

# Specs for 201

▼ Class	
🕒 Created	@Oct 26, 2020 12:30 PM
🔗 Materials	
☑ Reviewed	<input type="checkbox"/>
▼ Type	

## Project Roadmap

---

Welcome to the first ever EECS 201 Project! This is one of the four options that we'll have available to choose from, and it deals with concepts important to a lot of your upper levels and potentially your future career: the web!

What we're going to ask you to do is to make a personal website and blog, where you automate most, if not all of it!

To complete the project, here's the general roadmap of what you need to do:

### 1. Read the Project Introduction

- This project will take some time, as we do not want to assume you have web experience and want you to learn more tools.
- Be sure to read the entire project completely before starting - certain requirements might require you to do things in a certain order, or they might require you to organize your project in a certain way.
- Feel free to talk to us! We're here to see you succeed, and OH has been relatively quiet these past few weeks...

### 2. Implement each section entitled "sub-assignment".

- Each of these sections will give you resources, along with a general idea as to what you need to implement for the project.
- You're allowed to use as many libraries as you want or like, unless we state that you NEED to use X library or Y library. This is mainly as, for

some projects, we want you to learn something new, and we know that certain libraries start out a lot easier but end up being less versatile than others.

3. After implementing each sub-section, join us in OH for a chat!

- This is mainly so we know who is working on these and how far they are.
- Additionally, this lets us make sure that everyone is accountable for what they said they'd do, ensuring we can give a grade that is fair and reasonable to anyone who embarks on these tasks.
- Completing any sub-section of this project will be worth 10 points, so completing all of them WILL get you all the points you need for your assignment. However, each section builds on the previous sections, so to get the full points we suggest starting from the top and moving down.

## Sub-Assignment 1: Making a website

Before we start talking about automating a website, we'll need a website to work on!

What you're going to start off with here is a website functionally similar to the following:

All posts

Summaries on Course Material, Simplifications of Older Materials, and More!

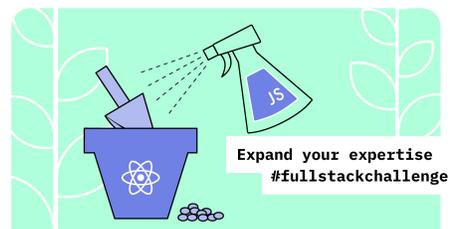
 <https://arav-agarwal2.github.io/>

If you want to learn web development as you go, we recommend the following websites and courses to look at.

Full stack open 2020

Parts 0-8 of the course material is written by Matti Luukkainen.  
The content of part 9 is written by developers from Terveystalo.  
Part 10 is written by Kalle Ilves. Numerous people have improved

 <https://fullstackopen.com/en/>



freeCodeCamp.org

Learn to code at home. Build projects. Earn certifications. Since 2014, more than 40,000 freeCodeCamp.org graduates have gotten jobs at tech companies including Google, Apple, Amazon,  <https://www.freecodecamp.org/>

freeCodeCamp 

Here's all that we're looking for to complete this portion:

1. Having two routes on the website:

- `/blog` , which will lead into a blog page with a few subpages as blog-posts.
- `/home` , which will lead into a home page detailing something about yourself, or whatever you'd like to place here.

2. Have the website hosted on either Heroku or Github Pages, two free hosting opportunities for students.

While we'd love to see some CSS and JS on your website, this can be done in pure HTML, so feel free to go as complicated or as simple as you'd like! The point here is to build a website that you could show off to a recruiter and feel proud of, so if that takes some time, it's bound to look really cool!

## Sub-assignment 2: Automation, Part 1

Now that we have some basic website set up, it's time to work on the more interesting part: Automation!

What we want for you to do is to automate this website a little bit. Instead of having to write HTML or JS files all day and night, wouldn't it be awesome if, when you commit a markdown file to a specific folder in your repository, you also convert it to an HTML file and add it to your website?

With what you've learned in this class, you can do precisely that!

Here's what we're looking for, and what you'd show to us to prove you've completed this portion of the assignment:

- You should have both a "source" directory and a "build" directory in your project. Your "build" directory, and all of the HTML files in it, shouldn't be

added to the Git Repository you're developing in.

