Event Delegation in JavaScript

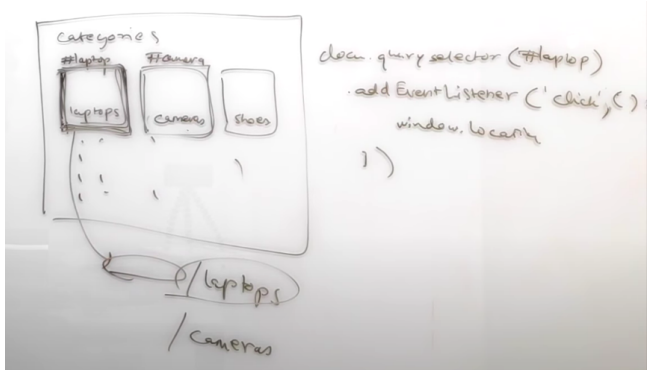
Friday, 25 August 2023 7:27 PM

→ Event delegation is the technique of handling events in a better way. This is only possible because of the way how event propagates up the DOM tree i.e because of event bubbling.

→ So event delegation is based on event bubbling - So, event delegation exists because of event bubbling.

→ As our code grows in size, there will be a lot of event listeners attached to our code & this may cause a performance bottleneck - So, event delegation is a great way of handling these hanging event listeners in our program.

→ Let us assume an e-commerce store:



→ Inside the website, we have different categories, when we click on that category, it takes us to that page.

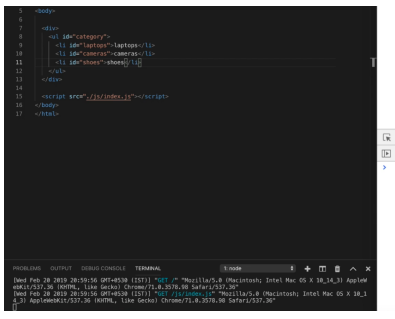
→ Now, each category has a click event attached to it whose callback fn has window.location.href which transfers us to that page.

→ But what if we have, an infinite scroll or lazy loading? This will result in too many event listeners hanging around & this can lead to much performance issues.

→ So, in order to handle this we use event delegation which states that instead of attaching event listener to each & every child elements individually we should rather attach event listener to the parent of these elements.

→ So, basically the events happening in child elements will bubble out their parent element & the event listener attached to parent will only run its callback fn.

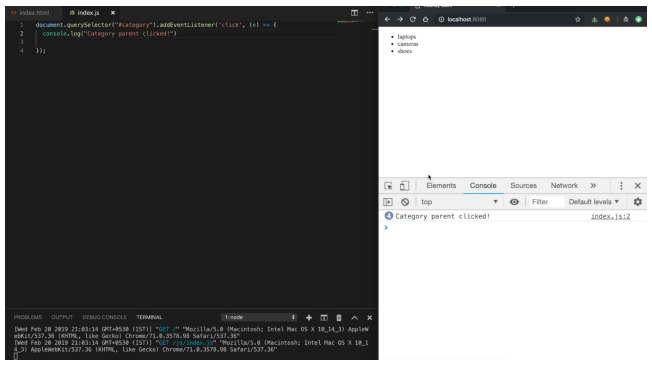
→ Code demonstration of event delegation:



→ HTML file

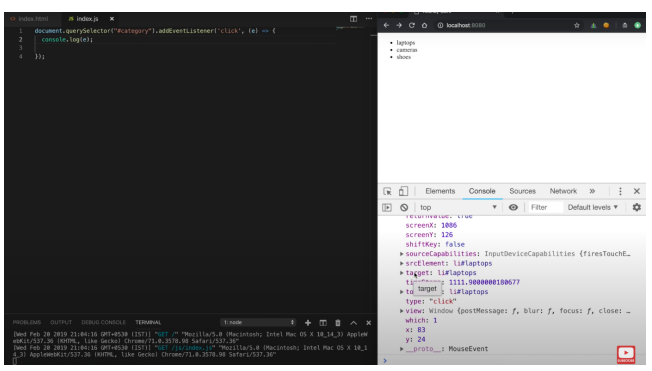
→ Now we need to redirect page when clicking individual 'li'. This could have been achieved by attaching an event listener to each individual 'li', but if 'li' would have been too many, then it would have caused performance issues due to many hanging event Listeners.

→ Therefore, we'll use event delegation (which works on the concept of event bubbling) and will just attach 1 event listener to the parent element.

