Tic Tackity Toe

Server

General

A server code is included for this assignment. The server is a socket.io server which acts as a middleman for the communications between multiple clients. There is a **README.md** document within the root of the server folder which could be useful to understand better how the server works.

Events

These are the types of listeners setup on the server:

- **users** - *Sends a list of users back to the user originally emitting the message*
- **matches** - *Sends a list of matches back to the user originally emitting the message*
- **game_challenge** - *Sends a game challenge to a user being challenged. Information about the challenger is included in the message.*
- **game_challenge_accepted** - *Notifies the user which originally challenged that his request has been accepted*
- **game_challenge_declined** - *Notifies the user which originally challenged that his request has been declined*
- **ready** - *Both users participating in a match should emit the **'ready'** event to notify that they are ready to play and as a result the **'assign_symbol'** event is emitted to the user which contains information about which symbol (**X** or **O**) he will play in this match*
- **game_move** - *When a user makes a move within the Tic Tac Toe he should emit the **'game_move'** event which notifies the other player within the match that he has made a move*
- **leave** - *When a user decides to leave he should emit a **'leave'** event notifying other connected users that he has left and therefore his nick is no longer occupied*

These are the types of events the server sends back to the clients:

- **users** - *Includes the list of users connected to the server*
- **matches** - *Includes the list of matches currently in playing or previous*
- **game_challenge** - *Forwards the game challenge to the user being challenged*
- **game_challenge_accepted** - *Forwards the acceptance to the user which challenged*
- **game_challenge_declined** - *Forwards the decline to the user which challenged*
- **assign_symbol** - *Sends the symbol to the user which originally emitted the **'ready'** event*
- **game_move** - *Notifies users within a match what the game move was*

- **new_match** - *Sends information to all connected users that a new match has been added*
- **match_ended** - *Sends information to all connected users that a match has ended and is ready to be updated*
- **connected_user** - *Sends information regarding the newly connected user to all connected users*
- **disconnected_user** - *Sends information regarding the newly disconnected user to all connected users*
- **user_left** - *Sends information to all connected users that a user has left and therefore can be removed from the active users list, his nick is no longer occupied*
- **session** - *Sends the session information to a user which connected, this is information which should be later on used to reconnect to the server. It would be a good idea to store these information within local storage*
- **connect_error** - *Sends the connect error when the server sends an Error() object, a connection error occurred or the nickname is occupied or not correctly provided*

Client

In this assignment, you should write a Tic-Tac-Toe game using React and socket.io. Below is an enlisting of all the functionality for this assignment:

- **(10%)** The welcome view (*available at /*) should display:
  - **(5%)** The user should upon arrival specify his/her nickname. If the nickname is free, i.e. no other user is currently active with the same nickname, he/she can proceed, otherwise a new nickname must be provided. *Note: no password is required, only nickname*
  - **(5%)** After the user has selected a nickname he should connect the socket and be redirected to a dashboard view
- **(10%)** The dashboard view (*available at /dashboard*) should display:
  - **(5%)** A list of all active and past matches
    - Each item should display:
      - Opponents
      - Status of the match (ongoing or finished)
      - The winner
  - **(5%)** A list of all connected users
    - Each item should display:
      - Nickname of the user
      - Connection status (connected or disconnected)

- **(15%)** Live updates within the dashboard view
    - **(2.5%)** When a new user connects, he should be added to the list of active users
    - **(2.5%)** When a user disconnects (not leave), he should show a status of disconnected
    - **(2.5%)** When a user leaves, he should be removed from the list of active users
    - **(2.5%)** When a match is created it should appear in the list of matches
    - **(5%)** When a match ends it should be updated within the list of matches, changing the status to finished and showing the winner of the match
- **(10%)** Challenging another player to a match
    - **(2.5%)** A user should be able to challenge another connected user to a game of Tic-Tac-Toe, which results in that user being notified and he can either accept or decline the match offer
    - **(2.5%)** A user should be able to accept the offer from another player which results in a notification to the other player
    - **(2.5%)** When a user accepts a match from another user he must create a unique identifier which will be the match id (*uuid is a good choice* **https://www.npmjs.com/package/uuid**) and send back to the server that he accepts the challenge along with the unique identifier and navigate to the match view. The user which originally challenged will receive a notification that his request was accepted and he will be redirected to the match view
    - **(2.5%)** A user should be able to decline the offer from another player which results in a notification to the other player
- **(30%)** Game of Tic-Tac-Toe (*available at /match/:id*)
    - **(15%)** A visual implementation of the Tic-Tac-Toe game
        - **(5%)** A 3 by 3 grid
        - **(5%)** Each player playing either X or O and it should be visible which symbol each player is playing during the match
        - **(5%)** First one to get three consecutive symbols in a horizontal, vertical or diagonal line wins
    - **(10%)** Live communications during the game (*See* **diagram** *for better understanding*)
        - **(2.5%)** When each player enters the match room he must emit the **'ready'** event in order to receive his symbol which he will play during this match
        - **(7.5%)** When each player makes a move he must emit the **'game_move'** event in order to notify the other player that he has made a move. This goes back and forth until the game ends with a game winning move
    - **(2.5%)** When the game ends both players should see who the winner of the match is and have an option to go back to the dashboard