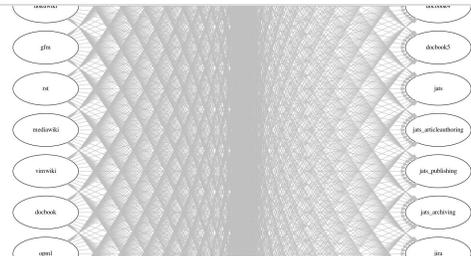
- When you commit a markdown file to your "source" directory, you should not only be able to both automatically generate the HTML page that will display your markdown, but also be able to automatically generate the HTML file that indexes all of your Markdown blog posts, allowing you to simply add a Markdown file, commit it to your repo, and get it to show up on your website!
- Have some decent error handling. If a markdown file is invalid, then you should find some way to let you know that, and probably prevent the commit from happening in the first place.

To help you get started, here is a list of resources that could be useful to accomplish this task. We'd like to say that, like the previous task, there are many, many ways to do this - so if you think you know an easy way to do so, do it, and show us!

#### About pandoc

If you need to convert files from one markup format into another, pandoc is your swiss-army knife.

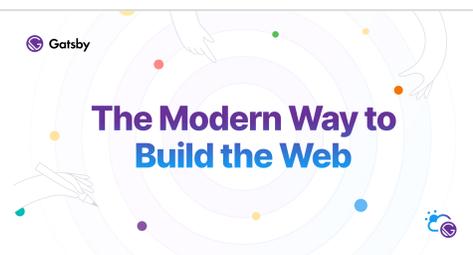
<https://pandoc.org/>



#### Static Site Generator

Learn what a static site generator is and why you might choose a static site generator, such as Gatsby, over other publishing tools. A static site generator is a software application that

<https://www.gatsbyjs.com/docs/glossary/static-site-generator/>



#### myles/awesome-static-generators

A curated list of static web site generators. Contribute to myles/awesome-static-generators development by creating an account on GitHub.

<https://github.com/myles/awesome-static-generators>



Again - while the sites above might lead to tools that can do a lot more than what we're asking for, we would like to stress that all that we're asking for can be done

with Pandoc and some command-line fu!

---

## Sub-assignment 3: Automation, Part 2

Alright! Woo!

You've done a lot to get here - but now let's kick it up one more notch.

Let's say your blog is getting more and more popular now, and you want to add some testing to your code. After all, making sure that your website works when we're automating it is a vital part of web-development, and doing it effectively is much more tricky than it can seem.

In particular, however, we want you to use Selenium to test your websites. While there are many, many other website-testing frameworks, Selenium is unique in that it directly simulates a browser, and also allows you to see your tests happen in real-time.

As this task will require you to learn a new library and, potentially, start learning a new language, we just want your Selenium test script to do the following:

1. Navigate to your blog-page
2. Click on all of the links, and report the title of each blog post to the command line.
3. If any of the links fail, return 1 and let us know that a certain link does not lead to any page yet, or it leads to a 404 page.

Additionally, we want this script to be part of your build process in the previous sub-assignment, so when you try to build your website, you end up not just building it, but testing that everything works out and exists.

To help you out, here's the documentation for Selenium with Python, the most common version of the testing framework:

### Selenium with Python - Selenium Python Bindings 2 documentation

Note This is not an official documentation. If you would like to contribute to this documentation, you can fork this project in GitHub and send pull requests. You can also send your feedback to my email: baiju.m.mail AT gmail DOT com. So far 50+ community members have contributed to this project (See the

 <https://selenium-python.readthedocs.io/>

## Sub-assignment 4: Automation, Part 4!

By now, you have a ridiculously slick little website - one that creates new pages, tests them, and prints all of those results to you on your command line!

This is all super useful, but there's still one set of things we can do to make our website the most over-engineered personal website ever - Lighthouse testing.

Lighthouse is a tool created by Google that vets your website to make sure that it follows certain web "best practices", such as Search Engine Optimization, Accessibility, and so on.

All we ask is for you to automatically collect data from Lighthouse whenever you push to your GitLab/GitHub repo, and have it displayed as a part of the GitHub/GitLAB CI, or Continuous Integration, pipeline whenever you commit to your website's repository.

To help with this, here's the tutorial from the Lighthouse GitHub page:

### GoogleChrome/lighthouse-ci

This document provides a step-by-step walkthrough on how to setup Lighthouse CI on your repository. After this guide, your build system will be running Lighthouse on your project's URLs

 <https://github.com/GoogleChrome/lighthouse-ci/blob/master/docs/getting-started.md>

