Portal Chess

Strike Team Java

Team Members:
Mike Cunningham
Jared Adams
Cory Dahn
Conor Hart
Jonathan Quirk

What Is Portal Chess?

- Portal Chess is a variant of the classic game of chess
- All the basic rules are the same including movement of pieces, capture of pieces, castling, pawn promotion, check, and winning the game through checkmate
- The key difference is that portal chess adds a piece for each player called a portal
- When a normal piece lands on a portal, it is teleported and comes out of the other portal
- If the portal pieces occupy the same space, they create a 'black hole' that removes all pieces in the spaces immediately surrounding the portals
- For more information on Portal Chess visit Wikipedia: https://en.wikipedia.org/wiki/Portal_chess

Project Goals & Key Features

The overall goal of the project is to create an application that will allow individuals to play games of portal chess against other individuals remotely. Based on the requirements given to us, the final product has the following features and functionality:

- A user can create an account and login/logout of their account.
- A user can search for other players and send invites to play matches
- Once invited, a player can decide to accept or reject the invite
- If an invite to play is accepted, a game of portal chess will begin
- The players can then play the game according to the rules
- The state of the game is saved so players can login/logout and come back to the game
- Users have profiles that track key stats such as wins and number of games played.

Design process

- Coding:
  - IntelliJ
  - Eclipse
  - Codesandbox.io
  - Wikimedia Commons (Icons)
- Continuous Integration:
  - Github
- Kanban:
  - Zenhub
- Agile methodology
  - Scrum
- Communication:
  - Microsoft Teams
- Design Artifacts:
  - Lucid Chart, CRC Maker
  - Microsoft Teams

High Level Design

- Server – running from Spring Boot
    - Java framework that allows a Client to communicate with a Java server with API Calls
    - This server is then automatically configured to communicate with a specific SQL database, and can query to update or obtain information.
- Client- running on React.js
    - Easy to build and use UI components, communicates with server with API request/response. Render changes with state components.
- Database- MySQL
    - Currently have an accounts, notifications, and matches table to store data. Getting and setting data is all done with Spring's data access object classes.

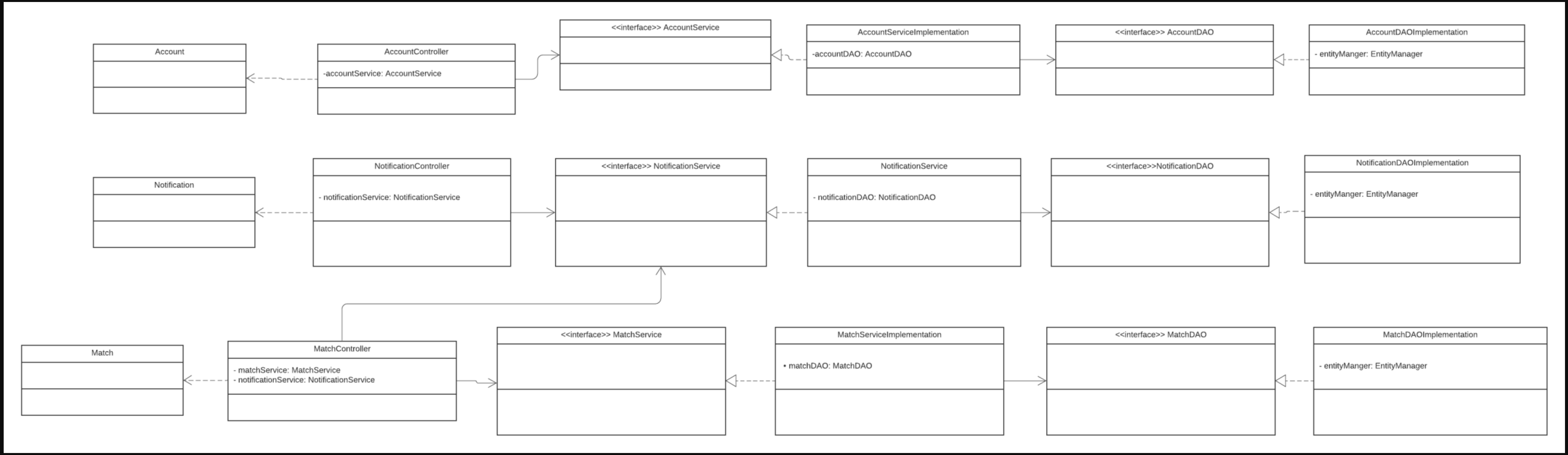
Design Decisions

- Refresh lists button on dashboard
- Our implementation follows the Ian Buckley ruleset
- Cannot move non-portal pieces onto portals
- Portals start in fixed positions

