CIS54x Unit Test Autograder Package

This package is intended to be used for eCornell's CIS54x online web design course. It contains unit tests intended to be used by the course's Codio environment.

The Unit Test AutoGrader runs off of Node.JS and can be installed into any machine, virtual or local, to grade assignments focused on HTML, CSS, and JavaScript.

Table of Contents:

---

1) Files, Node.JS Modules, and Other Contents Present within the CIS54x Package

Core Files and Folders

All of the files and/or directories listed here are necessary for the operation of the AutoGrader.

File/Directory	Purpose	Example Command
`app.js`	The main Node.JS file the AutoGrader runs off	`env TESTS=path/to/tests.json node app.js`
`runner.js`	The Mocha runner that runs all Mocha tests. Can be accessed separately from `app.js` via the command line.	`env TESTS=path/to/tests.json mocha path/to/runner.js`
`tests.json`	A JSON object file that contains all the necessary unit tests when running the AutoGrader	*none*

About "app.js"

`app.js` is the core of the AutoGrader application. It must be accessed via Terminal in order to function.

Running `app.js` requires 1 necessary [*] and 2 optional environmental arguments.

Argument	Purpose	Default	Additional Notes
TESTS*	Relative pathway to `tests.json`.	*none*	If `SINGLEDIR` env. variable not defined, the AutoGrader will look at the root directory of the `tests.json` file defined by `TESTS`.
SINGLEDIR	Relative pathway to single submission that needs to be graded.	If not provided, AutoGrader will look at root directory of `tests.json` as its SINGLEDIR directory	Must be a directory - files will return an error.
	Relative		

SUBMISSIONS	pathway to directory containing multiple submissions to be graded.	If provided, will overwrite SINGLEDIR declaration regardless.	Must be a directory containing directories - files will be ignored

A generic command line that will initialize the AutoGrader using `app.js` is divided into the following parts:

```
env
TESTS=path/to/tests.json
[SINGLEDIR=path/to/single/submission/directory/]
[SUBMISSIONS=path/to/directory/with/multiple/submissions/]
node app
```

> For more information in how to run the AutoGrader, look at the section titled Running the AutoGrader On Your Local Machine.

About "runner.js"

If you wish to skip using `app.js` and directly refer to mocha testing output from unit tests, you must run the AutoGrader using `runner.js` instead.

The benefit of running `runner.js` instead of `app.js` is that `runner.js` will output `console.log()` into the Terminal directly. Running the AutoGrader with `app.js` surpresses all `console.log()` output from any of the user tests. HOWEVER, `runner.js` will only look at one submission with every command line initialization.

Running `runner.js` requires 1 necessary [*] and 1 optional environmental arguments.

Argument	Purpose	Default	Additional Notes
TESTS*	Relative pathway to `tests.json`.	*none*	If `DIR` env. variable not defined, the AutoGrader will look at the root directory of the `tests.json` file defined by `TESTS`.
	Relative pathway to	If not provided, AutoGrader will look	

DIR	single submission that needs to be graded.	at root directory of `tests.json` as its testing directory	Must be a directory - files will return an error.

Notice that unlike `app.js`, `runner.js` only needs 2 environmental variables at maximum. This is because `runner.js` operates by looking at only one submission only.

A generic command line that will initialize the AutoGrader using `app.js` is divided into the following parts:

```
env
TESTS=path/to/tests.json
[DIR=path/to/single/submission/directory/]
mocha runner.js
```

> For more information in how to run the AutoGrader, look at the section titled Running the AutoGrader On Your Local Machine.

About "tests.json"

In order for the AutoGrader to run, it needs some parameters by which it can run tests - instructions, in other words. This is where `tests.json` comes into play.

`tests.json` provides the AutoGrader explicit instructions on how to utilize the tests to grade student submissions. A document called `sampleTest.json` has been provided to give an idea of how this file should be formatted.

