Extending the Submenu Functionality

Now we will extend the submenu functionality by fetching dynamic data and images from the server, and making submenus distinct.

WE'LL COVER THE FOLLOWING

- Rendering submenu items
- Different values for each menu

Rendering submenu items

The values that'll come from our server are as follows:

- The lists that go into each column
- The image matching the menu item

It'll take a fair bit of refactoring to go from the hardcoded version we have now to one that's populated by the server.

First, let's modify the server so that it can handle the requests we're about to make.

```
// Server

function getCategories(data) {
  if (data.category == 'top') {
    return [
      'Server apple',
      'Server banana',
      'Server pear',
      'Server orange'
    ];
  }
  if (data.category == 'additional') {
    return [
      'Server square',
      'Server circle',
      'Server oval',
      'Server diamond'
    ];
```

```
  }
  return [];
}

const endpoints = {
  "/categories": {
    "get": getCategories
  }
}

function getFunction(url, data, callback) {
  const domain = url.substring(0, url.indexOf("/"));
  const endpoint = url.substring(url.indexOf("/"), url.length);

  callback(endpoints[endpoint]["get"](data));
}

const api = {
  get: getFunction
};
```

We define a new endpoint called “categories,” and it gives a list of values based on the “category” it received in the data field.

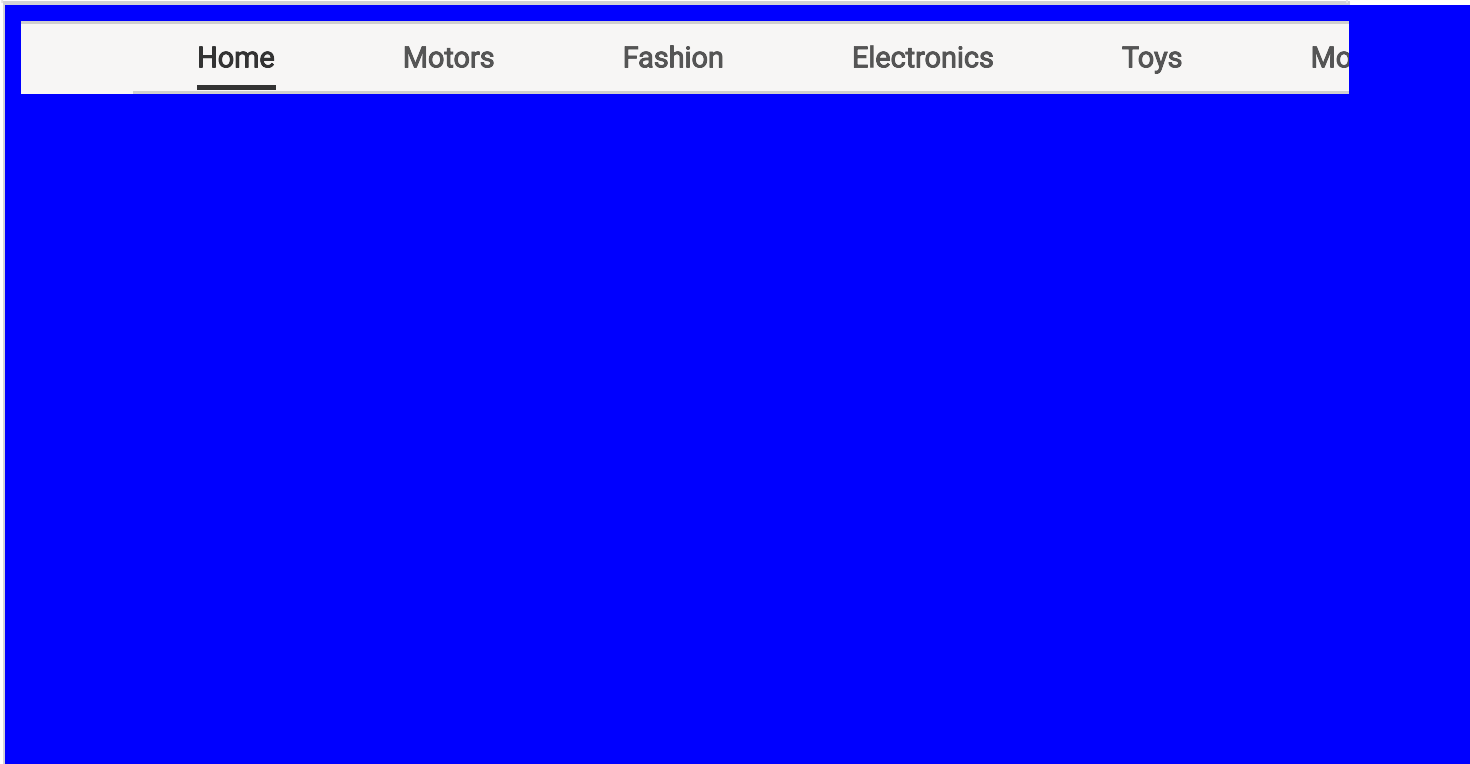
Next, let’s remove the hardcoded values and populate it on the JavaScript directly.