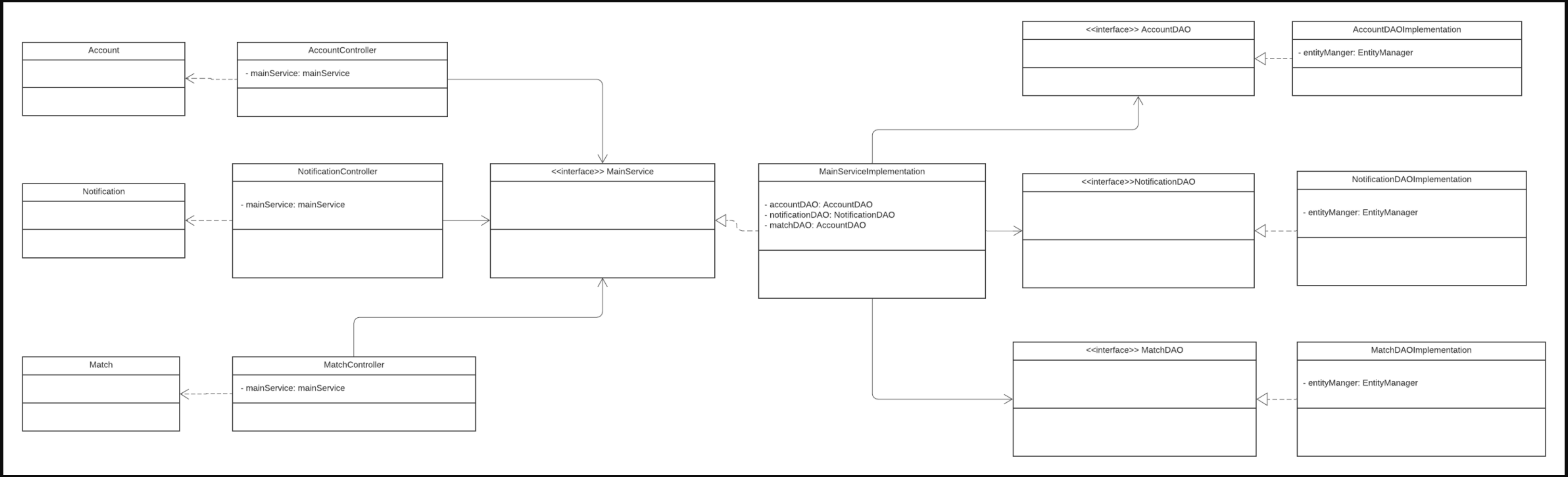
Design Patterns

- There are 3 request types one for each main object, each type has a Controller that will process the request from the client
- Each of these controllers has a service class that acts as an interface to communicate with the database. This allows a controller to access a specific table in the database.
- Late into the project we discovered this could have been done with having one service class that acts as an interface for all three controllers

How we did it


Account
AccountController
-accountService: AccountService
<<interface>> AccountService
AccountServiceImplementation
-accountDAO: AccountDAO
<<interface>> AccountDAO
AccountDAOImplementation
- entityManger: EntityManager
Notification
NotificationController
- notificationService: NotificationService
<<interface>> NotificationService
NotificationService
- notificationDAO: NotificationDAO
<<interface>>NotificationDAO
NotificationDAOImplementation
- entityManger: EntityManager
Match
MatchController
- matchService: MatchService
- notificationService: NotificationService
<<interface>> MatchService
MatchServiceImplementation
• matchDAO: MatchDAO
<<interface>> MatchDAO
MatchDAOImplementation
- entityManger: EntityManager

