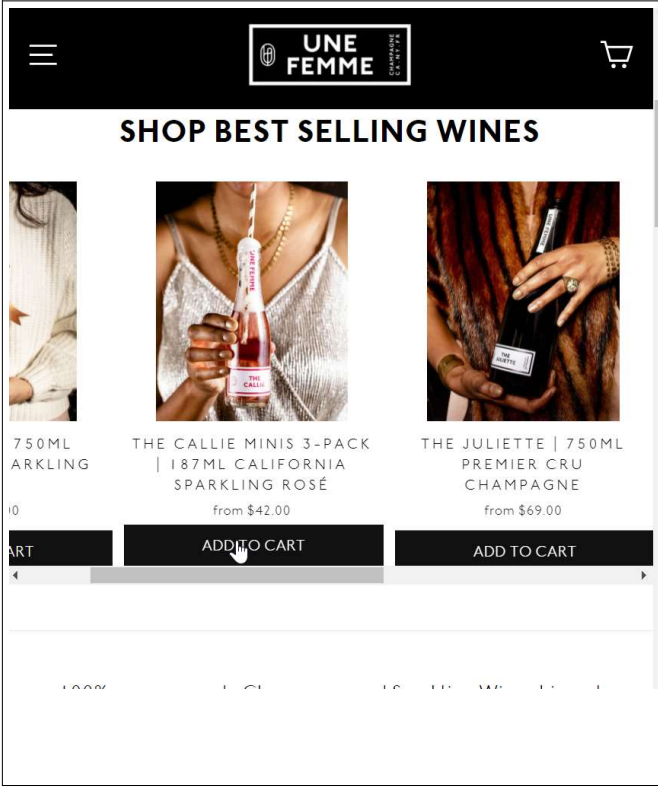
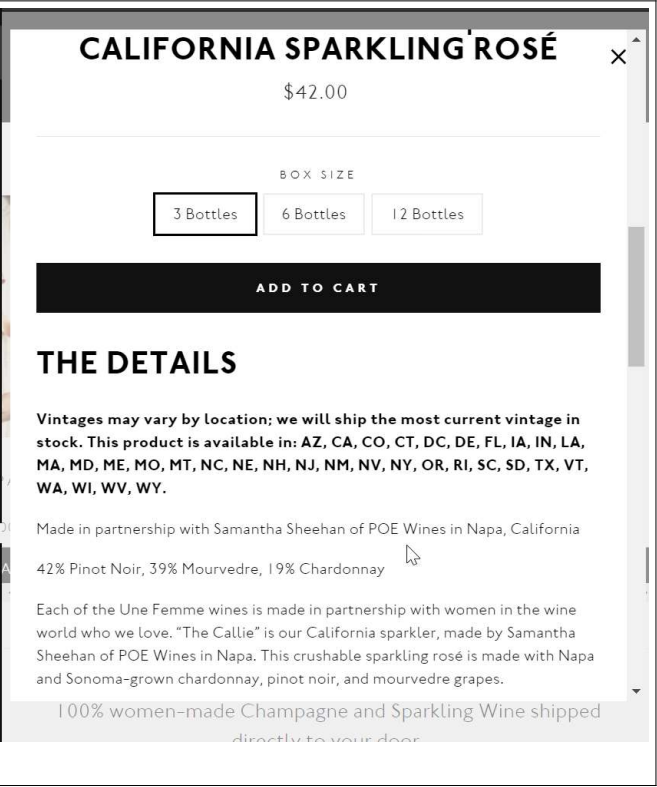