Output

JavaScript

HTML

CSS (SCSS)



I realized that to target the individual columns, I was missing a modifier CSS class, so I added `menu__sub__categories__items--top` to the top categories list (and the corresponding class to the additional categories list).

For the JavaScript, the part to look at is in `showSubmenu`. When we show the submenu, we'll populate the values from the server. Take a moment to make sure you understand it. The syntax with backticks is simply string interpolation, where we can define a regular string but insert dynamic values with `${dynamicValue}` syntax. We're basically just calling the API, taking the list of results, creating an `li` element with each item, and then finding the right list to put them under.

Note that in a real application, having the server fetch information at the exact moment the information is needed should be avoided when possible. It just adds wait time for the user. There are certain scenarios where it's not possible, like when you're selecting plane tickets, and the user won't know which plane routes to look up until you've filled out the fields. However, in our case, there's only a handful of menu items, and we can call this as soon as the page is loaded and cache it for when the user hovers over the item. You can tell that even eBay has a slight delay when you hover over the menu item because it's getting the images at the exact moment you need it.

Different values for each menu

The last thing we need to do before we can call this done is to vary the response based on the menu item.

Again, let's first modify the server so it can respond accordingly.

```
function getCategories(data) {
  if (data.category == 'top') {
    if (data.menuItem == 'Motors') {
      return [
        'Car',
        'Motorcycle',
        'Plane',
        'Trucks',
        'Wheels'
```

```
      ];
    }
    if (data.menuItem == 'Fashion') {
      return [
        'Women\'s tops',
        'Men\'s tops',
        'Jeans',
        'Hats'
      ];
    }
    return [
      'Server apple',
      'Server banana',
      'Server pear',
      'Server orange'
    ];
  }
  if (data.category == 'additional') {
    if (data.menuItem == 'Motors') {
      return [
        'Tires',
        'Windshields',
        'Ski racks',
        'Doors',
        'Windows'
      ];
    }
    if (data.menuItem == 'Fashion') {
      return [
        'On sale',
        'Red stuff',
        'Gucci',
        'New Arrivals'
      ];
    }
    return [
      'Server square',
      'Server circle',
      'Server oval',
      'Server diamond'
    ];
  }
  return [];
}
```

I've modified the function so that it looks at an additional data field: `menuItem`.

To make the client modifications, we just need to parse the active menu item and include it in our call.

Output

JavaScript

HTML

CSS (SCSS)

CSS (SCSS)

I need to parse out the active menu item name - which is being done on `line 15` of the Javascript file - since the variable actually holds the entire element, which looks like this.

```
<div class="menu__main__item menu__main__item--motors">
  <a class="menu__main__item__link" href="#">Motors</a>
</div>
```

Getting the first child gives me the `<a>` element, and `innerHTML` gives me `Motors`.

At this point, we have a dropdown menu that meets the requirements we set out to accomplish!

