What is react element?
It is an object

Root is the place where only react runs.

Render
It modifies the DOM and it overrides all the code.

* CrossOrigin attribute
The 'crossorigin' attribute is used to allow a webpage to make a request to a different domain in order to load a resource. This is necessary because of the same ~~policy~~ origin policy, which prevents a web page from making requests to a dif domain than the one that served the page.

The 'crossorigin' attribute was introduced to allow web developers to work arounds the same ~~policy~~ origin policy in a controlled way.

By specifying the 'cross origin' attribute, web developers can tell the browsers to include the 'origin' header in the request, and can also specify whether or not to send any credentials (such as cookies) along with the request.

This allow web developers to make requests to different domains in a way that is safe and secure, while still respecting the security measures in place to protect against cross-site request forgery (CSRF) attacks.

* **Emmet**

Emmet is a toolkit that helps you to write HTML and CSS faster. It is based on a set of abbreviations that expand into full HTML and CSS code when you press the tab key.

* **Async and Defer**

The 'async' and 'defer' attributes are used to specify the load loading behaviour of external Javascript scripts. They can be added to the '<script>' element to indicate whether or not the script should be loaded asynchronously, and whether or not the script should be executed as soon as it is loaded.

// The script will be loaded asynchronously, and will be executed as soon as it is loaded

```
<script src="script.js" async></script>
```

// The script will be loaded asynchronously, and will be executed after the page has finished loading

```
<script src="script.js" defer></script>
```

* Difference b/w Library and framework.

Library: Library is something which is limited to module, class, objects functions and pre written code.

Ex: Jquery, react.js etc.

CDN = Content Delivery Network.

Page:
Date: / /

Framework: Frameworks comprises lots of API's, compiler's, support program, library.
Ex:- Angular JS, Spring, Node JS etc.,

* **What is CDN? Why we use it.**
A Content Delivery Network (CDN) is a system of distributed servers that deliver webpages and other web content to a user based on the geographic locations of the user, the origin of web page and a content delivery server.

CDN's are the way to improve the performance of a website by reducing the distance between the server and the user and by reducing the load on the origin server.

?? CDN's are used to improve the performance, availability, security and cost-effectiveness of websites.

* **Why is React known as react?**
React is a javascript library for building user interfaces. It was developed by facebook and is often referred to as "React".

The name "React" is short for "Reactive", which refers to the way that React components are designed to react to changes in data.

React was designed to be easy to use and to make it possible to build complex user interfaces by breaking them down into smaller, reusable components. It is also designed to be fast, scalable and easy to maintain, which has contributed to its popularity among developers.

Date / /

??? * **Dif b/w React and ReactDOM**

React and ReactDOM are two different libraries that are commonly used together to build user interfaces (UIs) using React JS Library.

ReactDOM is a separate library that is used to render React components to the DOM. The DOM is a tree like structure that separates the HTML of a webpage, and ReactDOM is responsible for updating the DOM to reflect the changes made to a React component.

In short, React is a library for building UI components, and React DOM is a library that is used to render those components to the DOM. They are commonly used together, but they are distinct libraries with different functions.

??? Before making our application production ready. What we need to do?

- Minify our app
- Optimisation
- clean console.log
- Bundle
- Cacheing in our app

Creat-react-app = uses babel, webpack

What is parcel?
It is bundler. It is known as package
- module
- package of javascript file
- some piece of code.

NPM: does not stands for node package manager.
but everything else.
↳ NPM is basically used for managing our packages.

Why we use it?
Because we need want many packages in our app. Because our React app cannot be build by just injecting React into our web pages.

Why do we use NPM?
Because our React app is powered by a lot of things lot of packages, supposed we have to minify it, suppose we have to bundle up things, we have to remove console logs, optimised our app. So for that we need lot of helper packages and those helper packages come inside npm.

A react app is a huge app, it just not run on React. There are lots of super powers in React, which we need and those super powers comes from packages, which packages are present by using npm.

Parcel is a BEAST

→ My project depends on packages

Dependencies :

All the packages that our project needs. it means my project is dependent on something. and parcel is one of such dependency.

devDependencies means, we need our bundler/package in our developer environment.

- "^" - caret → will update to all future minor/patch versions without inc. the major version
- "~" - Tild → will update you to all future patch versions,
- " " - don't want any updates without increament minor versions.

'package.json' is a file that contains meta data about a project. It specifies the dependencies that the project requires and allows you to define scripts that can be run during the project lifecycle.

'package-lock.json' is a file that is automatically generated when you install dependencies in a project using 'npm'. It records the exact version of each dependency that was installed, so that the same versions can be installed when the project is run on another machine.

In summary, 'package.json' specifies the dependencies that our project needs while 'package-lock.json' is a record of the exact versions of these dependencies that are installed.

Page:
Date:

Node modules :- → This is the place where all the from all the superpowers are come.
↳ It just like a database for our npm
↳ It has many helper functions

- npx → execute using npm.