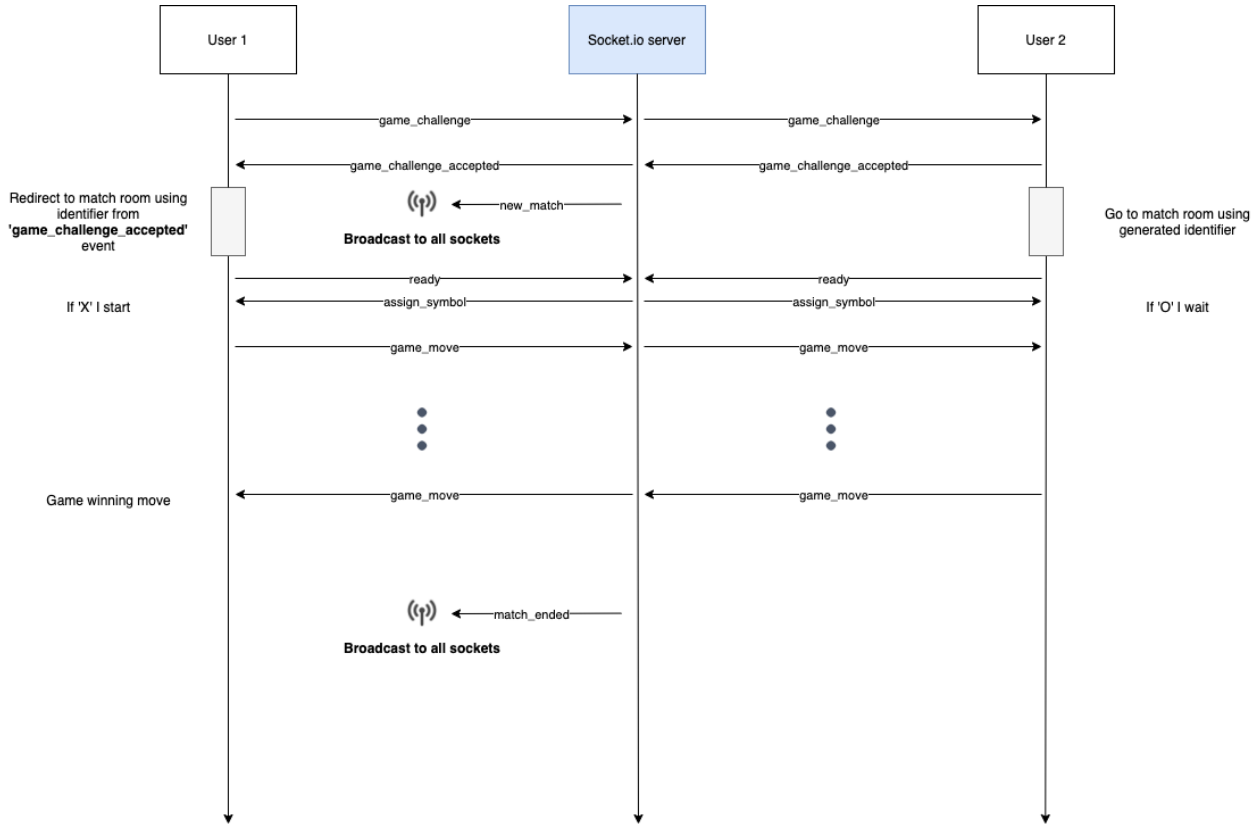
- **(2.5%)** When it is not the user's turn he should not be able to make a move and this also applies to both players once the game has ended

- **(10%)** Authorization
    - **(5%)** A higher-order component should be setup which encapsulates the functionality for the views which should be authorized. The authorized views are **/dashboard** and **/match/:matchId**
        - On mount it should check whether the user has an active session (*the session can be retrieved through **Redux***)
            - If he does have an active session
                - Reconnect to the server
                - Provide the session id
                - Dispatch new session information
                - Display wrapped component
            - If not, redirect back to the welcome view (**/**)
    - **(5%)** A higher-order component should be setup which encapsulates the functionality for the views which should not be authorized. The unauthorized view is **/**
        - On mount it should check whether the user has an active session (*the session can be retrieved through **Redux***)
            - If he does have an active session
                - Redirect to the dashboard view (**/dashboard**)
            - If not, display the wrapped component
- **(5%)** Navigation bar
    - **(2%)** The nickname of the user should be visible
    - **(3%)** A button which allows the user to leave should be visible
        - On click
            - Emit the **'leave'** event
            - Disconnect the socket
            - Remove the session information
            - Redirect back to the welcome view (**/**)

- **(10%)** Structure
  - **(5%)** All state which is truly global should reside in the **Redux** store state - therefore **Redux** must be setup in this project and used accordingly
  - **(2.5%)** Each component should reside in a single folder, where the implementation of the component is and all styles for that component
  - **(2.5%)** Views and components should follow **S**ingle **R**esponsibility **P**rinciple and therefore shouldn't be packed with unrelated logic

Diagram

This diagram shows how communications take place before a match of Tic-Tac-Toe begins and until it finishes, in regard to socket.io



Penalties

If the **node_modules/** is a part of the submission there will be a penalty of -1 on the total grade of the assignment. Please don't forget this, as there will be no chances after the submission date has passed.

Dependencies

All dependencies which are used to assist your implementation are allowed in this assignment. You cannot use dependencies that actually do the main work for you.

Submission

A single compressed file (***.zip, *.rar**) should be submitted in **Canvas**. Exclude the server code.