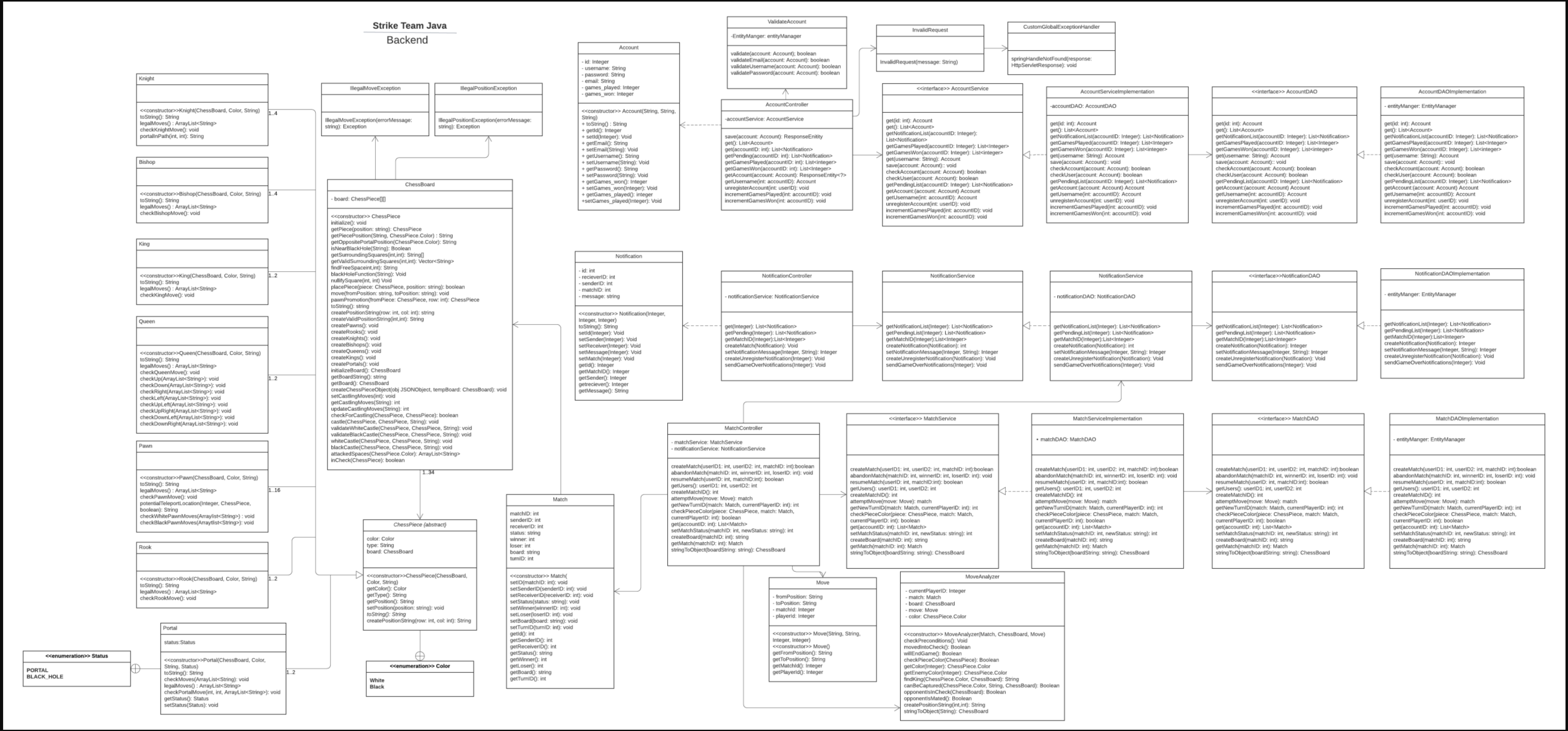
How we should have done it


Account
AccountController
- mainService: mainService
Notification
NotificationController
- mainService: mainService
<<interface>> MainService
Match
MatchController
- mainService: mainService
MainServiceImplementation
- accountDAO: AccountDAO
- notificationDAO: NotificationDAO
- matchDAO: AccountDAO
<<interface>> AccountDAO
AccountDAOImplementation
- entityManger: EntityManager
<<interface>>NotificationDAO
NotificationDAOImplementation
- entityManger: EntityManager
<<interface>> MatchDAO
MatchDAOImplementation
- entityManger: EntityManager