"we cannot import or export script tags"
module ⇒ can import and export.

Parcel

→ HMR - Hot Module Reloading Replacement
↳ parcel will keep track of which all the files which we are updating. If we go and change anything, HMR will re track.

→ File Watcher Algorithm - written in C++

→ .parcel-cache -
The cache feature of parcel is used to stores the results of certain operations in order to speed up the build process.
When Parcel run, it will check the cached version instead of running the operation again. This can significantly speed up the build process, especially for large projects with many dependencies.

→ Bundling
→ Minify
→ Cleaning our code.
→ Compression
↳ renames our files

→ Dev and production build.
→ Super Fast build algorithm
→ Image Optimization
→ Caching while development
↳ media (images) should in cache

→ dist - (short form for "distribution")
It is typically the directory where the final, production-ready version of a web application is built.
It usually contained the final, minified and optimised version of the application's code, as well as any assests such as images and fonts that are needed for the application to run.
The contents of the "dist" directory are usually what is deployed to a web server in order to make the application available to users.

→ Compatible with older version of browser
→ HTTPs on dev → zero config
→ Manages ~~in~~ port numbers → Tree shaking
↳ removing unwanted codes
→ Consist Hashing Algorithm

* Why is react is so fast?

1. Virtual DOM:
React uses a virtual DOM to minimize the number of DOM mutations required to update the views.

2. One way data flow:
In react, data flows in single direction from the parent component to the child components. This helps to prevent unnecessary re-renders of components that are not directly affected by change in state.

3. React.Memo and UseMemo:
This react feature allows us to ~~minimize~~ optimize our components by memoizing the output of expensive calculations. This means that the calculations will only

be performed again if the input values has changed, which can help to improve the performance of our application.

Pure Components:

React provides a way to create "pure components" that only re-renders when the props they receive change.

Server side rendering:

It uses bundlers which also helps in increasing the performance of the React app.

Transitive dependencies:

Transitive dependency is a dependency that a software package has on a library or module that is needed by one of its own dependencies.

Example:- if package A depends on package B and package B depends on package C, then package a has a transitive dependency on package C

Browserlist:

Browserlist is a thing which makes our code compatible for a lot of browsers

Render:

↳ Something updating in the DOM is known as render.

// React.createElement ⇒ object ⇒ HTML (DOM)

converts into

facebook developers built this....

Page :
Date : / /

* **Is JSX HTML in Javascript ?**

JSX is a HTML like syntax but it is not HTML inside javascript. JSX is used to define the structure of a components UI's

JSX	HTML
• It is syntax extension for javascript	HTML is a markup language used to structure and format content on the web. HTML does not have this capability.
• JSX uses curly braces to include JS expressions and values. JSX can include dynamic values and expressions	
• In JSX, we use camelCase notation for attributes Ex. "className"	In HTML, Kebab-case is used. Ex. "class" whereas html cannot
• JSX includes self closing tags. Ex. <img />	Ex. <img> </img> whereas HTML cannot
• JSX includes Javascript expressions and functions as attributes. Ex. <Component onClick={ someFunction } />	

JSX ⇒ Babel React.createElement ⇒ object ⇒ HTML(DOM)

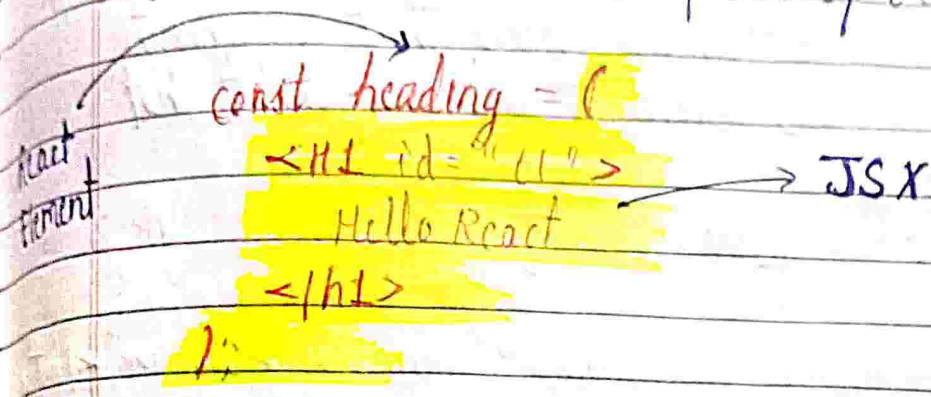
* **Babel :-**

Babel is a Javascript library which reads our code and gives us another code.

JSX is just a react element

* Functional Component

Functional component is just a normal function which is just returning piece of JSX as it can also return piece of react element



* Component Composition :—

If I have to use a component inside a component that is known as component composition.

