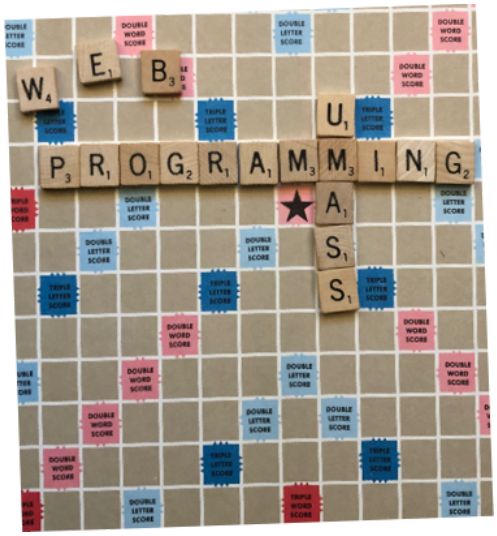
COMPSCI 326 - Web Programming
Homework 6 - Fetch & Rack - *individual assignment*
due October 25, 2021, 11pm EDT
(GitHub classroom link: https://classroom.github.com/a/1zj_owMI)



This is the sixth part of a series of assignments around the game of Scrabble. We hope that it will be a fun experience in progressively learning all pieces of modern web development, so as to engineer a fully functional game. In this assignment, you will work on incorporating previous work into the project, and starting to build player interaction.

Please submit this assignment on GitHub Classroom. There will be an automated grader that will check the functionality of your submission. It will be helpful to come up with test cases, and we encourage you to share them amongst each other; this will make everyone's code better and is actually how Quality Assurance (QA) can work in practice. However, this is an individual assignment and you **cannot share code**; submissions will be run against plagiarism detection tools. Additionally, we will be spot checking the code for good coding practices. It is expected your code **does not** contain (1) extraneous variables/code, (2) missing semicolons, (3) missing curly braces, (4) use of double equals, (5) use of `let` when a `const` would suffice, (6) use of `var`. Furthermore, you should use whitespace consistently and to make the code legible. Now that you've learned how to use ESLint, it should be easy to satisfy these requirements.

You will find a (mostly empty) template when you create your repository. You should import the files you previously used for homeworks 4 and 5. **Please do not rename any of the existing files or change the directory structure.** You are free to create more files and import them. However, you **cannot** use any external modules beyond those provided without prior permission.

1. Loading the JSON dictionary in the browser

In this first part, you will make it possible to import `scrabbleUtils.js` in your browser code. The current problem is that we cannot import the dictionary file used in various utility functions in the browser with an `import` statement (we had to replace it with a JavaScript file). To load the dictionary as a JSON file in the browser, your code will need to make an HTTP request in JavaScript to the `dictionary.json` file. This used to be fairly complex, but luckily with modern APIs, it is almost trivial: use the `fetch` API. We recommend you load the dictionary before executing anything else (like initializing the board, etc.), and set it as a global variable.

2. Rendering and updating a rack

In this second part, you will render the rack below the board. You should initialize the rack by taking 7 random tiles from the bag. You will then connect it to the controls in the following way:

- When the play button is clicked, your code should check if the word is valid (i.e., if it is in the dictionary) *and* if it can be played with the available tiles. If it cannot be played, you should tell the user (via `alert`).
- If the word can be played, the code should place it on the board (as with HW#4), remove the tiles used from the available tiles, and draw enough tiles from the bag to go back to 7 tiles.

You will find below an example of what this should look like.

3. Hints

In this last part, you will add a basic hint system. Add a third button to the controls, labelled “help” or “hint”. When clicked, it should display below the available tiles one or more hints, obtained using `bestPossibleWords`. In the example below, a random element from `bestPossibleWords` is displayed as the hint.

4. Shuffle

As you may have noticed, the shuffle function in `shuffle.js` is only used in `game.js`. It would therefore make sense to place it in that file, and keep it “private” (which in JS just means to not export it). Therefore, please move the function to `game.js`, which will also reduce the number of files we are working with.

Example board: