

## Project Three: Adding Unit Tests

In this project you extend project two by adding unit tests and code coverage. The unit tests will use the mocha test framework and the chai assertions package. The code coverage will use the istanbul package.

**Since the project is an extension of project one and two, if something was not correct in your project one solution or project two solution, you need to fix it in your project three solution. If any feature of project one or project two is not correct in your project three solution, then that is considered an error in project three.**

### Project Directory Structure

- ❖ On agora create a project three directory that has in it a README text file and a subdirectory, named app, for your project source code. The README file lists the author names, date, that this is Project 3, explains how to run your program (including showing the exact command to be entered) and any aspects of the program that do not work. You will lose fewer points for parts that do not work that you tell me about.
- ❖ In your app directory run the command

**npm init**

to create your package.json file. Then run the command

**npm install --save-dev mocha**

to install the mocha test framework. There are two dashes before the word “save” without any space between them. Since mocha is the first package being installed, the node\_modules subdirectory is also created to hold the mocha package.

- ❖ Run the command

**npm install --save-dev chai**

to install the chai test assertion package which you will use in your unit tests. There are two dashes before the word “save” without any space between them.

- ❖ Run the command

**npm install --save-dev istanbul**

to install the istanbul code coverage package. There are two dashes before the word “save” without any space between them. You can learn more about istanbul at <https://github.com/gotwarlost/istanbul>

Running the following command can also be helpful, if you are to create scripts to run the tests.

**npm install --save-dev nyc**

- ❖ In the app directory you will have two files

data\_structures.js  
make\_poem.js

as in project two. The one difference is that you need to make make\_poem.js a node module also (not just make data\_structures.js a node module) and export its functions

- main
- makePoem
- pickFirstWord
- pickNextWord

and any related functions you decide you need. Making make\_poem.js a module is necessary so that you can import it (using the require() function) into your mocha test file for make\_poem.js which will be in the qa subdirectory. You will need to import data\_structures.js (using the require() function) into your mocha test file for data\_structures.js which will be in the qa subdirectory.

- ❖ Create a directory named qa (for “quality assurance”) in the app directory. In app/qa create a file named tests\_make\_poem.js and a file named tests\_data\_structures.js. These files will contain your unit tests. In other words,

#### qa subdirectory

tests\_data\_structures.js  
tests\_make\_poem.js

- ❖ Your app/qa/tests\_make\_poem.js and app/qa/test\_data\_structures.js test files needs to have the following properties
  - **at least one test for each function and**
  - **100% code coverage using the code coverage definition of number of statements in code being tested that are executed at least once during at least one test; so every statement in every function needs to be executed in at least one test**
  - **each test needs to be specific in that you are checking whether a function returns the specific expected value when provided particular input. For example, given particular input does pickFirstWord() return the string “red”. Just checking that the return value is of a type object is not acceptable. You must check for the exact required return value, e.g. “red”.**
- ❖ Your mocha tests can use either the tdd user interface or the bdd user interface supported by mocha. You can use either the “assert” style or the “expect” style from the chai assertions module.
  - The example in the Logic Testing section of Chapter 5 of brown2014 (page 48) uses the tdd mocha user interface and the expect chai style. The example in the Page Testing section of Chapter 5 of brown2014 (pages 42-43) uses the tdd mocha user interface and the assert chai style.
  - If you go to the chai website, <http://chaijs.org>, you will see more examples of the expect and assert chai styles.

- If you go to the mocha website, almost all the examples on the main page use the bdd user interface and the expect chai assertions style.

## Running Mocha

- ❖ Your `npm install --save-dev mocha` command installed the mocha executable in `app/node_modules/.bin/mocha`. Assuming that you are in the app directory of your project and assuming you want to use the bdd (Behavior-Driven Development) mocha user interface and the spec reporter of mocha, you can run the tests in your `tests_make_poem.js` tests by the command. Note that the command starts with a period, `./`. Note that the `“qa/”` is needed since the `tests_make_poem.js` file is in your `qa` subdirectory.

