CS 341: Homework 3 – Software Tools

Up until now your web app consists of only two files: a static .html and .css file. As projects get larger, they get a lot more complex. For example:

- Web applications consist of many files that must be managed by source control.
- Web applications have a front end (runs in the web browser) and a back end (that runs on a server). Software is required to run and configure the server.
- Source code will need be tested regularly. For this, a third party library is required.
- Apps use many third party libraries. These libraries have to be updated as required and their interdependencies managed.

In this assignment you will setup several software tools and create a project to manage what will eventually be a very simple, complete web application.

Note: This may seem like a really long assignment because this document is long. But it's not as long as it looks as it's mainly about following step-by-step instructions. However, it's still wise to start early so you can catch problems early!

Specification Part 1: Installing and Configuring Software

You'll need to setup some software to use:

GitHub

At this point, you should already have an account on GitHub (github.com) that you've used for previous classes. This account should be tied to your up.edu email address.

Node.js

Node.js is one of many different libraries that are used for the backend of a web application. We've been using Node for this assignment since it is the most popular such library. Install Node.js on your computer via this link: https://nodejs.org/en/download/. If you're using the engineering computer lab, this should already be installed.

Express Framework

The Express framework is a JavaScript library and toolset designed to make it easy to create and manage a back-end for your Node.js web application. You will install it using the Node.js. Package Manager (npm) that was installed as part of the Node.js software. Run this command:

```
npm install express-generator -g
```

Specification Part 2: Creating a GitHub Repository

It's time to create a GitHub repository for your code. Use the form on this page: https://github.com/new and fill it out as follows:

- Check the box to initialize the repository with a README file.
- Add a .gitignore file. Specifically, use the template for Node projects.
- Add a license. I recommend the Apache or MIT license.

Once the project is created, clone it onto your computer, edit the README.md file, commit your changes and push them to the repository.

Specification Part 3: Creating a Project

To create a new Express project follow these steps:

1. In root folder your git repository, run this command:

   ```
   express hw3
   ```

This creates a folder named `hw3` that contains an initial application.

2. Change to the `hw3` folder
3. Install the app that was generated with this command: `npm install`
   This command will read the package.json file in your app's root folder and install all the required libraries into the node_modules folder. One of these is the express library which we will use heavily.
4. Start your program with: `npm start`
   This command is the standard was to start a Node.js app.
5. Using a browser, visit `http://localhost:3000/` to verify your starter app is running. You should see a "Welcome to Express" message.

Any server-based app you create for this class (including this homework assignment) is expected to have been created initially with the Express application generator so that the application is laid out in a consistent way for all students.

Specification Part 4: Getting the Cheesecake Ordering App in Place

Examine the app.js file that was created for you by Express. This is the main script for your app. Find this line in the code:

```
app.use(express.static(path.join(__dirname, 'public')));
```

This line specifies what folder "static" content is stored in. Static content is server-side content that the client might download. It typically includes things like HTML, CSS and image files.

Follow these steps to get your cheesecake app setup in the Express project:

1. Copy your .html file from homework #2 into the hw3/public folder and rename it index.html.
2. Copy your .css file from homework #2 into the hw3/public/stylesheets folder
3. Modify the <link> tag in your newly copied index.html file so that it refers to the .css file's new location
4. Reload the app in your browser to see your cheesecake app instead of the "Welcome to Express" page

Now find this line in app.js: `app.use('/', indexRouter);`

This the code that instructs the app to load your index.html as the "root" page of the application. You should test the redirection is working by restarting the app and visiting this URL: http://localhost:3000/

To complete this assignment, you may need to learn more about the Express framework. You can find more information on Express's website: `https://expressjs.com/`

Specification Part 5: Organizing your Client Side JavaScript

Up until now, your client-side JavaScript code has been embedded directly in the HTML. To facilitate unit testing (and better organization), a web application should keep as much Javascript code as possible in separate files. It's time for you to do the same. Move the code you wrote for the previous assignment into separate file(s) in the hw3/public/javascript folder. The <script> tags in your HTML should use the src attribute to load your code.

Furthermore, properly define helper methods that perform discrete actions you want to take. For example, if your HTML code contains a <script> tag like this to "do stuff" whenever a "foo" element is clicked:

```
<script>
$(function() {
     $(".foo").click(function( event ) {
     /* do stuff */
     });
});
</script>
```

You would replace the script with a reference to a file like this:

```
<script src="javascripts/stuff.js"></script>
```

And then put the following code in hw3/public/javascripts/stuff.js:

```
eventHandler = function( event ) {
     /* do stuff */
}

$(function() {
     $(".foo").click(eventHandler);
});
```

Specifically, the "do stuff" has been moved to a helper method and the event handler connects that helper to the click event. From this point forward, you are expected to keep your JavaScript code in separate files.

Once you have moved the code to separate .js file, verify that the app is still working by reloading it in your web browser.

