

# HW3 Programming: Your First Crowdfunding Site:

## Instructions

---

When you're ready to submit your solution, go to the [assignments list](#).

## Background

In this programming assignment we are going to build and deploy an extremely simple crowdfunding site. The site will not actually be able to accept orders yet, but will be on the web. It will be written in basic HTML and CSS, and the presence/absence of particular HTML tags will be scored with an SSJS `grader.js` script.

The process will expose you to the mechanics of editing and deploying code in a Linux development environment. You will also learn the basics of writing simple command line applications in `node.js`. Note again that all assignments assume you are running code on AWS on a EC2 t1.micro instance running Ubuntu 12.04.2 LTS, unless otherwise specified.

This is the first step towards our final project. As noted in the [syllabus](#), you will be graded on the technical aspects of this process (presence and absence of HTML tags, correctness of JSON output), but it is the qualitative aspects that your customers will ultimately take into account.

## Part 1: Deploy some basic HTML edits

In this part we will modify the deploy from [this lecture](#) to read in an HTML file. This is meant to put together much of the content from Lectures 2-4, from heroku to emacs to git to github.

1. First, [fork](#) (or [duplicate](#)) the `node-js-sample` repository that you deployed in [this lecture](#) into your own Github account.
2. Rename it to `bitstarter` by going to the settings URL, which will be like `https://github.com/startup-class/node-js-sample/settings`, except with your own username rather than "startup-class")
3. Next, redo the deploy with this new `bitstarter.git`, such that you again get a Hello World site, as shown below.

The screenshot shows a terminal window on the left and a web browser on the right. The terminal window is running commands to create a Heroku app, push code, and deploy it. The web browser shows the deployed application at `fierce-reaches-1070.herokuapp.com` displaying "Hello World!".

```
[ubuntu@ip-172-31-36-95:~]$ cd bitstarter/
[ubuntu@ip-172-31-36-95:~/bitstarter]$ heroku create
Creating fierce-reaches-1070... done, region is us
http://fierce-reaches-1070.herokuapp.com/ | git@heroku.com:fierce-reaches-1070.git
Git remote heroku added
[ubuntu@ip-172-31-36-95:~/bitstarter]$ git push heroku master
Counting objects: 277, done.
Compressing objects: 100% (237/237), done.
Writing objects: 100% (277/277), 192.33 KiB, done.
Total 277 (delta 12), reused 277 (delta 12)

-----> Node.js app detected
-----> Resolving engine versions
    Using Node.js version: 0.8.25
    Using npm version: 1.1.65
-----> Fetching Node.js binaries
-----> Vendors node into slug
-----> Installing dependencies with npm
    npm WARN package.json node-example@0.0.1 No README.md file found!
    npm WARN package.json node-connect@1.9.2 No README.md file found!
    express@2.5.11 /tmp/build_1o4sbc72oogi/node_modules/express
    connect@1.9.2 /tmp/build_1o4sbc72oogi/node_modules/express/node_modules/connect
    qs@0.4.2 /tmp/build_1o4sbc72oogi/node_modules/express/node_modules/qs
    mime@1.2.4 /tmp/build_1o4sbc72oogi/node_modules/express/node_modules/mime
    formidable@1.0.11 /tmp/build_1o4sbc72oogi/node_modules/express/node_modules/formidable
    mkdirp@0.3.0 /tmp/build_1o4sbc72oogi/node_modules/express/node_modules/mkdirp
    Dependencies installed
-----> Building runtime environment
-----> Discovering process types
    Procfile declares types -> web

-----> Compiled slug size: 3.8MB
-----> Launching... done, v4
    http://fierce-reaches-1070.herokuapp.com deployed to Heroku

To git@heroku.com:fierce-reaches-1070.git
* [new branch] master -> master
[ubuntu@ip-172-31-36-95:~/bitstarter]$
```

- Now use emacs to edit the relevant line in `bitstarter/web.js` to become "Hello World 2!"; here's the file of interest:

The screenshot shows a terminal window with the command `cat web.js` executed, displaying the contents of the `web.js` file. The file contains a simple Express.js application that listens on a specified port and responds with "Hello World!".