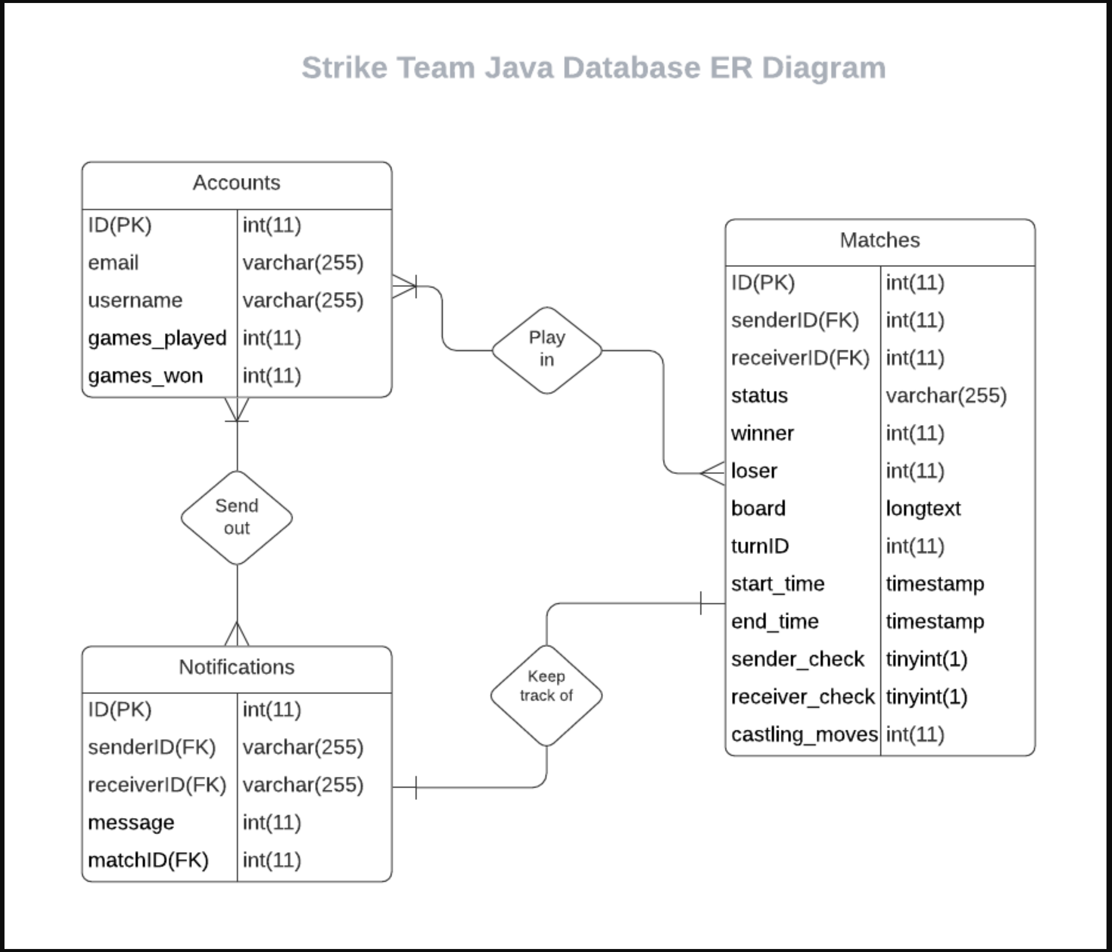
Design Artifacts

- CRC cards
- UML Class Diagram
- Database ER Diagram
- Frontend Diagram

UML Class Diagram


Strike Team Java
Backend
Knight
<<constructor>>Knight(ChessBoard, Color, String)
toString(): String
legalMoves() : ArrayList<String>
checkKnightMove(): void
portalInPath(int, int): String
1..4
Bishop
<<constructor>>Bishop(ChessBoard, Color, String)
toString(): String
legalMoves() : ArrayList<String>
checkBishopMove(): void
1..4
King
<<constructor>>King(ChessBoard, Color, String)
toString(): String
legalMoves() : ArrayList<String>
checkKingMove(): void
1..2
Queen
<<constructor>>Queen(ChessBoard, Color, String)
toString(): String
legalMoves() : ArrayList<String>
checkQueenMove(): void
checkUp(ArrayList<String>): void
checkDown(ArrayList<String>): void
checkRight(ArrayList<String>): void
checkLeft(ArrayList<String>): void
checkUpLeft(ArrayList<String>): void
checkUpRight(ArrayList<String>): void
checkDownLeft(ArrayList<String>): void
checkDownRight(ArrayList<String>): void
1..2
Pawn
<<constructor>>Pawn(ChessBoard, Color, String)
toString(): String
legalMoves() : ArrayList<String>
checkPawnMove(): void
potentialTeleportLocation(Integer, ChessPiece, boolean): String
checkWhitePawnMoves(Arraylist<String>) : void
checkBlackPawnMoves(Arraytlist<String>): void
1..16
Rook
<<constructor>>Rook(ChessBoard, Color, String)
toString(): String
legalMoves() : ArrayList<String>
checkRookMove(): void
1..2
Portal
status:Status
<<constructor>>Portal(ChessBoard, Color, String, Status)
toString(): String
checkMoves(ArrayList<String): void
legalMoves() : ArrayList<String>
checkPortalMove(int, int, ArrayList<String>): void
getStatus(): Status
setStatus(Status): void
1..2
<<enumeration>> Status
PORTAL
BLACK_HOLE
IllegalMoveException
IllegalMoveException(errorMessage: string): Exception
IllegalPositionException
IllegalPositionException(errorMessage: string): Exception
ChessBoard
- board: ChessPiece[][]
<<constructor>> ChessPiece
initialize(): void
getPiece(position: string): ChessPiece
getPiecePosition(String, ChessPiece.Color) : String
getOppositePortalPosition(ChessPiece.Color): String
isNearBlackHole(String): Boolean
getSurroundingSquares(int,int): String[]
getValidSurroundingSquares(int,int): Vector<String>
findFreeSpaceint,int): String
blackHoleFunction(String): Void
nullifySquare(int, int) Void
placePiece(piece: ChessPiece, position: string): boolean
move(fromPosition: string, toPosition: string): void
pawnPromotion(fromPiece: ChessPiece, row: int): ChessPiece
toString(): string
createPositionString(row: int, col: int): string
createValidPositionString(int,int): String
createPawns(): void
createRooks(): void
createKnights(): void
createBishops(): void
createQueens(): void
createKings(): void
createPortals(): void
initializeBoard(): ChessBoard
getBoardString(): string
getBoard(): ChessBoard
createChessPieceObject(obj JSONObject, tempBoard: ChessBoard): void
setCastlingMoves(int): void
getCastlingMoves(String): int
updateCastlingMoves(String): int
checkForCastling(ChessPiece, ChessPiece): boolean
castle(ChessPiece, ChessPiece, String): void
validateWhiteCastle(ChessPiece, ChessPiece, String): void
validateBlackCastle(ChessPiece, ChessPiece, String): void
whiteCastle(ChessPiece, ChessPiece, String): void
blackCastle(ChessPiece, ChessPiece, String): void
attackedSpaces(ChessPiece.Color): ArrayList<String>
inCheck(ChessPiece): boolean
1..34
ChessPiece {abstract}
color: Color
type: String
board: ChessBoard
<<constructor>>ChessPiece(ChessBoard, Color, String)
getColor(): Color
getType(): String
getPosition(): String
setPosition(position: string): void
toString(): String
createPositionString(row: int, col: int): String
<<enumeration>> Color
White
Black
Account
- id: Integer
- username: String
- password: String
- email: String
- games_played: Integer
- games_won: Integer
<<constructor>> Account(String, String, String)
+ toString() : String
+ getId(): Integer
+ setId(Integer): Void
+ getEmail(): String
+ setEmail(String): Void
+ getUsername(): String
+ setUsername(String): Void
+ getPassword(): String
+ setPassword(String): Void
+ getGames_won(): Integer
+ setGames_won(Integer): Void
+ getGames_played(): integer
+setGames_played(Integer): Void
Notification
- id: int
- recieverID: int
- senderID: int
- matchID: int
- message: string
<<constructor>> Notification(Integer, Integer, Integer)
toString(): String
setId(Integer): Void
setSender(Integer): Void
setReceiver(Integer): Void
setMessage(Integer): Void
setMatch(Integer): Void
getId(): Integer
getMatchID(): Integer
getSender(): Integer
getreciever(): Integer
getMessage(): String
Match
matchID: int
senderID: int
receiverID: int
status: string
winner: int
loser: int
board: string
turnID: int
<<constructor>> Match(