Sanity Check: Before proceeding to the next part, commit and push your changes to the git repository. There should be about a dozen new files and folders. Then, test that your repository is checked in properly by: cloning it into a different folder, navigating to the hw3 subfolder in your new clone, running `npm install` to download the needed libraries, running `npm start` to launch the app and reloading the app in your browser and testing it again. Now, delete your test clone so you don't get confused about where you are working. :)

Specification Part 6: Write a Simple Unit Test

Jest is a unit testing framework for JavaScript. You'll write some unit tests for this assignment and eventually use Jest for your project in CS341. To install Jest into your project, navigate to the hw3 folder in your project and run this command:

```
npm install --save-dev jest
```

Next, you want to create a subfolder of `hw3` named `tests`. This folder will contain all the unit tests for your project. Jest will automatically run any test files you place in this folder as long as the file name ends with `test.js` (example: `validate.test.js`).

In the `public/javascripts` folder, create a file called `sum.js`. In that file, write a little code that adds to integers to this file (This example was taken from the Getting Started section on Jest's website: https://jestjs.io):

```
function sum(a, b) {
      return a + b;
}
module.exports = sum;
```

In your tests folder, create a file named `sum.test.js` that contains this code:

```
const sum = require('../public/javascripts/sum.js');
      test('adds 1 + 2 to equal 3', () => {
            expect(sum(1, 2)).toBe(3);
});
```

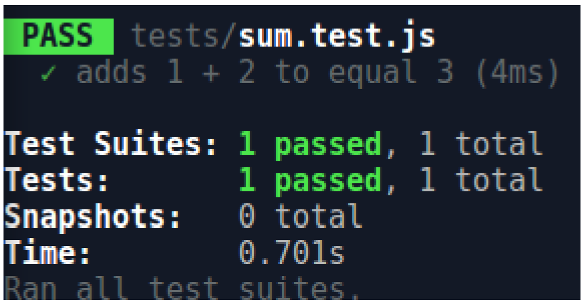
In your hw3 folder, locate the file named package.json. This is the main configuration file for a Node.js project. Edit this file and add a test script to the scripts section as shown below:

```
"scripts": {
      "start": "node ./bin/www",          ← Don't forget to add the comma.
      "test": "jest"                      ← Then, add this line.
},
```

In the hw3 folder, issue this command to verify that all is working:

```
npm test
```

You should see a message like the one in the image at right.



```
PASS  tests/sum.test.js
  ✓ adds 1 + 2 to equal 3 (4ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.701s
Ran all test suites.
```

Specification Part 7: Unit Testing for Client Side Code

The example sum.js file you created is fine but things get more complicated with a real app. The code you wrote for the previous assignment is designed to be included via a <script> tag. It assumes that a previous script tag has already loaded node.js and that the $ variable is available to access the DOM (i.e., the HTML document).

However, when running unit tests on the server side the code is not being executed inside a web browser. Therefore, there is no DOM and jQuery has not been loaded. Jest uses a "mock" DOM wherein you define a tiny HTML document and use that to test your code. Unfortunately this mock DOM is configured to ignore scripts. Your instructor has burned 20+ hours trying to figure out how to get around this to test your scripts. He has given up. If you can figure it out an effective and appropriate solution, expect to receive heaps of extra credit.

In the meantime, let's write a unit test to give you a taste of how a mock DOM would be used. Using the code below, create a second unit test in your project that also passes.

```
var fs = require('fs');

test('test selectEvent', () => {
       //Read the index.html file into a string
       var html = fs.readFileSync('public/index.html', 'utf8');
       expect(html).toEqual(expect.anything()); //any non-null value is okay

       //put the HTML into a testing DOM and do a sanity check
       document.body.innerHTML = html;
       const $ = require('jquery');
       expect($('h1').html()).toBe("Cheesecake Order Form");
```

To get the above to work, you'll need the jquery library to be part of your project. You could install it manually by issuing the command: `npm install jquery`. Unfortunately, that means it has to be manually re-installed each time that you download the app from version control. To make it a permanent part of your project, add it to the dependencies list in package.json. You also need to use the jsdom package: Update the package.json

```
"test":"jest --env=jsdom"
```

Comments and Formatting
Follow good programming practice with this assignment. Your HTML, CSS and Javascript code should contain helpful comments to make it easier for humans to read. Indent your code properly, particularly so that a reader can see where sections of your document begin and end.

Turning in your Homework
You are responsible for following these instructions correctly and turning in your homework assignments properly. I reserve the right to penalize your score for failing to do so. It also will frustrate those who are grading the assignment.

1. Be sure your name is in a comment at the top of each new file you created for this assignment.
2. Download a .zip file of your project from `github.com`. Rename the .zip file so that the filename contains your UP userid. Example: smithj22-CS341HW3-main.zip
3. Verify that your .zip is correct by unzipping it in a new location, issuing the `npm install` and `npm start` commands in that folder, and verifying that the server starts properly and you can use your app via a browser. The grader is not responsible for debugging your code. If the grader can not load your app this way you will receive a zero score.
4. Submit your appropriately named .zip file via the associated link on the course web site.
   ► Note: If the .zip file is too large, delete the node_modules folder from your project and re-zip it again.