Passing a component into a component is called composition / component composition

* JSX

It is a syntax extension for JavaScript that allows you to write HTML like code in our javascript files. When our code is compiled, the JSX is transformed into regular JavaScript code that can run in a web browser.

example JSX code

const button = <button> Click me! </button>;

This JSX code will be transformed into JavaScript code

const button = React.createElement("button", null, "click me");

JSX ⇒ React.CreateElement ⇒ Object ⇒ HTML (DOM)

Page
Date / /

* What does JSX behind the scenes?

JSX is transformed into regular JavaScript code by a compiler, such as Babel.

When we write JSX in our code, we are actually creating a syntax tree that represents the structure of the element or component that we are defining.

For example:- JSX code ↓

```
const button = <button className="primary">click me!</button>
```

This JSX code will be transformed into a syntax tree that looks something like this: ↓

```
{
  type: "button",
  props: {
    className: "primary",
    children: "click me!"
  }
}
```

This syntax tree is then transformed into following JavaScript code:

```
const button = React.createElement("button", { className: "primary" },
  "click me!");
```

This JavaScript code uses the 'React.createElement()' function to create a new Element with the specified type (in this case "button") and props (an object containing a 'className' property. The 'createElement()' function returns an object that represents the element, which can then be rendered to the DOM

JSX expressions must have one parent element. ???
Any piece of JSX component that we wright, there can only be one parent.

Suppose if we are waughting any JSX, like This ↓

```
const jsx = <h1> JSX1 </h1><h1> JSX2 </h1>
```

This is invalid JSX

⇒ JSX component should have only one parent element ↓

```
const jsx = <div><h1> JSX1 </h1>
              <h1> JSX2 </h2>
            </div>
```

Here → is a parent element, it is only one parent element and it has two childs.

So JSX can only have one root, one parent element, there is something known as React.fragment which offers react to us
So, what is React.fragment ????
React.fragment is a component which is exported by React
↳ import React from "react";

```
const jsx = (
  <React.Fragment>
    <h1> JSX </h1>
    <h1> JSX2 </h1>
  </React.Fragment>
);
```

It is coming from the React library.

So, React.fragment is like a empty tag.
But React.fragment is looks so ugly in the code. So, React developers, let's change it in some news, ↓

```
Const jsx = (
  <>
    <h1> JSX </h1>
    <h1> JSX2 </h1>
  </>
);
```

→ This is also a React.fragment element
→ It is also called as empty tag
→ shorthand property

*

```
const styleObj = {
  backgroundColor: "red",
};

Const jsx = (
  <div style = {styleObj}>
    <h1> JSX </h1>
    <h2> JSX2 </h2>
  </div>
);
```

Inline CSS

And it is same as

```
const jsx = (
  <div style = {{ backgroundColor: "red" }}>
    <h1> JSX </h1>
    <h2> JSX2 </h1>
  </div>
```

Page :
Date: / /

Can I use React fragment inside another react fragment.???

Yes, we can use a React fragment inside another fragment. A react fragment lets us group a list of children without adding extra nodes to the DOM.

```
const MyFunction = () => {
    return (
        <React.Fragment>
            <h1> heading 1 </h1>
            <React.Fragment>
                <h1> heading 2 </h1>
                <h2> heading 3 </h2>
            </React.Fragment>
        </React.Fragment>
    );
};
```

In this example, the outer fragment groups the 'h1' element and the inner fragment, and the inner fragment groups the two elements i.e 'h1' and 'h2'. This will results in the following DOM structure

```
<h1> heading 1 </h1>
<h1> heading 2 </h1>
<h2> heading 3 </h2>
```

Config driven UI → Interviews

* join() :-

To join elements of an array into a single string, we can use the 'join()' method of the 'Array' object.

This method takes an optional separator string as an argument, which is used to separate the elements of an array in the resulting string.

```
const fruits = ['apple', 'banana', 'mango'];
const fruitString = fruits.join(', ');
console.log(fruitString);
// output: "apple, banana, mango"
```

We can also use the 'join' method without an argument, in which the case elements of the array are separated with a comma by default.

```
const fruits = ['apple', 'banana', 'mango'];
const fruitString = fruits.join();
console.log(fruitString);
// output: "apple, banana, mango"
```

* Optional Chaining :- (?.)

In JS, Optional chaining is a feature that allows you to access properties of an object that may or may not exist, without causing an error. It uses the '?.' operator, lets you safely access a property of an object without having to check if the object is 'null' or 'undefined' first.

Example let obj = {
prop1 : {
prop2 : {
prop3 : "Hello"
}
}
};
let value = obj?.prop1?.prop2?.prop3;
console.log(value); // "Hello"

If the any of the properties is not exist, then the result will be 'undefined'.

* Props = = object

Props (short for "properties") are a way of passing data from a parent component to a child component. A component can receive props, which are passed in as an object as an argument to the component's function, and then use them to render dynamic content.

React wraps up all the properties into a variable known as "prop".

"When we pass any props to functional component it is received as a parameter which is prop."

(props) => ({destructuring}) =>

const {name, cuisines} = destructuring data

diff algorithm, Reconciliation

React fiber → Reconciliation engine
↳ React 16 (new)
↳ Responsible for 'diff' algorithm

Why react is so fast ???

