary research I originally conducted allowed for me to create a simple yet effective solution for this problem.

Motivation and ethic is starting to come back me, after a couple of bad days with development things are getting back on track. I am happy to know that one of the solutions I used for this project was to have a file which stored common game attributes, a solution I may not have thought of before embarking on this project but, is without doubt a much more efficient solution. The readability and exception handling of my code is also starting to improve. Communications between myself and Jesse have been improving, our conversation length has been decreasing but the amount of information we have been communicating with one another has been increasing.

Something I should keep in mind in the future is that although my development skills are growing allowing for me to develop more things, I should not always try to exceed the expectation of the end user. If I was to consistently exceed the objectives set out by my end user and add things to the project which were not requested then I may end up dissatisfying the end user and causing additional time constraints for the future.

Personal Evaluation

In the future, I should not always try to exceed the expectation of the end user. If I was to consistently exceed the objectives set out by my end user and add things to the project which were not requested then I may end up dissatisfying the end user and causing additional time constraints for the future.

Section 5.6: Appraisal

Since my project came to an end, I provided my end-user with a questionnaire so that he could provide feedback on the process that we underwent together. This allowed for Jesse to give his opinion on the development process, the final application, and ways that we could improve in the future.

Here is his response:

User Appraisal – Computer Science NEA
Please rate the following aspects of the Chess system out of five:

Aspect	Rating	Comment
User Interface	7/10	1/2
Usability	9	3/3
Leaderboards	10	4/4
Online Chess	9	5/5
Chess vs Computer	9	6/6
Maintenance	8	7/7
Security		

How did you find the duration of this development process? Is there anything that you would have liked to be changed in the future?
It was fun to see this creation from start to finish.
The development cycle was long, and Rayyan needed a lot of input from us in order for him to craft this to our exact specifications, which we understand was necessary. At the end of the process it was amazing to finally see what had been made.
Does the application meet the requirements of the proposed system and if so, what are the features you like and dislike about the application?
Yes this application exactly met our needs. We are happy with the design, gameplay, AI, leaderboards, and user login system. There are no disliked features.
Is there any additional feedback you would like to leave?
If improvements were to be made, it could be more responsive in terms of Web Design via animation / UX and a chat system.

Signed: Jesse Stimpson Date: 3/20/17

- 1) We love the UI, although customizations like color would give each user a unique feel.
- 2) The overall usability is great, everything is right in front of you. Some day we'd like to move our communication from Discord to this service itself.
- 3) The leaderboards are amazing, we love the glicko 2 elo you chose. We'd like to see a share feature in the future so players may brag on their own / others stats.
- 4) 100% perfect, WebSockets are so responsive!
- 5) We love this feature but Man is that computer good :)
- b) Very easily maintained, clean codebase using Node.js and MongoDB
- 7) We are aware server-side data is protected but we'd like to some day see some form of 2FA like gAuth in the future :)

Analysis of end-user feedback

User Interface

Jesse rated the user interface for the system a 7/10 but said that there is a lack of customization for the users. His point is quite valid, I am not the strongest front end designer in the world and the website certainly lacks responsiveness and customization. Although this score remains quite high especially considering that it was not a top priority for the proposed application.

In the future, I could improve upon Jesse's request by making the website more responsive to different monitor sizes. The website should be able to detect the proper screen size of the end-user and resize the elements accordingly. Another potential improvement for the user

interface would be to incorporate a customizable colour palette whereby users can change the colour scheme of the website depending on their own personal preference.

Usability

Jesse rated the usability element of the application a 9/10 and commented that in the future, their members would like to move their communications from Discord to the application itself. This comment by the society links to the additional feedback left by Jesse which was that the application could have been improved if a chat system was incorporated into the website. A messaging and voice chat system would allow for the members of the society to have a solid medium of communication where they can both play and talk to one another.

This implementation would also be quite simple as real-time messaging can be easily integrated to the current client server model. I am very happy to say that this important aspect of the application has been reached to high standards.

Leaderboards

Jesse and his members rated the leaderboards in the application a 9/10 and commented that the implementation of Glicko was a really good idea. It appears that he and the prospective users of the application are more satisfied with the Glicko system compared to an Elo system, most likely since it is a much more accurate and reliable system.