Each test required within a session must be a JSON object within the array titled "tests" within `tests.json`. Every test must contain the following variables (NOTE: Those marked * are required, others are not mandatory depending on the test used):

> For a more precise description of what every test requires, look at Inside "tests.json" within Running the AutoGrader On Your Local Machine

> For exact notes on which arguments and variables to use for each particular unit test, look at Unit Tests Provided With the Package.

Core Middleware

All of the files and/or directories listed here are necessary for the operation of the autograder.

File/Directory	Purpose	Additional Notes
`common.js`	A Node.JS module that contains functions and global variables that are commonly necessary among all unit tests.	NOT accessible via the command line - it is not a standalone Node.JS application
`package.json`	The Node.JS package information that contains information on all node modules currently installed within the Node.JS application	*none*
Directories `001-100/`, `101-200/`, and `301-400`	Directories that contain the unit tests necessary for the package	*none*
`app.css`	A CSS stylesheet that contains modified styling used by Mochawesome's HTML reports.	This file is to replace the default "*mochawesome-rport-generator*" package that comes with Mochawesome.
`utils.js`	A JavaScript file, a modified version of the one used by Mochawesome that produces the output needed for the HTML and JSON reports Mochawesome prints out.	This file is to replace the default *utils.js* file used by the Mochawesome package.

Core Node.JS Modules That Need to be Installed Globally

The autograder runs off of the following Node.JS modules that allow for the tests to run:

Node.JS Package	Purpose	Online Resource(s)
`mocha`	Testing Framework for this CIS54x Autograder Package	Website

Optional Files/Folders/Modules

While these files, folders, and modules are not necessary, some of these mentioned ARE necessary for certain unit tests. Which unit tests require the following files/folders/modules are mentioned in parentheses next to each file.

Optional File/Directory/Modules	Type	Purpose	Additional Notes
`vnu.jar`	File	Used to validate HTML and CSS for errors	Unit Tests 001, 101
`tests/`	Directory	Contains other files and directories used for testing the unit tests themselves	*none*
`testJSONS/`	Directory	Contains the `tests.json` files used to test the unit tests of the autograder	*none*
`testcafe`	Global module	A Node.JS test framework used for certain unit tests. Separate from Mocha, but the autograder utilizes this framework temporarily when testing DOM properties or JavaScript functionality on the DOM	Unit Test 103 `npm install -g testcafe`
`http-server`	Global module	a Node.JS module that allows files to be hosted on a localserver. Used in conjunction with TestCafe to test DOM properties and JavaScript functionality on DOM elements	Unit Test 103 `npm install -g http-server`
`Google Chrome`	Application	Used in conjunction with `testcafe` and `http-server` to host websites for testing DOM properties and JavaScript	*[1]*

		functionality	
`mocha.opts`	File	The operations file Mocha uses to define its settings and operations.	*Currently deprecated*

[1] To install Google Chrome via the Bash Terminal Command Line, you must execute the following commands in order:

```
wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | sudo apt-key add -
sudo sh -c 'echo "deb https://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/
sudo apt-get update
sudo apt-get install google-chrome-stable
```

2) Unit Tests Provided With the Package

The CIS54x Unit Test Autograder Package comes with the following unit tests pre-packaged. *Those contained within brackets [] are optional variables*

001-100 Unit Tests --- HTML-Related

001-100 Unit Tests	Function	Variables	Necessary Additional Files/Modules
`001-validate-html.js`	Looks for HTML errors in an HTML file or all HTML files in a given directory	`[HTML_PATH]` : *string*: file or directory. - default = directory of `tests.json`, or otherwise specified within command line variables SINGLEDIR or SUBMISSIONS	`vnu.jar`