setID(matchID: int): void
setSenderID(senderID: int): void
setReceiverID(receiverID: int): void
setStatus(status: string): void
setWinner(winnerID: int): void
setLoser(loserID: int): void
setBoard(board: string): void
setTurnID(turnID: int): void
getId(): int
getSenderID(): int
getReceiverID(): int
getStatus(): string
getWinner(): int
getLoser(): int
getBoard(): string
getTurnID(): int
ValidateAccount
-EntityManger: entityManager
validate(account: Account); boolean
validateEmail(account: Account): boolean
validateUsername(account: Account): boolean
validatePassword(account: Account): boolean
InvalidRequest
InvalidRequest(message: String)
CustomGlobalExceptionHandler
springHandleNotFound(response: HttpServletResponse): void
AccountController
-accountService: AccountService
save(account: Account): ResponseEnitity
get(): List<Account>
get(accountID: int): List<Notification>
getPending(accountID: int): List<Notification>
getGamesPlayed(accountID: int): List<integer>
getGamesWon(accountID: int): List<Integer>
getAccount(account: Account): ResponseEntity<?>
getUsername(int: accountID): Account
unregisterAccount(int: userID): void
incrementGamesPlayed(int: accountID): void
incrementGamesWon(int: accountID): void
<<interface>> AccountService
get(id: int): Account
get(): List<Account>
getNotificationList(accountID: Integer): List<Notification>
getGamesPlayed(accountID: Integer): List<Integer>
getGamesWon(accountID: Integer): List<integer>
get(username: String): Account
save(account: Account):: void
checkAccount(account: Account): boolean
checkUser(account: Account): boolean
getPendingList(accountID: Integer): List<Notification>
getAccount:(account: Account) Account
getUsername(int: accountID): Account
unregisterAccount(int: userID): void
incrementGamesPlayed(int: accountID): void
incrementGamesWon(int: accountID): void
AccountServiceImplementation
-accountDAO: AccountDAO
get(id: int): Account
get(): List<Account>
getNotificationList(accountID: Integer): List<Notification>
getGamesPlayed(accountID: Integer): List<Integer>
getGamesWon(accountID: Integer): List<integer>
get(username: String): Account
save(account: Account):: void
checkAccount(account: Account): boolean
checkUser(account: Account): boolean
getPendingList(accountID: Integer): List<Notification>
getAccount:(account: Account) Account
getUsername(int: accountID): Account
unregisterAccount(int: userID): void
incrementGamesPlayed(int: accountID): void
incrementGamesWon(int: accountID): void
<<interface>> AccountDAO
get(id: int): Account
get(): List<Account>
getNotificationList(accountID: Integer): List<Notification>
getGamesPlayed(accountID: Integer): List<Integer>
getGamesWon(accountID: Integer): List<integer>
get(username: String): Account
save(account: Account):: void
checkAccount(account: Account): boolean
checkUser(account: Account): boolean
getPendingList(accountID: Integer): List<Notification>
getAccount:(account: Account) Account
getUsername(int: accountID): Account
unregisterAccount(int: userID): void
incrementGamesPlayed(int: accountID): void
incrementGamesWon(int: accountID): void
AccountDAOImplementation
- entityManger: EntityManager
get(id: int): Account
get(): List<Account>
getNotificationList(accountID: Integer): List<Notification>
getGamesPlayed(accountID: Integer): List<Integer>
getGamesWon(accountID: Integer): List<integer>
get(username: String): Account
save(account: Account):: void
checkAccount(account: Account): boolean
checkUser(account: Account): boolean
getPendingList(accountID: Integer): List<Notification>
getAccount:(account: Account) Account
getUsername(int: accountID): Account
unregisterAccount(int: userID): void
incrementGamesPlayed(int: accountID): void
incrementGamesWon(int: accountID): void
NotificationController
- notificationService: NotificationService
get(Integer): List<Notification>
getPending(Integer): List<Notification>
getMatchID(Integer):List<Integer>
createMatch(Notification): Void
setNotificationMessage(Integer, String): Integer
createUnregisterNotification(Integer): Void
sendGameOverNotifications(Integer): Void
NotificationService
getNotificationList(Integer): List<Notification>
getPendingList(Integer): List<Notification>
getMatchID(Integer):List<Integer>
createNotification(Notification): int
setNotificationMessage(Integer, String): Integer
createUnregisterNotification(Notification): Void
sendGameOverNotifications(Integer): Void
NotificationService
- notificationDAO: NotificationDAO
getNotificationList(Integer): List<Notification>
getPendingList(Integer): List<Notification>
getMatchID(Integer):List<Integer>
createNotification(Notification): int
setNotificationMessage(Integer, String): Integer
createUnregisterNotification(Notification): Void
sendGameOverNotifications(Integer): Void
