**Hot Potato**

In this project, you will have the opportunity to practice using WebSockets to create a four-player hot potato game. The logic of the game is as follows:

* When a new player joins the server, they will be assigned an avatar indicated by the star with the text “You”.
* The game requires exactly 4 players to begin. No more, no less. When the 4th player joins, the game starts
* When the game starts, one player will be chosen randomly to hold the potato and the clock will begin counting down from 30.
* When the clock strikes 0, the player holding the potato loses, so pass it quickly!
* If your avatar is holding the potato, click on another player’s avatar to pass the potato.

To get you started, we’ve provided some starter code with the basic application structure:

* **public/index.html**, where you will write code to run in the browser
* **server.js**, where you will write code to run on the server
* **utils/constants.js**, where constants for each message type have been defined for you

Because the focus of this project is on WebSockets (and not on software engineering more broadly), we’ve encapsulated the majority of the game logic and DOM manipulation in various helper functions.

Your task is to implement the WebSocket logic to pass messages between the server and clients and to call the provided game logic helper functions in response to these messages. In addition to reinforcing the mechanics of connecting a WebSocket server and client, this project will help demonstrate the process of designing a system of messages and payloads sent between clients and the server.

**Setup Instructions**

This project should be completed on your own computer instead of on Codecademy. You can download what you’ll need by clicking the “Download” button below. If you need help setting up your computer, read our [article about setting up a text editor for HTML/CSS development](https://www.codecademy.com/articles/visual-studio-code).

Once you’ve downloaded the project, open up the project folder in your text editor. Then, [use the command line](https://www.codecademy.com/learn/learn-the-command-line" \t "_blank) to navigate to the root of the starter code directory, and run the following commands:

npm install  
node server.js

If these commands are successful, you should see the message Listening on: http://localhost:8080 displayed in your terminal.

Navigate to [http://localhost:8080](http://localhost:8080/" \t "_blank) in your browser to view the game and then click on “Start”!

**Getting Started**

**1.**

Before diving in, spend some time familiarizing yourself with the code we’ve provided. First, take a look at **public/index.html** where we’ve defined the variables and helper functions below for you:

VARIABLES:

* + wsClient will store an instance of the browser’s WebSocket client.
  + clientPlayerIndex will store a number assigned to a player when the WebSocket server accepts that player into a game (0 through 3).
  + potatoHolderIndex will store the index of the player currently holding the potato (eg. the player with clientPlayerIndex = 1 is holding the potato if (and only if) potatoHolderIndex = 1.

WS LOGIC:

* + init() contains all logic to initialize the WebSocket client and define its responses to WebSocket events.
  + passThePotatoTo() updates the current potatoHolderIndex and sends the data to the server.

DOM HELPER FUNCTIONS (you will not need to modify these functions):

* + updateDisplay() updates the text shown in the center of the screen. Used by a number of the helper functions below.
  + setPlayerIndex() assigns each client’s clientPlayerIndex and renders the star next to their avatar.
  + updateCurrentPotatoHolder() updates the potatoHolderIndex and updates the image of the potato holder.
  + countDown() renders the current time with a different color as the clock counts down.
  + endGame() displays a game over message.

**2.**

Next, take a look at **server.js** where we’ve defined the variables and helper functions below for you:

VARIABLES:

* + nextPlayerIndex is used to provide the clientPlayerIndex for the next player to join and can be used to keep track of the number of players in the game.

HELPER FUNCTIONS:

* + handleNewUser() determines what to do when a new player joins the server. Until there are 4 players in the game, the server will accept a new client into the game by sending them a clientPlayerIndex value. Once there are 4 players in the game, a random player will be assigned to hold the potato and the game will start. If there are 4 or more players in the game, the server will let them know that the game is full.
  + passPotatoTo() should broadcast to all clients the newPotatoHolderIndex when a player passes the potato.
  + startTimer() starts a timer interval that “ticks” every 1 second from 30 down to 0. It should broadcast the current time to each client connected to the server and notify all players when the game is over.

**3.**

Finally, take a look at **utils/constants.js** where the various message types used in this application are defined:

CLIENT

* + .MESSAGE.NEW\_USER is sent from the client to the server when the client joins the server
  + .MESSAGE.PASS\_POTATO is sent from the client to the server when the client passes the potato to another player.

SERVER

* + .MESSAGE.PLAYER\_ASSIGNMENT is sent from the server to a single client when the client joins the server
  + .MESSAGE.GAME\_FULL is sent from the server to a single client when they attempt to join a full game
  + .BROADCAST.COUNTDOWN is broadcast to all clients each time the timer ticks
  + .BROADCAST.NEW\_POTATO\_HOLDER is broadcast to all clients when the potato is passed
  + .BROADCAST.GAME\_OVER is broadcast to all clients when the game ends (the timer reaches 0)

**Create a WebSocket Server**

**4.**

Now that we have the starter code set up, let’s begin implementing the WebSocket logic. First, in the WS LOGIC section of **server.js**, declare a constant variable called wsServer. Then create a new instance of the WebSocket.Server class and store it in wsServer.

You should use the provided server as the HTTP server that the WebSocket connection will be made over.

Hint

You will need to use the WebSocket.Server class exported by the ws package. The constructor for this class accepts an object with a server property whose value should be the server created above.

Your code may look like this:

const option = { server: myHttpServer };  
const myWebSocketServer = new WebSocket.Server(options);

**5.**

Prepare your newly created server to respond to client 'connection' events by calling the .on() method on your wsServer.

For now, the server should print out the message 'A new client has joined the server' each time the 'connection' event is detected.

Hint

Remember, the .on() method accepts two arguments:

* + a string specifying an event (in this case, 'connection') to respond to
  + a callback that will run whenever that event occurs. The callback will receive one argument—socket— corresponding to a single server/client connection.

Your code may look like this:

myWebSocketServer.on('connection', (socket) => {  
  // respond to connection events here  
});