**W05 Testing and Debugging**

**JavaScript Novice to Ninja, Chapter 10**

<https://www.sitepoint.com/premium/books/javascript-novice-to-ninja-2nd-edition/read/15/k01nwvxw>

Design your programs in a way that prevents the user from making errors.

Graphical user interface, application

Description automatically generated with medium confidence

Graphical user interface, website

Description automatically generated

Warnings are messages that allow the program to continue running.



Graphical user interface, application

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

You can use Linting tools to test the quality and consistency of your code.

*Feature detection.*

Feature detection is where you test a browser for support of a JavaScript feature.

Graphical user interface, text, application

Description automatically generated

*Debugging*

You can use alerts, console.logs, and breakpoints. Console.logs allow you to create your own stack trace. Console.trace() will log an interactive stack trace for you.

To set a breakpoint, you can use the debugger keyword, which will pause the execution in the code at that point and allow you to see the values. While paused, you can hover over variables and see their values. You can then restart the program by clicking the play button.

Remove any debugging code prior to shipping.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, letter, email

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, website

Description automatically generated

Graphical user interface

Description automatically generated

*MVC Application with modules and objects example*

<https://byui-cit.github.io/cit261/resources/hikes-MVC.html>

**Hikes: Model-View-Controller**

Here is a simple implementation of the MVC architecture of our Hikes app. It will also show examples of using classes and ESModules.

Start by creating a simple HTML file. You can use the code below as a starting point.

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="utf-8" />**

**<title>Great Hikes</title>**

**<meta name="viewport" content="width=device-width, initial-scale=1" />**

**</head>**

**<body>**

**<main>**

**<h1>Great Hikes</h1>**

**<ul id="hikeList" ></ul>**

**</main>**

**</body>**

**</html>**

Create one more file: **index.js** and add it with a script element into your html.

**Activity**



**MVC**

Before we take the next steps, let's think a moment about how we would like to organize our code. Not taking this time will almost always end up in what is known as 'Spaghetti code'. Hard to follow, maintain, and understand.

For this project we will use the Model-View-Controller (MVC) architecture. It provides a nice separation of concerns and will help us stay organized. This is a simple application and so we will probably only need one model, one view, and one controller. Create three more files in your project called:

* + **HikeModel.js**
  + **HikesView.js**
  + **HikesController.js**

In MVC models are responsible to handle all of the interactions with our data source. They often contain schema information describing the data as well.

Views take information (usually from the Model) and display it in whatever format is needed.

Controllers act as the glue between the model and view. Their job is to request information from the model when needed, and pass it on to the view to be displayed.



**HikeModel.js**

Let's begin by building the model: HikeModel.js. We will use a **class** to represent our model. I've provided a bit of code below to get you started. Copy it into your file.

**// normally the model would have more going on...retrieving the hikes from a database, adding hikes, editing hikes, filtering, etc. Our model will be very simple. We could simply export the hikeList, but a better pattern would be to create a 'getter' function to do it instead. That way as our model changed...we could simply change the getter function and anything using it should be able to remain the same.**

**const hikeList = [**

**{**

**name: 'Bechler Falls',**

**imgSrc: 'falls.jpg',**

**imgAlt: 'Image of Bechler Falls',**

**distance: '3 miles',**

**difficulty: 'Easy',**

**description:**

**'Beautiful short hike along the Bechler river to Bechler Falls',**

**directions:**

**'Take Highway 20 north to Ashton. Turn right into the town and continue through. Follow that road for a few miles then turn left again onto the Cave Falls road.Drive to the end of the Cave Falls road. There is a parking area at the trailhead.'**

**},**

**{**

**name: 'Teton Canyon',**

**imgSrc: 'falls.jpg',**

**imgAlt: 'Image of Bechler Falls',**

**distance: '3 miles',**

**difficulty: 'Easy',**

**description:**

**'Beautiful short hike along the Bechler river to Bechler Falls',**

**directions:**

**'Take Highway 33 East to Driggs. Turn left onto Teton Canyon Road. Follow that road for a few miles then turn right onto Staline Raod for a short distance, then left onto Alta Road. Veer right after Alta back onto Teton Canyon Road. There is a parking area at the trailhead.'**

