Tell me about yourself –

Currently I develop pages in react like basic pages, article pages, home page, product details pages and others. So first we develop one any kind of page with ui using styled components and than upload content for like 35 pdp page so upload for 35 and more than 100article pages than upload content for all.

For this we have two ways either

Tcs built has ui kind of things we called it canvasX so either you can upload content using ui and also can use vscode – I prefer vscode its fast copy paste and change things.

Mostly UI is made for client only so they can also change content according to their need – so they no need to raise long list of change request for content change, only ui change request they raise.

While doing this I learned creating api using express

Mongodb connection using mongoose

Connecting both frontend and backend

How to use react redux and start creating small projects and than created ecommerce project.

Ecommerce –

1. Search functionality –

onSubmit => If(keyword.trim()) { navigate(`/search/${searchTerm}`)

###onChange searchTerm will update and onSubmit will navigate to search page.

Api call /api/products?keywords=””&pageNumber=””

In backend page = req.query.pageNumber and keyword = req.query.keyword.

Will count for products related to keyword –

const count = await Product.countDocuments({ ...keyword })

and than also find products for page 1 first 10 products and so on.

const products =

await Product.find({ ...keyword }).limit(pageSize).skip(pageSize \* (page - 1))

and will send response => res.json({products, page, pages:Math.ceil(count/pageSize)})

2. Home page –

On load – dispatch(listProducts())

const { data } = await axios.get(

      `/api/products?keyword=${*keyword*}&pageNumber=${*pageNumber*}`

In carousel – dispatch(listTopProducts())

3. pagination on homepage –

[…Array(pages).keys()].map((x)=>({x+1}))

4. Login page –

In frontend component – using useState() for handling data and onSubmit will post api will call

/api/users like – axios .post(‘/api/users’,{name,email,password}, {headers:{‘Content-Type’:’application/json’}})

Than on api – const {name,email,password} = req.body

Const user = await Users.findOne({email})

If(user && user.matchPassword(password)){

res.json({})}

5. register –

In controller findOne if found user already exists else create new user and also generate token.

5. product details page –

Normal 3 column layout, data is coming from redux state,

For select dropdown […Array(countInStock).keys()].map((x)=>(x+1))

onClick – addToCartButton => will navigate(‘/cart/productid?qty=${qty}’)

6. on Cart page –

After adding to cart will automatically navigate to cart page with productId and Quantity

And there first get cart data from redux to display available items in cart

And also update cart state in useEffect first render.

And also update state if user change orderedQuantity. For this

First find the product in cart state

existItem = state.cartItems.find((x)=>x.productId === productId)

if(existItem) return {…state, cartItems: state.cartItems.map((x)=>x.productId === productid ? item : x)}

else return {…state.cartItems,item}

And if user click on delete icon – will dispatch an action to remove product from cart.

Cartremoveaction => return {…state, cartItems: state.cartItems.filter((x)=>x.product !== product)}

onClick on checkoutHandler = > will navigate(‘/login?redirect=shipping’)

so if userInfo exist it will got shipping page. Here will save details in state and will navigate(‘payment’)

on payment screen – here we will save payment method and if shipping address is not present will redirect to shpping address page else will redirect to place order page.

Q1. Differentiate between Real DOM and Virtual DOM.

Q2. What is arrow function in React? How is it used?

A. Arrow functions are more of brief syntax for writing the function expression. They are also called ‘fat arrow‘ (=>) the functions. These functions allow to bind the context of the components properly since in ES6 auto binding is not available by default. Arrow functions are mostly useful while working with the higher order functions.

|  |
| --- |
| //General way  render() {  **return**(          <MyInput onChange={**this**.handleChange.bind(**this**) } />      );  }  //With Arrow Function  render() {  **return**(          <MyInput onChange={ (e) => **this**.handleOnChange(e) } />      );  } |

### ****Q. Explain the lifecycle methods of React components in detail.****

Some of the most important lifecycle methods are:

1. **componentWillMount()**–Executed just before rendering takes place both on the client as well as server-side.
2. **componentDidMount()**–Executed on the client side only after the first render.
3. **componentWillReceiveProps()**– Invoked as soon as the props are received from the parent class and before another render is called.
4. **shouldComponentUpdate()**–Returns true or false value based on certain conditions. If you want your component to update, return **true** else return **false**. By default, it returns false.
5. **componentWillUpdate()**– Called just before rendering takes place in the DOM.
6. **componentDidUpdate()**–Called immediately after rendering takes place.
7. **componentWillUnmount()**– Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

