Prototype Autograder

This prototype is based on Node.JS and requires the following external dependencies not provided within the default Node.JS library:

- glob Link to Page
- he Link to Page
- node-html-parser Link to Page
- css Link to Page
- ncp Link to Page
- empty-folder Link to Page

The prototype contains several key files and subdirectories:

- **package.json:** the necessary Node.JS file that indicates which dependencies are required for this Prototype
- **app.js:** the main JavaScript file that you must initialize if you want to start up the Prototype
- **.validate/validate.js:** a module file that contains and exports the HTML and CSS validation & indentation functions needed inside *app.js*
- **findFiles.js:** a module file that contains and exports the Finding All Necessary files functions needed inside *app.js*
- **./gradeParser/gradeParser.js:** a module file that exports a function to read JSON files, as well as update our assignment details into the proper format.
- **vnu.jar:** a Java-based program that validates HTML and CSS files - this is necessary for checking for errors and warnings in HTML and CSS.
    - Full documentation of the vnu.jar file: Link to Page
    - The **vnu.jar** file can be operated via Bash Terminal - refer to the documentation linked above to determine how
- **node_modules:** a subdirectory that contains node.js modules that were downloaded during testing
- **logistics:** a subdirectory that contains necessary files (i.e. list of students, config file)
    - **students.json:** a JSON file that contains a list of test students for the Prototype
    - **config.json:** a JSON file that contains our settings for grading students - contains info such as point allocation, toggles, etc.
- **grading:** a subdirectory that contains all test files to use for the Prototype.
    - Each assignment within this folder MUST contain the following files and subdirectories:
    i. '*submissions/*' = The subdirectory that contains all student submissions
    ii. '*assignment_details.json*' = A JSON file that contains info on which files must be

present in each submission and what those files should or should not contain (if applicable).
    - If this file links to particular files within its 'files' array, you must provide a '*sampleSubmission/*' subdirectory that contains those files!

iii. '*testSubmission/*' = A subdirectory where the grader will copy and paste student submissions into whenever it grades a student

Preparing the Prototype

You must download the prototype and, using a Bash terminal, set your work directory to the prototype root folder.

STEP 1: FILE STRUCTURE AND ORGANIZATION

You must have your grader file structure so that it matches the following.
NOTE: Those marked with an asterisk (*) can have their paths modified in STEP 2

```
- app.js
- logistics/
    - students.json
    - config.json
- node_modules/
- validate/ *
    - validate.js
    - vnu.jar
- gradingPrepare.js *
- findFiles.js *
- grading/ *
    - assignment_details.json
    - submissions/
      - (student submissions divided by folders)
    - testSubmission/
      - sampleSubmission/
         - (whatever files 'assignment_details.json' needs)
```

STEP 2: SETTING UP PATHWAYS, VARIABLES, AND FILES

2.1: "config.json" Setup

The file **config.json** is currently set up with the following characteristics:

- **"gradingDirectory":** Tells the autograder which file to look into when grading
- 3 object lists named "validation", "inline", and "best_practices" that are crucial to how the autograder works:
  - **validation:** determines how files are checked for validation errors
  - **inline:** check if all HTML files contain either inline CSS or inline JS
  - **best_practices:** check if indentation should be checked or not
- Each of these contains a "toggle" value that tells the autograder if they should grade a student submission based on that quality.
  - if "toggle" is set to *false*, it won't grade students for that particular attribute.
- There is a four necessary object named "points" - this contains the point distribution between the categories named above

2.2: "assignment_details.json" Setup

An example of how the file **"assignment_details.json"** is set up is displayed below:

```
{
     "details":{
           "rootFile":"index.html"
     },
     "files":[
      {
        "type":"file",
        "fileName":"index.html",
        "extension":".html",
        "fileType":"html",
        "careAboutDirectory":false,
        "directory":"",
        "contains":"index.html",
        "notContains":["<title>Replace Title</title>"]
     },
     {
       "type":"file",
       "fileName":"all.css",
       "extension":".css",
       "fileType":"css",
       "careAboutDirectory":false,
       "directory":"styles/",
       "contains":"all.css"
     }
  ]
}
```

The JSON structure breaks down like so:

- **details:** an Object that contains information pertinent to how the autograder should

process details on each student submissions
    - **rootFile:** The file used as a reference by the autograder to determine where the root of the student's web files are located
- **files:** An array of objects that contains what each student must contain within their submissions
    - Each object must contain the following values:
    - *type:* is this a file or text?
    - *fileName:* what is this file's intended basename?
    - *extension:* what is this file's intended extension?
    - *fileType:* what should this file's contents be parsed as (i.e. html, css, js)?
    - *careAboutDirectory:* should we care about directory when looking for this file within a student submission? (true or false)
    - *directory:* if we care about directory, where should the file be located ('' = root directory)
    - *contains:* either null, an Array, or a String pathway leading to a files
        - tells what this file in the student's submission must contain
    - *notContains:* either null, an Array, or a String pathway leading to files
        - tells what this file in the student's submission must NOT contain

2.3: 'app.js' Setup

Within **app.js**, there contains the following variables:

- var **assignment_url**
- var *studentSubmissions*
- var *testSubmissionLocation*
- var *sampleSubmissionLocation*
- var *assignment_details_url*
- var *config_url*
- var *students_url*

These variables are the pathways that the autograder uses to locate core essential files.

> NOTE: **assignment_url** receives its value from **config.json**, particularly from **gradingDirectory**.
> All other variables (in italics) are all determined by what **assignment_url** receives from **config.json**.

There are also several *require()* statements declared within *app.js*:

> const **gradingPrepare** = require("./gradingPrepare/gradingPrepare.js");

```
const validate = require("./validate/validate.js");
const findFiles = require("./findFiles.js");
const inline = require('./inline.js');
```

You may alter these to suit your needs, though this is inadvisable.

Using the Autograder Prototype

In order to use the autograder, once you've prepared the autograder appropriately, you can initialize the autograder with the following bash command:

```
node app.js
```

Make sure your set working direcctory is still the root directory of the prototype, where "app.js" is set up. You should see output appearing within the Bash terminal if the autograder has been prepared properly.

What the Autograder Does

The autograder does several grading procedures automatically:

1. Grab **config.json** and set **assignment_url** to the proper pathway
2. Grab additional files (**student.json**, **assignment_details.json**)
3. Loop through each student, grading them

When a student is being inspected by the autograder, these processes automatically happen regardless of what is mentioned inside **config.json**

1. Copy a students submission into **testSubmission/**, emptying it if necessary.
2. Collect all files submitted by the student
3. Determine the root directory of the student's web files
4. Find necessary files and the elements in those files that a student should and shouldn't contain

From here all other processes are dictated by **config.json**:

5. Check for validation errors and indentation in all HTML files
6. Check for validation errors and indentation in all CSS files
7. Check all HTML files for inline CSS and JavaScript

More functions to come in time