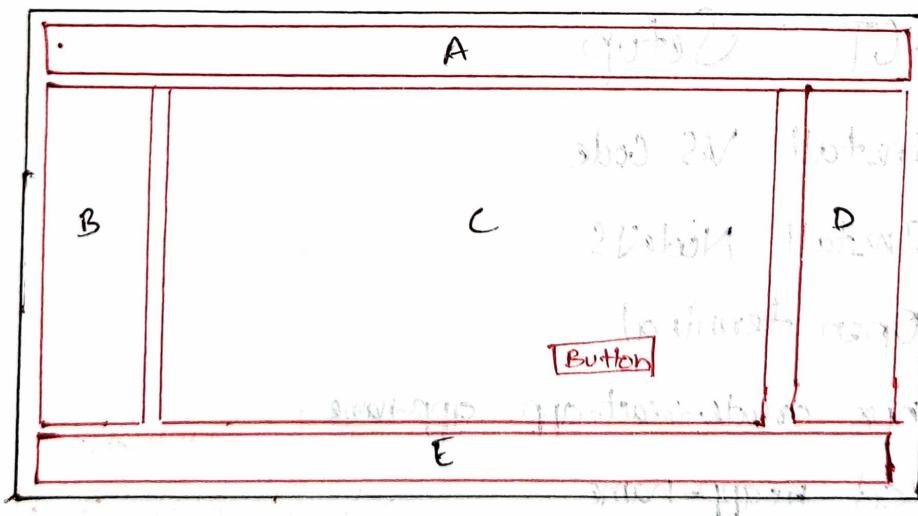


REACT

React is based on JS library used to create UI
React is all about components.
It is a component based architecture.
Components lets you split the UI into independent, reusable pieces, and think about each piece in isolation.
Conceptually, components are like JS functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.



A, B, C, D and E are components.
Button is also a component inside a component C.

Why we need React?

→ JS is based on Imperative approach but react is based on declarative approach.

→ Declarative approach in React means that we need to tell the end state only and React will work automatically.

Imperative means line by line.

→ SPA approach (Single Page Application)

→ Faster development, etc.

Advantages of SPA

* REACT Alternatives

Porter provides alternatives to React JS

→ Angular

→ JS

* REACT Setup

→ Install VS Code

→ Install NodeJS

→ Open terminal

→ npx create-react-app app-name

→ cd app-name

→ npm start

commands to

run in terminal

Difference between React and Angular

React has been developed

Folder ↗

v src

File → index.js → It is the entry point of the JS.

It is the first file to execute.

To use any file we import it first.

package.json → contains dependencies & scripts

v public

index.html → It is the main HTML file.
(contains root element)

v src

index.js → It fetches the root element from
index.html. (root.render(<APP />));
(It contains 'APP' component)

APP.js → It contains the working of App component.

App.css → CSS file for styling

Note: To use any function, first you need to export
from where it is created and import where
you want to use.

To create a component always create a folder
for that component and create JS and CSS
files in it. This folder is created inside
src folder.

Create Component

To create a component we create a js file which contains a function, to use that function we export in the same file and import where we want to use it.

The function we created for the component

returns the JSX code: `id ← (stateless)`

Code (Component file)

each time we edit code, `id ← (functional)`

`import './item.css';` here we use className because `const name = "Rishabh";` class is a reserved keyword in JS
`function Item() {`

`return <p className='rishabh'> Rishabh </p>`

`} return <p className='rishabh'> {name} </p>`

`export default Item;`

we use {} when data changes dynamically.

App.js file (code)

`import './App.css';`

`import Item from './components/item';`

`function App() {` this function has no parent

`return <div> {` It is mandatory to have a

`<div>` parent, when you have multiple children

`<Item> <Item>`

`<ItemDate> <ItemDate>`

`</div>`

Another component

`export default App;`

* Use of Props

Props are arguments passed into REACT Components.

Props stands for properties.

Code (Component file)

```
function Item(props) {
```

```
    const itemName = "Rishabh";
```

```
    return (<p className="Name">{itemName}</p>);
```

and now export

```
export default;
```

Code (App.js file)

```
import Item from './Item';
```

```
function App() {
```

```
    return (
```

```
        <div>
```

```
            <Item name="Rishabh"></Item>
```

```
        </div>
```

```
    );
```

```
}
```

⇒ Another way (App.js)

```
function App() {
```

```
    const response = [
```

```
        {
```

```
            itemName: "Kushwaha",
```

```
        ]
```

```
return (
  <div>
    <Item name={response[0].itemName}></Item>
  - - -
  </div>
);
}
```

⇒ But what happens when we do this

```
<Item name={response[0].itemName} I am Rishabh</Item>
  <div> I am Rishabh </div>
```

This will not be
visible.

To make this visible, we

need to add this in item.js file.

```
<div>
  <p className="Rishabh">{itemName}</p>
  &props.children
</div>
```

className="Rishabh" - Div will act as

child

;

;

single node without children

if child is present

I = single term

{ "name": "Rishabh" } = multiple

```
return (
  <div>
    <Item name={response[0].itemName}></Item>
  </div>
);
```

Index of object
of array

⇒ But what happens when we do this

```
<Item name={response[0].itemName} I am Rishabh</Item>
<div> <span>I am Rishabh</span> </div>
```

This will not be visible.

To make this visible, we

need to add this in item.js file.

```
<div>
  <p className="Rishabh">{itemName}</p>
  {props.children}
</div>
```

* React Events

Handling events with React elements is very similar to handling events on DOM elements.

React has the same events as HTML: click, change, mouseover etc. (All the events start with on like onClick, etc.)

→ React events are written in camelCase syntax:
onClick instead of onclick.

→ React event handlers are written inside curly braces:

onClick={shoot()} instead of onclick="shoot()".

Q. What happens when we use onClick={shoot()} prop

instead of onClick={shoot} prop?

In this case the function shoot() will be called and it will call the function even if we didn't click on button.

(*) state = [value, setState] hook

* Hooks

A hook is a special function that lets you "hook" into React features. For example, useState is a hook that lets you add React state to function components.

* USESTATE

The React useState Hook allows us to track state in a function component.

This hook is used for storing variables that are part of your application's

state and will change as the user interacts with your website.

The state change happens per component instance basis.

State is used when we want to manage changing data in an application.

Ex. We need to import useState first.

import {useState} from "react";

function FavColor() {

const [color, setColor] = useState("black");

↓ ↴

It is the function that updates our state. It is the function that updates our state.

HW UseState is a asynchronous hook, it will wait for the component to finish its cycle, don't re-render, and then it will update the state.

Q Why const is used and still the value changes?

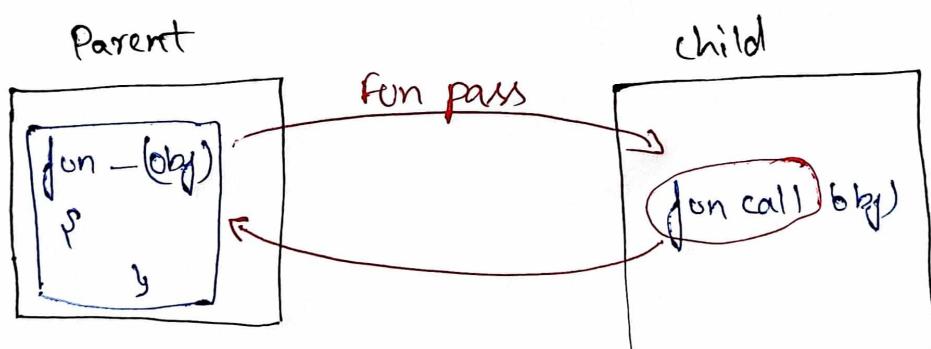
When a component is rendered, the function is executed again, creating a new 'color' variable, which has nothing to do with the previous variable.

* ~~onlick~~ onChange

This event in REACT detects when the value of an input element changes.

* Lifting State Up

Sharing state is accomplished by moving it up to the closest common ancestor (parent) of the components that need it.



Step 1: Function will be passed from parent to child.

Step 2: Function will be called in child component with input parameter.

Step 3: Value will be accessed.

* useEffect Hook

The useEffect hook allows you to manage side effects in your components.