Bugfix goal: Opening quickbuy on mobile -David Weru

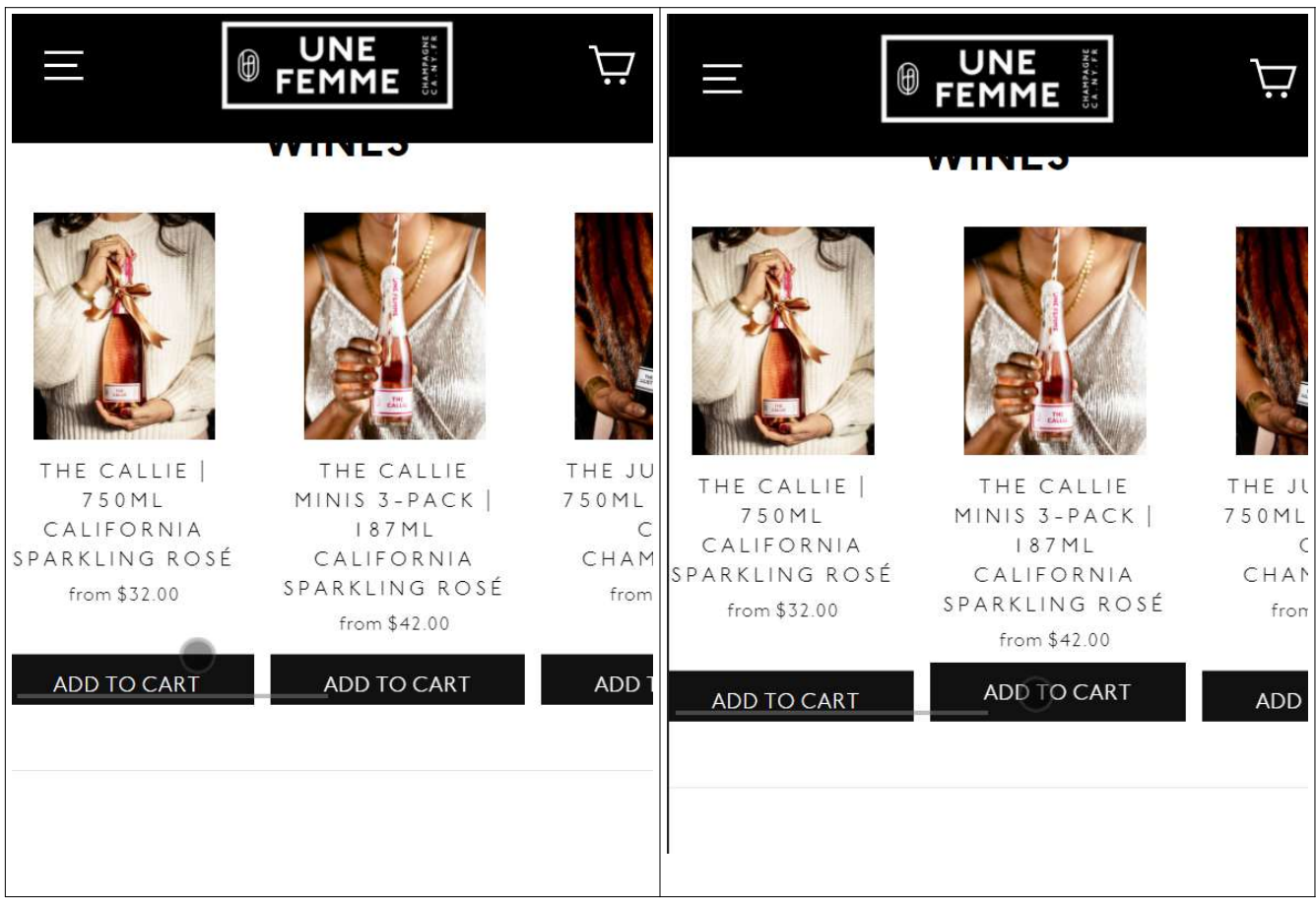
The problem here on the surface was rather simple.

On desktop, clicking the buy button, opened the quickbuy box

(click)	(result open)
	

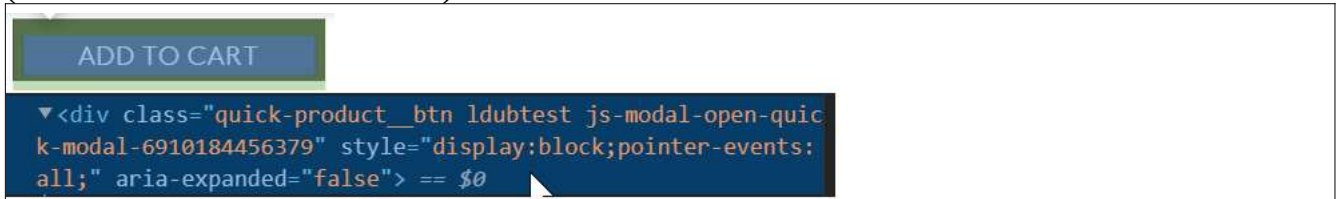
On mobile, the click did nothing.

(click)	(click again. Nothing)
---------	------------------------



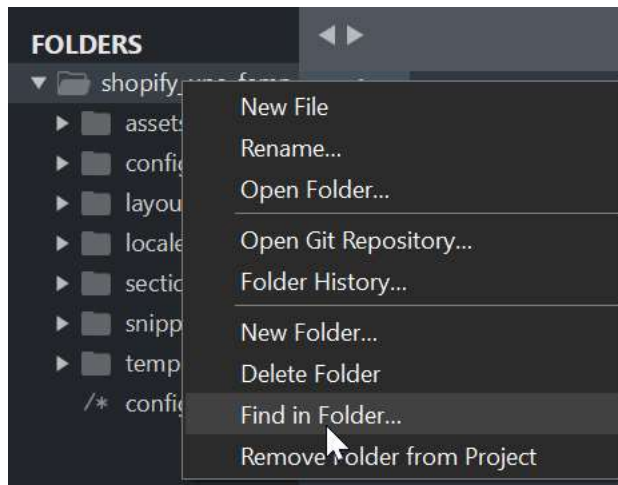
So we first look at the element, and sure enough, the elements do not have their events coded on them directly.

