61)Saving our data in local storage: part 1

Here we will learn how we can save user’s data. So next time they visit our page, they can pick up right where they left off. For ex lets say we type something in text box, then we refresh the page or we close the browser, that data is gone. We will see how we can save that data and fetch it later. We are going to be doing that using local storage.

Crud operations are basics operations on any data storage. To use local storage we will use localstorage object. this is provided by browser.

CRUD

C – create

localStorage.setItem('location', 'Philadelphia');

we have key value pair. location is key and philadelphi is value. here data will be saved in form of string.

R- read

localStorage.getItem('location');

even if we reload our page, we can still read data.

We can also access local storage from browser and we can change it.

D- delete

localStorage.removeItem('location');

if we delete all items-

localStorage.clear();

as we said local storage can only store string. What if we want to store array of objects? For this we have to convert our object into string. This is how we save a object in local storage-

const user = {

name: 'Sumeet',

age: 27

};

const userJSON = JSON.stringify(user);

console.log(userJSON);

localStorage.setItem('user', userJSON);

we use**JSON.stringify** to convert objector array into string.

Now how do we convert json back into js object code-

const userJSON = localStorage.getItem('user');

const user = JSON.parse(userJSON);

console.log(user);

now we will see how to integrate this part into application.

62)Saving our data in local storage: part 2

Now we will integrate local storage with our 2 apps, how we save our data as user adds it.

let notes = [];

const filters = {

searchtext: ''

};

const notesJSON = localStorage.getItem('notes');

if (notesJSON !== null) {

notes = JSON.parse(notesJSON);

}

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');

if (note.title.length > 0) {

noteEl.textContent = note.title;

} else {

noteEl.textContent = 'Unnamed Note';

}

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

});

};

renderNotes(notes, filters);

document.querySelector('#create-note').addEventListener('click', (e) => {

notes.push({

title: '',

body: ''

});

localStorage.setItem('notes',JSON.stringify(notes));

renderNotes(notes, filters);

});

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

filters.searchtext = e.target.value;

renderNotes(notes, filters);

});

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

// e.preventDefault();

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

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

// });

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

// console.log(e.target.checked);

// });

document.querySelector('#filter-by').addEventListener('change', (e) => {

console.log(e.target.value);

});

Here we load our data from local storage.

63)Splitting our application code

We created a file named notes-functions.js. now we will split our app into multiple files. Both of these files they share same namespace. Js files runs in order in which they are defined in index.html.

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

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

If we do

console.log(notes);

in notes-function.js, we will get refrence error. Because notes is defined in notes-app.js and this file runs after notes-function.js. if we reverse the order in which files are loaded, everything works fine.

But in our case we need functions file before app file, because in app file we will call functions that we have defined in function file.

64)Debugging Our Applications

Lets say we want to stop program execution at some line. To that use –

debugger;

with this line, program execution will stop at this line. File in which debugger is placed will be opened in debugger tools .Now in dev tools you can execute code line by line. You can use watchlist. You can see value of variables by taking cursor over them. You can execute any statement in console. Lets say we want to check value of variable a, then type a in console and press enter.you will get value of that variable. lets say you are in loop, in each iteration you can check value of i by typing i on console. To open console press **esc.**

One thing to be noted is, execution will pause only if our chrome dev tools are opened. If dev tools are not opened, then debugger statement won’t be taken into account.

66)Complex DOM rendering

In notes app along with each note we want to have render a button. Because we have divided our app into functions, we just need to change one function, i.e generateDOM. So we do this-

const generateDOM = (note) => {

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

const button = document.createElement('button');

button.textContent = 'x';

if (note.title. length > 0) {

noteEl.textContent = note.title;

} else {

noteEl.textContent = 'Unnamed Note';

}

noteEl.appendChild(button);

return noteEl;

};

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 = generateDOM(note);

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

});

};

Here buttons will be displayed after notes text. But lets say we want to have button before note text. Its not simple as appending child before we setup textcontent. If we do this-

const generateDOM = (note) => {

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

const button = document.createElement('button');

button.textContent = 'x';

noteEl.appendChild(button);

if (note.title. length > 0) {

noteEl.textContent = note.title;

} else {

noteEl.textContent = 'Unnamed Note';

}

return noteEl;

};

Then we dnt see button in DOM. Why this? It is because when we set textContent after appending button, we are appending it, we are overwriting everything in that paragraph which in this case includes the button. So how does we show button before note text?

We going to create a container element, this is going to be div element. Inside it we will have our 2 elements.This is going to make it possible to append one before the another and to switch up the order. This is how we solve the problem-

const generateDOM = (note) => {

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

const textEl = document.createElement('span');

const button = document.createElement('button');

button.textContent = 'x';

noteEl.appendChild(button);

if (note.title. length > 0) {

textEl.textContent = note.title;

} else {

textEl.textContent = 'Unnamed Note';

}

noteEl.appendChild(textEl);

return noteEl;

};

Istead of using p tag we used span. This is because p tag makes the note text to come in next line after button. While span solves this problem.

68)Setting up third party library

Now we want to remove todo items when we click on ‘x’ button. For this we will give unique identifier to each todo. To generate these unique id, we are going to use third party library.

Google this- github uuid

Open this link-

<https://github.com/kelektiv/node-uuid>

here we have code of library. Then we have documentation. There are different methods in which we can get started, there is a browser ready version. As we are going to use it in browser, so we will go with this one. Now there are different versions of algo, we will use 4th one. It says-

<script src="http://wzrd.in/standalone/uuid%2Fv4@latest"></script>

<script>

uuidv4(); // -> v4 UUID

</script>

So we need to import a script, then we need to call function and we will get id.

What we will do is when we are adding new note to notes array then we add new property called id and we use this library to generate unique id.

document.querySelector('#create-note').addEventListener('click', (e) => {

notes.push({

id: uuidv4(),

title: '',

body: ''

});

saveNotes(notes);

renderNotes(notes, filters);

});

69) Targeting by uuid

Here we deleted the note. First we delted the note in array with given unique id then we saved that notes array in local storage,then we called render function.

70)Checkbox Challenges

Here we used checkbox to mark event as completed or uncompleted. We used change event on checkbox. This event is called whenever checkbox is checked or unchecked. On this event, we toggle value of filter, save new array into local storage andre render dom.

On input change event is called when we focus out from that input.

71)Edit Note Page: Part1

Here we will create new page in notes pp. user will be redirected to this new page whenever user wants to change a note or add a note. So we create edit.html. inside this html we have this-

<a href="/index.html">Click Me</a>

/index.html means take me to index.html file which is on server.

Now we convert p tag to a tag in index.html and we send user to edit.html. now we want to do same when we click on create note button i.e we want user to be sent to edit.html page.

When user clicks on this button we are going to save note in local storage, then we are going redirect user to edit page. To do this redirect w eare going to use another global variable provided by browser, its called location. this object has function called assign. It takes a string as a value and takes user to user that we have give in argument.

72)Edit Note Page: Part2

Local storage is shared for all pages in a domain.

73)syncing data across pages

Lets say we have multiple tabs open. If we change note in one tab w, we want chnge to reflect in other tab. We will do it by listening to changes in local tab.

Window variable is representation of browser window. It also contains other browser variables that we have used.

Window.innerHeight, window.innerWidth

On window we also have other global variables made avalaible to use by browser.like window.console.log(‘sumit’), we just use shortcut to use console. We have other variables in window like window like document, if we do-

Console.log(window.document === document)

We get true. It means both are same object.

We will use window to attach event listener. We attach event listener on click event like this-

**Window.addEventListener(‘click’, () => {**

**Console.log(‘Clicked’);**

**});**

This event will be executed when we click on anywhere. We want to listen to **storage** event. It is fired when data in local storage changes. It is important to know that storage event fires on other pages. If I have 2 tabs open and tab 1 changes local storage, event listener won’t be fired for tab1, it is only going to be fired for tab 2.

If we print event object on console, there we hve some proeprties related to storage like which key was changed, old and new value etc.

notes-edit.js-

window.addEventListener('storage', (e) => {

if(e.key === 'notes') {

notes = JSON.parse(e.newValue);

note = notes.find((note) => {

return note.id === noteId;

});

if (note === undefined) {

location.assign('/index.html');

}

titleElement.value = note.title;

bodyElement.value = note.body;

}

});

So if we have 2 tabs of notes-edit.html open and we change note in one of them ,then other tab is automatically updated when ever we make changes in another tab.

Now what we want to do is, we want to update index.html, to whenever we make changes in notes-edit.html in another tab. We can make this happen by listening to storage event in index.html.

74)Javascript Dates

const now = new Data();

this gives current date.it contains time as well. Open mdn docs for functions on date object. there we hve different methods for setting data and different methods for getting data. We can get day, month year etc from date. We can also set these.

const now = new Date();

console.log(now);

console.log(`Year: ${now.getFullYear()}`);

console.log(`Month: ${now.getMonth()}`);

console.log(`Year: ${now.getDate()}`)

similarly we can get hours, mintes and seconds. Note that getMonth(0 returns 0 for jn and so on.

Now we will focus on how we can create date in different point of time. Not just current one. For this we just need to provide some arguments to Date function. First thing that we can put in date function is string representation of point of time. So we can do like this-

const now = new Date('January 21 2001 6:25:01');

passing a string to date is’t very common. When we are passing around date related information in our code. We typically store it as a number. It makes lot of sense. It makes saving information easy. If I want to store a cratedAt and last editedAt value on each note, all I really need to do is add two new properties whose values are numbers. Now missing piece to puzzle is what’s known as Unix epioc. It is spefic point in time that is in past. This point in time is January 1 1970 at 0:00. This is unix epic. When we represent time as a number we use positive numbers for all points in time after this and negative numbers for all points in time before this time. 0 represents this time i.e January 1 1970 at 0:00. As number increases we increase the number of milli seconds. So if we want to move one second in future to that time. I will use number 1000. Lets say we want to represent today’s time in number form. It will be large number.

const now = new Date('January 21 2001 6:25:01');

console.log(now.getTime());

**980038501000**

With getTime we can get number representation of date. now lets say we have number representation of a date and we want to get year, month and day of that day. First thing that we need to do is to convert that number back to date. We do y=this by passing that number to Date function. like this-

const now = new Date('January 21 2001 6:25:01');

const timestamp = now.getTime();

const myDate = new Date(timestamp);

console.log(myDate);

now we know how we can get these details from a date.