A side effect is a change that affects something outside the components being rendered.

Ex- Fetching data, directly updating the DOM, and timers.

Syntax-1

```
useEffect(() => {
  // code
});
```

Every render

Syntax-2

```
useEffect(() => {
  // code
}, []);
```

first render

Syntax-3

```
useEffect(() => {
  // code
}, [name]);
```

First render +
when dependency changes

Dependency

Syntax - 4

```
useEffect(() => {
```

Ⓐ [if some code]

Ⓑ [return () => {
 }
});

To handle unmounting
of a component

Here Ⓑ will execute before Ⓒ

Form Handeling

```
import { useState } from "react";
```

```
function App() {
```

```
    const [firstName, setFirstName] = useState("");
```

```
    const [lastName, setLastName] = useState("");
```

```
    function FirstchangeHandler(event) {
```

```
        setFirstName(event.target.value);
```

```
}
```

```
    function LastchangeHandler(event) {
```

```
        setLastName(event.target.value);
```

```
}
```

```
    return (
```

```
        <div>
```

```
            <form>
```

```
                <br/>
```

```
                <input
```

```
                    type="text"
```

```
                    placeholder="First name"
```

```
                    onChange={FirstchangeHandler}
```

```
>
```

```
<br/>
```

```
<input type="text" placeholder="last name" onChange={LastchangeHandler}/>
        );
    </div>
</form>
);
}

export default App;
```

The problem with above code is that for every input field we are defining different onChange Handler functions and multiple useState hooks.

Let say we have input field and other form components in a very large number then writing onChange handler for every component is not the best way to create a form.

The more optimized code is given below.

Code

```
function App() {
```

```
const [formData, setFormData] = useState({  
    firstName: "",  
    lastName: "",  
    email: "",  
    isVisible: true  
});
```

name attribute is provided in every element

|| Here the whole data of the form is being tracked i.e. in formData

```
console.log(formData);
```

```
function changeHandler(event) {
```

```
    const {name, value, checked, type} = event.target
```

```
    return setFormData((prevFormData) => {
```

Here it is

storing the ...prevFormData

Previous state of the form with the help of ...spread operator

```
    );
```

```
return (
```

```
<div>
```

```
<form>
```

```
<input type="text" value=""/>
```

type = "text"

placeholder = "first name"

onChange = {changeHandler}

name = "firstName"

value = {formData.firstName}

/>

With the help of value you can track the state of individual element.

```
<br/>
```

Controlled components

```
<input type="text" placeholder="last name"/>
```

type = "text"

placeholder = "last name"

onChange = {changeHandler}

name = "lastName"

value = {formData.lastName}

/>

```
<br/>
```

```
<input type="email" value=""/>
```

type = "email"

placeholder = "enter your email"

onChange = {changeHandler}

name = "email"

value = {formData.email}

/>

<input

type = "checkbox"

onChange = {changeHandler}

name = "isVisible"

id = "isVisible"

checked = {formData.isVisible}

/>

<label htmlFor="isVisible">

Am I visible?

</label>

~~<form>~~

~~<label>~~



'htmlFor' attribute is used
to attach the label with
checkbox with the help of
'id' attribute i.e. isVisible.

II Controlled Components

II It maintains state inside the component

<fieldset>

<legend> Mode: </legend>

<input

type = "radio"

onChange = {changeHandler}

name = "mode"

```
value="Online-mode"
id="Online-mode"
checked={formData.mode === "Online-Mode"}  
    </input>  
> <label htmlFor='Online-mode'> Online Mode </label>  
  
<input type="radio" value="Offline-mode" id="Offline-mode" checked={formData.mode === "Offline-Mode"}>  
    </input>  
> <label htmlFor='Offline-mode'> Offline Mode </label>  
  
</fieldset>  
> <label htmlFor='favCar'> Favorite car </label>-  
  
<select  
    name="favCar"  
    id="favCar"  
    value={formData.favCar}  
    onChange={changeHandler}>  
    <option value="Scorpio"> Scorpio </option>  
    <option value="Tharr"> Tharr </option>  
</select>  
> <button> Submit </button>  
</form>  
</div>
```

REACT ROUTER

If is a standard library for routing in React.