(element doesn't show onclick="")



The next step is to find if there is a function calling the object by it's classname or element id.

```
<div id="_par_une-femme-the-juliette" class="quick-product  
btn ldubtest quick-product_btn--not-ready js-modal-open-  
quick-modal-6910184456379 " style="display:block;pointer-ev
```



```
Searching 184 files for "js-modal-open-quick-modal-"  
D:\projects\shopify_une_femme1\snippets\product-grid-item.liquid: |  
193 </a>
```

Nothing. -Above is the code that renders the buttons. It's just the machinery that builds the button that we click.

But nothing here opens the popup.

It's going to be a very long day. Let me explain something first about the events.

There are many ways to structure a program.

Javascript and html and other server languages are programs.

The programs written for this environment tend to only do something when someone does something.

That is called "Event driven" programming.

Events are things like, scrolling, clicking, and typing. Input events.

Event driven programs can be written without a clear structure starting out.

Let me explain.

You're an electrician. A tenant wants a new switch for a new light bulb.

What do you do? Add lightbulb. Then add switch. Connect to power. Done.

Did you have to know how the rest of the house is wired? no.

That is the beauty of event driven programming.

There's another tool loved by event driven programmers.

Object oriented programming.

Javascript doesn't explicitly give you things like classes,

(A class is the house's floorplan.

A class "instance" is the particular house the electrician was working on.

The lightbulb and switch is an object, and the event of turning on the light is the method)

Object oriented programming lets people write code without knowing the rest of the code.

Like that electrician.

Event driven programming is like making a car and a way to drive it, and leaving the driver to decide where they want to go.

In a crude way, it's like walking into a garage with all the tools just laying there if you need them.

Why did I talk about classes? Because with all these events, things can get pretty chaotic. So you need to make a controller. Think of this like where you sit when you drive a car, or play a game.

All your controls are in one place and have been layered out so you cannot give conflicting instructions.

Below is a snippet of the file which handles these events for our website:

Searching 184 files for "theme.js"

D:\projects\shopify_une_femme1\assets\theme.js:

```
2 @license
3   Impulse by Archetype Themes (https://archetypethemes.co)
4:   Access unminified JS in assets/theme.js
5
6   Use this event listener to run your own JS outside of this file.
.
12 */
13
14: console.log("using unminified theme.js");
15
16 window.theme = window.theme || {};
```

D:\projects\shopify_une_femme1\assets\theme.min.js:

```
2 @license
3   Impulse by Archetype Themes (https://archetypethemes.co)
4:   Access unminified JS in assets/theme.js
5
6   Use this event listener to run your own JS outside of this file.
```

D:\projects\shopify_une_femme1\layout\theme.liquid:

```
83
84   {% comment %} <script src="{ 'theme.min.js' | asset_url }" defer="defer"></script> {%
endcomment %}
85:   <script src="{ 'theme.js' | asset_url }" defer="defer"></script>
86
87   {%- if request.page_type contains 'customers/' -%}
```

4 matches across 3 files

(Example: cannot turn a light on and off at the same time)

Now. This control area is where things get complicated.

Imagine that light switch the electrician installed was extremely modern. So modern that after it was installed, it was added to your home's wifi and you could use an app to turn on and off that light.

Say you had all your items and appliances accessible on this app.

Forgot to turn off the stove? Turn off the stove. Not sure what else you left on? Turn off power to the kitchen. Want to make it colder, flip a switch on your phone.

This app checks your instructions before running them. It runs them through a filter.

Light bulb only goes to 30? only goes to 30 even if your app can set it to 50. Your buddy can't turn on the ac? Well, no ac for you.

Now you see, that last example is what is called an event handler.

You turn on a light, but the event handler/manager ultimately decides if that light goes on regardless of user permissions.

Now that we have the preliminaries out of the way, we can describe our mobile button problem, and why we'll be here all night.

You see, programming is a weird thing. You can give someone all the controls to your smart house without having the electrician first installing the lightbulb.

Yes. You turn on a lightbulb.

A loading animation plays.

A lightbulb gets ordered from your online store.

Then gets installed.

Then the loading screen goes away.

Then the lightbulb goes to the on setting.

Now this is why we will be here all night.

There is a process to create that lightbulb that is different for each house that wants it.

The mobile house MIGHT NOT be compatible with that lightbulb, but the switch is there.

You will press that button. The lightbulb will do nothing.
(Or in this case, the quickbuy popup for our store.)