React uses which is something known as Virtual DOM. Virtual DOM is a representation of actual DOM. It is not actual DOM.

React uses something known as Reconciliation. Reconciliation is the process where we have a 'diff' algorithm which finds out the difference between the trees one tree and the other tree. Now when we have found out the difference, it will re-render only the portion which is required.

Why we use key ???

In React, the "key" prop is used to identify which items in a list are unique, so that React can efficiently update the list when it changes.

So, in short, keys help React identify which item have changed, are added, or are removed, so that it can efficiently update the list, without having to re-render the entire list.

Why we do not use index key as key in React ???

When we use index value as 'key' attribute when creating a list:—

1. Performance Issues due to unnecessary re-renders
2. Issues in Data mapping in case list items are sorted, filtered, or deleted.

When it is safe to use index as key in a list?

1. Data static
2. When you know reordering of lists: sorting, filtering is not going to happen.
3. In the absence of an id.

default import
named import

Two way binding

Page :
Date : / /

Config.js ==> Constant.js

→ React uses one way data binding

If we need to maintain a variable, that changes itself, then we need to maintain a variable that is a React Variable

React variable is a kind of state variable
Every component in React maintains a state and we can puts all the variables into our state.
And every time we have to create a local variables to use state in it.

??? What is hook ???
Hook is a special function that allows a component to hook into React features such as state and life cycle methods.
Hooks introduced in React 16.8

Hooks are nothing but normal kinds of functions

??? What is useState ???
'useState' is a hook in React that allows us to add state to functional components.
'State' is a way to store and manage data that can change in a React Component.

With 'useState', we can create a state variable and a function to update it

Date: / /

The 'useState' function takes an initial state as its argument, and returns an array with two item; the current state, and a function that can used to update the state.

→ From React library.

useState() → is used to create state variable

e.target.value → whatever we write in Input box.

Why do I need this state variable???

Because whenever we a variable our component will not update and when we use state variable and click on it, our component thought that something is updated in the state and prints it.

Why the variable is not updated???

Because when we update our variable the component is not re-render and it always remain freez, so for updating use state variables.

- 'useState' is a hook which allows us to maintain, update etc in the state.
- state is a container which stores the data.
- state is a ~~public~~ local variable

* Difference between 'named export' and 'default export' and '* as export';

⇒ named export: name export allow a module to export multiple values, each with a unique name. These can be imported into another module using same name.

For example:

```
// File: math.js
export const sum = (a, b) => a + b;
export const multiply = (a, b) => a * b;

// File: main.js
import { sum, multiply } from './math.js';
console.log(sum(2, 3)); // 5
console.log(multiply(2, 3)); // 6
```

⇒ Default export :-

It allows a module to export a single value as the default export. This value can be imported into another module using any name. For example.

```
// File: math.js
const sum = (a, b) => a + b;
export default sum;

// File: main.js
import mysum from './math.js';
console.log(mysum(2, 3)); // 5
```

"* as" :-

This syntax allows to import all the exports from a module and bind them to a specific object.

```
// File: math.js
export const sum = (a, b) => a+b;
export const multiply = (a, b) => a*b;

// File: main.js
import * as math from './math.js';
console.log(math.sum(2,3)); // 5
console.log(math.multiply(2,3)); // 6
```

* Monolithic architecture :-

Monolithic architecture refers to a software design where all components of the application are combined into a single, unified executable. In this architecture, all components of the application such as there user interface, database access code, and business logic are all combined into a single codebase. This approach is relatively single simple and easy to understand, but can become difficult to maintain and scale as the application grows.

Microservices architecture :-

It is a software design pattern in which large application is broken down into a set of small, independant services that communicate with each other over a network. Each service is responsible for specific business capability, and can be developed,

Page load ⇒ call the API ⇒ fill the data

Page:
Date: / /

deployed and scaled independently of the other services. This approach allows for greater flexibility and scalability, but can also be more complex to design and manage, as it requires a robust infrastructure for communication and coordination between the services.

- Easier to test
- easy to maintain
- single responsibility

* There are 2 ways to the page load the page and call the API.

① Loads the page —300ms→ call the API —500ms→ Render Page.

② Loads the page —100ms→ render the page —300ms→ call the API —500ms→ Update the UI

Best method is 2nd method

↳ To do in this way we use useEffect() hook in React.

* **UseEffect hook** → is called after the component is rendered.

useEffect is a function, but we call this function by passing another function. This another function is a callback function. (This will not called immediately it will called whenever useEffect will be called).

Whenever our component renders, and re-renders continously. Our components code of the useEffect

is called and after every render it will call the function that we passed in it (useEffect).

When will my component renders?

There 2 times when my component renders,

1. either on my state changes
2. or my props changes.

After every re-render our 'useEffect' function is called and it is the bad way to call it, and on every re-renders.

