JavaScript Part 6: The DOM and Events

A web page is not just a picture: the browser keeps it as a structure that JavaScript can read and change while the visitor watches. That structure is called the Document Object Model, or DOM, and the actions of a visitor are called events. Four steps connect code to the page: select, change, listen and build. Each step answers one simple question, and the browser console can run every line on the page itself. In this part, Fadi, a web developer at a network of medical clinics, uses them to make the appointment slots of a booking page respond to clicks.

1. Select

The browser turns the HTML of a page into a tree of elements, and the document object is the entry point to that tree. The querySelector method finds the first element that matches a selector, which is the same pattern language that CSS uses, such as a class name or an id. The querySelectorAll method returns all the matching elements. The question is: which element on the page do we want to work with? If nothing matches, querySelector gives null, and using that result later causes an error, so it is worth checking. The code must run after the elements exist, which is why scripts are usually placed at the end of the page or loaded with the defer attribute.

Example: A stage manager holds a cast list with many actors and needs to call one of them to the stage. The manager does not read the whole list aloud but gives one description: the person in the red coat. Exactly one actor from the crowd steps forward, and if nobody wears a red coat, nobody comes, which is like null. A selector works in the same way: it describes an element by its class name or id, and the page hands back the matching element.

At work: Fadi's booking page has a list of appointment slots and an area for messages. He asks the Select question, selecting the message area by its id and all the slot buttons by their shared class name. He checks in the console that the selection is not null before writing more code. The later steps can now rely on these two selections.

2. Change

Once we hold an element, we can change what the visitor sees. The textContent property reads or sets the text of an element, and it treats everything as plain text. The classList property adds, removes or toggles class names, which is how code switches styles prepared in CSS on and off. The setAttribute method changes attributes such as disabled. The question is: what should look or read differently, and is a class name enough? The innerHTML property can insert HTML, but if it receives text typed by a visitor, a malicious visitor could inject harmful code, so textContent is the safer default. Keeping the look in CSS and only switching class names keeps the code short.

Example: A hotel receptionist flips the sign of a room from available to occupied when a new guest arrives. The receptionist does not rebuild the door or the room. The door stays exactly the same, and only its status changes, so people see something different. The classList property does that for an element: adding or removing one class name flips its look and leaves the element itself in place.

At work: When a patient clicks a slot, Fadi changes the page. He asks the Change question, adds a class named selected to the chosen slot so that it is highlighted, and sets the text of the message area to confirm the time. Because he uses textContent, a name typed by a patient into a field can never be run as code. The highlight style stays in CSS, and the script only switches it on.

This document is a property of Asaseeyat Academy - © Copy Right Protected.

3. Listen

An event is something that happens on the page, such as a click, a key press or a change in a field. The addEventListener method attaches a function, called a handler, to an element and an event type. The handler runs every time that event happens, and it receives an event object that tells us more, for example which element was clicked. The question is: what should happen when this event occurs? For example, `button.addEventListener("click", showSlots)` runs the function named showSlots on every click. We pass the function itself without parentheses, because adding parentheses would run it immediately instead of waiting for the click. One listener on a parent element can also handle the clicks on all its children, even children added later, which is called event delegation.

Example: A restaurant kitchen has a bell, and every time a customer rings it, the staff respond in the same way. The staff do not watch the customer all the time, and they do not act before the bell. An event listener waits in the same way and reacts only when the event happens.

At work: Fadi attaches one click listener to the list of slots instead of one to every button. He asks the Listen question and uses the event object to find out which slot was clicked. The handler highlights that slot and removes the highlight from the previous one. Because the listener sits on the list, slots that are added later work without extra code.

4. Build

Often the elements do not exist in the page at the start, because they come from data. The createElement method makes a new element in memory, and we set its text and class names in the same way as for existing elements. Then the append method places it inside a parent element, and only then does the visitor see it. The question is: which data should become elements, and where should they go? A loop over an array of objects, as in the previous part, creates one element for each record. Building many elements one by one is fine for a booking page, and the code stays readable.

Example: A florist receives a list of orders on Monday morning, and no bouquet exists yet. For each order, the florist prepares one bouquet card with the customer's name and the date, and puts it on the right shelf. The cards do not exist before the list arrives, and a longer list simply means more cards. Creating and placing elements works in the same way: the createElement method makes each new card, and the append method puts it on its shelf.

At work: Fadi receives the free slots of the day from the clinic system as an array of objects. He asks the Build question, loops over the array and creates a button for each slot with its time as the text. Each button gets the shared class name and is appended to the slot list. When the clinic adds a new slot, the page shows it without any change to the code.

Conclusion

Fadi's booking page now responds to what the patient does, and its slots come from data. Select asks which element we want, and change asks what should look or read differently. Listen asks what should happen when the event occurs, and build asks which data should become elements and where they go. In the next part, we use the same ideas to handle forms, to validate what visitors type and to remember it with local storage.

Sources

- MDN Web Docs: Introduction to the DOM – https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
- MDN Web Docs: Document.querySelector() – https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector
- MDN Web Docs: EventTarget.addEventListener() – https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener
- MDN Web Docs: Element.classList – https://developer.mozilla.org/en-US/docs/Web/API/Element/classList
- MDN Web Docs: Node.textContent – https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent
- MDN Web Docs: Element.append() – https://developer.mozilla.org/en-US/docs/Web/API/Element/append

This document is a property of Asaseeyat Academy - © Copy Right Protected.