<<interface>>NotificationDAO
getNotificationList(Integer): List<Notification>
getPendingList(Integer): List<Notification>
getMatchID(Integer):List<Integer>
createNotification(Notification): Integer
setNotificationMessage(Integer, String): Integer
createUnregisterNotification(Notification): Void
sendGameOverNotifications(Integer): Void
NotificationDAOImplementation
- entityManger: EntityManager
getNotificationList(Integer): List<Notification>
getPendingList(Integer): List<Notification>
getMatchID(Integer):List<Integer>
createNotification(Notification): Integer
setNotificationMessage(Integer, String): Integer
createUnregisterNotification(Notification): Void
sendGameOverNotifications(Integer): Void
MatchController
- matchService: MatchService
- notificationService: NotificationService
createMatch(userID1: int, userID2: int, matchID: int):boolean
abandonMatch(matchID: int, winnerID: int, loserID: int): void
resumeMatch(userID: int, matchID:int): boolean
getUsers(): userID1: int, userID2: int
createMatchID(): int
attemptMove(move: Move): match
getNewTurnID(match: Match, currentPlayerID: int): int
checkPieceColor(piece: ChessPiece, match: Match, currentPlayerID: int): boolean
get(accountID: int): List<Match>
setMatchStatus(matchID: int, newStatus: string): int
createBoard(matchID: int): string
getMatch(matchID: int): Match
stringToObject(boardString: string): ChessBoard
<<interface>> MatchService
createMatch(userID1: int, userID2: int, matchID: int):boolean
abandonMatch(matchID: int, winnerID: int, loserID: int): void
resumeMatch(userID: int, matchID:int): boolean
getUsers(): userID1: int, userID2: int
createMatchID(): int
attemptMove(move: Move): match
getNewTurnID(match: Match, currentPlayerID: int): int
checkPieceColor(piece: ChessPiece, match: Match, currentPlayerID: int): boolean
get(accountID: int): List<Match>
setMatchStatus(matchID: int, newStatus: string): int
createBoard(matchID: int): string
getMatch(matchID: int): Match
stringToObject(boardString: string): ChessBoard
MatchServiceImplementation
• matchDAO: MatchDAO
createMatch(userID1: int, userID2: int, matchID: int):boolean
abandonMatch(matchID: int, winnerID: int, loserID: int): void
resumeMatch(userID: int, matchID:int): boolean
getUsers(): userID1: int, userID2: int
createMatchID(): int
attemptMove(move: Move): match
getNewTurnID(match: Match, currentPlayerID: int): int
checkPieceColor(piece: ChessPiece, match: Match, currentPlayerID: int): boolean
get(accountID: int): List<Match>
setMatchStatus(matchID: int, newStatus: string): int
createBoard(matchID: int): string
getMatch(matchID: int): Match
stringToObject(boardString: string): ChessBoard
<<interface>> MatchDAO
createMatch(userID1: int, userID2: int, matchID: int):boolean
abandonMatch(matchID: int, winnerID: int, loserID: int): void
resumeMatch(userID: int, matchID:int): boolean
getUsers(): userID1: int, userID2: int
createMatchID(): int
attemptMove(move: Move): match
getNewTurnID(match: Match, currentPlayerID: int): int
checkPieceColor(piece: ChessPiece, match: Match, currentPlayerID: int): boolean
get(accountID: int): List<Match>
setMatchStatus(matchID: int, newStatus: string): int
createBoard(matchID: int): string
getMatch(matchID: int): Match
stringToObject(boardString: string): ChessBoard
MatchDAOImplementation
- entityManger: EntityManager
createMatch(userID1: int, userID2: int, matchID: int):boolean
abandonMatch(matchID: int, winnerID: int, loserID: int): void
resumeMatch(userID: int, matchID:int): boolean
getUsers(): userID1: int, userID2: int
createMatchID(): int
attemptMove(move: Move): match
getNewTurnID(match: Match, currentPlayerID: int): int
checkPieceColor(piece: ChessPiece, match: Match, currentPlayerID: int): boolean
get(accountID: int): List<Match>
setMatchStatus(matchID: int, newStatus: string): int
createBoard(matchID: int): string
getMatch(matchID: int): Match
stringToObject(boardString: string): ChessBoard
Move
- fromPosition: String
- toPosition: String
- matchId: Integer
- playerId: Integer
<<constructor>> Move(String, String, Integer, Integer)
<<constructor>> Move()
getFromPosition(): String
getToPosition(): String
getMatchId(): Integer
getPlayerId(): Integer
MoveAnalyzer
- currentPlayerID: Integer
- match: Match
- board: ChessBoard
- move: Move
- color: ChessPiece.Color
<<constructor>> MoveAnalyzer(Match, ChessBoard, Move)
checkPreconditions(): Void
movedIntoCheck(): Boolean
willEndGame(): Boolean
checkPieceColor(ChessPiece): Boolean
getColor(Integer): ChessPiece.Color
getEnemyColor(Integer): ChessPiece.Color
findKing(ChessPiece.Color, ChessBoard): String
canBeCaptured(ChessPiece.Color, String, ChessBoard): Boolean
opponentIsInCheck(ChessBoard): Boolean
opponentIsMated(): Boolean
createPositionString(int,int): String
stringToObject(String): ChessBoard