So if we don't want to call after every re-render, pass in a indepency array in to it. This is knows as dependency array.

```
useEffect( () => {
    //API call

}, [ ]);

console.log("render");
```

- empty dependency array ⇒ once after render
- dependency array [searchText] ⇒ once after initial render + everytime after render (my searchText changes)

"Best part in React is 'Reconcilliation' "

- JS expression & statement
- Early Rendering

Page :
Date : / /

??? **Why React is so fast?**

There are so many things which makes React very fast.

- There are bundlers, which is doing
  - minification
  - removing console.log()
  - image Optimization
  - load of bundling

Parcel is doing all that

And, we have React in our applications, which has,

- Virtual DOM
- Reconciliation
- diff algorithm
  - which tracks the on every state update
  - diff of the trees are calculated
  - only that particular portion on my DOM is changed and React does all that very fast
  - DOM updates in React is very fast...

?? **What happens if we don't have dependency array?**

If we don't have dependency array that means our 'useEffect' is not dependent on any anything and 'useEffect' default behaviour is to be called after render.

So every time my component rendered, my 'useEffect' will be called.

"It will be called after each rendered".

" React says that never create a component inside another component."
" Never ever write 'useState()' inside if else condition."
" Never write "useState()" in for-loop "

→ React gives us 'useState' hook to create state variables inside our component (functional component).
'Never use 'useState()' outside our functional component'.

Why CDN is a good place for hosting images.

CDN (Content Delivery Network) is a good place for hosting images because it allows for faster and more reliable delivery of the images to users. Here are a few reasons why :—

1. Distributed servers
2. Caching
3. High availability
4. Security
5. Cost effective

What is 'createBrowserRouter' ?

'react-router-dom' library

'createBrowserRouter' is a function provided by the 'react-router-dom' library that creates a '<BrowserRouter>' component. It is used to handle client side routing in a React application.

This is the recommended router for all react router web projects. It use DOM History API to update the URL and manage the history stack

??? What is RouterProvider?

'RouterProvider' is a custom component and it is not part of 'react-router' package. It is a higher order component that is used to provide the 'router' object and other routing related context to the other components in a React Application

All router objects are passed to this component to render our app and enable the rest of the API's

??? useRouterError

'useRouterError' is a hook in the React Route library that allows us to handle errors that occuring during the rendering of a route

⇒ There are two types of routing

1. Client side routing
2. Server side routing

Something
→ we don't want to load from server

- **client-side routing:**

This type of routing is handled by the client's web browser, and allows for the use of different URLs for different pages within a single page application (SPA).

Example of client-side routing libraries include React Router and Vue Router.

- **Server Side Routing:** all our comes from server

This type of routing is handled by the user, and maps URLs to specific server side actions or views.

Examples :- Express.js (for Node.js) and Ruby on Rails.

'Link' in React

In React, a Link is a component that is used to navigate b/w diff pages or routes in a web application.

It is similar to a anchor tag (<a>) in HTML, but with additional functionality provided by React Router.

The 'Link' component allows us to navigate to different pages or routes within our application without causing a full page refresh, which improves the overall user experience.

Outlet :— → It is component

An <Outlet> should be used in parent route elements to render their child route elements. This allows nested UI to show up when child routes are rendered. If the parent route matched exactly, it will render a child index route or nothing if there is no index route.

All the children will go into the 'outlet' according to the route.

useParams() :—

In React, the '**useParams**' hooks allow a component to access the dynamic Parameters of a route. It is part of the 'react-router-dom' library and is used in conjuction with the 'Route' component.

Class Based Components :— are normal JavaScript classes

"We can't create a class based component without render method"

Inheriting something from R.C

React

```
class Profile extends React.Component {
    render() {
        return <h1> Profile Class Component </h1>
    }
}

export default Profile
```

Whatever we return from 'render()' method will be injected into DOM

* **Constructor**

In a class component, the constructor is a special method that is called when an instance of the component is created. It is used to initilize the component's state and setup any necessary bindings

The constructor method is defined with the keyword "constructor" and it called automatically by React when the component is created. It typically receives one argument, **props**, which is an object containing the component's properties.

The best place to initilize and set state variables in a class component is inside the constructor method. This ensures that the state is properly setup before the component is rendered.

* **Why we use super(props)**

In class based components, the 'super(props)' call is used to pass the props argument to the parent's class constructor.

This is necessary because when a class extends another class, the child class's constructor needs to call the parent's class constructor in order to properly set up the child class's 'this' object.

"React uses one big object to hold state whole state, even in Functional Component's React still uses one big object behinds the scenes to manage all states."

// We DO Not Mutate State Directly
// Never DO this.state = something

*→ **Life cycle method**

"First of all 'Constructor' is called then rendered

Constructor ⟶ Rendered ⟶ componentDidMount

* **componentDidMount** :-

componentDidMount is the best place to call API's in class Based Components because it is called immediately after the component has been rendered on the page. This means that the component has been fully rendered and all of its child components have been mounted as well.