An additional feature that the user would have liked in the system would be the ability to share scores to social media. I assume that by this, the group means sharing to social media or potentially an Eastland social media platform such as Yammer on Office 365. Overall, the users seem quite pleased with the leaderboard system and so this main element has also been successfully achieved.

Online Chess

The rating for the online chess was a perfect score, a 10/10 which is without doubt the most crucial element of the application. Their response simply indicates that the Chess system has been designed exactly to their specifications with elements that exceed the proposed requirements set out by Jesse.

Although this element was scored 10, I do believe that additional changes could be made to further improve this unit of the application. The matchmaking system could be improved by allowing for users to queue up against specific ranks for example, allow for users to queue against players of rankings that are a number below or above them. For example, users could be allowed to play a matchmaking game but only play against players who have a Glicko rating that is the same as theirs or 100 rating points above.

Chess vs Computer

The training option of the application was rated a 9/10 and the feedback left by the members of the group is quite vague. They seem to be big fans of the training feature on the system and left a slightly sarcastic comment about how difficult the Computer can get. I do agree with them; the Computer can be very tough.

Although there were no improvements mentioned by Jesse, we did discuss a potential improvement for the application at a previous point in time. It would be a nice feature if users had the ability to choose the orientation on the board as currently, the computer will always be playing as black and the user white. It would allow for the application to be more

customizable and allow for interfaces to be added which may consequently improve the user interface rating.

Overall though, the two Chess aspects of the application seem to be highly commended and the user is very satisfied with the results.

Maintenace

The maintenance rating is an indication of how easy the system is for Jesse to manage in the future. Although it is no longer a requirement for me personally to maintain the system, I will most likely assist in maintaining the system for a small period in the future. As Jesse is also a software developer, he found the code very readable, efficient and easy to understand so that he can make any appropriate changes in the future. This is also a big element of my project and so it is very helpful to know that this has been improved.

In terms of improving my code, more error handling could have been implemented and certainly some function names could be slightly adapted so they are easier to understand.

Security

The final box was a rating for the system's security. Jesse being a security advocate rated the application's security an 8/10 and this was also a key part of the system itself. He commented that there is strong server side protection in place to ensure that the data on the server is secure in case of a potential leak but there is certinaly security holes in the system.

There is never enough security in a system as the number of malicious attacks grows day by day. The use of gAuth or a 2FA system would allow for user accounts to be more secure and would certainly be a great addition in the future. In addition, backup systems should be put in place to ensure that data will always be secure, this data could be stored on an external hard drive in a secure location. Overall, this is still a very good rating!

Additional Comments – Changes for the future

I also provided Jesse with a couple of questions to ask for his opinions on the development cycle. Since he is a developer, he is most likely more than able to provide some valuable information about our process and how I can improve as a developer in the future. In the first question, he mentioned that there was an excessive amount of communication between myself and him although he understands that this was only necessary due to the specifications set out for this project.

This process could have been completed much faster to the same degree of complexity in a much shorter time frame if all the documenting was not required. In the future, it would have been nicer if we could have developed this in a shorter timeframe as I know that Jesse and the members of his group were adamant to see the program finished.

Has the application met your expectations?

The group said that the application has exactly met their specifications and they are extremely happy with the system. He states that there are no downsides to the system but as seen in the comments provided by Jesse, additional features would not go without praise. The project seems to have been successfully completed.

Is there any additional feedback you would like to leave?

Jesse said that if there were to be improvements, as a priority, it would be to the website design. The users would like for the application to be more responsive and I can see their point as depending on the device being used to access the application, there may be some whitespace located in the footer of the page caused by lack of content. This could be fixed with a fair bit of additional code but, front end development is really not my forte and would be more suited for a front-end developer.

A chat system would have also been a good feature to be added, it would allow for the users to be able to communicate and play on one medium. It's likely that in the future, when I have the time I will implement this feature for Jesse. Sadly, this project is coming to an end so it is most likely not going to be documented.

Proposed System Analysis

Now that Jesse and the prospective users of the system have left some valuable feedback about the development process and the final application. I will conduct a full evaluation of the proposed application objects against the completed system. Using the feedback provided by Jesse and my own personal opinions, I can create a table to indicate how successfully this problem has been solved.

