Demo Day Skills ✅

All graduates of Savvy Coders should be able to implement a majority of the following objectives on their demo day projects. This may not apply in *all projects,* and this list is not exhaustive.

Ultimately, the greatest emphasis is placed on JavaScript and Single Page Application principles. This is indicated below as 'minor'/'major'.

💡 For evaluation purposes, a students project should be judged upon said criterion and meet at least 70% of these, with at least 50% coming from 'major' objectives (JS and SPA).

HTML Objectives (minor)

- Write *semantic* HTML. Although this could vary greatly from project to project, some examples of this *might* include the use of: `<section>`, `<article>`, `<aside>`, `<figure>`, and/or `<figcaption>`. At the least, we might see: `<nav>`.
- Apply principles of 'basic' accessibility and SEO principles. As a bare minimum, this should include use of `alt` *attribute* on all `<img>`s 👆 and 'matching' `<title>` and `<h1>`s as appropriate.

CSS Objectives (minor)

- Use appropriate fonts and icons. Specifically, in our core curriculum, we learned to apply Google Fonts and Font Awesome.
- Apply appropriate structure to the layout of both major and minor elements. Specifically, in our core curriculum, we learned both CSS flexbox and CSS grid.
- Build a navigation menu with 'dropdown' functionality. This includes the use of `:hover` to hide/show a 'dropdown'.
- With regards to accessibility once again, we should remember to include `:focus` whenever we use `:hover`.

JS Objectives (major)

- Use of `=>` when appropriate in *callback functions* and/or if using *function expressions*.
- Use of ` `` ` for any/all *string concatenation*.
- Interact with the *Document Object Model* from JS using `document` API.
- Respond to events using *Event Listeners*. This could include 🖱 or ⌨️ interactions from the user or data-driven events.
- *Functions*. Although not wholly applicable in all cases, *functions* should adhere to the following principles whenever possible as these approaches lend themselves to writing code that is modular and composable:
  - Use explicit `return`s.
  - Avoid mutations and/or affecting the *global scope* in any way.
  - Write terse functions that focus on performing one specific task with consistent output for given input.
- Use `fetch` or `axios` to retrieve/send data from and to a REstful JSON API.

Single Page Application (major)

💡 Students should endeavor to mimic the approach used in our core curriculum. That is, building a state-driven single page application with uni-directional data flow that follows a model-view-updater architecture.

💡 Moreover, students' projects should feature a 'data store' that serves as a 'single source of truth' for the entire application. This 'store' (model) should update itself in response to **update**s from user-driven events and/or data-driven events (`fetch` or `axios`). Finally, as `state` is updated, the application should `render` using modular components (views).