It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL without refreshing the page.

⇒ Installation Command

```
npm install react-router-dom
```

⇒ Then go to index.js file and wrap BrowserRouter around App i.e.

<BrowserRouter>

<App/>

</BrowserRouter>

⇒ Now create routes in App.js file using

<Routes>

<Routes>

tag

To create single route we use following tag

```
<Route path="/" element=<div> Home </div>>/>
```

From the above routing method we have to change the link in order to make changes in the page. i.e to go from Home to About.

To avoid this problem we will use `<NavLink>` tag to show link of the various other links.

By using above tag we can automatically add active class in the link which we have clicked. We can also use nesting in `<Route>` tag but by default it will not show the children of the parent `<Route>` element.

In order to make the visible we will use `<Outlet>` tag in MainHeader.js

Ex `<Routes>` Parent Route

```
<Route path="/" element=<MainHeader>>>
```

It makes it `<Route index element=<Home>>>`

default route `<Route path="/About" element=<Support>>>`

`</Routes>` signs function to break child route

```
</Routes>
```

When you use `<Outlet>` tag it show the child content and parent content also, so if you want to remove the parent content from every child content.

Then you have to remove the content from the parent element. From this when you go to child element you won't see the content of the parent element.

But when you click on the parent element you'll show nothing because you have already removed the content.

To avoid this we create a MainHeader on route and give path of Home to it and add the content to the child route. Same as we did in the previous example.

Use Navigate Hook (`useNavigate()`)

This hook helps you to go to specific URL, forward or backward pages in the React Router. (For better understanding implement this)

Context API

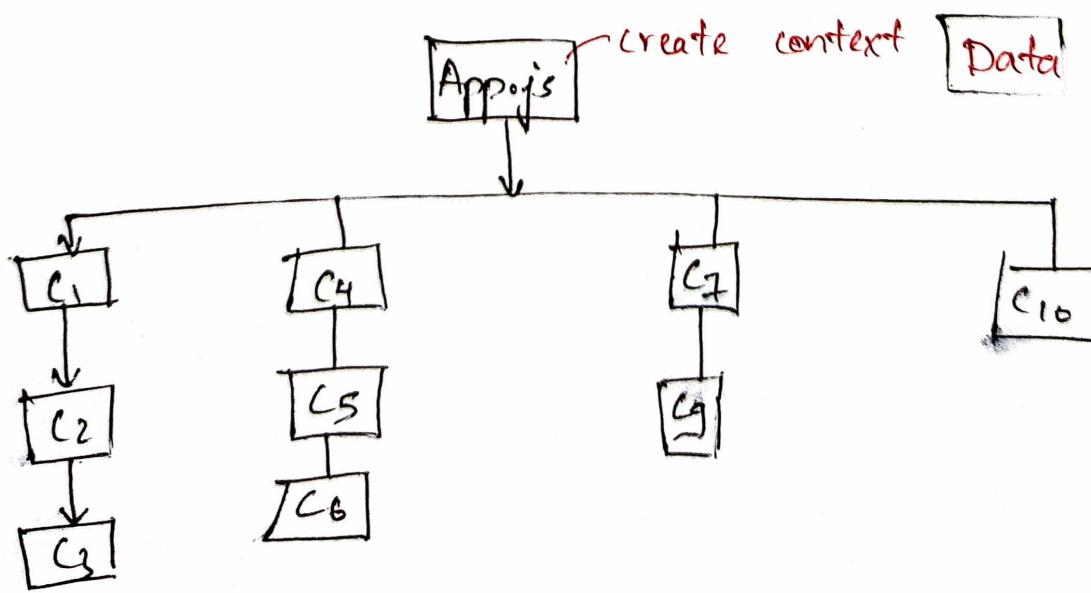
The React Context API is a way for a React app to effectively produce global variables that can be passed around.

This is an alternative to prop drilling or moving props from grandparents to child to parents, and so on.

It allows us to pass down and use data in whatever component we need in our React app without using props.

'Context' means snapshot of data.

'API' is like a function which does a certain work.



Here, a Dart context is created in App.js file and it can be accessed to any of its childs.

