Assignment 1
CS-455
Spring 2021

Due Date: 4/9/2021 at 11:59 pm (No extensions)
You may work in groups of 4

Goals:

1. **Broader goal:** examine concepts of web application technologies and security issues associated with them.
2. **Specific goals:** to provide introduction to:
   (a) HTTP protocol
   (b) Web application architecture: front-end vs back-end
   (c) Using JavaScript `strict-mode`
   (d) Using JavaScript objects
   (e) Web application development using secure coding practices.
   (f) Encoding and transmission of web data using `GET` and `POST` methods
   (g) To practice all of the above by implementing a rock-paper-scissors game using Node.js

Overview

You shall be implementing a simple Rock Paper Scissors game using Node.js. Doing so will familiarize you with handling HTTP requests and responses and with submitting form data using HTTP `GET` and `POST` methods. Furthermore, your implementation shall feature all secure coding practices covered in class.

Web App Specifications

Your web app architecture shall comprise a front-end and back-end. The front- and back-end components shall have the following functions:

Front-end: When the user navigates to the homepage, the homepage will display a form consisting of three radio buttons:

- Rock
- Paper
- Scissors

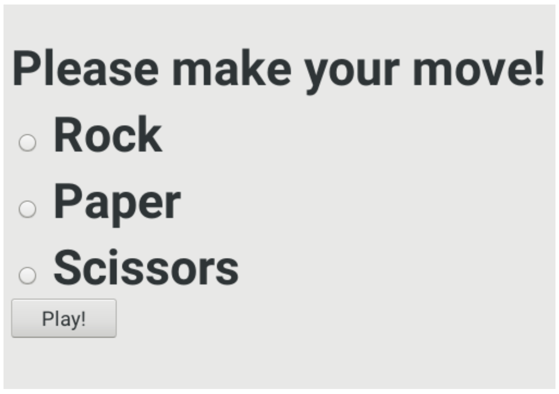


Figure 1: The homepage

and a `Play!` push button. Figure 1 shows an example. The player shall make a choice and click `Play!`, which will use the `POST` method to send the choice to the Node.js server.

The Back-end: The back-end shall consist of the Node.js server listening on the specified port. The server shall do the following:

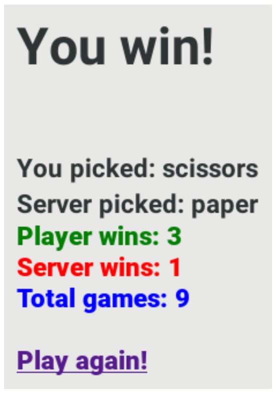
1. Receive the HTTP request
2. Extract the player's choice from the body of the message
3. Randomly choose rock, paper, or scissors
4. Compare the server's choice to the player's choice and apply the rules of Rock Paper Scissors to determine if the player wins, the server wins, or whether the game is a draw
5. Dynamically generate a page consisting of:
   - The result of the game
   - The server's choice
   - The player's choice
   - The number of player wins
   - The number of server wins
   - The total number of games played since the server was started
   - A link back to the home page allowing the player to play another game.

The Figure 2 gives an example of how the page should look.

The server shall be invoked using `node rps.js`. The homepage should be accessible from the front-end by typing `http://localhost:3000` into the browser URL bar.

Coding Requirements

The following is a list of the specific coding requirements:


You win!

You picked: scissors
Server picked: paper
Player wins: 3
Server wins: 1
Total games: 9

Play again!

Figure 2: The game outcome

1. Your code must use the express framework in order to implement all communications and must use the `ejs` templating engine for generating pages.

2. Your back-end implementation must be encapsulated in the class object that stores the state of the game. The object shall store the number of player wins and losses, choices of the player and server, all functions necessary for handling the HTTP requests and responses, and any other needed functionality.

   The game server is brought up by creating an instance of your game class. It must be done using the syntax similar to:

```
let myGame = new RPS(3000);
```

   where `RPS()` is the constructor for the game class and 3000 is a sample port number on which the HTTP server will listen for requests.

3. The back-end implementation must feature all the secure coding practices covered in class. These include:

   - Ensuring correct arithmetic and preventing effects of IEEE 754 encoding (where applicable).
   - Properly testing for `Infinity`, `NaN`, etc. (where applicable).
   - Proper parsing of strings.
   - Using `let` or `const` instead of `var`.
   - Properly converting strings to integers and/or floats.
   - Using `===` instead of `==`.
   - Using object prototypes in order to avoid unnecessary code duplication across instances.
   - Using strict mode correctly.
   - To protect the integrity of your object and help enforce encapsulation, restrict the modification of properties that are expected to remain constant. These include properties such as the port number, template names, file paths, etc. To achieve this, please make sure to set the `writable`, `enumerable`, and `configurable` property descriptors as appropriate.

- Prohibiting external addition and deletion of properties to/from the object.

Tips and Resources

- You can set property descriptors and then choose to freeze or seal or prevent extensions to the object (please use the one that is most appropriate) using the approach similar to the one below:

```
.
.
.
function RPS(....)
{
  Object.defineProperty(...); // Define the first property
  Object.defineProperty(...); // Define the second property
  .
  .
  .
  Object.seal/freeze/preventExtensions() // Lock the object as appropriate
                    // using seal, freeze, or preventExtensions.
                    // Please use the method that is most appropriate
                    // for your design and meeting the above-stated
                    // requirements.
}
.
.
.
```

- Rules of Rock Paper Scissors https://bit.ly/2DycjNI
- HTML forms and radio buttons https://bit.ly/2gmUh9x
- Generating a random number in JavaScript https://bit.ly/2LWju4M
- To see examples of handling `POST` requests and other Node.js basics please see the sample codes from the Node.js lecture https://bit.ly/2GCpUa2

BONUS:

Extend your game to allow the user to:

1. Prior to commencing a round of the game, the user can enter a chosen user name and the number of games they would like to play in this round.
2. After the round is complete, the user's information and score is saved to a file. If the file already contains a user with the specified name, that record should be replaced with the current record.
3. Finally, the user is re-directed to a score-board that ranks all users according to the highest percentage of games won.

SUBMISSION GUIDELINES:

- This assignment must be completed using Node.js. You must use `express` package and `body-parser` to implement your commununications code. No other packages may be used.
- Please hand in your source code for the html page implementing the front-end and your Node.js JavaScript code. To TITANIUM.
- **If you worked in a group, only one person within each group should submit.**
- Write a README file (text file, do not submit a .doc file) which contains
  - Names and email addresses of all group members
  - How to execute your program
  - Whether you implemented the extra credit
  - Anything special about your submission that we should take note of
- Place all your files under one directory with a unique name (such as `p1-[userid]` for assignment 1, e.g. `p1-mgofman1`).
- Tar the contents of this directory using the following command. `tar cvf [directory_name].tar [directory_name]` E.g. `tar -cvf p1-mgofman1.tar p1-mgofman1/`
- Use TITANIUM to upload the tared file you created above.

Grading guideline:

- Node.js code starts without errors: 5'
- Correctly implemented front-end meeting all of the above-described requirnments: 15'
- Correctly implemented back-end meeting all of the above-described requirnments: 75'
- README file included: 5'
- BONUS: 5' points
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.