### ****Q. How**** ****are forms created in React?****

React forms are similar to HTML forms. But in React, the state is contained in the state property of the component and is only updated via setState(). Thus the elements can’t directly update their state and their submission is handled by a JavaScript function. This function has full access to the data that is entered by the user into a form.

handleSubmit(event) {

    alert('A name was submitted: ' + **this**.state.value);

    event.preventDefault();

}

render() {

**return** (

<form onSubmit={**this**.handleSubmit}>

            <label>

                Name:

                <input type="text" value={**this**.state.value} onChange={**this**.handleSubmit} />

            </label>

            <input type="submit" value="Submit" />

        </form>

    );

}

### ****React practical ques.****

## Q1. Display Simple Data with JSX

export default function App() {

return (

<div className="App">

<h1>JSX Example!</h1>

</div>

);

}

## Q. Loop Over and Display Data with JSX

// App.js

import users from "./users-data";

function App() {

return (

<div className="App">

<div className="page-deets">

<h2>Loop Over and Display Data with JSX</h2>

</div>

{/\* Iterate over imported array in userData \*/}

<div className="users">

{users.map((user, index) => (

<div key={index}>

<h3>{user.name}</h3>

<p>{user.location}</p>

<p>{user.car}</p>

</div>))} </div> </div>);}

## Q. Use React State to Update the DOM

function App() {

const [search, setSearch] = useState("");

return (

<div className="App">

<h2>Update Data from an input</h2>

<div className="input-display">

Seached Keyword: <b>{search}</b>

</div>

<div className="inputs">

<input

className="input"

type="text"

value={search}

placeholder="Seach Here"

onChange={(e) => setSearch(e.target.value)}

/>

</div>

</div>);}

## Q. Show an Alert Based on an Input

function App() {

const [phrase, setPhrase] = useState("");

if (phrase === "Hello React") {

alert("You may pass!");

}

return (

<div className="App">

<h2>What's the secret phrase?</h2>

<input

type="text"

value={phrase}

onChange={(e) => setPhrase(e.target.value)}

placeholder="Enter a secret"

/>

<p>

Hint: It's <strong>Hello React</strong>

</p>

</div>

);

}

## Q. Add Two Numbers

function App() {

const [number1, setNumber1] = useState();

const [number2, setNumber2] = useState();

const [total, setTotal] = useState(number1 + number2);

function calculateTotal() {

setTotal(number1 + number2);

}

return (<div className="App">

<h1>Adding Two Numbers</h1>

<div>

<input type="number"

value={number1}

onChange={(e) => setNumber1(+e.target.value)}

/>

<input type="number"

value={number2}

onChange={(e) => setNumber2(+e.target.value)}

/>

</div>

<button onClick={calculateTotal}>Add Them!</button>

<h2>Total: {total}</h2>

</div> );}

## Q. Fetch and Display from an API

const gitHubUrl = "https://api.github.com/users/deekshasharma";

function App() {

const [userData, setUserData] = useState({});

useEffect(() => {

getGitHubUserWithFetch();

}, []);

const getGitHubUserWithFetch = async () => {

const response = await fetch(gitHubUrl);

const jsonData = await response.json();

setUserData(jsonData);

};

return (

<div className="App">

<header className="App-header">

<h2>GitHub User Data</h2>

</header>

<div className="user-container">

<h5 className="info-item">{userData.name}</h5>

<h5 className="info-item">{userData.location}</h5>

<h5 className="info-item">{userData.blog}</h5>

<h5 className="info-item">{userData.company}</h5>

</div>

</div>

);

}

## Q. React Simple Counter

const App = () => {

const [counter, setCounter] = useState(0);

const handleClick = (type) => {

type === "increment" ? setCounter(counter + 1) : setCounter(counter - 1);

};

return (

<div>

<h2>Counter: {counter}</h2>

<div className="buttons">

<button onClick={() => handleClick("increment")}>Increment</button>

<button onClick={() => handleClick("decrement")}>Decrement</button>

</div>

</div>

);

};

Q To Do –

const App = () => {

let [item, setItem] = useState("");

const [list, setList] = useState([]);

const add = () => {

const l = [...list];

if (item === "") alert("enter some text");

else {

l.push(item);

setList(l);

setItem("");