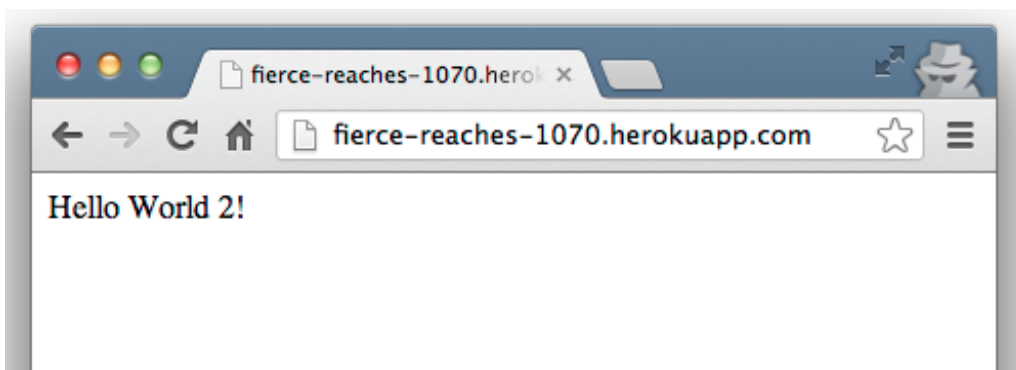
```
[ubuntu@ip-172-31-36-95:~/bitstarter]$ cat web.js
var express = require('express');

var app = express.createServer(express.logger());

app.get('/', function(request, response) {
  response.send('Hello World!');
});

var port = process.env.PORT || 5000;
app.listen(port, function() {
  console.log("Listening on " + port);
});[ubuntu@ip-172-31-36-95:~/bitstarter]$
```

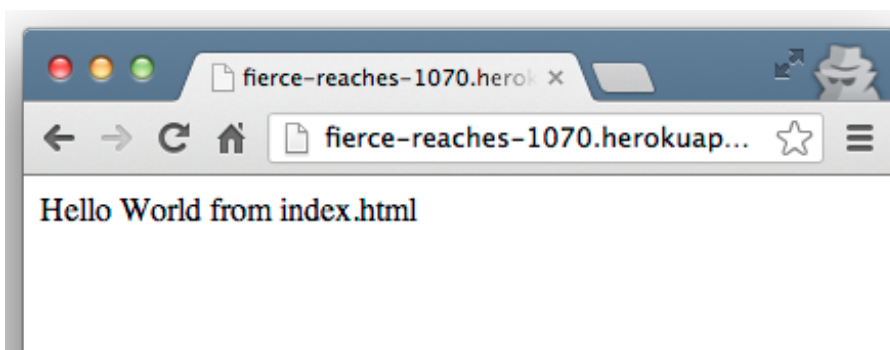
- Then execute `git add web.js`; `git commit -m "Updated web.js"` after you make the edit (see [here](#) and [here](#) to refresh). Finally, redo the deploy. You should see something like this:



6. Now create a file named "index.html" with the string "Hello World from index.html" in it, and `git` add it to the repo, as shown. From this point forward we won't remind you to `git add` all the time.

```
ubuntu@ip-172-31-36-95:~/bitstarter]$ cat index.html
Hello World from index.html
ubuntu@ip-172-31-36-95:~/bitstarter]$ git add index.html
ubuntu@ip-172-31-36-95:~/bitstarter]$ git commit -m "Added index.html"
[master 3071706] Added index.html
1 file changed, 1 insertion(+)
create mode 100644 index.html
ubuntu@ip-172-31-36-95:~/bitstarter]$
```

7. Next, look at [fs.readFileSync](#), the [Buffer.toString](#), and this [article](#). Modify `web.js` again, except this time have it read "index.html" from disk and use that data in place of "Hello World 2!". Now redo the deploy. You should see something like this:



8. Finally, do `git push origin master` to push your commits to your Github account. This is different from `git push heroku master` which actually deploys the code to Heroku.

**Submit:** Submit a comma-delimited file with your github repo URL and your Heroku URL. It should have one line and look like this:

```
http://github.com/startup-class/bitstarter, http://fierce-reaches-1070.herokuapp.com
```

## Part 2: Wireframe and Edit HTML/CSS

Now we are going to modify the deploy to get up a first cut of our bitstarter website.

