Learning Experience Design Table: Event Bubbling

A) Expected Learning Outcome	B) Activity or Practical usecase	C) Demo idea or Code snippet (with new learnings mentioned)

User understands the concept of event bubbling and how events propagate in the DOM.	Event bubbling is a mechanism in the DOM where an event triggered on the innermost (target) element propagates up through its ancestors to the outermost (root) element.	`body.addEventListener("click", ()=> {`
		`    console.log("Parent is clicked")`
		`})`
		`div.addEventListener("click", ()=> {`
		`    console.log("div is clicked")`
		`})`
		`form.addEventListener("click", ()=> {`
		`    console.log("form is clicked ")`
		`})`
		`h2.addEventListener("click", ()=> {`
		`    console.log("h2 is clicked")`
		`})`
		`label.addEventListener("click", ()=> {`
		`    console.log("label is clicked")`
		`})`
		`label.addEventListener("click", ()=> {`
		`    console.log("label is clicked")`
		`})`
		`gender.addEventListener("click", ()=> {`
		`    console.log("gender is clicked")`
		`})`

User should be able to identify problems where event bubbling can be utilised for better solutions	Event Bubbling can be used for like
	1. Form Validation
	2. Delegated Event Handling
	3. Password Visibility Toggle
	4. Handling Form Submission with Event Capturing
		`//Delegated Event Handling:`
		`form.addEventListener("click", ()=> {`
		`    console.log("form is clicked ")`
		`})`
		
		`form.addEventListener('click', (e) => {`
		`    const target = e.target;`
		
		`    // Check if the clicked element is a password toggle button`
		`    if (target.id === 'pass-toggle-btn') {`
		`        // Handle password toggle logic here`
		`        target.className = passwordInput.type === 'password' ? 'fa-solid fa-eye-slash' : 'fa-solid fa-eye';`
		`        passwordInput.type = passwordInput.type === 'password' ? 'text' : 'password';`
		`    }`

		`});`
User should be able to prevent event bubbling using event.stopPropagation()	Here, when the password visibility toggle button (passToggleBtn) is clicked, the event.stopPropagation() is used to prevent the click event from continuing to propagate up the DOM hierarchy. This is important to avoid unintended consequences, such as triggering the form submission.	`passToggleBtn.addEventListener('click', (event) => {` `passToggleBtn.className = passwordInput.type === "password" ? "fa-solid fa-eye-slash" : "fa-solid fa-eye";` `passwordInput.type = passwordInput.type === "password" ? "text" : "password";` `event.stopPropagation();// Event stopPropagation for the password input to prevent form submission` `});`

User understands the difference between event bubbling and event capturing	The concepts of event bubbling and event capturing are associated with the order in which events are handled as they propagate through the DOM hierarchy.
	
	The **event bubbling** order is demonstrated through the event listeners attached to various elements, such as the body, div, form, h2, label, and gender.
	
	Event capturing is the opposite of bubbling. It involves the event traveling down from the root of the DOM hierarchy to the target element.
	In the code, **event capturing** is demonstrated when handling the **form submission event.**
		`//Event bubbling` `body.addEventListener("click", ()=> {` `console.log("Parent is clicked")` `})` `div.addEventListener("click", ()=> {` `console.log("div is clicked")` `})` `form.addEventListener("click", ()=> {` `console.log("form is clicked ")` `})` `h2.addEventListener("click", ()=> {` `console.log("h2 is clicked")` `})` `label.addEventListener("click", ()=> {` `console.log("label is clicked")` `})` `label.addEventListener("click", ()=> {` `console.log("label is clicked")` `})` `gender.addEventListener("click", ()=> {` `console.log("gender is clicked")` `})`
		
		`// Handling form submission event` `//Event Caputring` `form.addEventListener("submit", handleFormData, true);`

User should be able to implement event delegation by leveraging	Event delegation is a powerful technique that involves placing a single event listener	`// Event delegation for password toggle buttons` `form.addEventListener('click', (e) => {` `const target = e.target;`

event bubbling	on a common ancestor of multiple elements and then using event bubbling to handle events for those elements. This approach is more efficient than attaching individual event listeners to each element	`    // Check if the clicked element is a password toggle button`
		`    if (target.id === 'pass-toggle-btn') {`
		`        // Handle password toggle logic here`
		`        target.className = passwordInput.type === 'password' ? 'fa-solid fa-eye-slash' : 'fa-solid fa-eye';`
		`        passwordInput.type = passwordInput.type === 'password' ? 'text' : 'password';`
		`    }`
		`});`