Work from this link because some of the other fruit stand folders are more complicated than I need:

<https://github.com/alchemy-bootcamp-one-summer-2018/classwork/tree/master/week02_DOM-templates-components/class09_components/fruit-stand>

1. **index.html:**

a. All of my stuff will eventually get put into the root id: <div id="root"></div>

b. Make sure that the javascript script tags are listed in the same order as Marty’s.

2. **js/html.js:** Copy and paste this file exactly – it does stuff I don’t quite understand but I know that I need it ;)

3. **index.js:** Copy and paste this file exactly – it inserts all of the html stuff that I’ll build later into the <root> tag up top.

4**. fruit-api.js**: I’ll call this store-api.js.

a. Put my initial store info in.

b. ADD THE HOURLY STUFF TO EACH STORE IN HERE, TOO, so that stores has key, location, max, min, avg, and HOURLY info in it.

c. Get rid of addKey stuff, I don’t need it.

d. storeAPI needs load and add functions but I don’t need the remove part unless I eventually want to include a remove button. fruits.push(fruit) is what’s important: it adds a new fruit to the list.

5. **app.js**:

a. Load the modules in I need.

b. Marty only has a header and a <main> in this html so I’ll do the same.

c. Make a class App that finds the main tag (I’ll eventually put my table stuff into it).

d. Make a storeList and a storeForm.

e. main.appendChild(fruitList.render());main.appendChild(fruitForm.render());

are super important lines! None of the table data or form data will show up on the page until I put these in.

6. **fruitcard.js**: I’ll call this storerow.js.

a. In the template function, Marty has a list element (<li> tag) in here so I’ll put my <tr> <td> ….</td> </tr> stuff in here to represent one row in my table.

b. (I can also calculate each row total in here to stick into the above tag if I decide not to do it somewhere else.)

c. What he calls FruitCard I’ll call TableRow. It needs to know the store info and it needs a render function to put the table info into the row template form above.

7. **fruit-form.js**: I’ll call this store-form.js.

a. Marty has all his form data in here so I’ll do the same.

b. I’ll make a fruit form class that has an onAdd property.

c. The render method will put the form into the template layout above.

d. Where Marty takes in a new fruit element (let fruit = { name:…color…image…}), I’ll do the same – I just need to make sure that the names match up exactly with what are in my form (ex: name = “avg\_cookies” in the form should be elements.avg\_cookies.name).

e. I’LL ALSO GIVE THIS NEW STORE ITS HOURLY COOKIE DATA HERE.

8**. fruit-list.js**: I’ll call store-list.js.

a. Marty has a <ul> tag in here so this is where he starts his list. Therefore, this is where I’ll start my table. I’ll need <table>, <thead>, <tbody>, and <tfoot> tags here.

b. The class StoreList will need the store info and the lastStores info.

c. I’ll clear out the table info using the “this.tbody.children[i].remove()” hint that Marty gave in a previous lab.

d. this.lastFruits = this.fruits.slice() is a special line. It makes a “shallow copy” of lastFruits. What that means is that at the instant that I write this line, the lastFruits list and the this.fruits lists are exactly the same, but they can be different later (meaning, if a new fruit gets added to this.fruits then it won’t get added to lastFruits. How does this help me? In these next lines:

if(lastFruits.includes(fruit)) continue;

this.updateFruit(fruit);

Continue statements are tricky: if the the statement inside the if statement is true, then continue makes it so that NONE OF THE CODE BELOW IT EXECUTES during that iteration. Meaning, if lastFruits (my old fruit list) contains a fruit that I already have, then IT WON’T NEED TO UPDATE THAT FRUIT ROW. However, if lastFruits (my old fruit list) contains a new fruit that I DON’T have yet, then it will update that row.

e. the updateFruit method will be my updateRow method. Where Marty references this.ul (a list tag), I’ll reference this.tbody for that part of the table. This is what will add a new row if a user enters a new store.

f. I can get rid of references to updateCount and spanCount because I don’t need them.

g. render is what prints the ORIGINAL store data to the screen when I first load the page.

9. **footer.js**: Will contain a template for what I want the footer to look like and a class Footer that has a render method. I can create an updateFooter method in my store-list.js file.