**},**

**{**

**name: 'Denanda Falls',**

**imgSrc: 'falls.jpg',**

**imgAlt: 'Image of Bechler Falls',**

**distance: '3 miles',**

**difficulty: 'Easy',**

**description:**

**'Beautiful short hike along the Bechler river to Bechler Falls',**

**directions:**

**'Take Highway 20 north to Ashton. Turn right into the town and continue through. Follow that road for a few miles then turn left again onto the Cave Falls road. Drive to until you see the sign for Bechler Meadows on the left. Turn there. There is a parking area at the trailhead.'**

**}**

**];**

**// Hike Model**

**export default class HikeModel {**

**constructor() {**

**// We need a constructor...but in this case it isn't doing much**

**}**

**getAllHikes() {**

**// should return a list of all the hikes.**

**}**

**getHikeByName(hikeName) {**

**// filter the hikes for the record identified by hikeName and return it**

**return ;**

**}**

**}**

Notice I included a list of hike data at the top for us to work with. More often the data would get pulled from a database.



**HikesView.js**

Now let's move on to the View. The view should get sent some data, then format it and display it in the current page. As before we will use a class, and I will give you a head start:

**// Hike View handler**

**export default class HikesView {**

**constructor(listElementId) {**

**// will need this**

**this.imgBasePath = '//byui-cit.github.io/cit261/examples/';**

**}**

**renderHikeList(hikeList, listElement) {**

**// loop through our list of hikes building out the appropriate HTML for each and append it to the listElement**

**}**

**renderOneHikeLight(hike) {**

**// this method will be used to create the list of hikes with less detail: name, image, distance, difficulty**

**}**

**renderOneHikeFull(hike, parentElement) {**

**// this method will be used to one hike with full detail...you will need this for the stretch goal!**

**}**

**}**



**HikesController.js**

Remember that that earlier I said that the Controller was like the glue between the Model and View? Well in order to do that the Controller has to know about both. Making our model and view using Classes and modules will make it easy to do that. As before copy/paste the code below into your controller to get started.

**import HikeModel from './HikeModel.js';**

**import HikesView from './HikesView.js';**

**// Hike controller**

**export default class HikesController {**

**constructor(parentId) {**

**this.parentElement = document.getElementById(parentId);**

**// this is how our controller will know about the model and view...we add them right into the class as members.**

**this.hikeModel = new HikeModel();**

**this.hikesView = new HikesView(parentId);**

**}**

**showHikeList() {**

**// this will get called each time we need to display our full hike list. It should grab the list of hikes from the Model, and then send them to the view.**

**}**

**showOneHike(hikeName) {**

**// use this when you need to show just one hike...with full details**

**}**

**addHikeListener() {**

**// for the stretch you will need to attach a listener to each of the listed hikes to watch for a touchend.**

**}**

**}**



**Putting it all together**

The next step is to let our HTML know about our modules. In the **index.js** file import in your Controller, create an instance of the class, then call the **showHikeList()** method. You may want to console.log something out from that method until we create it to help us know if everything is working correctly. If there are errors, fix them :) If not move on to the next step. Make sure that **showHikeList()** does not get called before the DOM is ready.



**Finishing up**

The last step is go back in our model, view, and controller and finish writing the methods as indicated in the comments. I would recommend starting with the Model. Complete the **getAllHikes** method first (model), test it, then move on to the **showAllHikes** and **renderHikeList** methods. Test again.

After that work on the controller or View methods as it makes sense. I would recommend following the same order...finish a method, then test. It will keep your errors under control and make your code easier to troubleshoot.

**CSS Troubleshooting in Six Easy Steps**

<https://www.sitepoint.com/premium/courses/css-troubleshooting-in-six-easy-steps-2869>

Validate your css in a validator, which will give you a list of syntax errors.

If your rule is still not being applied, try to turn on borders for the element.

Check the weight of the rule.

Isolate the problem.

You can test different browser versions be using emulation. If you don’t have a specific browser you can use hacks for this. Use the star hack or you can include this html in your header.

Text

Description automatically generated

Then, you can write specific rules for any version of IE.

Text

Description automatically generated

Another tool you can use is BroswerStack.com.

