event (emit)	where	listener (on)	where
[user goes on website]	Front	io.on('connection') → sets up listeners for all socket events	Back
[user enters name & clicks button] socket.emit('newPlayer', name)] → sends name entered to server	Front splash.js	socket.on('newPlayer', (name) => { → gets name from front end → creates new player object w/ received name + score of 0 → add new player to players array → tell player/user to view game page socket.emit('startGame') → update frontend w/ username and array index socket.emit('updateUser', {name: player.name, index: players.length - 1}) → update all players w/ current list of players and their scores io.emit('updateScores', players) → tell all users who the current judge is (name + index in array) & gif to be judged io.emit('newJudge', {name: currentJudge.name, index: currentJudgeIndex, currGif: currGif}) })	Back
socket.emit('startGame') → tell frontend to start the game	Back	socket.on('startGame', () => { → updates 'gameOn' state to true, causing <mainGame/> to render })	Front app.js
socket.emit('updateUser', {name: player.name, index: players.length - 1}) → tell frontend to update "global" (context) player variable w/ player's name and index	Back	socket.on('updateUser', (playerInfo) => { → update player object state w/ name + index received from server [will be shared w/ other components via context API] })	Front app.js
io.emit('updateScores', players) → give frontend of all players the array of players and their scores	Back	socket.on('updateScores', players => { → update state 'playerScores' w/ list of players + their scores })	Front Scoreboard.js

event (emit)	where	listener (on)	where
io.emit('newJudge', { name: currentJudge.name, index: currentJudgeIndex, currGif: currGif }) → tell front end of all players the name & index of the judge & the gif url to be judged	BACK	socket.on('newJudge', (judge) => { → updates 'currJudge' state with the name of the judge received from server → updates 'currGif' state with the link of the current gif → compare judge index (from server) w/ player's index (stored in playerContext) → if same (player is judge) set 'isJudge' state to true set 'isInputting' state to false set 'isWaiting' state to true set 'viewInputs' state to false → if not (player is not judge) set 'isJudge' state to false set 'isInputting' state to true set 'isWaiting' state to false set 'viewInputs' state to false → these states are used to 1. conditionally render components or 2. store variables received from server	Front MainGame.js
[user types in input box & submits] socket.emit('newInput', { input: input, index: player.index }) → send input phrase & player's index to backend to be stored in roundInputs array	Front SoloInput.js	socket.on('newInput', userInput => { → push player's index & input (received from front end) into roundInputs array → tell user's front end to be waiting socket.emit('waiting') → check if all user inputs have been submitted → if all players have submitted inform all users it's time to judge the round, sending all player index & inputs to front end io.emit('judgeTheRound', roundInputs)	BACK
socket.emit('waiting') → tell user's front end to be waiting	Back	socket.on('waiting', () => { → update 'isInputting' state to false → update 'isWaiting' state to true → these states are used to conditionally render components }	Front MainGame.js

event (emit)	where	listener (on)	where
io.emit('judgeTheRound', roundInputs) → if all players have submitted, inform all users it's time to judge the round, sending all player index & inputs to front end	BACK	socket.on('judgeTheRound', (allInputs) => { → receives player indices & phrases from server & updates state 'submittedInputs' → updates 'isWaiting' state to false → updates 'viewInputs' state to true → these states are used to 1. conditionally render components or 2. store variables received from server })	Front MainGame.js
socket.emit('roundWinnerChosen', { winnerIndex: props.playerIndex, winningPhrase: props.input }) → inform server that a phrase has been chosen & the corresponding winner's index & phrase (so that server can tell all other players also)	Front Submission.js	socket.on('roundWinnerChosen', (roundWinner) => { → get the player object associated with the winnerIndex from front end → increase that players score by 1 → get winners name → tell all players that a winner has been chosen & send winners name + phrase io.emit('roundWinnerChosen', winningInput) → tell front end of all players to update score board and send all player's names and scores io.emit('updateScores', players) → request a new gif from giphy & update currGif variable w/ new link → wait [10] seconds → check winning players score → if score >= [2] //changes → tell all players that game has ended & send winner's name io.emit('endGame', winningPlayer) → clear players array → clear roundInputs array → set currJudge index back to first player who will enter the room → if score < [2] → clear roundInputs array → update currJudge index by adding 1 (move to next player) → get player object based on currJudge index	Back

→ tell all players there is a new judge and send judge's name & index and the currGif link to be judged

```
io.emit('newJudge',
  { name: currentJudge.name
    index: currentJudgeIndex
    currGif: currGif
  }
)
```

}

```
io.emit('roundWinnerChosen', winningInput)
```
→ tell all players that a winner has been chosen & send winners name + phrase

BACK

```
socket.on('roundWinnerChosen', (winner) => {
```
→ receive round winner's name and phrase from server
→ update 'roundWinner' state with winner info
→ update 'viewInputs' state to false ⇒ causes rendering of WinRound component

}

Front
MainGame.js

```
io.emit('endGame', winningPlayer)
```

BACK

```
socket.on('endGame', (winner) => {
```
→ receive winner's name from server & update 'winner' state
→ update 'gameOn' state to false
→ update 'gameEnd' state to true
⇒ causes rendering of EndGame component

Front
APP.JS