David Cardona

DGMD E-28 – Developing Single-Page Web Applications

Assignment 4 – Wordle

Spring 2022

**Web Links**

<https://dcardonab.github.io/DGMDE-28_SPA/assignments/A4/assignment4.html>

**API**

I used WordsAPI for my implementation. It is deployed via RapidAPI. I used it due to the vast number of words available. It is also possible to provide a query string requesting random words, as well as the number in the requested word. In my implementation, I added a second fetch request to obtain a definition so that it is displayed when the game is over. Also, by ensuring that a word has a definition, I am narrowing the word selection only to words that aren’t slangs and have formal definitions.

**What was the most satisfying part of this assignment?**

I enjoyed working with an API and reading about what parameters can be specified. The ability of retrieving information via APIs make an implementation a lot stronger. I am still to learn how to implement an API into a commercial application without disclosing the API key. Something else that I enjoyed was working with a global keyboard event handler, avoiding explicitly using an input. It was also challenging to overcome this when deploying the app in a mobile device, but I found a useful condition as a workaround to insert a text field that would display the keyboard on mobile devices.

**Code**

assignment4.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Wordle</title>

<link href="static/styles.css" rel="stylesheet">

<script type="text/javascript" src="static/script.js"></script>

</head>

<body class="bg\_dark\_gray">

<div id="container">

<h1 class="center">Wordle</h1><hr class="bottom\_margin" />

<div class="bottom\_margin center" id="messages">Choose mode!</div>

<!-- Gameplay Buttons -->

<div class="bottom\_margin center" id="buttons">

<button id="new\_game\_button">New Game</button>

<button id="new\_game\_button\_debug">New Game - Debug</button>

</div>

<!-- Mobile Controls -->

<div class="bottom\_margin center" id="mobile">

<input type="text" id="mobile\_input" placeholder="Open Keyboard">

</div>

<!-- Main Content -->

<div class="center" id="game\_content">

<div class="float\_child bottom\_margin" id="grid"></div>

<!-- Info -->

<div class="float\_child">

<!-- Keys -->

<div class="bottom\_margin">

<h3 class="center">Used Keys</h3><br />

<span id="used\_keys"></span>

</div>

<!-- Debug -->

<div id="debug">

<div class="bottom\_margin">

<h3 class="center">Word to Guess</h3>

<span id="word"></span>

</div>

<div class="bottom\_margin">

<h3 class="center">Definition</h3>

<span id="definition"></span>

</div>

</div>

<!-- Score -->

<div class="score\_box">

<div class="bottom\_margin">

<h3 class="center">Score</h3>

Total - <span id="total\_score">0</span>

</div>

<div>

<table>

<th>Number of Guesses</th>

<tr><td>1 - <span id="score\_1">0</span></td></tr>

<tr><td>2 - <span id="score\_2">0</span></td></tr>

<tr><td>3 - <span id="score\_3">0</span></td></tr>

<tr><td>4 - <span id="score\_4">0</span></td></tr>

<tr><td>5 - <span id="score\_5">0</span></td></tr>

<tr><td>6 - <span id="score\_6">0</span></td></tr>

</table>

</div>

</div>

</div>

</div>

<!-- Instructions -->

<div>

<details>

<summary class="center">Instructions</summary>

<ul>

<li>You have six tries to guess a word.</li>

<li>A green background is displayed when a letter is in the word and in the correct position.</li>

<li>A yellow background is displayed when a letter is in the word and in the wrong position.</li>

<li>A gray background is displayed when a letter is not in the word.</li>

</ul>

</details>

</div>

</div>

</body>

</html>

styles.css

body {

color: #FFFFFF;

font-family: monospace;

}

button, input[type='text'] {

background: none;

border: 2px solid #FFFFFF;

border-radius: 5px;

color: #FFFFFF;

font-family: monospace;

height: 25px;

}

li {

margin-bottom: 10px;

}

table {

margin: auto;

}

#container {

padding: 0px 50px;

}

#game\_content, #mobile {

display: none;

}

.bg\_dark\_gray {

background-color: #2c2c2c;

}

.bg\_gray {

background-color: #3b3b3b;

}

.bg\_green {

background-color: #087e1e;