Babel helps in converting JSX ⟶ HTML code

Page :
Date :

VVI When react renderings things up it does in two phase

1. Render Phase
2. Commit Phase

1. Render phase :-
When the component's states and props are used to determine what changes need to be made to the virtual DOM. React then updates the virtual DOM with these changes, which creates a representation of how the actual DOM should look.

2. Commit Phase :-
The commit phase is when React takes the updated virtual DOM and makes the corresponding changes to the actual DOM. This is when the changes are actually visible on the screen.

In short, the render phase is where React decides what changes should be made, while the commit phase is where React makes those changes.

→ Render phase includes constructor and render method
→ Commit phase is the place where actually is modifying our DOM.

ComponentDidMount is called after the initial render has finished or after my component is on the browser.

Render Phase is faster than Commit Phase.
↳ modifying the DOM is very difficult task

Why we use asyn before componentDidMount function but not inside useEffect in callback function.

It is generally recommended to use 'async' in 'componentDidMount' when fetching data from the server. Because 'componentDidMount' is called only once and the data only needs to be fetched once, an 'async' function can be used to wait for the response.

On the other hand, 'useEffect' is called on every render, so it is better to use 'asyn' inside the 'callback function of 'useEffect' only if the data needs to be refetched on every render.

Component Did Update :- → called after every update

It is a life cycle method that is called after a component has been updated. It is called after the components props or state have been changed, and the component has re-rendered on the screen.

'ComponentDidUpdate' is a good place to put any code that needs to be run after component has been updated, such as updating the state based on the previous props or state, fetching new data based on the updated props, or triggering a side effect such as a network request.

Component Will Unmount :-

It is a life cycle method that is called just before a component is removed from the screen. This method is typically used to perform any necessary clean-up tasks, such as canceling network requests, removing event listeners, or cleaning up any other resources that were created in 'componentDidMount'

It's important to use this lifecycle method to clean up all the subscription, setTimeOut, setInterval, listeners and etc that were created in componentDidMount, otherwise it can lead to memory leaks, and it will cause the performance issues.

→ The return statement inside the 'useEffect()' hook is called a "Cleanup function". It is used to perform any necessary cleanup operations when the component using 'useEffect()' is unmounted or the component's props or state change.

```
useEffect( () => {

    const setInterval(() => {
        console.log("I Love React");
    }, 1000);

    // Cleanup function --- called
    return () => {
        clearInterval(timer);
    }
}, []);
```

* **Advantages of using hooks :-**

1. Improved code organization
2. Better Performance
3. Easier to test
4. Greater flexibility
5. Better debugging
6. Community support.

* Custom hooks

" We are creating custom hooks to return some logics. It does not return any JSX like functional comp.

* Why we need to create own hooks :-

- ↳ allows for reusability
- ↳ easier to understand
- ↳ easier to maintain
- ↳ easier for testing

" Whenever we add EventListener, we need to clean up the event listners.

" Whenever you feel like there is a feature that can abstract, create a hook.

*

// Chunking
// Code splitting
// Dynamic bundling
// Lazy Lodding
// On Demand Lodding
// Dynamic import

- Make my trip is a image heavy website.
- In paytM, every icon can be a different bundle.

" We we are loading our component in demand, react tries to suspend it."

// Upon On Demand Loading → upon render → suspend loading

* PostCss

PostCss is a tool for transforming styles with Javascript plugins. It allows developers to use modern css syntax, including future css features, and polyfill older browsers.

It is just a 'transpiler' that turns a special PostCss plugin syntax into a Vanilla CSS. We can think of it as the Babel tool for CSS

→ features and benefits.

- PostCss is fully customizable so we can use only the plugins and features we need for our application.
- It also produces fast build times compared with other pre-processors.
- We can write our own custom plugins.

* tailwind.config.js

→ content configuration

It is the place where we configure the paths to all of our HTML templates, Javascript components, and any other source files that contain Tailwind class names.

→ Theme configuration

It is the place where we define our projects colour palette, type scale, fonts, breakpoints, border radius values and more.

→ Plugins : —

Plugins are extensions that enhance the functionality of a software application. In the context of PostCSS, plugins are small javascript programs that can manipulate or transform the CSS code in a specific way.

Plugins let you register new styles for Tailwind to inject into the user's stylsheet using Javascript instead of CSS.

* ~~postcssrc~~

* Tailwind CSS
  - CSS on the go (in the same file)
  - reusability
  - less bundle size
  - flexible UI (customizable)
  -

"Tailwind CSS says that whenever you write CSS wrong in my way, I will override everything"

* @tailwind base;
@tailwind components
@tailwind utilities

These three lines are CSS import statements that bring in the base styles, component styles and utility styles of the tailwind CSS framework into a project.