} };

return (

<div>

<input

type="text"

value={item}

onChange={(e) => setItem(e.target.value)}

/>

<button onClick={() => add()}>add</button>

{list.map((lq, index) =>

<li key={index}>{lq}</li>)

}

</div>

);

};

export default App;

## Q. Search text based on list

const people = [

"Shashi Koshy",

"Dhriti Taneja",

"Dipa Mishra"

];

function App() {

const [searchTerm, setSearchTerm] = React.useState("");

const [searchResults, setSearchResults] = React.useState([]);

const handleChange = (e) => {

setSearchTerm(e.target.value);

};

React.useEffect(() => {

const results = people.filter((person) =>

person.toLowerCase().includes(searchTerm.toLowerCase())

);

setSearchResults(results);

}, [searchTerm]);

return (

<div className="App">

<input

type="text"

placeholder="Search"

value={searchTerm}

onChange={handleChange}

/>

<ul>

{searchResults.map((item) => (

<li>{item}</li>

))}

</ul>

</div>

);

}

## Q. Show or hide element in React

**Using React Class:**

class Toggle extends React.Component {

state = {

show: true

};

toggle = () =>

this.setState((currentState) => ({ show: !currentState.show }));

render() {

return (

<div>

<button onClick={this.toggle}>

Toggle: {this.state.show ? "Show" : "Hide"}

</button>

{this.state.show && <h2>Hello World!</h2>}

</div>

);

}

}

## Q. How can I update the parent state in React?

const App = () => {

const [name,setName] = useState("tarun")

return <div>

<Child name={name} set={setName}></Child>

</div>;

};

const Child = ({name,set})=>{

return <div>

{name}

<button onClick={()=>set("rathore")}>setName</button>

</div>

}

## Q. React prop validation for date objects

**PropTypes.instanceOf(Date):**

import React from "react";

import PropTypes from "prop-types";

const DateDisplay = ({ date }) => <p>{date.toString()}</p>;

DateDisplay.propTypes = {

date: PropTypes.instanceOf(Date)

};

export default function App() {

return (

<h3>

<DateDisplay date={new Date()} />

</h3>

);

}

## Q. How to access custom attributes from event object in React?

**event.target.getAttribute:**

export default class Header extends React.Component {

handleClick(event) {

console.log(event.target.getAttribute("name"));

}

render() {

return (

<h2 name="Header" onClick={this.handleClick}>

CLICK ME !

</h2>

);

}

}

## Q. Delete an item from state array in react

const App = ()=>{

const [users,setUsers] = useState(userData)

const deleteU=(id)=>{

setUsers(users.filter((user)=>id!==user.id))

}

return <div>

{users.map((u)=> <li key={u.id} onClick={()=>deleteU(u.id)}>{u.name}</li>)}

</div>

}

## Q. Sending the bearer token with axios

**Autherization token in axios:**

const api = 'your api';

const user = JSON.parse(sessionStorage.getItem('data'));

const token = user.data.id; /\*take only token and save in token variable\*/

axios.get(api , { headers: {"Authorization" : `Bearer ${token}`} })

.then(res => {

console.log(res.data);

.catch((error) => {

console.log(error)

});

## Q. Pass props in Link react-router

<Route exact path="/props-through-component"

component={() => <PropsPage title={`Props through component`} />}/>

<Route exact path="/props-through-render"

render={(props) => (<PropsPage {...props} title={`Props through render`} />

)}

/>

## Q. How to disable a button when an input is empty?

class App extends React.Component {

state = {

email: ""

};

handleChange = (e) => {

this.setState({

email: e.target.value

});

};

render() {

return (

<div>

<input

placeholder="Email"

value={this.state.email}

onChange={this.handleChange}

/>

<button disabled={this.state.email.length < 1}>Submit</button>

</div>

);

}

}

## Q. How can one tell the version of React running at runtime in the browser?

const REACT\_VERSION = React.version;

<h1>React version: {REACT\_VERSION}</h1>

## Q. How to start search only when user stops typing?

function App() {

const [value, setValue] = React.useState("");

const handleOnChange = (event) => {

setValue(event.target.value);

};

React.useEffect(() => {

const timeoutId = setTimeout(

() => console.log(`Search function called: "${value}"`),

300

);

return () => clearTimeout(timeoutId);

