Build Tailwind CSS Configuration

This Node.js script builds a Tailwind CSS configuration file by reading JSON files from three different directories: `/components`, `/base`, and `/utilities`. The resulting configuration file is written to a file named `tailwind-plugin.js` in the `/app` directory.

Prerequisites

- Node.js (v10 or later)

Installation

1. Clone this repository.
2. Run `npm install` to install the required dependencies.

Usage

1. Run `npm run build` to build the Tailwind CSS configuration file.
2. The resulting configuration file will be written to `./app/tailwind-plugin.js`.

Customization

You can customize the directories that are searched for JSON files by modifying the `configurations` object in the `buildTailwindConfig` function.

Code Explanation

Here's a detailed explanation of each function used in the code:

1. `const fs = require('fs')`: This line imports the Node.js `fs` module, which is used to read and write files.

2. `const find_files_recursively = require('../utilities/find_files_recursively')`: This line imports a custom utility function that recursively finds files in a directory.
3. `buildTailwindConfig()`: This function is responsible for building the Tailwind CSS configuration file. It does the following:
    - Sets the `dirname` variable to the current working directory using the `process.cwd()` function.
    - Defines the `configurations` object, which contains three keys (`myComponents`, `myBase`, and `myUtilities`) that map to the results of calling the `getStyles` function with different directories.
    - Defines the `getStyles` function, which takes an array of file paths and reads each file using the `fs.readFileSync` function. The resulting JSON objects are merged into a single object using `Object.assign`.
    - Initializes the `text` variable to an empty string.
    - Uses a `for...in` loop to iterate over the `configurations` object and generate a string of JavaScript code that assigns each key to its corresponding value.
    - Writes the resulting JavaScript code to a file named `tailwind-plugin.js` in the `/app` directory using the `fs.writeFileSync` function.
4.
5. `find_files_recursively(path, file_list = [], excludes = 'node_modules|dist|.git', file_type = '')`: This function takes a directory path, an optional array of file paths, an optional string of excluded directories, and an optional file type. It recursively finds all files in the directory that match the file type and are not in the excluded directories. The resulting file paths are added to the `file_list` array and returned.
6. `getStyles(files)`: This function takes an array of file paths and reads each file using the `fs.readFileSync` function. The resulting JSON objects are merged into a single object using `Object.assign`. The function returns the merged object.
7. `configurations`: This object contains three keys (`myComponents`, `myBase`, and `myUtilities`) that map to the results of calling the `getStyles` function with different directories.
8. `text`: This variable is initialized to an empty string and is used to store the JavaScript code that will be written to the `tailwind-plugin.js` file.
9. `for...in` loop: This loop iterates over the `configurations` object and generates a string of JavaScript code that assigns each key to its corresponding value. The resulting JavaScript code is added to the `text` variable.
10. `module.exports`: This object is used to export the `myUtilities`, `myComponents`, and `myBase` keys from the `configurations` object.

11. `fs.writeFileSync`: This function writes the resulting JavaScript code to a file named `tailwind-plugin.js` in the `/app` directory.

Further Reading

- Content Configuration: Learn how to configure the paths to all of your HTML templates, JavaScript components, and any other source files that contain Tailwind class names.
- Configuration: Learn how to customize your project's color palette, type scale, fonts, breakpoints, border radius values, and more.
- Adding Custom Styles: Learn how to customize your design tokens, how to break out of those constraints when necessary, adding your own custom CSS, and extending the framework with plugins.
- Functions & Directives: Learn how to use Tailwind's custom functions and directives to access Tailwind-specific values.