Page :
Date : / /

* Pros of Tailwind ess

1) Easy to debug
2) less code is shipped
3) No dublicate css
4) Bundle size is small
5) Time saving (faster development)
6) It is much costomizable when compare to Material UI something (flexible)
7) It is newer way of writing css

* Cons of Tailwind CSS

1) To much classes
2) It comes with a learning curve.
3) Can lead to over-reliance on pre-defined classes

* Props Drilling :-

Props drilling refers to the practise of passing data from a parent component to child component through props in Reactjs. This method is used to pass data from one component to another, especially when the components are nested and don't have a direct parent-child relationship.

→ How can we send data from child to parent Component :-
There are many ways such as :—
i) Local storage
ii) Can create custom hooks.

3) Not a good way

* Cons of props drilling :-

1) complexity :-
As the number of nested components increases, passing data through props can become complex and difficult to manage.

2. Performance :-
Excessive props drilling can negatively impact performance as it requires more re-renders of components.

3. Code duplication :-
When passing data through multiple components, there is a risk of duplicating code and passing the same props multiple times, making the code harder to maintain.

4. Cluttered code : With growing number of props, components can become cluttered and difficult to read, leading to decreased maintainability.

5. Lack of modularity :-
With props drilling, it can be difficult to reuse components in different parts of the application, leading to a lack of modularity in the code.

* **Lifting state up in React :-**

Lifting state up refers to moving the state management of a component to a higher level of component. This is done to share state among multiple components and to allow for better data management and modularity. By lifting the state, it becomes easier to reuse and manage shared state, rather than having to duplicate the state management code in multiple components.

"Context us 'God', It is Everywhere"

Page :
Date : / /

* Context in ReactJS :—

Context in ReactJS provides a way to share data that is required by multiple components in our application without the need to pass props down the component tree manually. It allows us to pass data to components at different levels in the component tree, without having to pass props down through every level.

ReactJS Context API consists of two components, a 'provider' component, and a 'consumer' component.
The 'provider' component holds the data that needs to be shared, and the 'consumer' component is used to access the data from the Providers.

- **Why we use it?**
"When we want some piece of data to be accessible anywhere in my app, we use context API."
And other reasons are :—

- Share data across multiple components:
- Reduce Prop drilling:
- Centralize data management:
- Provide a way to access global data:
- Simplify the component tree:

* What is redux?

Redux is a tool for managing the state (data) in a React application. It allows us to store all our application's data in one place (known as the "store"), and this makes it easier to build complex, real-world

React != Redux

applications and keep the state organized.

* **Cons of using Redux :—**
  - It takes time to learn and set up, especially for small or simple applications.
  - Debugging, the state updates can be difficult to follow and trace through the application.

That being said, Redux has been widely adopted and can be very beneficial for large, complex applications.

If our applications is very small, then we don't need redux, Only use when we have to build large scale applications involves a lot of data handling.

- Redux comes with lot of complexities.
- A huge learning curve.

React and Redux are both different things

* **Redux Toolkit :—**

Redux Toolkit is a library that makes it easier to work with Redux in a React application. It provides a set of tools and conventions for working with the state management and reduces the amount of boilerplate code that needs to be written.

Redux have a single store only, where we can store anything

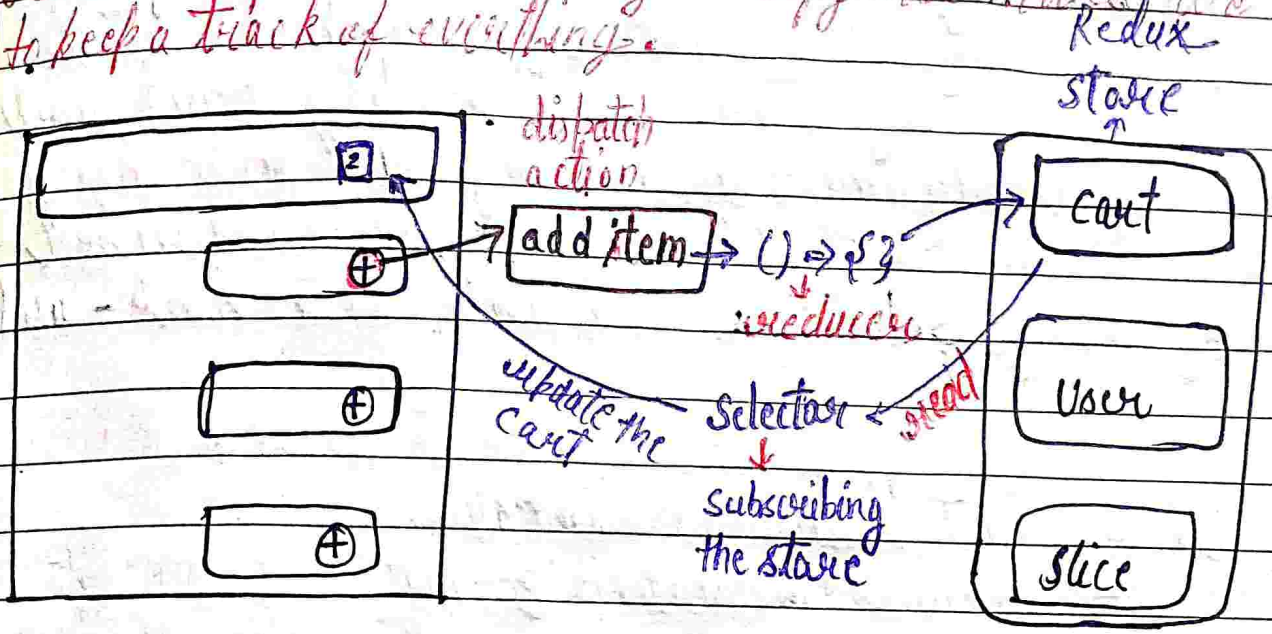
theme slice.