}, [value]);

return (

<>

<input onChange={handleOnChange} value={value} placeholder="Search" />

<h1>{value}</h1>

</>

);

}

Project – 1 Jobs

1. index.js –

 <Provider *store*={store}>

      <App />

    </Provider>

2 App.js – how to use react router

<BrowserRouter>

      <Routes>

        <Route *path*="/" *element*={<ProtectedRoute><Dashboard /></ProtectedRoute>}>

          <Route *index* *element*={<Stats />} />

          <Route *path*="all-jobs" *element*={<AllJobs />} />

          <Route *path*="add-job" *element*={<AddJob />} />

          <Route *path*="profile" *element*={<Profile />} />

        </Route>

        <Route *path*="/register" *element*={<Register />} />

        <Route *path*="/landing" *element*={<Landing />} />

        <Route *path*="\*" *element*={<Error />} />

      </Routes>

    </BrowserRouter>

How to use outlet –

function Dashboard() {

  return (

    <Wrapper><main *className*="dashboard">

        <SmallSidebar />

        <BigSidebar />

        <div><Navbar />

          <div *className*="dashboard-page"><Outlet /></div>

        </div>

      </main>

    </Wrapper>

  );

}

3. use of children –

function ProtectedRoute({ *children* }) {

  const navigate = useNavigate();

  const { user } = useSelector((*state*) => *state*);

  useEffect(() => {

    if (!user) {

      navigate("/landing");

    }

  }, [user, navigate]);

  return *children*;

}

4. update state by 1 line–

    setValues({ ...values, [*e*.target.name]: *e*.target.value });

5. create or login action –

const { data } = await axios.post(`/api/v1/auth/${*endPoint*}`, *currentUser*);

6. add or remove from local storage –

const addUserToLocalStorage = ({ *user*, *token*, *location* }) => {

  localStorage.setItem("user", JSON.stringify(*user*));

  localStorage.setItem("token", *token*);

  localStorage.setItem("location", *location*);

};

const removeUserFromLocalStorage = () => {

  localStorage.removeItem("token");

  localStorage.removeItem("user");

  localStorage.removeItem("location");

};

7. update user –

const { data } = await authFetch.patch("/auth/updateUser", *currentUser*);

8. create Job –

    await authFetch.post("/jobs", {position,company,jobLocation,jobType,status,})

9. update Job –

await authFetch.patch(

      `/jobs/${*state*.editJobId}`,

      {

        company,

        position,

        jobLocation,

        jobType,

        status,

      },

      {

        headers: {

          Authorization: `Bearer ${*state*.token}`,

        },

      }

    );

10. delete job –

await authFetch.delete(`/jobs/${*jobId*}`);

11. get all jobs –

let url = `/jobs?page=${page}&status=${searchStatus}&jobType=${searchType}&sort=${sort}`;

  if (search) {

    url = url + `&search=${search}`;

  }

const { data } = await authFetch(url, {

      headers: {

        Authorization: `Bearer ${token}`,

      },

    });

12. server.js –

const app = express();

app.use(express.json());

*// auth routes*

import authRouter from "./routes/authRoutes.js";

app.use("/api/v1/auth", authRouter);

import jobsRouter from "./routes/jobsRoutes.js";

app.use("/api/v1/jobs", authenticateUser, jobsRouter);

app.get("/", (*req*, *res*) => {

*res*.send("its goodw");

});

await mongoose.connect(process.env.MONGO\_URL);

    app.listen(port, () => {

      console.log(`server is listening on ${port}`);

*// console.log(db);*

    });

13. routes –

const authRouter = express.Router();

authRouter.route("/updateUser").patch(authenticateUser, updateUser);

14. register controller –

  const { name, email, password } = *req*.body;

  const userAlreadyExists = await User.findOne({ email });

  const user = await User.create({ name, email, password });

  const token = user.createJWT();

*res*.status(StatusCodes.CREATED).json({user});

15. login controller –

  const { email, password } = *req*.body;

  const user = await User.findOne({ email }).select("+password");

  const isPasswordCorrect = await user.comparePassword(password);

const token = user.createJWT();

  user.password = undefined;

*res*.status(StatusCodes.OK).json({ user, token, location: user.location });

16. update controller –

  const { email, name, lastName, location } = *req*.body;

  const user = await User.findOne({ \_id: *req*.user.userId });

user.email = email;

  user.name = name;

  user.lastName = lastName;

  user.location = location;

  await user.save();

  const token = user.createJWT();