1. Begin by taking a look at the websites for [Selfstarter](#), [Lockitron](#), and the [Light Table](#) project on Kickstarter. We have isolated a few common elements of these sites in this wireframe:

Bitstarter Wireframe											
Grid Layout of HTML Elements											
navigation	Logo		[Blank]						About		
heading	Heading										
subheading	Subheading										
pitch	Video						Thermometer				
							Order				
							Social				
section1	Marketing Section 1										
section2	Marketing Section 2										
faq	FAQ										
footer	Footer										

2. Our first step is to lay this out in HTML. Use emacs to modify `bitstarter/index.html`, literally typing in the file shown in these images.
  - NOTE: it's ok if you don't fully understand the code. In a nutshell, we're pulling in [Twitter Bootstrap](#) and defining some CSS styles in the head. Then we are laying out a series of `div` elements. By doing this out manually you'll have a greater appreciation and a point of reference for when we cover HTML/CSS theory.
  - NOTE ALSO: We know that typing it in is a pain, but in doing so you pay much more attention to detail than when simply copy/pasting; see [Zed Shaw](#). This will also give you practice editing a file in emacs. (That said, if and only if you have accessibility issues and actually can't type from a screenshot, you can start with the [file here](#).)

## index.html

```

1<!DOCTYPE html>
2<html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>My First Bitstarter</title>
6    <link type="text/css" rel="stylesheet"
7      href="https://d396qusza40orc.cloudfront.net/startup%2Fcode%2Fbootstrap-2.3.2.css">
8    <style type="text/css">
9      body {
10        padding-top: 20px;
11        padding-bottom: 40px;
12      }
13      .container {
14        width: 960px;
15      }
16      p.lead {
17        padding-top: 15px;
18      }
19      .navigation, .pitch, .section1, .section2, .faq, .footer {
20        padding: 10px 0px 10px 0px;
21      }
22      .video, .thermometer, .order, .social {
23        border: 1px dotted;
24        text-align: center;
25      }
26      .video {
27        /* Internal borders have 1px width, thus need to add 4 x 1px to 120px. */
28        height: 124px;
29        line-height: 124px;
30      }
31      .thermometer, .order, .social {
32        /* line-height to vertically center: http://phrogz.net/css/vertical-align/index.html */
33        height: 40px;
34        line-height: 40px;
35      }
36      div.row {
37        border: 1px solid;
38      }
39    </style>
40  </head>
41  <body>
42    <div class="container">
43      <div class="row navigation">