It is poor thing to keep all data at one place, but not it's not a poor thing. Because we will create logical separation into our store. We will create 'slices' of our store.
Each store will have different slices. What all slices can have in our app. It can have a,
- user slice
- authentication slice
- theme slice
- cart slice

A 'slice' is a small portion of the store.

→ Our component cannot directly modify the store. Suppose if we click on the '+' button, we cannot directly modify the store. We cannot modify the 'cart'.
So, what we can do — so, we have to do 'dispatch' and 'action'.
'action' can be supposed as 'add item' (for example) when we will click on 'add item' it means we are dispatching an action.
what will this action does —
It will call an ~~action~~ a function.
↳ normal JavaScript function.

And this function will modify our 'cart'. We cannot directly modify our app. why ???
So whenever there is large application, we don't want random components randomly modify our stores, we need to keep a track of everything.



When we click '+' button, we dispatch an action which calls a function known as reducer function, which modify or update the slice of the store.

There is something known as Selector, if we want to read cart, we have to call this selector, the selector will give us the information that will update the cart

⊕ → dispatches (action) —Calls→ reducer function —updates→ slice of the store

Cart ←update— selector ←read it

↳ retrieving the current state of the store without modifying it.

'Selector' means, we are selecting the slice out of the store, or we are selecting the portion of the store. Selector is a hook end of the day and hook is a normal javascript function. / useSelector()

There is one more jargon, that is - when we use selector - it is known as subscribing to the store.

subscribing the store $\xrightarrow{\text{means}}$ reading the store

So, basically, it is like async with the store. Whenever my store will modify, cart will automatically modify in my UI

It means, the component is subscribed by using **selectors**. and this is a hook - useSelector()

→ Let us install redux library.

→ npm i @reduxjs/toolkit

→ npm i react-redux

!? Why we are installing two libraries.

Core Job of Redux is to maintaine the store (data), create the slices.

npm i @reduxjs/toolkit

core of the redux

npm i react-redux

bridge between redux and react.

→ Now, we will create our store, so for creating a store we need configureStore() and this configureStore will come from "@reduxjs/toolkit"

This configureStore will contain slices.

Now, this store is different and our app is also different, so how we will connect our store to our app.

So will we need a provider to connect our store to with the app.

We can put this provider to our whole app or for certain components also.

So, here we will provide this provider to our whole app. So in our case, the root component is App layout. So we will provide this provider to our whole app.

Here, we will import a provider which comes from react-redux and in this way the Provider component will act as a bridge between our store and app for connecting them.

```
<Provider store={store}>
    <UserContext.Provider
      value={{
        user: user,
        setUser: setUser,
      }}
    >
      <Header />
      <Outlet />
      <Footer />
    </UserContext.Provider>
</Provider>
```

Now, lets create a slice for our cart. We create slices by using createSlice() and it comes from "@reduxjs/toolkit".
because creating slices is the core job of our redux toolkit.

For creating slice we need two things first
- the name of the slice
- and the initial state

```
1.  const cartSlice = createSlice({
2.      name : 'cart',
3.      initialState : {
4.          items : [],
5.      },
    })
```

So, now we have to modify our app. so will we use Reducer() function to modify our app. And remember when this reducer function calls - on dispatch of an action.
Now, we have to gave a name for this dispatch action, lets gives the name - addItem
So, it means, this reducer function will contain a mapping of actions (addItem) and our reducer function.

```
6.      reducers : {
7.          addItem : () => {
8.
9.          }
        }
```

So, How we can modify our store, this reducer function takes in two things

- state → (Initial state)
- action → (This is the place where I'll we get the item which will we have to add to our cart)

↳ action payload

```
reducers: {
    addItem: (state, action) => {
        // some logic
    }
}
```

* * * * * * * * * * * * *

```
reducers: {
    addItem: (state, action) => {
        state.items.push(action.payload);
    },
    removeItem: (state, action) => {
        state.items.pop();
    },
    clearCart: (state) => {
        state.items = [];
    },
},
});
```

all actions name

```
export const { addItem, removeItem, clearCart } = cartSlice.actions;
```

↳ from here we exports all actions.

```
export default cartSlice.reducer;
```

↳ from here we exports all reducers

Syntax