*res*.status(StatusCodes.OK).json({ user, token, location: user.location });

## Q. ***How do you check whether a string contains a substring?***

There are 3 possible ways to check whether a string contains a substring or not,

**a.) Using includes:** ES6 provided String.prototype.includes method to test a string contains a substring

var mainString = "hello", subString = "hell";

mainString.includes(subString)

**b.) Using indexOf:** In an ES5 or older environments, you can use String.prototype.indexOf which returns the index of a substring. If the index value is not equal to -1 then it means the substring exist in the main string.

var mainString = "hello", subString = "hell";

mainString.indexOf(subString) !== -1

**c.) Using RegEx:** The advanced solution is using Regular expression test method(RegExp.test), which allows for testing for against regular expressions

var mainString = "hello", regex = "/hell/";

regex.test(mainString)

## Q. ***How to convert string to title case with javascript?***

Title case means that the first letter of each word is capitalized. You can convert a string to title case using the below function,

function toTitleCase(str) {

return str.replace(

/\w\S\*/g,

function(txt) {

return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();

}

);

}

toTitleCase("good morning john"); // Good Morning John

## Q. ***How do you trim a string in javascript?***

JavaScript provided a trim method on string types to trim any whitespaces present at the begining or ending of the string.

" Hello World ".trim(); //Hello World

For eval argument should be a string.

var x = 10;

var y = 20;

var z = '50';

eval('x + y + 1'); // returns 31

eval(z); // returns 50

unique in array –

Array.prototype.getUnique = function() {

var o = {}, a = [], i, e;

for (i = 0; e = this[i]; i++) {o[e] = 1};

for (e in o) {a.push (e)};

return a;

}

## Q. ***How do you check if a string starts with another string?***

You can use ECMAScript 6 String.prototype.startsWith() method to check a string starts with another string or not. But it is not yet supported in all browsers. Let us see an example to see this usage,

"Good morning".startsWith("Good"); // true

"Good morning".startsWith("morning"); // false

## Q. ***Explain array methods [ join(), pop(), push(), shift(), unshift(), concat(), map(), filter(), reduce(), reduceRight(), every(), some(), indexOf(), lastIndexOf(), find(), findIndex(), includes() ]***

[“a”,”b”,”c”].join() – by default it will join with comma like - a,b,c

**d.) array.shift()**: The shift() method removes the first element from an array and returns that removed element. This method changes the length of the array.

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.shift();

console.log(fruits) // Output: Array ["Orange", "Apple", "Mango"]

**e.) array.unshift()**: The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.

var fruits = ["Banana", "Orange", "Apple"];

fruits.unshift("Mango","Pineapple");

console.log(fruits); // Output: Array ["Mango", "Pineapple", "Banana", "Orange", "Apple"]

**f.) array.concat()**: The concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

const array1 = ['a', 'b', 'c'];

const array2 = ['d', 'e', 'f'];

console.log(array1.concat(array2)); // Output: Array ["a", "b", "c", "d", "e", "f"]

**g.) array.map()**: The map() method creates a new array with the results of calling a provided function on every element in the calling array.

var array1 = [1, 4, 9, 16];

// pass a function to map

const map1 = array1.map(x => x \* 2);

console.log(map1); // Output: Array [2, 8, 18, 32]

**h.) array.filter()**: The filter() method creates a new array with all elements that pass the test implemented by the provided function.

var words = ['spray', 'limit', 'elite', 'exuberant', 'destruction'];

const result = words.filter(word => word.length > 6);

console.log(result); // Output: Array ["exuberant", "destruction"]

**i.) array.reduce()**: The reduce() method executes a reducer function (that you provide) on each element of the array, resulting in a single output value.

const array1 = [1, 2, 3, 4];

const reducer = (accumulator, currentValue) => accumulator + currentValue;

console.log(array1.reduce(reducer)); // Output: 10

console.log(array1.reduce(reducer, 5)); // Output: 15

**j.) array.reduceRight()**: The reduceRight() method applies a function against an accumulator and each value of the array (from right-to-left) to reduce it to a single value.

const array1 = [[0, 1], [2, 3], [4, 5]].reduceRight(

(accumulator, currentValue) => accumulator.concat(currentValue)