I will be using the key below:

Objective not met	Objective partially met	Objective met	Objective exceeded
-------------------	-------------------------	---------------	--------------------

Original Objective	Completed System
The application must be web-based for cross system compatibility	The application works on all operating systems that are running an internet browser which supports Javascript
The website should be user-friendly and have a minimalistic design	Although this was rated a 7/10, this was the lowest rating out of all the aspects of my developed system. The website still met the expectations set out by the end user but there is certainly room for improvement in terms of website design, customization and responsiveness.
The website should store information about users. Such as games played on the system, the number of players currently online and the number of games being played.	The application stores personal information about the users such as their first names, last names and passwords securely. There is also a game score allocated to each user which stores their total number of wins, losses and draws. Each game played on the system is stored along with time stamps indicating the time

	of when the game was played, how the game was ended and the winner of the game.
Users should be allowed to play a game of Chess against a computer with a difficulty level of their choice	Users can select a difficulty between 1 and 20 to train against a Computer.
Users should be allowed to play a game of Chess with a friend	Users can play against a friend by selecting the play vs friend option where they will be provided with a unique link. When another user opens this link, they will be playing against that user.
Users should be allowed to play a game of Chess against a random player on the system by joining a match making lobby	Users can play against with a random user of the system by playing the solo game mode. Each pair of users will be redirected and placed in a game, playing a random orientation on the board.
The system should keep track of a player's performance	<p>The system uses a complex mathematical formula to calculate the relative skill level of a player. This value can be used to calculate the performance of a player over a period.</p> <p>Depending on how a user performs in the long term will effect this number.</p>
Every game played by a user against another user will be used to track this performance. The system will store the total number of wins, losses and draws for each user of the system.	The system tracks all games played by a user that are against another player on the system. Games against a computer are not tracked as requested by the end-user.
Player's will have all been assigned an Elo rating, that is a mathematical representation of a player's skill level.	An Elo rating system has been implemented that has been built upon using cronjobs and Glicko-2 to perform more optimally than a generic Arpad based system.
The chess game should be multiplayer and update in real-time	The Chess game is multiplayer and updates in real-time. When a player moves a chess piece on the board, his opponent will see the move as it is made.
The system should display a leaderboard that contains a list of players' along with their performance and the player with the highest rating should be at the top of the leaderboard.	<p>The Leaderboard displays the list of all users from top to bottom, with the top player having the highest ranking of the players.</p> <p>Users can search users using a search bar, customize the number of users the leaderboard should display and change pages on the page.</p>

Users in a game of Chess should be allowed to export information about one of their games	<p>Users can export the portable game notation string of their game to a text file during or at the end of any game of Chess that they play.</p> <p>However, there is no current system in place that allows for users to export previous games which they played. This can be done manually by Jesse, and he has indicated that he is happy to do so as all games (along with their PGN strings) are stored in the database.</p>
Users in a game of Chess should be allowed to resign from a game of Chess	Users can forfeit a game by resignation.
Users in a game of Chess should be allowed to ask the other player on the board to end the match as a draw	Users can request for their opponent to end the match as a draw. If requested, the game will end as a draw.

Section 5.6: Conclusion

Now that my project has been completed, I would like to leave some feedback on how I found the process and some additional notes that may not have been documented in my project. For starters, I found that the testing section was quite unrealistic when it came to my agile development process as I usually debug my code while I code meaning that an extensive amount of testing has occurred throughout the duration of the project without being documented. More specifically, a lot of whitebox testing has occurred throughout the duration of this project to ensure functions are working as intended. Even though an agile development occurred, it is simply not possible to truly document all the testing that has occurred throughout the duration of the process.

I would create the solution for each iterative part of my project and test it fully before testing it on this document because I am no super experienced programmer who will it right the first time around. I also find it much easier for myself to test code as I go along since I am quite familiar with how the code is operating at the time of developing.

Overall, this project has been quite useful. I have learned a lot about time management, development and more throughout the duration of this processs but I do believe that agile development or any waterfall development strategies are quite difficult to approach. Personally, I try to specialise with a specific element of agile development similarly to how one individual would complete the design section of a project whereas, I must complete all the four elements. The front-end design for the website was by far the hardest task even though it was not close to being the most complex, I am not a huge designer and I am not the most creative designer. Also, I found that the guide for this project was very generalised and did not cover a lot of the methodologies I was using to tackle my project such as using a NoSQL databasing system and certain front-end templating models. However, this was a great project and a good way to spend my time and I have learned a lot about developing software, industry and coding asynchronously.

Thank you for reading my project!