Rules to follow while using Context API

- ① Create context
- ② Provide context
- ③ Consume context

useSearchParams()

The `useSearchParams()` hook is a built-in hook in the `react-router-dom` package that allows you to access and update the search parameters in the current URL query string.

The hook may return an array with two items:

1) The first item is an instance of the

`URLSearchParams` class that represents the search parameters in the current URL.

2) The second item is a function that can be used to update the search parameters.

When this function is called with a new `SearchParams` instance or an object of key-value pair, the search parameters in the URL are updated accordingly.

useLocation()

The `useLocation()` hook is a built-in hook in the `react-router-dom` package that allows you to access the current location in your React components. The `useLocation()` hook returns an object with the following properties:

- * `pathname`: The current URL pathname (excluding the domain and query parameters)
- * `search`: The query string in the current URL (including the "?" character).
- * `hash`: The anchor portion of the current URL (including the "#" character).
- * `state`: An optional state object that was passed to the current location.

Redux Toolkit

Redux Toolkit is a set of tools that helps simplify Redux development.

It is a library.

It includes utilities for creating and managing Redux stores, as well as for writing Redux actions and reducers.

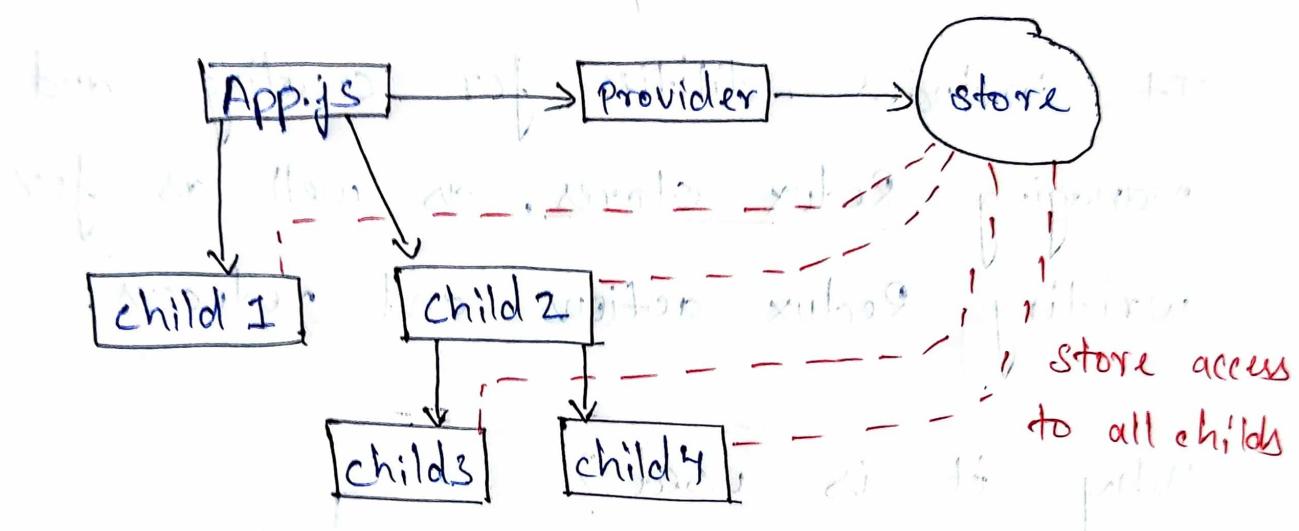
Why it is used?

Redux Toolkit makes it easier to write good Redux applications and speed up development, by baking in your recommended best practices, providing good default behaviours, catching mistakes, and allow you to write simpler code.

Its main purpose is to maintain and update data across your applications for multiple components to share, all while remaining independent of the components.

What is the Redux store?

The Redux store is the main, central bucket which stores all the states of an application.



Actions

The only way to change the state is to emit an action, which is an object describing what happened.

Reducers

Reducers take in two things: previous state and an action. Then they reduce it (return it to one entity) the new updated instance of state.

Slice

A slice is a portion of redux code that relates to a specific set of data and actions within the store's state.

useDispatch()

This hook returns a reference to the dispatch function from the Redux store. You may use it to dispatch actions as needed.

useSelector()

It allows you to extract data from the Redux store state, using a selector function.