Zachary Stevens
Project Proposal
Connect 4

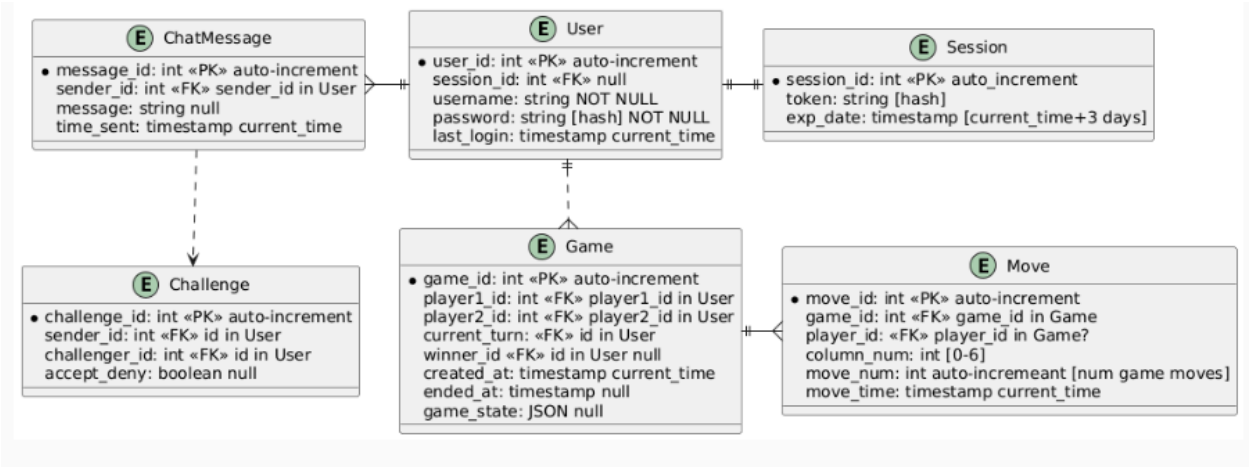
Description:

Connect 4 is a turn-based, 2 player game. Each player will take turns dropping their colored piece onto a 6 rows by 7 column board. The player's color will drop to the first available row (gravity). Once a move has occurred, the move cannot be changed. The first player to obtain 4 of their colored pieces in a row wins the game.

Technology Description:

Normal HTML, CSS, and JavaScript will be used for the frontend due to its easy use for data fetching using AJAX and SVG rendering. Node.js with Express will be used in the backend along with a MySQL database to manage the users, game state, etc. Also, Node.js can handle WebSockets that will be used once a connection for a game has been confirmed which will make playing the game and communication smoother. Sessions will be managed by creating a token using things like IP address, username, etc.and saving the hash along with an expiration date for 3 days from now. When someone completes the login, the checksum for the session will be updated and moving from the lobby page with the messages to the game page, the session will be validated.

ER Diagram



Session to User is a one to one relationship, in order for a session to exist a user must exist, but a session_id in user can be null in the case of an account being manually put into the database or if the user were to logout. Sessions will be generated every login. A User can send multiple chat messages, in order for a ChatMessage to exist, the user must exist. A Challenge I have labeled as a ChatMessage mainly to see that a challenge will be sent in the chat to the challenger. There must be 2 Users for every Game, a Game cannot have more or less than that. In the future, a solution to adding bots could be unique, set player_ids for the bots. In Game there is a field called game_state, game_state will be a simple JSON storage of all of the moves (most likely color, column and list from latest to most recent). Moves are stored in the database

alongside of the game_state field, this is made as a backup in case the JSON fails or if the Moves fail the JSON can load the moves. Obviously, exactly one game has many moves, and a move cannot exist without a game.

API Documentation

Users:

/createaccount
POST
Request: {username: “String”,
password: “HashedString”}
Response: 201 - {message: “New account made message.“}
400 - {error: “Account could not be created message.””}

/login
POST
Request: {username: “String”,
password: “HashedString”}
Response: 201 - {message: “logged in”}
400 - {error: “username or password is incorrect”}

Sessions:

session/validate
POST
Request: {user_id: Int,
session_token: “HashedString”
current_time: timestamp}
Response: 200 - {message: “Valid”}
400 - {error: “Invalid Token”}
403 - {error: “Session Expired”}

session/update
POST (not updating the old session)
Request: {user_id: Int,
session_token: “HashedString”
current_time: timestamp}
Response: 200 - {message: “Session Updated”}
400 - {error: “Session Not Updated”}

Game:

game/gamestate
GET
Request: {game_id: int}
Response: 200 - {message: game.gamestate}
400 - {error: “game was not found”}

game/opponent
GET
Request: {user_id: int
Game_id: int}
Response: 200 - {message: user.username}
400 - {error: “user was not found”}

Move:

move/create
POST
Request:{game_id: int,
player_id: int,
column_num: int [0-6]}
Response: 201 - {message: "Move created successfully" }
400 - {error: "Invalid move" }

move/history
GET
Request: {game_id: int}
Response: 200 - {message: [{ move_id: int, player_id: int, column_num: int,
move_num: int, move_time: timestamp }] }
400 - {error: "Game not found or no moves available" }

ChatMessage:

chat/sendmessage
POST
Request:{user_id": int,
message: "String”}
Response: 201 - {message: "Message sent successfully" }
400 - {error: "Message could not be sent" }

chat/history
GET
Request: {user_id: int}
Response: 200 - { message: [{ message_id: int, sender_id: int, message:"String",
time_sent: timestamp }] }
401 - {error: "Not able to access messages" }

Challenge:

challenge/send
POST

Request: {user_id: int,
challenger_username: “string”}
Response: 201 - {message: "Challenge sent successfully" }
400 - {error: "Challenge could not be sent" }

challenge/reply
POST
Request: {user_id: int
challenge_id: int
accept_dent: boolean
Response: 200 - {message: "Response recorded" }
400 - {error: "Challenge response failed" }

Logout:

logout/
PUT (changing the session_id in user to null)
Request: {user_id: int}
Response: 200 - {message: “Logged out successful‘}
400 - {error: “Logout unsuccessful”}