}

.bg\_yellow {

background-color: #877c0a;

}

.bottom\_margin {

margin-bottom: 30px;

}

.cell {

padding: 10px;

margin: 10px;

display: inline-block;

border: 1px solid;

font-size: 32px;

text-align: center;

}

.center {

text-align: center;

}

.float\_child {

float: left;

width: 50%;

}

.score\_box {

border: 1px solid #FFFFFF;

padding: 20px 0px;

}

@media (max-width: 420px) {

.cell {

margin: 5px;

padding: 5px;

}

}

@media (max-width: 800px) {

.float\_child {

float: none;

width: 100%;

}

}

script.js

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// CLASSES

class Cell {

constructor(row, column) {

this.row = row;

this.col = column;

this.value = SPACE; // Initial content is a space to maintain cell size.

this.bg\_color = 'bg\_dark\_gray';

}

display = () => `<div class="cell ${this.bg\_color}" id="r${this.row}\_c${this.col}" >${this.value}</div>`;

set\_background\_color = color => this.bg\_color = color;

set\_value = value => this.value = value;

}

class Row {

constructor(row\_num) {

this.row = row\_num;

this.cells = [];

}

display = () => {

// Defer display calls to each cell

let cells = this.cells.reduce((s, c) => s + c.display(), '');

return `<div class="row" name="${this.row}">${cells}</div>`;

};

}

class Grid {

constructor() {

this.create\_grid();

}

create\_grid() {

this.rows = [];

// Create rows

for (let i = 0; i < 6; i++) {

let r = new Row(i+1);

// Add 5 cells per row

for(let j = 0; j < 5; j++)

r.cells.push(new Cell(i+1, j+1));

this.rows.push(r);

}

}

display = () => {

// Defer display calls to each row

let s = this.rows.reduce((s, r) => s + r.display(), '');

s = `<div class="grid">${s}</div>`;

document.getElementById('grid').innerHTML = s;

};

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// HELPER FUNCTIONS

add\_key = key => {

ROW\_KEYS.push(key);

display\_GUI();

};

consolidate\_keys = () => {

USED\_KEYS = USED\_KEYS.concat(ROW\_KEYS); // concat() returns a new array.

ROW\_KEYS = []; // Reset ROW\_KEYS

};

display\_element = (id, mode) => document.getElementById(id).style.display = mode;

display\_GUI = () => {

// Update keys and grid display after modifications

display\_keys();

GRID.display();

};

display\_keys = () => {

let ALL\_KEYS = [...ROW\_KEYS, ...USED\_KEYS];

ALL\_KEYS.sort();

document.getElementById('used\_keys').innerHTML = ALL\_KEYS.join(', ');

};

// Messages display is never set to none, but they are cleared by inserting

// white space to ensure that spacing is retained.

display\_message = message => document.getElementById('messages').innerHTML = message;

key\_once\_in\_row = (key) => {

// Count how many times the key to remove is in a given row

let key\_count = 0;

for (let i = 0; i < 5; i ++) {

if (GRID.rows[CURRENT\_CELL['row']-1].cells[i].value === key)

key\_count++;

}

if (key\_count === 1)

return true;

return false;

};

remove\_key = key => {

// Remove key from array if it was added in this row, and only played once.

// Keys played in previous rows will not be removed as they are in a different array.

if (ROW\_KEYS.includes(key) && key\_once\_in\_row(key)) {

const index = ROW\_KEYS.indexOf(key);

// This usage of splice removes 1 (second argument) item at the given index

ROW\_KEYS.splice(index, 1);

display\_GUI();

}

};

update\_current\_cell = (row, col) => {

// Update cell and activate it

CURRENT\_CELL['row'] = row;

CURRENT\_CELL['col'] = col;

};

update\_score = () => {

document.getElementById('total\_score').innerHTML = ++SCORE;

document.getElementById(`score\_${CURRENT\_CELL['row']}`).innerHTML = ++GUESSES[CURRENT\_CELL['row']];

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// GAME FUNCTIONS

async function check\_key(e) {

/\*

\* 'e' is the Keyboard event and it has multiple attributes.

\* One of them is the 'keyCode' property, containing the ASCII code of any given key.

\* The 'key' property contains the actual character, which will be added to an array

\* containing the keys that have been used.