		`[SUPPRESS]` : *boolean*: suppress errors involved with doctype or missing title in head. - default = `false`	
`002-element-exists.js`	Looks if an HTML tag is present within an HTML file	`HTML_PATH` : *string*: path to HTML file `SELECTOR` : *string*: CSS selector of HTML element needed to be searched for `[EXISTS]` : *boolean* : Should the element exist or not exist? - default = `true`	*none*
`003-resource-exists.js`	Checks if the src or href files of certain elements load properly in the browser for the index page of a website	`SELECTOR` : *string*: CSS selector `[DIR_PATH]` : *string* : path to the root directory of a website - default = directory of `tests.json` , or otherwise specified within command line variables SINGLEDIR or SUBMISSIONS `[FILES]` : *array of strings:* all HTML files that needs to be inspected - default = `["index.html"]`	- `TestCafe` - `http-server` - `Google Chrome`
		`HTML_PATH` : *string*: path to file `ORDER` : *Object list of*	

`004-order-exists.js`	Given a PARENT SELECTOR and an array of CHILDREN SELECTORS, check if the parent's children match an expected order of HTML elements provided	*string and array pairs*: parent selector, and list of children selectors `[PARENT]` : *string*: selector of some HTML element that acts as a parent of elements - default = `"html body"` (first instance of Parent only) `CHILDREN` : *array of strings/object lists*: list of either strings that match immediate children selectors or object lists that contain their own PARENT and CHILDREN	- `Cheerio` - `node-html-parser`

101-200 Unit Tests --- CSS-Related

101-200 Unit Tests	Function	Variables	Necessary Additional Files/Modules
`101-validate-css.js`	Looks for CSS errors in a CSS file or all CSS files in a given directory	`[CSS_PATH]` : *string* : file or directory - default = directory of `tests.json` , or otherwise specified within command line variables SINGLEDIR or SUBMISSIONS	`vnu.jar`
		`CSS_PATH` : *string* : path to CSS file `PROPERTY` : *string* : CSS property needed to be searched for	

`102-declaration-exists.js`	Looks if an CSS declaration is present within a CSS file	`[SELECTOR]` : *string* : looks for specified PROPERTY inside any CSS declarations matching the SELECTOR - default = `null` `[VALUE]` : *string* : looks for specified PROPERTY with specified VALUE - default = `null` `[EXISTS]` : *boolean* : Should the declaration exist or not exist? - default = `true`	*none*

301-400 Unit Tests --- File-System-Related

301-400 Unit Tests	Function	Variables	Necessary Additional Files/Modules
`301-file-exists.js`	Checks if a given file or directory exists	`PATH` : *string* : file or directory	*none*
`302-image-files.js`	Checks if all images are located in the same base directory	`[ROOT_DIR]` : *string* : path to root directory of website - default = directory of "tests.json", or otherwise specified within command line variables SINGLEDIR or SUBMISSIONS `[IMAGES_DIR]` : *string* : directory where all images must be based inside - default = null (will just look for common directory)	- `is-image`

`303-formatting.js`	Checks if the indicated files are formatted properly via fuzzy testing and file comparisons	`PATH` : *string* : file or directory `[SIMILARITY]` : *string/double/integer*: parameter of similarity, files must be >= suggested format in similarity - default = 0.75 `[INDENTATION_ONLY]` : *boolean*: Check for only indentation or whole format - default = false `[FILETYPES]` : *string/array of strings*: which types (html or css or both) AutoGrader should check for - default = `["html", "css"]`	- `diff` - `cssbeautify` - `html`
`304-compare-files`	Compares two files to get degrees of difference between the two	`CHECK_PATH` : *string*: file to be checked `TRUE_PATH` : *string*: file to be compared against `[SIMILARITY]` : *string/double/integer*: parameter of similarity, files must be >= suggested format in similarity - default = 0.75 `[ONLY_CONTENTS]` : *boolean*: check only contents or	- `diff`

		whole format - default = false	

3) Running the AutoGrader On Your Local Machine

Base Installation

While installation is relatively easy, there are certain actions that MUST be taken for the program to work as intended.

1. In your Bash Terminal, change the working directory to the root folder of where you have placed the CIS54x package.

2. Use the following command to install the necessary Node.JS modules:
   npm install

3. You must install mocha globally into your local machine. You may do so with the following command:
   sudo npm install -g mocha

4. You must replace the following two files over their defaults. If you are feeling paranoid, feel free to create copies of the original mochawesome and mochawesome-report-generator folders before replacing the following files.

