EE461L HW 5

Frontend Technologies

Task 1: Web Development Fundamentals

HTML, CSS, and JavaScript are the three foundational technologies that drive the web. HTML is what we use to structure a page, CSS is what we use to design it, and JavaScript is the language we use to implement its functionality and behavior. These days we typically use web *frameworks* to create web applications, but all of those frameworks still fundamentally build on HTML, CSS, and JavaScript.

Knowing the basics is helpful when you move on to learning more powerful frameworks, and it also comes in handy when you need to build simpler pages that don't require the full capabilities of a web framework. One great use case for a static HTML/CSS page is a *personal website*.

With that in mind, for this task you'll be:

1. Building a single-page personal website in HTML/CSS (no JavaScript, for now)
2. Deploying it with GitHub Pages[1]

Getting Started

1. Closely follow the instructions for GitHub Pages to create an empty repo on your GitHub account that you can clone to your computer and start building your website in. When you commit changes and push this repo, your website is deployed on the web at the URL: https://yourgithubusername.github.io.
2. Once you've cloned your new, empty repo from GitHub, create two files in it:
    a. `index.html` – your homepage
    b. `stylesheet.css` – the external stylesheet containing your page's styles
3. Now, build and deploy a personal site meeting the requirements in the following section.

[1] GitHub Pages is a free and popular place to host personal websites for software engineers. It's also used to host the documentation for many many popular open-source projects.

Requirements

Feel free to put whatever content on your personal website you deem appropriate (within reason, of course). It can show off your hobby, highlight your professional experience, or whatever else you feel comfortable with. Just be sure it meets the following requirements:

Requirement	**Points**
Use a CSS selector to apply style(s) to at least one HTML tag.	0.2
Create at least one class, apply it to at least two elements, and style it.	0.2
Create at least one id, apply it to an element, and style it.	0.2
Use `<div>` to create at least two sections of content on the page.	0.2
Use `<a>` to link to another website.	0.2
Use a CSS selector to change the color of `<a>` when you hover the cursor over it.	0.2
Change at least two box model properties of any tag, class, or id.	0.2
Include an image (something safe for work).	0.2
Correctly use a `<meta>` “description” tag to describe your site to search engines.	0.2
Deploy the page to yourgithubusername.github.io using GitHub Pages.	0.2

Note that some of these requirements overlap. Some clever web design will allow you to satisfy multiple requirements at once.

Deliverable

- Submit a link on Canvas to the GitHub repo hosting your site.
    - **Note 1:** If you have already used your GitHub Pages repo for a past project, create your website in a different repo on GitHub and submit a link to that along with a link to your existing GitHub Pages site.
    - **Note 2:** Do not push changes after the deadline (until the assignment is graded). We will consider the date and time of the last commit as the turn-in date.

Optional Reading

If you want to learn more about modern static site design (for your portfolio, user documentation, a self-hosted blog, etc.), consider looking into Jekyll or Hugo. These are *static site generators* which take a collection of templates and Markdown files and compile them into basic HTML and CSS for a static site. This is beyond the scope of our class, but interesting nonetheless.

Task 2: Intro to React

Having grasped the basics of web development, it's time to move on to React. React is one of many JavaScript frameworks used to build more powerful web applications. These frameworks use OOP-esque programming models to better structure, organize, and embed logic into the frontend. While the resulting application is still fundamentally HTML, CSS, and JavaScript under the hood, frameworks provide a lot of useful abstraction and functionality that helps developers focus on building feature-rich products.

The reason we are teaching React is because it has the least steep learning curve of the most popular JavaScript frameworks. It is nonetheless highly capable, evidenced by its use at Facebook (where it was created), Uber, AirBnb, and Netflix, among others.

With that in mind, for this task you'll be:

1. Setting up the toolchain needed to get started with React development
2. Following a tutorial to build part of a tic-tac-toe game in React

Getting Started

1. Follow the instructions for setting up the "Create React App" development toolchain.
2. Run `npx create-react-app tictactoe` on the command line to initialize a starter React app somewhere on your drive.
3. Verify that it works by running `npm start` in the `tictactoe` directory. You should be able to access it in the browser at http://localhost:3000 if it's working.

Requirements

Your sole requirement for this task is to complete part of the tic-tac-toe tutorial from the React documentation and submit the resulting code.