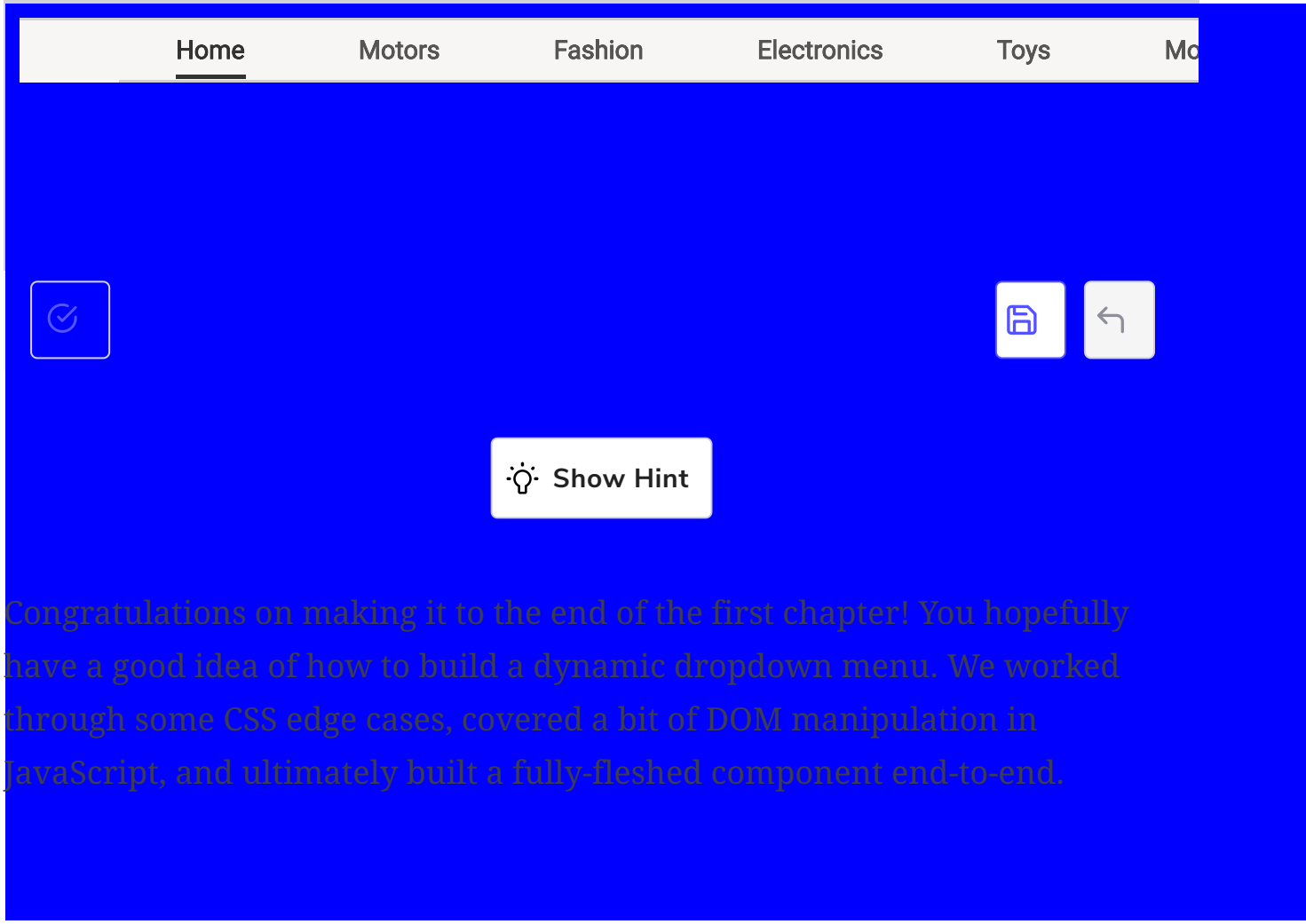
There's one bug, which is that when the submenu goes away, the active menu item still appears active. Can you think of what needs to be done?

Output

JavaScript

HTML

CSS (SCSS)



Congratulations on making it to the end of the first chapter! You hopefully have a good idea of how to build a dynamic dropdown menu. We worked through some CSS edge cases, covered a bit of DOM manipulation in JavaScript, and ultimately built a fully-fleshed component end-to-end.