   - utils.js
     - *This file replaces:* `node_modules/mochawesome/dist/utils.js`
   - app.css
     - *This file replaces:* `node_modules/mochawesome-report-generator/dist/app.css`

Sample Command Lines to Operate the AutoGrader

Here are some sample commands that can be used to operate the AutoGrader, for people who may not necessary be comfortable with how to work the AutoGrader.

Testing A Single Submission w/ a tests.json WITHIN the Root of the Submission Directory

Let us say that `submissions/` is the directory containing all student submissions and we are trying to grade `student1/`, which coincidentally happens to contain `tests.json`:

```
env
TESTS=submissions/student1/tests.json
node app
```

> Explanation: Since `tests.json` is located within the root of the submission folder (`submissions/student1`), the AutoGrader knows where the submission is located relative to `app.js` and will grade the submission properly.

Testing A Single Submission w/ a tests.json NOT Located Within the Same Root Of the Submission Directory

Let us say that `submissions/` is the directory containing all student submissions and we are trying to grade `student2/` but `tests.json` is not located within `student2/`:

```
env
TESTS=testFiles/tests.json
SINGLEDIR=submissions/student1/
node app
```

> Explanation: `tests.json` is located within another directory called `testFiles/`, while the student submission `student2/` is contained within `submissions/`. We need to signify that we want to test `submissions/student2/`, so we use `SINGLEDIR` within the command line.

Testing MULTIPLE Submissions w/ a tests.json That Can Be Located Anywhere

Let us say that `submissions/` is the directory containing all student submissions and we are trying to grade BOTH `student1/` and `student2/`. `tests.json` is located within `testFiles/`:

```
env
TESTS=testFiles/tests.json
SUBMISSIONS=submissions/
node app
```

> Explanation: We want to test multiple submissions, so we must use `SUBMISSIONS` in our command line, not `SINGLEDIR`. In this scenario, the location of `tests.json` does not matter, but we must still define where it is.

Workflow Of the AutoGrader

The CIS54x Unit Test AutoGrader performs its functions in the following order by default:

1. User executes a command to initialize the AutoGrader (ex. `env TESTS=tests.json node app.js`) while the current working directory within the Bash terminal is the root folder where `app.js` is located
2. `app.js`:
    - Gets the path to `tests.json` via the Environmental Variable executed alongside the command above
    - Determines whether to test a single submission or multiple submissions, depending on whether `SUBMISSIONS` was defined in the command line or not.
    - Executes a Child Process command: `env TESTS=tests.json mocha --... ... runner.js` for every submission required to be graded.
3. `runner.js`:
    - Scans `tests.json` for the list of unit tests it must execute - Ends prematurely with Exit Code 1 if no `tests.json` was provided.
    - Synchronously runs each unit test provided inside `tests.json` - every unit test is its own mocha test suite
    - Prints out Mocha test results using `mochawesome` - produces a `testReport/` folder with `report.html`, alongside other files
4. Back to `app.js`
    - Once the Runner finishes its testing, the Autograder performs according to whether the tests returned any failed suites or not.
    - If the runner returns any failed suites, the program terminates with Exit Code 1 and with any error messages outputted via the terminal
    - If the runner doesn't return any failed suites, the program terminates with Exit Code 0.

To view the results of the Autograder's testing, simply open up `report.html` that is located within the newly made `testReport/` directory.

Additionally, it is entirely possible to run `runner.js` without relying on `app.js` - one simply needs to execute `env TESTS=tests.json mocha --opts mocha.opts runner.js` within the Bash Terminal, as the program will output a `report.html` that can be viewed regardless.

Adjusting Settings

Should the need arise, certain options can be altered to allow the Autograder to work per the needs of the user.

