Cumulative Project: Beat Mix

In this project, you'll build a web app drum machine using your knowledge of JS functions and arrays.

Beat Mix

Project Overview

In this project, you will write functions to implement logic for a beat-making music machine. By the end, you will have an application that can loop over a 16-step grid of four drum types and play them when they are activated. You will also write the functionality to invert each row of drums, clear each row of drums, or clear the entire board. Finally, you will build the functionality to retrieve and save presets to a server.

In this project, you'll be writing both front-end and back-end JavaScript code, but all your knowledge of syntax should work in both places! We've set up the project structure and taken care of the front-end user interface and the actual audio code. You'll be writing functions that will be called elsewhere in the code so you can turn the project from a nice interface into an actual functioning app.

You can view a video demonstration of the final app here:

How To Begin

To start, download the starting code for this project here. After downloading the zip folder, double click it to uncompress it and access the contents of this project.

Once you have the project downloaded, you'll need to run some terminal commands to get the application started. First, open the root project directory in your terminal. Run `npm install` to install the dependencies of this project. Once it has finished installing, you can run `npm run start` to begin your server. You'll see `Server listening on port 4001` in the terminal. You'll have to have your server running to ensure that the code you write runs correctly and that you get to hear your drum beats! You can end your server process with a Ctrl + C shortcut in the terminal window.

To see the application in its initial, non-working state, simply open **index.html** in a web browser. You will be writing JavaScript code that uses modern ES6 syntax. You should use Google Chrome (at least version 60) or Firefox (at least version 55). If your version of Chrome or Firefox is too old, correctly-written code may still not run as expected. The links above will let you download the latest release of either browser if you do not have it or are unsure of which version you're running.

Implementation Details

To complete this project, your code will need to contain the following in **public/js/script.js**:

- Four variables to represent the arrays of drum pads. These arrays are named after the drums they represent: `kicks`, `snares`, `hiHats`, `rideCymbals`. These arrays should all have a length of `16` and be filled with `false`.
- a function called `toggleDrum` that takes two arguments: a string representing the array name (`'kicks'`, `'snares'`, `'hiHats'`, or `'rideCymbals'`), and an index number. This function should flip the boolean value in the correct array at the specified index.
- A function named `clear` that takes an array name string and sets all values in the correct array to `false`.
- A function named `invert` that takes an array name string and flips the boolean value of all elements in the correct array.

In addition, you will write some server-side code to handle saving and retrieving drum machine presets in **presetHandler.js**:

- a function named `presetHandler`. This function will be called from within your server to get an existing preset or create/update a preset.
    - `presetHandler` takes up to three arguments. The first argument is a string representing the request type: `'GET'` or `'PUT'`. The second argument is the array index of the `presets` array. For `'PUT'` requests, a third argument, `newPresetArray` will also be passed in, representing the new drum preset array to save at that index.
    - `presetHandler` should return an array. This array will have one or two elements depending on how it is called. If `presetHandler` is called with an invalid `index`, it should return an array with `404` as the first element, meaning that that array index is Not Found. If `index` is valid, the first element of the return array should be `200`, meaning the request was OK.

- If `presetHandler` is called a method that is not `'GET'` or `'PUT'`, it should return an array with `400` as the first element, meaning that it was a Bad Request.
- If the index was valid, `presetHandler` should also return a second element in the array. for `'GET'` requests, that element should be the preset array at that array index. For `'PUT'` requests, it should save the `newPresetArray` to that index and then also return it as the second element.
- If you are testing presets with the app itself, you will need to stop and re-start your server to see the changes you write in **presetHandler.js** take effect.

Bonus

As a bonus, you can choose to implement a function in **script.js** to play multiple synthesizer tones at once by writing:

- a function called `getNeighborPads` that accepts an x, y, and a size parameter. In the application, these values refer to the synth grid: `x` and `y` zero-indexed from the bottom left of the grid, and `size` is a number representing the number of rows/columns in the square. `getNeighborPads` should return an array of neighbors, each in the form `[xValue, yValue]`. Neighbors are the squares immediately to the left, right, above, and below a grid position.

To work on the bonus with tests, you will need to remove their pending status. Open the **test/test.js** and edit the line that begins `xdescribe('BONUS: getNeighborPads() function'` (it should be around line 360 in the test file). If you delete the `x` (so that the line simply starts with `describe(` and save the test file before re-running, your bonus tests will now be active.

Testing

As you work, check your work! While you're working in **script.js**, you should only need to refresh your browser to see changes, but once your move to writing code in **presetHandler.js**, you will need to restart your server to test out your work. To stop your server running, use the `Ctrl + C` key command in your terminal and start it again with `npm run start`.

A testing suite has been provided for you, checking for all essential functionality and edge cases. To run these tests, first open the root project directory in your terminal. Then run `npm install` to install all necessary testing dependencies (you will only need to do this step once). Finally, run `npm run test`.

You will see a list of tests that ran with information about whether or not each test passed. After this list, you will see more specific output about why each failing test failed.

As you implement functionality, run the tests to ensure you are creating correctly named variables and functions that return the proper values. The tests will additionally help you identify edge cases that you may not have anticipated when first writing the functions.