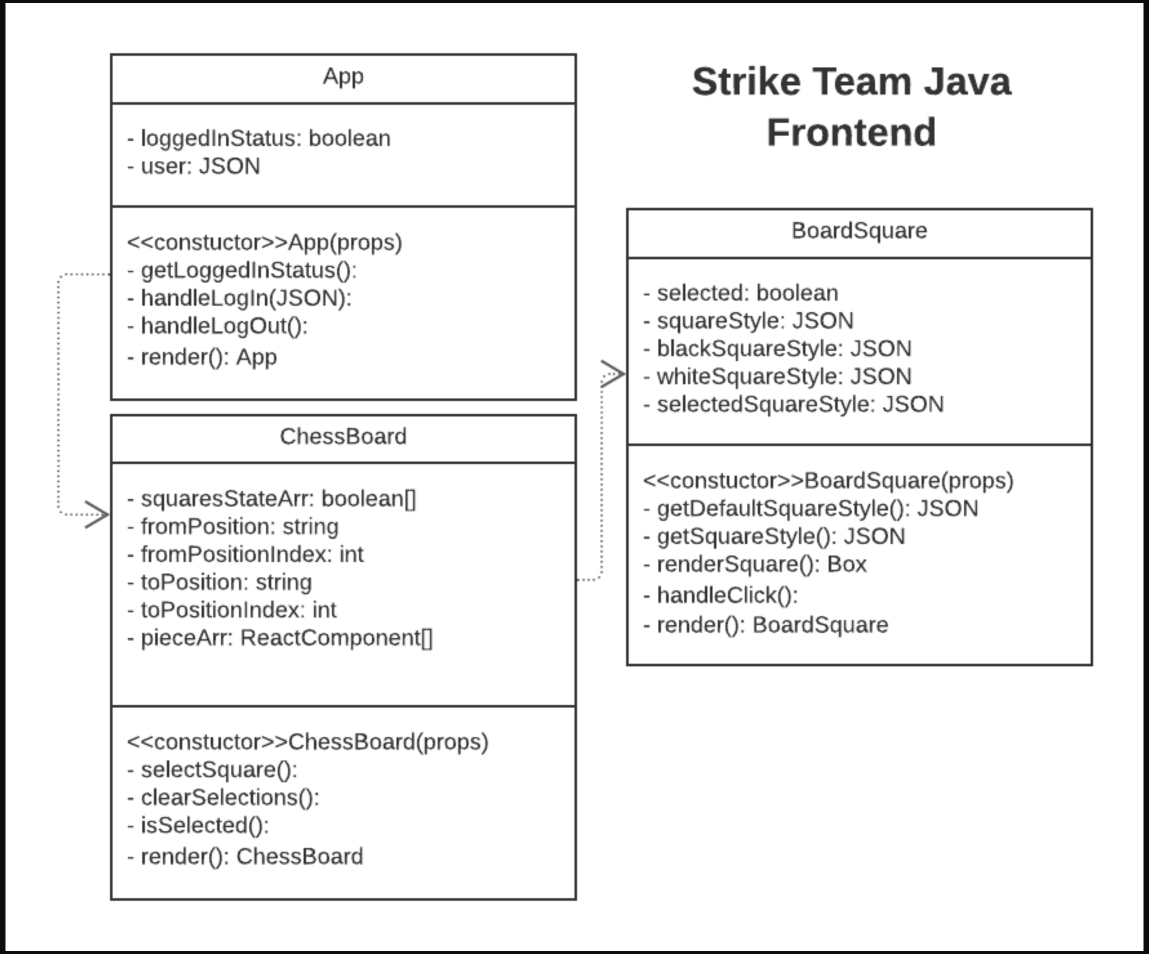
Database and Frontend Diagrams



Accounts	
ID(PK)	int(11)
email	varchar(255)
username	varchar(255)
games_played	int(11)
games_won	int(11)

Matches	
ID(PK)	int(11)
senderID(FK)	int(11)
receiverID(FK)	int(11)
status	varchar(255)
winner	int(11)
loser	int(11)
board	longtext
turnID	int(11)
start_time	timestamp
end_time	timestamp
sender_check	tinyint(1)
receiver_check	tinyint(1)
castling_moves	int(11)

Notifications	
ID(PK)	int(11)
senderID(FK)	varchar(255)
receiverID(FK)	varchar(255)
message	int(11)
matchID(FK)	int(11)



App

- loggedInStatus: boolean
- user: JSON

<<constuctor>>App(props)
- getLoggedInStatus():
- handleLogIn(JSON):
- handleLogOut():
- render(): App

ChessBoard

- squaresStateArr: boolean[]
- fromPosition: string
- fromPositionIndex: int
- toPosition: string
- toPositionIndex: int
- pieceArr: ReactComponent[]