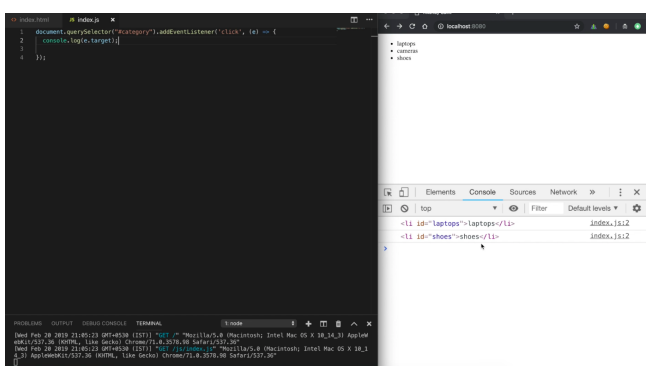


→ So, when we click on the children elements, because of event bubbling, the event bubbles up to its parent & 'category parent clicked' is consoled.

→ In order to detect which element was clicked, we use 'event' object (here, 'e')



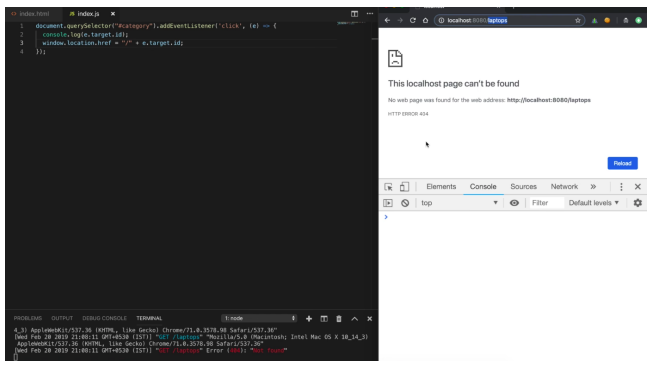
↳ o/p on clicking laptops.



↳ o/p on clicking laptops & then shoes.

→ We can access the ids using (e.target.id)

→ So, in order to navigate to the particular page, we can do:



→ We've achieved this just through 1 event listener.

→ So, we've achieved event delegation through the concept of event bubbling.

→ We can improve the above code by adding a condition, so that the

Page redirects only when we click on 'li' element.

```
index.html   index.js
1  document.querySelector("#category").addEventListener('click', (e) => {
2    console.log(e.target);
3    if (e.target.tagName == 'LI') {
4      window.location.href = "/" + e.target.id;
5    }
6  });
```

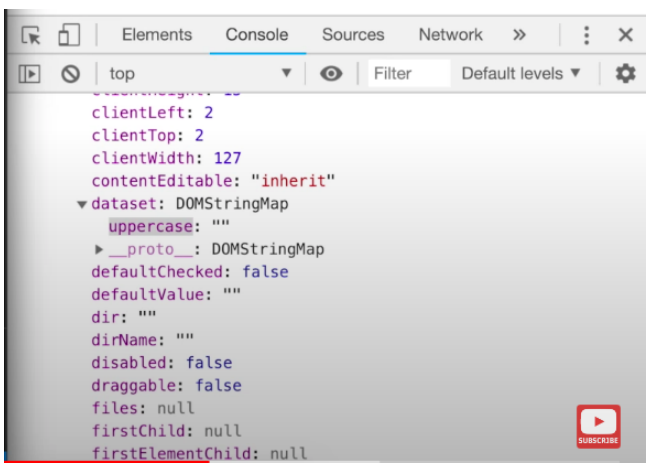
→ Another example:

```
index.html   index.js
1   <!DOCTYPE html>
2   <head>
3     <title>Akshay Saini</title>
4   </head>
5   <body>
6
7     <div id="form">
8       <input type="text" id="name" data-uppercase>
9       <input type="text" id="pan">
10      <input type="text" id="mobile">
11    </div>
12
13    <script src="./js/index.js"></script>
14  </body>
15  </html>
```

→ HTML code.

→ Here we need to change text inside input-box to uppercase in whichever input tag we get dataset uppercase.

→ In order to access data-uppercase, we can do so by (e.target.dataset)



→ So now we can check that if uppercase is not undefined then we make value of text to uppercase.

→ So we've made this functionality once but we can reuse this functionality to all input boxes.

→ This is known as behaviour pattern which can be achieved using event delegation.

→ Benefits of event delegation:

1) It uses a lot less memory & thus mitigates the case of performance bottleneck.
2) Writing less code
3) DOM manipulation, i.e now we've attached just 1 event listener to the parent otherwise it would've been so difficult to attach an event listener to all child elements in case of infinite scrolling.

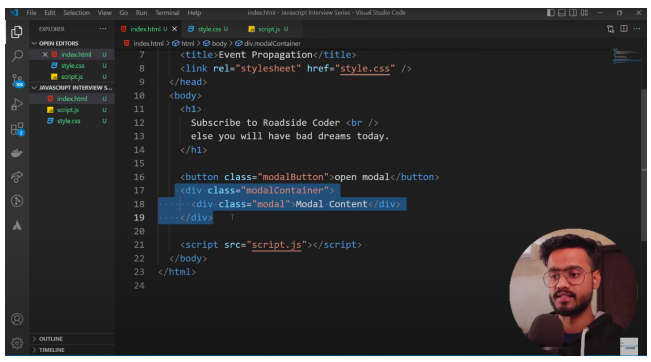
→ Limitations of event delegation:

1) All events are not bubbled up. eg: blur(), focus(), resizing of window, so on.