Fixing this will be easy IF the machinery to install that lightbulb is already there.

If not, we'll be there all night (I won't have to explain why in this case because the machinery was there)

So our first step was to find if someone just had an exception for mobile.

They did (pic) (-que sarcasm:-no really! How bizarre..)

```
function productMouseover(evt) {  
  var el = evt.currentTarget;  
  // No quick view on mobile breakpoint  
  
  if (true){// (!theme.config.bpSmall) {  
    }, 100);  
  }
```

That was commented out, and I just set that condition to true on default. Nothing broke.

Tested it and the popup still did nothing. That exception stayed removed.

That filtered function loads information onto a hidden div when you hover your mouse over it.

If you are on mobile, you don't have a mouse to hover over elements of a page. When the information is not loaded, it cannot be displayed when you click on the button.

Okay, second step.

Force run the next pieces of the machinery directly. Below is the function that is called when a mouse hovers on a button. (Bypass the event handler and just turn the switches on ourselves.)

It's the "theme.preloadProductModal" function. It needs 3 things to run.

"productId" "handle" and "btn"

-basically, the product id, the product name (it's given in the button), and the button element.

To run it, all you have to do is call it with an "onclick" event. It will run, because this function is defined before the page already loads. It'll answer because it's already listening in the mouseover event function. So we'll just call it directly because we can.

```
var productId = el.dataset.productId;
var handle = el.dataset.productHandle;
var btn = el.querySelector('.quick-product__btn');
theme.preloadProductModal(handle, productId, btn);

// alert(productId+'\n'+handle+'\n'+btn);
}
};

theme.preloadProductModal = function(handle, productId, btn) {
```

Below is the button element loaded with these direct-to-function switches. -starting at "theme.preloadProductModal("handle","productid","btn");" below. We'll talk about "to_the_point" later below.

```
-product__label" onclick= to_the_point("_par_une-femme-
the-callie-750ml","6910187995323");theme.preloadProductM
odal("une-femme-the-callie-750ml", "6910187995323", docu
ment.getElementById("_par_une-femme-the-callie-750ml"));
document.getElementsByClassName("js-modal-open-quick-mod
al-6910187995323")[0];">Add to Cart</span> == $0
```


A glimmer of hope. It now takes: 2 clicks. Fantastic.

Now we're sure that the lightbulb has to be ordered before it can be used.

What if I just click twice in the code? Or 3 times? Or until it opens?

Well, you can, if the lightbulb is there. But first, let's order it.

I'll now introduce you to asynchronous programming. (We're about to talk about "to_the_point")

It works like this: You throw a ball, and I throw it back. You can't throw a ball unless you have one, and you can't catch one unless it's thrown at you.

When someone says AJAX, this is pretty much the play. You send for stuff, wait, then do something when you get it.

You can't keep throwing the ball until you get it back. You have to do the "throwing event", then the "waiting event", then the "catching event".

If you throw repeatedly, you will be stuck at the "throwing event" stage, and only one "throwing event" will be sent -the last one. Also, your program will be slow.

So this is how we click twice from the code. (to_the_point)

```
function to_the_point(id,pid){    I
//you clicked, preloadProductModal(a,b,c) has been called. And a reply is on the way.
//this is the one second waiting room... -we'll click again once for good measure.
    document.getElementById(id).click(); //you click

    setTimeout(function(){ //wait one second (timeout function)
        //this lets the function from preloadProductModal(a,b,c) run and get an AJAX reply

        //check to see if modal has arrived/replied
        var a=document.getElementById("QuickShopModal-"+pid).getAttribute("class");
        a.includes("modal--is-active")==true?"do_nothing":document.getElementById(id).click(id);

        //(modal--is-active for your product id) -it's here, don't click.

        //(modal--is-active for another product id)click again -yours has not arrived
    }, 1000);
}
```

(function screenshot)

We click, then wait (there's an event for that)

Then look at what arrived. (See if the arriving package is ours)

If not, we click through the code.

If it is, we're done clicking.

It's midnight now, so go to bed. (I'm kidding. Don't miss your bedtime. Please.)

If there was no machinery for mobile, this is what we would have to do.

We would simply request our lightbulb, make adjustments to it, then install it for mobile and hope there was a good reason why this wasn't built in. Maybe the user interface was too small for mobile.

Either way, this is where our lesson ends.

When the user taps the button on mobile, the popup appears as it should.

Lessons here are: Event driven programming is great for the programmer, but fixing bugs in a unified event handler can get interesting.

For one, the code in the event handler is like a road. It doesn't care what size car passes it, but you'll know that a car will pass here. It's ambiguous code that is meant to handle different kinds of things so long as they have a compatible interface. If car, wheels and steering wheel. If lightbulb, lightbulb and switch.

If popup -you get the idea.