```

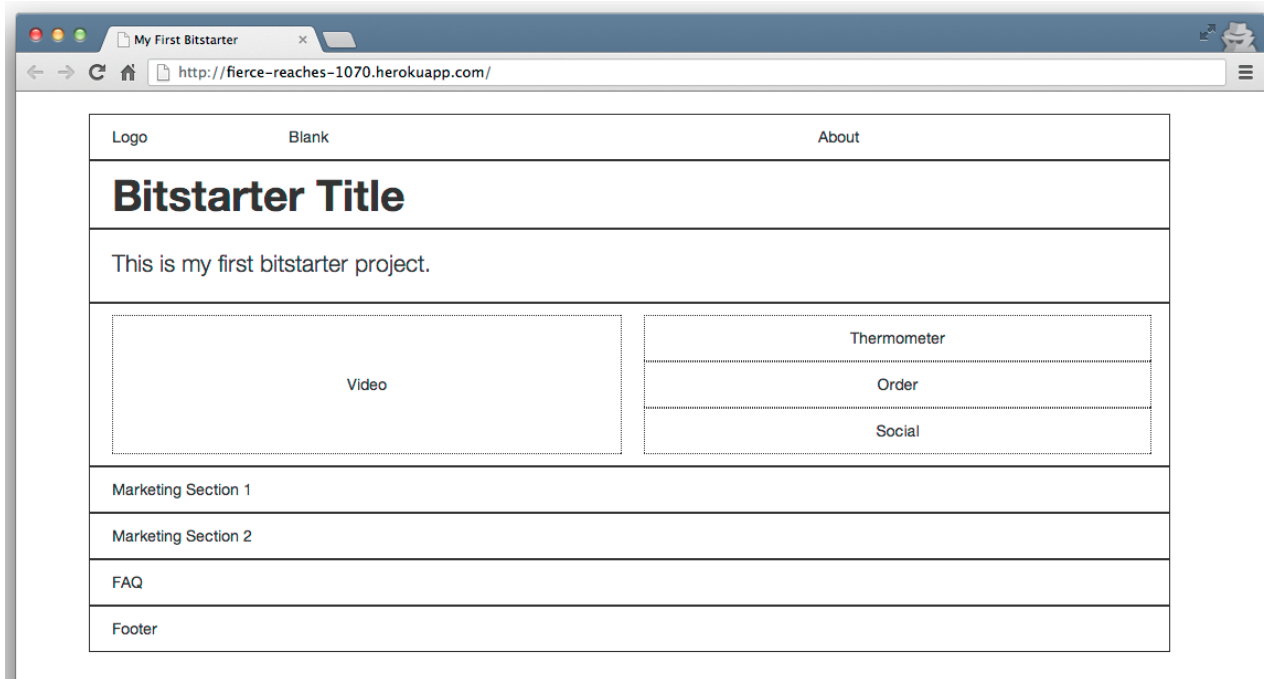
```

44     <div class="span2 logo">
45         Logo
46     </div>
47     <div class="span6 blank">
48         Blank
49     </div>
50     <div class="span4 about">
51         About
52     </div>
53 </div>
54 <div class="row heading">
55     <div class="span12">
56         <h1>Bitstarter Title</h1>
57     </div>
58 </div>
59 <div class="row subheading">
60     <div class="span12">
61         <p class="lead">This is my first bitstarter project.</p>
62     </div>
63 </div>
64 <div class="row pitch">
65     <div class="span6 video">
66         Video
67     </div>
68     <div class="span6">
69         <div class="thermometer">
70             Thermometer
71         </div>
72         <div class="order">
73             Order
74         </div>
75         <div class="social">
76             Social
77         </div>
78     </div>
79 </div>
80 <div class="row section1">
81     <div class="span12">
82         Marketing Section 1
83     </div>
84 </div>
85 <div class="row section2">
86     <div class="span12">
87         Marketing Section 2
88     </div>
89 </div>
90 <div class="row faq">
91     <div class="span12">
92         FAQ
93     </div>
94 </div>
95 <div class="row footer">
96     <div class="span12">
97         Footer
98     </div>
99 </div>
100 </div>
101 </body>
102</html>

```

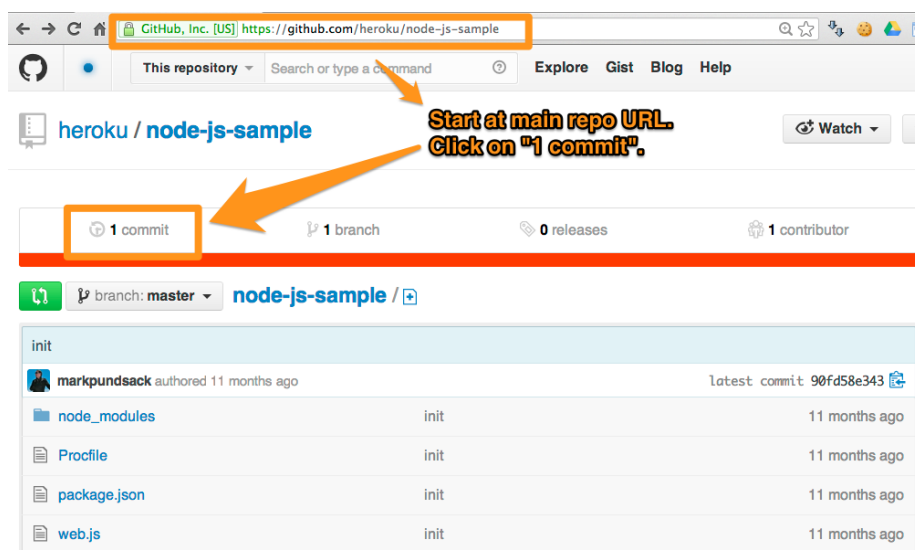
3. Commit your changes along the way and periodically do `git push origin master`. You can preview the `index.html` file by using `scp` or `rsync` to copy it locally and then using [Chrome](#) to look at it. Or you can use [rawgithub](#), which is even more convenient.