\*/

// Only allow keyboard input when the game isn't over.

if (!GAME\_OVER) {

// Destructure CURRENT\_CELL into local variables for easy access.

// Curly brackets are used for destructuring associative arrays and objects.

let {row, col} = CURRENT\_CELL;

// Convert key to lowercase to process uppercase and lowercase keys the same way.

const key = e.key.toLowerCase();

// 65 and 90 are the 'a' and 'z' keys respectively

if (e.keyCode >= 65 && e.keyCode <= 90) {

// Allow keys to be input until filling up the fifth column

if (GRID.rows[row-1].cells[4].value === SPACE) {

// Add key to displayed

if (!USED\_KEYS.includes(key) && !ROW\_KEYS.includes(key))

// Add key to USED\_KEYS array

add\_key(key);

// Update cell value

GRID.rows[row-1].cells[col-1].set\_value(key);

// Update input for it to be written in the next column

if (col <= 5)

update\_current\_cell(row, col+1);

display\_GUI();

}

}

// 8 is the 'delete/backspace' key

else if (e.keyCode == 8) {

// Allow keys to be deleted up to the first column

if (GRID.rows[row-1].cells[0].value != SPACE) {

// Clear messages if they are showing

display\_message(SPACE);

// Remove key will only remove a key that has been played once this row.

remove\_key(GRID.rows[row-1].cells[col-2].value);

GRID.rows[row-1].cells[col-2].set\_value(SPACE);

// Update input for it to be written in the previous column

if (col >= 1)

update\_current\_cell(row, col-1);

display\_GUI();

}

}

// 13 is the 'enter/return' key

else if (e.keyCode == 13) {

// Ensure that the last cell of the row was populated

if (GRID.rows[row-1].cells[4].value != SPACE) {

let word = await check\_word();

if (!word)

display\_message("The word you input is not in the dictionary.<br />Please use the delete key and try again.");

else {

if (compare\_word(word))

game\_over(true);

else {

// If the word is not a match, check if that was the last try.

// Display new game buttons if it is, otherwise update the row.

row === 6 ? game\_over(false) : update\_current\_cell(row+1, 1);

// Pass keys from ROW\_KEYS to USED\_KEYS.

consolidate\_keys();

// If on mobile, clear input bar for next word.

if (/Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(navigator.userAgent))

document.getElementById('mobile\_input').value = '';

}

display\_GUI();

}

}

else

display\_message("Your guess must be 5 letters long.");

}

}

}

async function check\_word() {

// Retrieve word in current row

let word = GRID.rows[CURRENT\_CELL['row']-1].cells.reduce((s, c) => s + c.value, '');

// The success flag will be switched if there is a status code of 200.

let status;

// For this fetch call, retrieving only the status suffices,

// because it will return 200 if the word exists, and 404 if it doesn't.

await fetch(`https://wordsapiv1.p.rapidapi.com/words/${word}`, API\_OPTIONS)

.then(response => status = response.status)

.catch(err => console.error(err));

// Return false if the status code of the API request is not successful.

if (status != 200)

return false;

// Otherwise, return the word for comparing it.

else

return word;

}

function compare\_word(word) {

let matches\_counter = 0;

let letters = [];

// Count how many of each letters are present in the word.

for (let letter in WORD) {

// Check if a key has already been created for a given letter.

if (!(WORD[letter] in letters))

letters[WORD[letter]] = WORD.split(WORD[letter]).length - 1;

}

for (let i = 0; i < 5; i++) {

// If the letter is present in the word to be guessed and in the right position,

// color the background green.

if (WORD[i] === word[i]) {

GRID.rows[CURRENT\_CELL['row']-1].cells[i].set\_background\_color('bg\_green');

letters[word[i]]--;

matches\_counter++;

continue;

}

// If the letter is present in the word to be guessed and in the wrong position,

// color the background yellow.

// Check if the letter is present in the substring, ensuring that there are only

// as many yellow cells as the amount of letters in the word to be guessed.

if (word[i] in letters && letters[word[i]] != 0) {

letters[word[i]]--;

GRID.rows[CURRENT\_CELL['row']-1].cells[i].set\_background\_color('bg\_yellow');

continue;

}

// If the letter is not present in the word to be guessed,

// color the background gray.

