47)Setting up a web server

We will install a software which will serve our folder as wensite.it means we can access all of those files in browser. Run this command-

**npm install -g live-server**

create a folder called note-app. In this folder create a new file called index.html. now go to path where we have notes-app folder. Now run this command-

**live-server notes-app**

index.html will open in browser. In url bar you will see url like this-

<http://127.0.0.1:8080/>

this works locally in your machine only. Other people cannot access it. Now when ever we change index,html and save it. We see new content without refreshing page.

49)DOM Manipulation

DOM stands for document object model. Document in DOM is html document, object is js object. So we have object in js that models our html document. That’s what dom is. Lets see how we can do DOM manupiation using that js object. This js object is called document. This is object provided to us by browser. So lets remove paragraph tag. Code-

Html-

<!DOCTYPE html>

<html>

<head>

<script src= "notes-app.js"></script>

</head>

<body>

<h1>Notes App</h1>

<h2>Take notes and never forget</h2>

<p>This application is created by Andrew Mead</p>

</body>

</html>

Js-

const p = document.querySelector('p');

p.remove();

but we will get error. This is because our js code executed before our html was rendered. So it cannot find p tag, because there is no tag at that time. As soon as we import the js file our code is executed.so we import our js file after html is rendered.

Html-

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h1>Notes Apps</h1>

<h2>Take notes and never forget</h2>

<p>This application is created by Andrew Mead</p>

<script src="notes-app.js"></script>

</body>

</html>

By using querySelector we get only first element that matches the result. If we want to delete all p tags then we use querySelectorAll , it gives us lists of all matching nodes. Code-

ps.forEach((p) => {

p.remove();

});

We can aslo read and change text of our tags. Code-

const ps = document.querySelectorAll('p');

ps.forEach((p) => {

console.log(p.textContent);

p.textContent = 'sumeet';

});

51)Adding new elements via DOM

const ps = document.querySelectorAll('p');

ps.forEach((p) => {

console.log(p.textContent);

p.textContent = 'sumeet';

});

const newParagraph = document.createElement('p');

newParagraph.textContent = 'This is new paragraph from javascript';

document.querySelector('body').appendChild(newParagraph);

here first we created a new element, then gave it some text content. Now we have to give it a place , where we render it. So we append it to body tag. Now body has existing childs. So our new element will be added in the end.

Note that in our code first we are changing the text of all paragraphs to sumit, however our newly created paragraph is not created at time when we change text. So it wnt be affected by this code.

52)Handling User Interaction

This is how we add event to a button-

document.querySelector('button').addEventListener('click', () => {

console.log('Button Clicked');

})

Go to documentation. Google – addEventLister Mozilla. On left hand side you can see all events. Arguments in square brackets means that they are optional.

Whenever a event happens a argument is passed to our listener function. this argument represents the event. If we print it on console, there are lot of properties.we will focus on important ones. We have property called target. Target is actually representation of that DOM element. So like we get element back from querySelector we get the elment on which event was fired from target property. So we can access element on which event was fired and we can do things to it. Like –

document.querySelector('button').addEventListener('click', (e) => {

e.target.textContent = 'Button is clicked';

})

53)Advanced Queries

Lets say we have 10 buttons in our page and we want to add Evemt listener for one. We do this bt selecting that button by id. Html-

<button id="remove-all">Remove All Note</button>

Js-

document.querySelector('#remove-all').addEventListener('click', (e) => {

console.log('Delete all');

})

Note that, id should be unique. That means there can be only one element with a given id.

We can also use class. But multiple elements can have same class. This is how we select all elements with given class. Code-

document.querySelector('#remove-all').addEventListener('click', (e) => {

document.querySelectorAll('.note').forEach((note) => {

note.remove();

});

})

Note that **querySelectorAll** returns array of elements, but lets say we want to select only first element then use this-

document.querySelector('button')[1].addEventListener('click', (e) => {

console.log(‘Button clicked);

})

As there can be multiple elements we use, querySelectorAll. We can also use combinations of queryselectors like this-

**p#order**

this will target paragraph tag which has id of order.

**button.inventory**

this will target all buttons with class of inventory.