4. When you finish and do `git push heroku master` it should look like this:



Great job. You now have a simple v1 crowdfunder up. You can edit the text further to start figuring out what you want to market. As noted in the first lecture, the text (aka [marketing copy](#)) that you write within the HTML tags will not be graded, but will serve to promote your crowdfunder.

**Submit:** Submit the permalink to your final `index.html` file in your github repository by using the github.com web UI. See below:





Now click "Browse code" to look at a specific commit

1) Note that the URL has a SHA1 in it. We are now browsing a specific commit.

2) We can now click an individual file to get a true permalink. Do this for your bitstarter repo, for your final index.html file.

This is the kind of URL you want. A specific permalink for a piece of code within a given git commit, globally unique and specified.

But do it for your index.html, not web.js

```

1  var express = require('express');
2
3  var app = express.createServer(express.logger());
4
5  app.get('/', function(request, response) {
6    response.send('Hello World!');
7  });
8
9  var port = process.env.PORT || 5000;
10 app.listen(port, function() {
11   console.log("Listening on " + port);
12 });

```

So, you should submit a URL that looks something like this:

```
https://github.com/YOUR-USERNAME/bitstarter/blob/90fd58e343c29e54fc1d4e19036595500b2bdc21/index.html
```

It should NOT look like this with master in the URL:

```
https://github.com/YOUR-USERNAME/bitstarter/blob/master/index.html
```



See [here](#) for more details; in short the "master" URL will keep changing as you make more commits, so it won't always point to the same data. The link with the SHA1 hash in it will not have that problem.

## Part 3: Write SSJS headless grader

- Now that you have an extremely basic crowdfunding site up and running, we are going to write a simple node.js command line application which takes your crowdfunding URL as an input and returns a list of boolean flags in JSON format that check for the presence of HTML tags.
- Use `npm` at the command line to install two new node libraries, like we did last time with `restler`:

```
$ npm install cheerio
$ npm install commander
```

- We are going to call this script `grader.js`. Start with the following skeleton. Note that we are using a new node.js library for developing a command line application, and that we express the main routine as a function of the arguments given on the command line.
- You should read the code closely and use the [emacs SSJS REPL](#) to interact with it and figure out what the snippets do. In particular, reading a bit about JSON ([1](#), [2](#)) may be helpful.

### `grader.js`

```
#!/usr/bin/env node
/*
  Automatically grade files for the presence of specified HTML tags/attribut
  es.
  Uses commander.js and cheerio. Teaches command line application developmen
  t
  and basic DOM parsing.

  References:

  + cheerio
    - https://github.com/MatthewMueller/cheerio
    - http://encosia.com/cheerio-faster-windows-friendly-alternative-jsdom/
    - http://maxogden.com/scraping-with-node.html

  + commander.js
    - https://github.com/visionmedia/commander.js
    - http://tjholowaychuk.com/post/9103188408/commander-js-nodejs-command-
    line-interfaces-made-easy

  + JSON
    - http://en.wikipedia.org/wiki/JSON
```

```
- https://developer.mozilla.org/en-US/docs/JSON
- https://developer.mozilla.org/en-US/docs/JSON#JSON_in_Firefox_2
*/

var fs = require('fs');
var program = require('commander');
var cheerio = require('cheerio');
var HTMLFILE_DEFAULT = "index.html";
var CHECKSFILE_DEFAULT = "checks.json";

var assertFileExists = function(infile) {
    var instr = infile.toString();
    if(!fs.existsSync(instr)) {
        console.log("%s does not exist. Exiting.", instr);
        process.exit(1); // http://nodejs.org/api/process.html#process_
        process_exit_code
    }
    return instr;
};

var cheerioHtmlFile = function(htmlfile) {
    return cheerio.load(fs.readFileSync(htmlfile));
};

var loadChecks = function(checksfile) {
    return JSON.parse(fs.readFileSync(checksfile));
};

var checkHtmlFile = function(htmlfile, checksfile) {
    $ = cheerioHtmlFile(htmlfile);
    var checks = loadChecks(checksfile).sort();
    var out = {};
    for(var ii in checks) {
        var present = $(checks[ii]).length > 0;
        out[checks[ii]] = present;
    }
    return out;
};

var clone = function(fn) {
    // Workaround for commander.js issue.
    // http://stackoverflow.com/a/6772648
    return fn.bind({});
};
```

```
if(require.main == module) {
  program
    .option('-c, --checks <check_file>', 'Path to checks.json', clone(
assertFileExists), CHECKSFILE_DEFAULT)
    .option('-f, --file <html_file>', 'Path to index.html', clone(asse
rtFileExists), HTMLFILE_DEFAULT)
    .parse(process.argv);
  var checkJson = checkHtmlFile(program.file, program.checks);
  var outJson = JSON.stringify(checkJson, null, 4);
  console.log(outJson);
} else {
  exports.checkHtmlFile = checkHtmlFile;
}
```