Changing Mochawesome Settings

If you wish to alter the Mochawesome settings used by the AutoGrader to cater to your needs, you must alter the `child_process()` command lines used within `app.js`. These particularly call the function `exec()`, so you may use this to easily search for where the commands are within the file.

While `mocha.opts` is currently deprecated, it contains the original Mochawesome reporter settings used in previous versions of the AutoGrader. There were several defined options pre-set to work by default. These options are meant to alter the behavior of the Mocha testing framework and the Mochawesome reporter used in conjunction with Mocha. You may use this as a means of experimenting with what settings is which.

Option	Description	Default Value	Ac
`--reporter`	Tells Mocha which reporter it should use	`node_modules/mochawesome/dist/mochawesome.js`	The M frame a fork versi Moch on pu this is altera the Moch repor for th Autog
			- `sho` show pend from - `ena` show code withi test

`--reporter-options`	Defines options for Mochawesome - MUST be a single string delimited by commas (for this readme, separated by newlines)	`showPending=false`
		`enableCode=false`
		`reportDir=testReport/`
		`reportFilename=report`
		`charts=false`
			- `rep`
			defin
			which
			`repor`
			and i
			files a
			save
			-
			`repor`
			what
			Moch
			outpu
			shoul
			save
			(does
			.exte
			`chart`
			show
			graph
			detail
			many
			each
			passe
			failed

`--timeout`	Sets the timeout period for each unit test	`20000`	Reco to be at lea than allow time- tests to pe opera

A full list of additional options for Mochawesome that can be altered are provided here and here.

Inside "tests.json"

Within `tests.json` are the unit tests that are meant to be executed by the Autograder upon

each run. You may link to any unit test file provided by the Autograder, or any custom unit test that you may wish to use. For each unit test you wish to define, here are the common values you must define for the unit test to operate properly.

- Note: Those wrapped by brackets [] are optional; those marked with an asterisk `*` are those where certain unit tests have predefined these and cannot be altered via `tests.json`

Be aware that any links provided are relative to `runner.js`, meaning that if a unit test file is located one directory lower than `runner.js` you must adjust properly.

Value	Description	Value Type	Example
`title`	The title that defines your unit test	string	"title":"P tag existence"
`test`	Pathway to where the unit test file is located	string	"test":"001-100/002-element-exists.js"
`[statement]`	Message that usually defines what you may expect for the test to return a success	string	"statement":"Expect p tag to be present"
`[error_message]`	The error message that appears if a unit test returns a failure	string	"error_message":"P tag was not found!"
	Defines whether any hints should		

`[hints]` *	appear for failed tests `001` and `101` are predefined to output hints regardless	string or `false` (boolean)	"hints":"Check for mispellings\nMake sure you haven't forgotten to add them into your code"
`variables`	Each unit test requires certain unique variables to operate properly Refer to Unit Tests Provided With the Package for a list of variables for each unit test	object list, containing various value types	"variables":{ "HTML_PATH":"./example.html", "SELECTOR":"p", "EXISTS":true }

Inside "common.js"

There is a single variable named `vnuPath` defined within this file. All that is necessary is to ensure that the defined pathway to `vnu.jar` is set so that it is relative to the unit test that requires it, not `runner.js` or `app.js`.

4) Warnings and Considerations

There are several things to keep track of when using the package within your local machine:

1. Make sure you've set your working directory to the root directory of the CIS54x package
2. Make sure that you have the most up-to-date Node.JS modules installed
3. Make sure that you have the necessary modules installed globally (mocha)

4. Make sure that tests.json contains the appropriate pathways and variables
    - Remember: all pathways defined within this file must be treated as if you were relative to `runner.js`
5. Make sure that you've replaced app.css and utils.js appropriately
6. Make sure that the `vnuPath` variable defined within `common.js` is linked properly to `vnu.jar`, relative to the unit tests that require that file
7. If you wish to run the tests from runner.js and not app.js, then make sure that `tests.json`, `mocha.opts`, and `runner.js` have their paths properly defined