**h1#title.application**

this will target h1 that have id of title and class application.

**h1.application#title**

same as above. So order does not matter.only thing that matter’s is that your tag comes first when we are setting up multiple different things in our selector otherwise it would look pretty weird.

54)Text Inputs and live data filtering

Html-

<input id="search-text" type="text">

Js-

document.querySelector('#search-text').addEventListener('change', (e) => {

console.log(e);

});

This event is fired when I click in text input and then I focus out. We printed event object on console. We can see various property of this object, like type,target etc. if we want to get text typed in input then use value property on event object. **change** event is not going to work for us, we want to filter notes as we type in text box. So we use **input** event, it is fired when we type keys in text box.

55)Rendering our filtered data

Here our notes will be filtered as we type in input field. Html-

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h1>Notes Apps</h1>

<h2>Take notes and never forget</h2>

<input id="search-text" type="text" placeholder="Filter todos">

<div id="notes" ></div>

<button id="create-note">Create Note</button>

<button id="remove-all">Remove All Note</button>

<script src="notes-app.js"></script>

</body>

</html>

Js-

const notes = [{

title: 'My next trip',

body: 'I would like to go to spain'

}, {

title: 'Habbits to work on',

body: 'Exercise. Eating a bot better'

}, {

title: 'Office Modification',

body: 'Get a new seat'

}];

const filters = {

searchtext: ''

};

const renderNotes = (notes, filters) => {

const filteredNotes = notes.filter((note) => {

return note.title.toLowerCase().includes(filters.searchtext.toLowerCase())

});

document.querySelector('#notes').innerHTML = '';

filteredNotes.forEach((note) => {

const noteEl = document.createElement('p');

noteEl.textContent = note.title;

document.querySelector('#notes').appendChild(noteEl);

});

};

renderNotes(notes, filters);

document.querySelector('#search-text').addEventListener('input', (e) => {

filters.searchtext = e.target.value;

renderNotes(notes, filters);

});

Previously we were adding new element to end of body by using **appendChild** method. But here we want new element to be added before buttons. So we added div tag there. And now we are appending elements to div tag. Also we want to remove previously rendered element before adding new ones. Now we cannot remove all elements of body. So we removed all elemnts of div –

document.querySelector('#notes').innerHTML = '';

57)working with forms

Html-

<form>

<input type="text" placeholder="first name" name="firstName">

<button>Submit</button>

</form>

We do not have js for it. When we type some name and click on button. Our page reloads. All data is gone. Our url cnaged to-

<http://127.0.0.1:8080/?firstName=sumeet>

from this-

<http://127.0.0.1:8080/>

our form data appears in url. this is default behaviour. This was used in past when js support in browsers was not so great. So this method was used to get our data to server. But today we have great javascript support. So we want to handle form submission via client side javascript code. So we want to override default behaviour. We can validate fields and display error messages without wiping user data. We do this by-

html-

<form id="name-form">

<input type="text" placeholder="first name" name="firstName">

<button>Submit</button>

</form>

Js-

document.querySelector('#name-form').addEventListener('submit', (e) => {

e.preventDefault();

console.log(e.target.elements.firstName.value);

e.target.elements.firstName.value = '';

});

Form element has property called elements. In code we can see how to use it. We use **prevent.Default** method to prevent the default behaviour of form , that is to reload webpage.

58)CheckBoxes

<input type="checkbox">

With this we will get checkbox. But we also need to some text to display. For this we use label tag.

<label>

<input type="checkbox"> Check me for fun

</label>

Now even if we click on label text our checkbox will be checked or unchecked. If we replace label with some other text, we won’t get this functionality. We can also use label with other types of inputs like this-

<label>

Search:<input id="search-text" type="text" placeholder="Filter todos">

</label>

Now when we click on text. Cursor appears on input.

Now lets see how to get value of checkbox in js.

We used changed event. Now this event will be called, whenever checkbox is checked or unchecked.html-

<label>

<input id="for-fun" type="checkbox"> Check me for fun

</label>

Js-

document.querySelector('#for-fun').addEventListener('change', (e) => {

console.log(e.target.checked);

});