## checks.json

This is the *initial* checks.json file. You will modify this to add more checks (see below).

```
["h1",
 ".navigation"]
```

## Sample usage

Here is a sample use case.

```
[balajis@jiunit:~]$head -n3 index.html
<!DOCTYPE html>
<html lang="en">
  <head>

[balajis@jiunit:~]$cat checks.json
["h1",
 ".navigation"]

[balajis@jiunit:~]$./grader.js --checks checks.json --file index.html
{
  ".navigation": true,
  "h1": true
}
```

Modify this script as follows:

1. First, add the initial versions of `grader.js` and `checks.json` to the same `bitstarter` github repo and do `git add`, `git commit`, `git push`.
2. Update `checks.json` to include checks for all the following classes in the wireframe. Here's the list of checks; make sure to encode as JSON in `checks.json`.

```
h1
.navigation
.logo
.blank
.about
.heading
.subheading
.pitch
.video
.thermometer
.order
.social
.section1
.section2
.faq
.footer
```

You can probably figure out how to encode these in `checks.json` from context, but in a bit more detail: The idea here is that in the `checkHtmlFile` function, with the current `checks.json` we are only confirming that there is at least one `<h1>` tag ([see here](#)) and at least one HTML element with a `.navigation` CSS class ([see here](#)) in our `index.html` document. However, we want to test that the `index.html` file also has `.thermometer`, `.pitch`, and all the other classes depicted in the [wireframe](#). So we need to add more rules to `checks.json`. (This really isn't a hard problem; just explaining what the idea is). We'll cover HTML/CSS in far greater detail soon, but try messing around with [cheerio](#) in your Emacs node REPL to get a sense of how this testing process works programmatically.

3. Update `grader.js` to take either a file or a URL as an input on the command line. Use the [restler](#) library from the previous HW to download the URL. Your final output should look like this, except it should include all the checks and not just the one for `navigation`.

```
[balajis@jiunit:~]$ ./grader.js --checks checks.json --url http://fierce-reaches-1073.herokuapp.com
{
  ".navigation": true,
  "h1": true
}
```

**Submit:** Submit the JSON output from running it on your sample URL (Output Submission) and

your revised `grader.js` (Additional Submission). Yes, we know there are ways you can game this, but we're trying to be nice :) However, we'll look into setting up external automated graders for the next assignment.

## Hints and Tips

Here are some sample files to help in your debugging.

```
$ wget https://spark-public.s3.amazonaws.com/startup/data/hw3/test.tar.gz
$ tar -xvzf test.tar.gz
$ cd test
$ wget https://spark-public.s3.amazonaws.com/startup/code/grader.js
$ chmod u+x grader.js
```

Then execute the following commands

```
$ ./grader.js -c checks-test-1.json -f test1.html
{
  ".navigation": false,
  "h1": true
}

$ ./grader.js -c checks-test-1.json -f test2.html
{
  ".navigation": true,
  "h1": true
}

$ ./grader.js -c checks-test-2.json -f test1.html
{
  "div div div h1": true,
  "div div.subheading": true,
  "div.footer": true,
  "title": true
}

$ ./grader.js -c checks-test-2.json -f test2.html
{
  "div div div h1": true,
  "div div.subheading": false,
  "div.footer": false,
  "title": true
}
```