**`./node_modules/.bin/mocha qa/tests_make_poem.js`**

This is equivalent to the command

**`./node_modules/.bin/mocha -u bdd -R spec qa/tests_make_poem.js`**

since bdd is the default user interface and spec is the default reporter.

- ❖ Similarly, you can run the tests in your `tests_data_structures.js` file by the command

**`./node_modules/.bin/mocha qa/tests_data_structures.js`**

- ❖ Mocha's default user interface is bdd, but if you want to use the tdd (Test-Driven Development) user interface instead, then the command for running the tests in your `tests_make_poem.js` file adds the `“-u tdd”` argument and is (assuming we want to use the default reporter which is spec)

**`./node_modules/.bin/mocha -u tdd qa/tests_make_poem.js`**

See <http://mochajs.org/#interfaces> for more about mocha's user interfaces.

- ❖ In Mocha the reporter selected determines how the output of your test runs is displayed. The default reporter is spec, so not listing a reporter in your command-line is equivalent to including `“-R spec”` in the command-line. See <http://mochajs.org/#reporters> for more information about your reporter.
- ❖ Suppose we want to use the json reporter which displays the test output as a JSON object. We would use the following command to run the tests in your `tests_make_poem.js` file (assuming we want to use the bdd user interface).

**`./node_modules/.bin/mocha -R json qa/tests_make_poem.js`**

- ❖ Recall that there are two possible assertion styles from the chai package that you can use: `assert` or `expect`. Which of those your tests use is not a command-line argument to the mocha command. Instead you specify the assertion style using the `require()` function at the start of your test file (that is, `tests_make_poem.js` and `tests_data_structures.js`).
- ❖ As mentioned above each unit test function has check if the return value of the function being tested is exactly the expected value, e.g. `“red”` if `“red”` is supposed to be returned. If the returned value is a javascript object checking that it is exactly correct. Chai has a set of assertion statements that can handle this. See the list at <http://chaijs.com/api/bdd> and `.deep` as in

```
expect(foo).to.deep.equal({ bar: 'baz' });
    expect({ foo: { bar: { baz: 'quux' } } })
        .to.have.deep.property('foo.bar.baz', 'quux');
```

## Code Coverage with Istanbul

### Running istanbul

For code coverage you need to use istanbul, <https://github.com/gotwarlost/istanbul> . See that website for direction about installation.

To see test coverage for test\_make\_poem.js and test\_data\_structures.js enter:

(From the app directory) Note: All commands are one line.

```
./node_modules/.bin/istanbul cover --reporter=text ./node_modules/.bin/_mocha -u bdd -R spec qa/test_make_poem.js
```

(For make\_poem coverage. The above command has to be entered as a single line.)

Note:-x excludes data\_structures from coverage report because it is used by make\_poem.js, but we aren't worried about it's coverage in the make\_poem tests.

```
./node_modules/.bin/istanbul cover --reporter=text ./node_modules/.bin/_mocha -u bdd -R spec qa/test_data_structures.js
```

(For data\_structures coverage. The above command has to be entered as a single line.)

## Viewing Code Being Tested with Missing Coverage

**When you run in the app directory istanbul with the "cover" argument which is what we have been doing, a directory named coverage is created.** If you cd into that directory, there is a subdirectory called lcov-report. Copy that subdirectory into your public\_html directory on agora. You will need to then go into public\_html and change the permissions modes all the files and subdirectories in lcov-report. Directories need to be 755 and files 644.

Then start your browser and enter

```
http://agora.cs.wcu.edu/~your_user_name/lcov-report/
```

which will bring up the index.html file. You then click on the links on that web page to navigate to a particular file. The code of that file and which statements are being missed will be shown.

## Test Scripts (Optional)

To make testing easier you can also add tests to your **package.json** under the Scripts element so you may run scripts by name. For example, by having the following in my **package.json**:

```
"scripts": {
  "test": " ./node_modules/.bin/mocha -u bdd -R spec qa/bdd_Person.js",
  "atest1": " ./node_modules/.bin/mocha -u bdd -R spec qa/bdd_Person.js",
```

```
}
```

I can now run tests from the project directory by stating:

```
npm run atest1
```

This is far easier than the long commands needed to run a unit test.