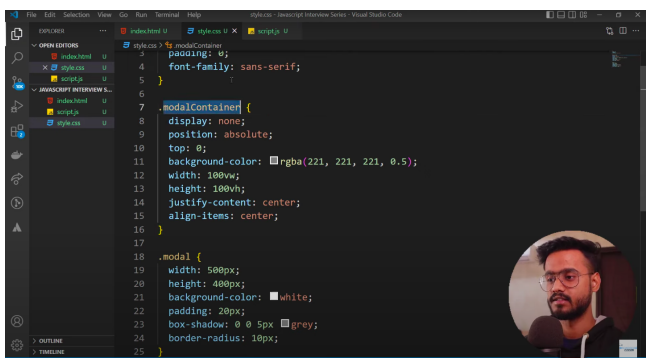
2) We cannot use stopPropagation() in order to stop event propagation at a particular child element because for event delegation to work we cannot stop event bubbling.

Q: Create a modal which opens on clicking a button and closes only when click on a space outside the modal (negative space):

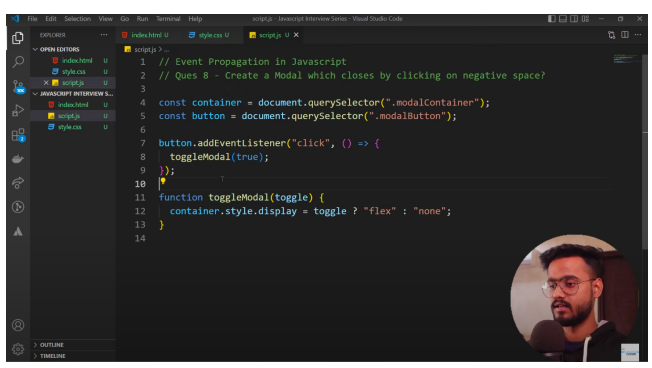
ans.



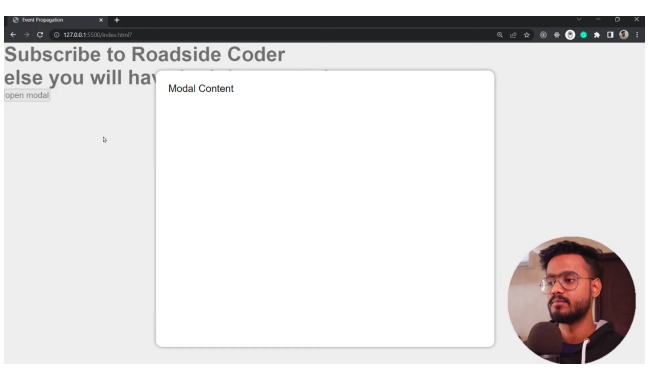
→ HTML file having a button & a modal inside a modalContainer



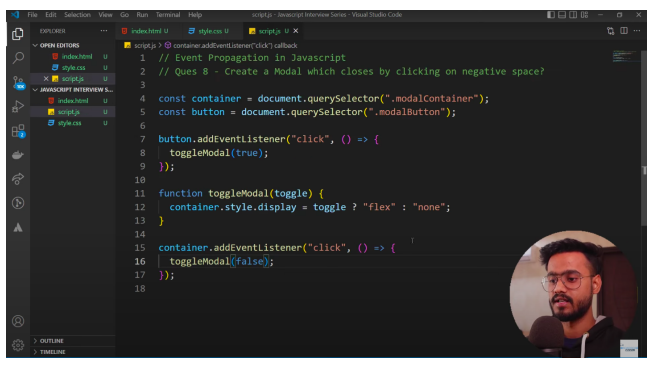
→ some styles. Note that modalContainer would be hidden initially.



→ We toggle the modal to true when button is clicked & inside toggleModal() fn we set 'modalContainer' class's 'display' style to flex. otherwise modal will be hidden if 'toggle' is false



→ output



→ Now we add eventListener to 'modalContainer' class & toggle the Modal to false.

→ But note that, if we do this then modal will close whenever we click anywhere once the modal is opened.

→ So, we close the modal only when class name is modalContainer (when we click on modal, we get class name as 'modal', when we click outside we get class name as 'modalContainer',

→ So, modal gets closed when we click outside the modal.

Note: If something is wrapped inside 2 HTML tags eg:
  Hello , then to access tag we can use, 'event.target.closest'.