);

console.log(array1); // Output: Array [4, 5, 2, 3, 0, 1]

**k.) array.every()**: The every() method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

function isBelowThreshold(currentValue) {

return currentValue < 40;

}

var array1 = [1, 30, 39, 29, 10, 13];

console.log(array1.every(isBelowThreshold)); // Output: true

**l.) array.some()**: The some() method tests whether at least one element in the array passes the test implemented by the provided function. It returns a Boolean value.

var array = [1, 2, 3, 4, 5];

var even = function(element) {

// checks whether an element is even

return element % 2 === 0;

};

console.log(array.some(even)); // Output: true

**m.) array.indexOf()**: The indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.

var beasts = ['ant', 'bison', 'camel'];

console.log(beasts.indexOf('camel')); // Output: 2

console.log(beasts.indexOf('giraffe')); // Output: -1

**n.) array.lastIndexOf()**: The lastIndexOf() method returns the index within the calling String object of the last occurrence of the specified value, searching backwards from fromIndex. Returns -1 if the value is not found.

var paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';

var searchTerm = 'dog';

console.log('The index of the first "' + searchTerm + '" from the end is ' + paragraph.lastIndexOf(searchTerm));

// Output: "The index of the first "dog" from the end is 52"

**o.) array.find()**: The find() method returns the value of the first element in the provided array that satisfies the provided testing function.

var array1 = [5, 12, 8, 130, 44];

var found = array1.find(function(element) {

return element > 100;

});

console.log(found); // Output: 130

**p.) array.findIndex()**: The findIndex() method returns the index of the first element in the array that satisfies the provided testing function. Otherwise, it returns -1, indicating that no element passed the test.

var array1 = [5, 12, 8, 130, 44];

function isLargeNumber(element) {

return element > 20;

}

console.log(array1.findIndex(isLargeNumber)); // Output: 3

**q.) array.includes()**: The includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.

var array1 = [1, 2, 3];

console.log(array1.includes(2)); // Output: true

var pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('at')); // Output: false

## Q. ***What is the difference between for..in and for..of?***

* **for in**: loops over enumerable property names of an object.
* **for of**: (new in ES6) does use an object-specific iterator and loops over the values generated by that.

Both for..of and for..in statements iterate over lists; the values iterated on are different though, for..in returns a **list of keys** on the object being iterated, whereas for..of returns a **list of values** of the numeric properties of the object being iterated.

Example:

let list = [4, 5, 6];

for (let i in list) {

console.log(i); // "0", "1", "2",

}

for (let i of list) {

console.log(i); // "4", "5", "6"

}

## Q. ***When to use reduce(), map(), foreach() and filter() in JavaScript?***

* **forEach():**

It takes a callback function and run that callback function on each element of array one by one.

Basically forEach works as a traditional for loop looping over the array and providing array elements to do operations on them.

var arr = [10, 20, 30];

arr.forEach(function (elem, index){

console.log(elem + ' comes at ' + index);

})

// Output

10 comes at 0

20 comes at 1

30 comes at 2

* **filter():**

The main difference between forEach() and filter() is that forEach just loop over the array and executes the callback but filter executes the callback and check its return value. If the value is true element remains in the resulting array but if the return value is false the element will be removed for the resulting array.

Note: filter does not update the existing array it will return a new filtered array every time.

var arr = [10, 20, 30];

var result = arr.filter(function(elem){

return elem !== 20;

})

console.log(result)

// Output

[10, 30]

* **map():**

map() like filter() & forEach() takes a callback and run it against every element on the array but whats makes it unique is it generate a new array based on your existing array.

Like filter(), map() also returns an array. The provided callback to map modifies the array elements and save them into the new array upon completion that array get returned as the mapped array.

var arr = [10, 20, 30];

var mapped = arr.map(function(elem) {

return elem \* 10;

});

console.log(mapped)

// Output

[100, 200, 300]

* **reduce():**

reduce() method of the array object is used to reduce the array to one single value.

var arr = [10, 20, 30];

var sum = arr.reduce(function(sum, elem) {

return sum + elem;

});

console.log(sum); // Output: 60

var marks = [50, 20, 70, 60, 45, 30];

function findMin(arr) {

return Math.min.apply(null, arr);

}

function findMax(arr) {

return Math.max.apply(null, arr);

}

console.log(findMin(marks));

console.log(findMax(marks));