## **PATH Environment Variable (Optional)**

- ❖ You can make invoking mocha simpler by adding the path to the mocha executable to your PATH environment variable. The following directions explain how to do this.
- ❖ Step 1: On agora at the command prompt enter

```
printenv PATH
```

to see the current value of your PATH environment variable.

- ❖ Step 2: In your home directory on agora edit your .profile file by adding at the end of it the line

```
PATH = “./node_modules/.bin:$PATH”
```

This inserts the path to the mocha executable in front of all the paths that are already in the PATH environment variable. The mocha executable is in the directory node\_modules/.bin in your home directory. **Note we are editing the .profile file, not the .bashrc file!**

- ❖ Step 3: At the command prompt enter

```
source .profile
```

which causes your shell to read your .profile file. Now do

```
printenv PATH
```

again and you will see ./node\_modules/.bin added to the front of the sequence of file pathnames that are examined for executables. Your .profile file will automatically be read by your shell each time you logon to agora.

- ❖ Step 4: Now (since the PATH environment variable knows where to look in the file system) you can run your mocha tests using the commands (assuming you want to use the bdd user interface and the spec reporter)

```
mocha qa/tests_make_poem.js
```

```
mocha qa/tests_data_structures.js
```

These commands only work if you have added ./node\_modules/.bin to your PATH environment variable in your .profile file in your home directory (that is, ~/.profile).

## Other Comments

- ❖ All of your code should be in function definitions except for your calls to the main() function. In other words, the only global code should be the calls to the main() function. This style is more modular and easier to understand.
- ❖ Make sure to use good programming style and comments. There should be at least one blank line between the end of one function's body and the header of the next function. Follow the airbnb style guide for javascript.

<https://github.com/airbnb/javascript>

- ❖ Your project will be graded based on how it runs on agora when I test it.
- ❖ Remember you need a JSDoc comment (that is, a `/**` style comment) at the start of each source file or test file and just above each function header. In the JSDoc comment at the start of each file you need `@author` for the author names, `@version` for the date, and a good description of what the code in the file does. In the JSDoc above each function you need a description and your `@param` and `@return` attributes if there are any parameters or a return value.
- ❖ JSDoc is very similar to javadoc. See the following links for more details
  - <https://en.wikipedia.org/wiki/JSDoc>
  - <http://usejsdoc.org/>
  - <https://github.com/jsdoc3/jsdoc>
- ❖ Your code must not be placed on github since github repositories are public.

## Administrative

- The project is due at 5 pm on Friday, the 20th of April.
- If you have not submitted the project by the time it is due, you can submit once a late project submissions. Late points will be deducted. Every twenty-four hour period starting at 5 PM is one late day which is five points deducted.
- Weekends and holidays do not count towards late points in the following sense. A project submitted from 5:01 PM on Friday until 5:00 PM on Monday counts as one extra late day. For holidays the approach is similar.
- A maximum of twenty late points will be deducted. At which time more than 20 late points will be accrued, the project is no longer viable for submission.
- You may work in teams of one or two students from the class. You may talk with other students in the class about the concepts involved in doing this project, but anything involving actual code needs to just involve your team. In other words, you can not show your team's code to students outside of your team and you can not look at the code of another team.
- Submit your project directory as a gzipped tar file via handin on agora. So the tar file will contain your README file, your tests files in the qa subdirectory, and your source files. If your project is in a directory named project3 and you are in its parent directory, then you can use

**`tar -cvzf project3.tar.gz project3`**

to create the file project3.tar.gz. The 'c' means to create a tar file, the 'v' means verbose, the 'z' means to turn the tar file into a gzipped file, and the 'f' specifies that the next command line argument is the name of the resulting file. Then use the following command to submit the file.

**`handin.253.1 3 project3.tar.gz`**