With number representation if a number is greater than another then that means that day comes after another. So w ecan use < or > operator.

Here lets say I want to get month like may, now inbuilt date object do not having any built in method for that. It will gives us a number then we have check index, in other words it is lot of work. To make life easier we will use moment.js, which is in built js library.

75)Moment

Here we will see that how moment js makes it easier to work with dates in js.

Now google momentj and open its official site then click on documentation. There we can see various methods to get moment js. go to browser method. Then click on cdn.js to get cdn for library. Here you will find many links. We need to get link that says- moment-with-locals-min.js at end. It is at bottom.

Now we will play around with this library. Because of this library we have moment function avalaible to us.

Now istead of using Date function, we can create dates like this-

const now = moment();

console.log(now.toString());

its similar to using new Date, without any arguments.

Now lets explore how we can manipulate date with momentjs.lets go to docs. Docs are good. there is lot to this library. So it is important to learn to navigate through documentation. If you want to perform getting and setting operations then go to **get+set** section of docs. We methods form miilisecods to year allowing us to get and set those values. Lets se minute method. On right hand side click on method. If we call this method with argument we set value and we call it without argument then we get value. if we do-

Now.minute(5),

Then we are setiing minute to 5 in time. Same is true for hour and so on.

const now = moment();

console.log(now.toString());

now.minute(27);

console.log(now.toString());

console.log(now.minute());

now there are other kinds of modification methods. They allow us to add or subtract to where we relatively are. Click on manipulate section on right hand side. Here we can see methods like add and subtract. Both add ans subtract take 2 arguments, first is amount and second is unit. We can also chain multiple method calls.

const now = moment();

console.log(now.toString());

now.add(1,'year').subtract(20, 'days');

console.log(now.toString());

**Sat May 12 2018 14:29:29 GMT+0530**

**Mon Apr 22 2019 14:29:29 GMT+0530**

Now will see methods related to formatting.in documentation there is entire section on it. This section is display. Open it-

<https://momentjs.com/docs/#/displaying/>

we have format method. We can call it without argument but we can also pass a string to it. Which tells how our time should be displayed. Open link above and see docs for details.

Let assume we want our date to be displayed in this format-

November 3rd, 2003

Code-

const now = moment();

console.log(now.toString());

now.add(1,'year').subtract(20, 'days');

console.log(now.format('MMMM Do, YYYY'));

**Sat May 12 2018 14:40:45 GMT+0530**

**April 22nd, 2019**

There is one more way we can format things that I want to expore. So under docs, in display section go to time from now. Time from ow returns a message like you in instagram , pic was liked 2 minutes before. When use fromNow it takes your moment and it figures out exactly how far in the past or future and then it returns a string. Code-

const now = moment();

console.log(now.toString());

now.add(1,'year').subtract(20, 'days');

console.log(now.fromNow());

output-

**Sat May 12 2018 14:51:19 GMT+0530**

**in a year**

here we manipulated the date away from current moment in time. we added a year to our date and subtracted 20 days. Then we run **fromNow** function, without any arguments on it. We get in a year. so you can see that it’s summarizing the difference between the 2 dates in a way that could be very useful for a user. It is not saying year minus 20 days because 20 days are small enough to be ignored as compared to year. so it is smart enough to see what units matter and which do not. lets change code –

const now = moment();

console.log(now.toString());

now.subtract(1,'week').subtract(20, 'days');

console.log(now.fromNow());

**Sat May 12 2018 15:57:06 GMT+0530**

**a month ago**

last thing to learn is how we can work with time stamp in moment. In docs go to UnixTimestamp. We have it for milliseconds and seconds. In js it is more common to work with milliseconds.

const now = moment();

console.log(now.valueOf());

here we will get time in number form. That is no of milliseconds that are passed after 1970. We can also create date out of this time stamp. Code-

const now = moment();

const nowTimestamp = now.valueOf();

const date1 = moment(nowTimestamp);

console.log(date1.toString());

lets change present date to your DOB-

const now = moment();

now.date(27).month(5).year(1991);

console.log(now.format('MMM D, YYYY'));

76)Intergrating Dates: Part1

Here we will use dates in our app. We will show user when last time a date was edit like edit last hour or something like that. First we created properties createdAt and updatedAt on our note object. then we are updating updateingAt whenever we edit a note. Then we use this updated info, to show span which tells how long ago that note was updated. We are also updating the span, when there change in local storage i.e when we update note in other tab span is update.

77)Intergrating Dates: Part2