**Google Chrome DevTools Documentation**

<https://developer.chrome.com/docs/devtools/>

**Assignment: CSS Utilities**



### Prepare

There is a bit more setup we need to do before we can start with Stylelint. Open a command prompt for your computer and run the following:

npm install -g stylelint-config-standard

If you get an error while installing, try just using this command: ***npm install -g stylelint.* If you get a permissions error, put sudo in front.**

Then change to a directory where you have a .css file you would like to check. Create a file there called .stylelintrc.json Put the following inside that file:

{

"extends": "stylelint-config-standard"

}



### Lint your code

Make sure you are still in the directory where created the .stylelintrc.json file (you may not see the file in the file manager, but if you do a ls -la on Mac or dir on Windows you should see it.) and follow the directions in CSS Mastery to run the stylelint utility on the file. How many errors did you have? Try outputting the errors to a file using this syntax:

stylelint stylesheet.css > csslintoutput.txt

Practice interpreting the error messages by fixing them until the test comes back clean.

Linters have become quite popular not just with CSS, but also HTML and Javascript. One thing to understand however is that they are **not** validators. They will find some errors, but they are more concerned with helping you write clean optimized, readable code. They tend to be very opinionated as a result. They don't just encourage you to write valid code, but correct code by avoiding bad practices...according to whomever setup the rules of the Linter.

You may or may not agree with all these opinions however, and so most Linting packages are configurable...you can use your own opinions instead of someone else's. Instructions on how to modify Stylelints defaults can be found at [Stylelint docs](https://github.com/stylelint/stylelint/blob/master/docs/user-guide/get-started.md).



### Analyze-Css

The next tool we will use is Analyze-Css. With the same CSS file as the last step run the analyze-css utility. "How?" you may be asking..."that tool wasn't in the reading and I've never seen it before!" Read on...

Try running each of these utilities without any arguments...just type the command. You should see some hints on how to use the command. This is pretty typical for these type of utilities and learning how to read this type of concise help is a good skill to develop.

if you run it for analyze-css for example you will see something like this:

analyze-css --url <url> [options]

Options:

--url Set URL of CSS to analyze

--file Set local CSS file to analyze

--ignore-ssl-errors Ignores SSL errors, such as expired or self-signed certificate errors

--pretty, -p Causes JSON with the results to be pretty-printed

--no-offenders, -N Show only the metrics without the offenders part

--auth-user Sets the user name used for HTTP authentication

--auth-pass Sets the password used for HTTP authentication

--version, -V Show version number and quit

--help, -h This help text

There are certain conventions that are almost always followed with these helps. Here are a few of the more common conventions:

* + Things written in ALLCAPS or in **bold** often means type it exactly as it appears,
  + and lowercase, italics, or surrounded with < > can mean you are supposed to change it to something specific.
  + Something surrounded in [ ] is going to be an optional part of the command. It will work with our without that part.
  + A | (pipe) between two parameters means that you can use one or the other, but not both.

We want to run the tool against a local file instead of a URL...and I see an option for that above. Something like this should work:

analyze-css --file filename.css

The results are not very readable however. How would you modify the command to give better output?

Most of what Analyze-css gives back is informational. It tells you how many selectors you are using, how specific they are (less specific is usually better), if you are using any prefixed css that you don't need to anymore, and then it will point out offenders and duplicates...selectors that you really should look at and optimize.



### Uncss

Finally run the uncss utility on your file. It is a great optimization tool. It looks at your css and finds any rules that were not used...then removes them! Try this for example. Take a look at the CSS for the [Foundation css framework](https://byui-cit.github.io/advcss/resources/foundation.css). Then take a look at this simple [example page](https://byui-cit.github.io/advcss/resources/foundation-all.html) that has the Foundation css linked in. Now run Uncss on it.

uncss http://byui-cit.github.io/advcss/resources/foundation-all.html

You may even want to send the results to a file like this:

uncss http://byui-cit.github.io/advcss/resources/foundation-all.html > optimized.css

What was the result?

Graphical user interface, text, application

Description automatically generated

**Instructor’s solution MVC App**

<https://github.com/byui-cit/cit261/tree/gh-pages/solutions/week05>