Requirement	**Points**
Follow the tutorial to create the tic-tac-toe game through the end of the “Declaring a Winner” section. “Adding Time Travel” onward is optional.	2

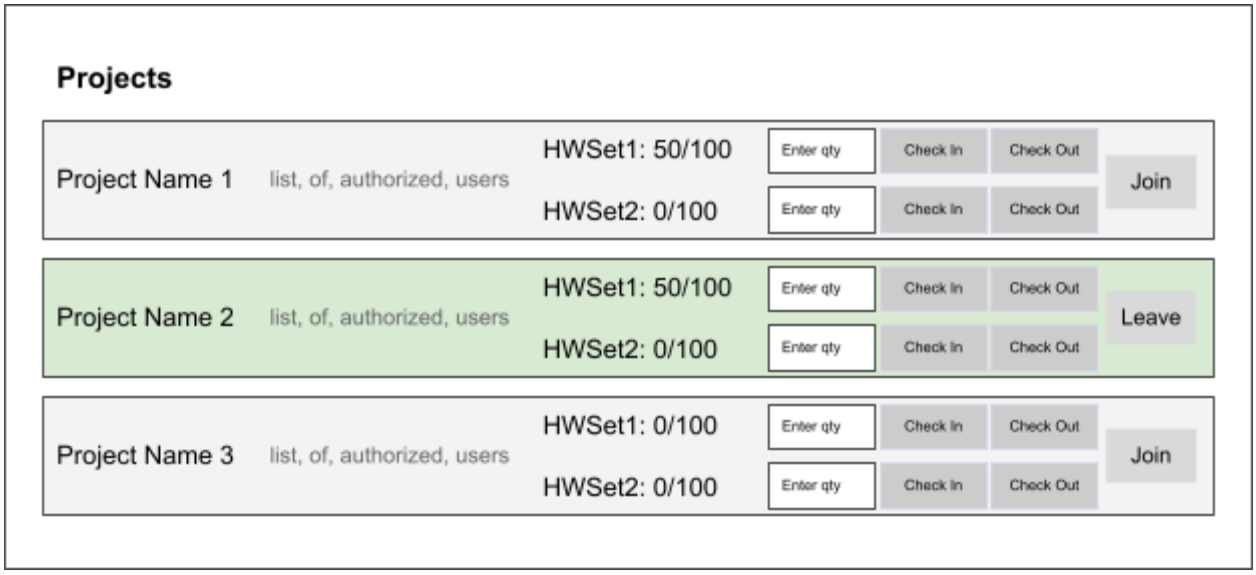
Deliverable

- Submit a zip of the `tictactoe` folder *without* *`node_modules`*
    - Remember: thanks to the development toolchain, you can always use `npm install` to reinstall the dependencies in `node_modules`. You should distribute your projects without `node_modules`, because they take up a lot of space.

Task 3: Custom React App

Now comes the interesting part. By now you’ve learned a bit about the fundamentals and have also figured out how to get started with React. In this task, you’ll use your knowledge to start building some React components for the frontend portion of your team project.

Below is a sketch of the `Projects` component where users can see a list of projects they are authorized to join, view the hardware checked out from each hardware set, and join and leave projects. Think for a moment about how this page could be broken up into a React component hierarchy. While there may be a lot on the page, it can be built with just a few reusable components. You may find it helpful to draw it up on a whiteboard or on paper. Once you’ve done that, you can move on to completing this task: implementing the `Projects` page.



Getting Started

1. Run `npx create-react-app yourappname` to initialize a starter React app somewhere on your drive.
2. Create a new component called `Projects` that is returned by the `render` function in the `App` component created by `create-react-app`.
3. Implement the `Projects` component and any necessary child components consistent with the requirements in the following section.

Requirements

Requirement	**Points**
Use at least two components (e.g., `Button`) from a library like Material UI.	0.4
Implement *at least* two components other than `Projects`.	0.4
Reuse one of your custom components multiple times.	0.4
Pass props from a parent to a child at least twice (reused components only count once toward this requirement).	0.4
Use a custom event handler to modify a component's state at least once.	0.4

Some notes:

- You are *only* implementing the frontend for this task. Your projects page doesn't need to pull from a database yet. That means you can hardcode the state, for now.

Deliverable

- Submit a zip of the React app folder you created *without* `node_modules` *included*
    - Remember: You should distribute your projects without `node_modules`.
    - **Note:** While you can use parts of this exercise in the team project, this homework assignment is to be completed and submitted individually.