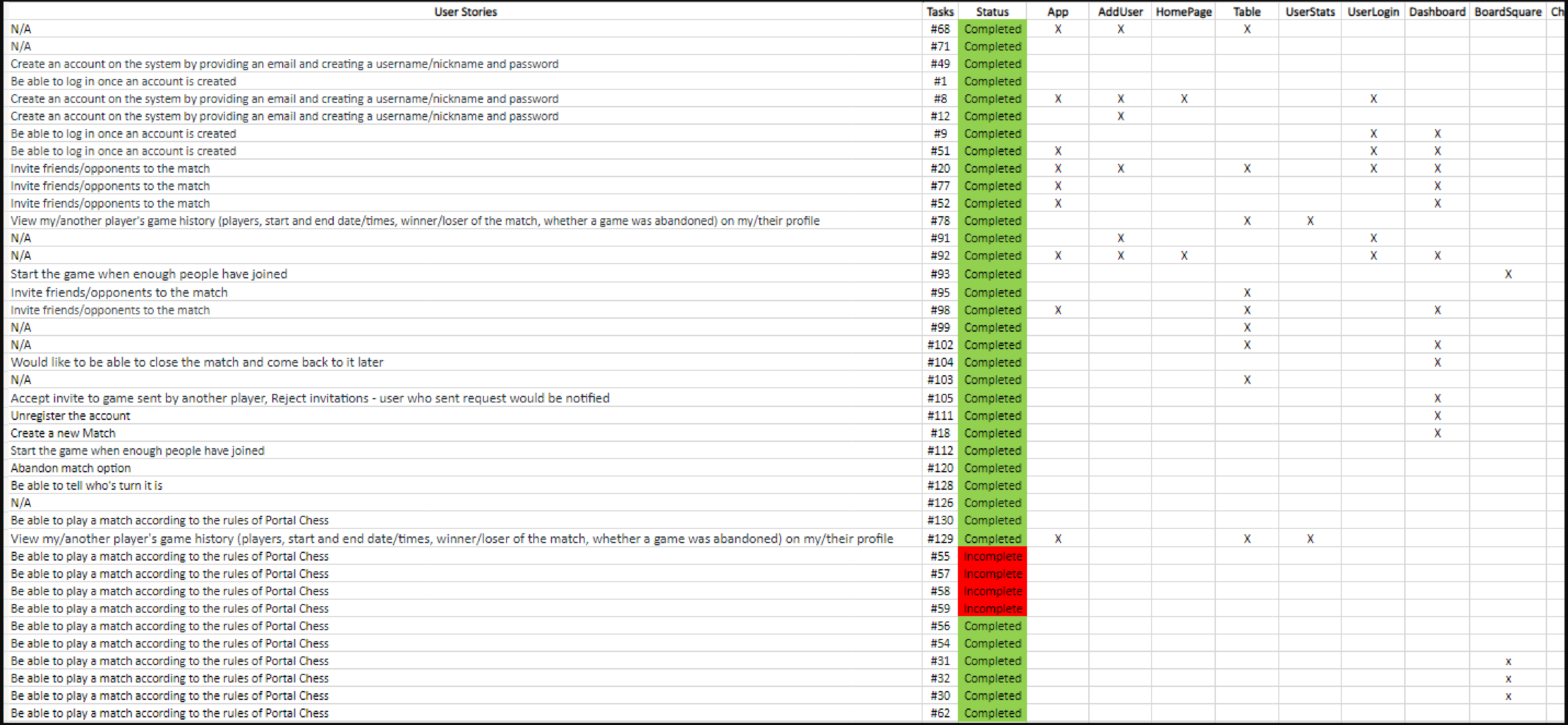
<<constuctor>>ChessBoard(props)
- selectSquare():
- clearSelections():
- isSelected():
- render(): ChessBoard

BoardSquare

- selected: boolean
- squareStyle: JSON
- blackSquareStyle: JSON
- whiteSquareStyle: JSON
- selectedSquareStyle: JSON

<<constuctor>>BoardSquare(props)
- getDefaultSquareStyle(): JSON
- getSquareStyle(): JSON
- renderSquare(): Box
- handleClick():
- render(): BoardSquare

Traceability link matrix



User Stories	Tasks	Status	App	AddUser	HomePage	Table	UserStats	UserLogin	Dashboard	BoardSquare	Ch
N/A	#68	Completed	X	X		X					
N/A	#71	Completed									
Create an account on the system by providing an email and creating a username/nickname and password	#49	Completed									
Be able to log in once an account is created	#1	Completed									
Create an account on the system by providing an email and creating a username/nickname and password	#8	Completed	X	X	X			X			
Create an account on the system by providing an email and creating a username/nickname and password	#12	Completed		X							
Be able to log in once an account is created	#9	Completed						X	X		
Be able to log in once an account is created	#51	Completed	X					X	X		
Invite friends/opponents to the match	#20	Completed	X	X		X		X	X		
Invite friends/opponents to the match	#77	Completed	X						X		
Invite friends/opponents to the match	#52	Completed	X						X		
View my/another player's game history (players, start and end date/times, winner/loser of the match, whether a game was abandoned) on my/their profile	#78	Completed				X	X				
N/A	#91	Completed		X				X			
N/A	#92	Completed	X	X	X			X	X		
Start the game when enough people have joined	#93	Completed								X	
Invite friends/opponents to the match	#95	Completed				X					
Invite friends/opponents to the match	#98	Completed	X			X			X		
N/A	#99	Completed				X					
N/A	#102	Completed				X			X		
Would like to be able to close the match and come back to it later	#104	Completed							X		
N/A	#103	Completed				X					
Accept invite to game sent by another player, Reject invitations - user who sent request would be notified	#105	Completed							X		
Unregister the account	#111	Completed							X		
Create a new Match	#18	Completed							X		
Start the game when enough people have joined	#112	Completed									
Abandon match option	#120	Completed									
Be able to tell who's turn it is	#128	Completed									
N/A	#126	Completed									
Be able to play a match according to the rules of Portal Chess	#130	Completed									
View my/another player's game history (players, start and end date/times, winner/loser of the match, whether a game was abandoned) on my/their profile	#129	Completed	X			X	X				
Be able to play a match according to the rules of Portal Chess	#55	Incomplete									
Be able to play a match according to the rules of Portal Chess	#57	Incomplete									
Be able to play a match according to the rules of Portal Chess	#58	Incomplete									
Be able to play a match according to the rules of Portal Chess	#59	Incomplete									
Be able to play a match according to the rules of Portal Chess	#56	Completed									
Be able to play a match according to the rules of Portal Chess	#54	Completed									
Be able to play a match according to the rules of Portal Chess	#31	Completed								x	
Be able to play a match according to the rules of Portal Chess	#32	Completed								x	
Be able to play a match according to the rules of Portal Chess	#30	Completed								x	
Be able to play a match according to the rules of Portal Chess	#62	Completed									

Wiki Address

- https://github.com/mdcham/cs414-f20-Strike-Team-Java/wiki/P3

Demo