GRID.rows[CURRENT\_CELL['row']-1].cells[i].set\_background\_color('bg\_gray');

}

if (matches\_counter === 5)

return true;

}

function game\_over(victory) {

// Switch status to prevent keyboard input.

GAME\_OVER = true;

if (victory) {

// Print message depending on whether the player won or not.

display\_message('You won!');

update\_score();

}

else

display\_message('You lost. Would you like to try again?');

// Display buttons to start a new game.

display\_element('buttons', 'block');

display\_element('debug', 'block');

}

async function get\_word() {

// Ensure that there a no spaces in the word retrieved.

// Also ensure that the retrieved word has a definition.

// The do-while loop will ensure that the code runs at least just once,

// thereby replacing the word when running a new game.

do {

// The 'random' and 'letters' paramaters in the query string are requesting

// a random word with a given number of letters

await fetch('https://wordsapiv1.p.rapidapi.com/words/?random=true&letters=5', API\_OPTIONS)

.then(response => response.json())

.then(response => WORD = response.word)

.catch(err => console.error(err));

await fetch(`https://wordsapiv1.p.rapidapi.com/words/${WORD}/definitions`, API\_OPTIONS)

.then(response => response.json())

.then(response => DEFINITION = response.definitions.length != 0 ? response.definitions[0].definition : '')

.catch(err => console.error(err));

} while (!WORD.includes(' ') && DEFINITION === '');

// Add word and definition to debug field.

document.getElementById('word').innerHTML = WORD;

document.getElementById('definition').innerHTML = DEFINITION[0].toUpperCase() + DEFINITION.substring(1);

}

function new\_game(debug) {

// Check if player is on mobile device. If so, add event handlers to display the keyboard.

// This is important since the app does not rely on an input field on the computer.

// REF: https://stackoverflow.com/questions/3514784/what-is-the-best-way-to-detect-a-mobile-device

if (/Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(navigator.userAgent))

// Show field to get keyboard input

display\_element('mobile', 'block');

// Reset arrays with used keys.

USED\_KEYS = [];

ROW\_KEYS = [];

// Reset initial cell.

update\_current\_cell(1, 1);

// Retrieve a new word.

get\_word();

// Reset board.

GRID.rows.forEach(r => r.cells.forEach(c => c.set\_value(SPACE)));

GRID.rows.forEach(r => r.cells.forEach(c => c.set\_background\_color('bg\_dark\_gray')));

// Display grid and information sections.

display\_GUI();

// Hide buttons and clear messages

display\_element('buttons', 'none');

display\_message(SPACE);

// Display word if on debug mode.

debug ? display\_element('debug', 'block') : display\_element('debug', 'none');

// Display game content.

display\_element('game\_content', 'block');

// Enable keyboard input.

GAME\_OVER = false;

}

function set\_event\_listeners() {

// Add global event listener for the keyboard

document.addEventListener('keyup', check\_key);

// Add new game listener to on-screen button

// Note that the anonymous function is calling the 'new\_game' arrow function.

// This is due to the need of passing parameters. As such, a reference to

// the function needs to be included in the anonymous function.

document.getElementById('new\_game\_button').addEventListener('click', () => new\_game(false));

document.getElementById('new\_game\_button\_debug').addEventListener('click', () => new\_game(true));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// GLOBAL CONSTANTS AND VARIABLES

const API\_OPTIONS = {

method: 'GET',

headers: {

'X-RapidAPI-Host': 'wordsapiv1.p.rapidapi.com',

'X-RapidAPI-Key': 'f8867426bfmshf169cf12937c05ap1c351cjsnd38f2ac86236'

}

};

const SPACE = '&nbsp;';

let GAME\_OVER = true;

let GRID = new Grid();

let WORD;

let DEFINITION;

let CURRENT\_CELL = {

'row': 1,

'col': 1

};

// Arrays to keep track of used keys.

let USED\_KEYS = [];

let ROW\_KEYS = [];

// Score variables

let SCORE = 0;

let GUESSES = {

1: 0,

2: 0,

3: 0,

4: 0,

5: 0,

6: 0

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// INIT

window.onload = () => set\_event\_listeners();