REACT

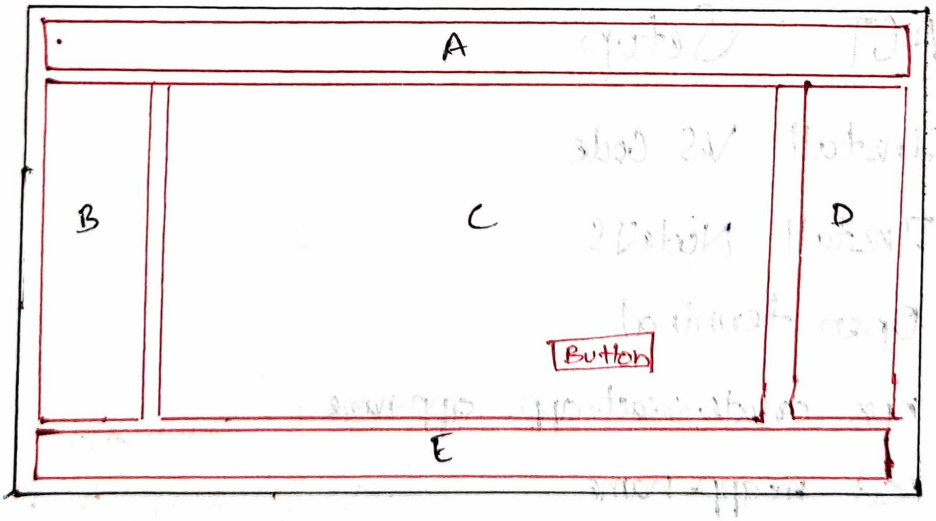
React is a JS liberary used to create UI

React is all about components.

It is a component based architechure.

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 imparative 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 autometically.

→ Imperative means line by line.

→ SPA approach (Single Page Application)

→ Faster development, etc.

* REACT Alternatives

→ 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

Folder →
v src
File → | index.js → It is the entry point of 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.

Code (component file)

```
import './item.css';
const name = "Rishabh";
function Item() {
    return <p className='rishabh'> Rishabh </p>
    return <p className='rishabh'> {name} </p>
}

export default Item;
```

here we use className because class is a reserved keyword in JS

we use {} when data changes dynamically.

App.js file (code)

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

function App() {
    return (
        <div>
            <Item></Item>
            <ItemDate></ItemDate>
        </div>
    );
}
export default App;
```

It is mandatory to have a parent, when you have multiple child.

Another component

* Use of Props

Props are arguements passed into REACT components.

Props stands for properties.

Code (component file)

```
function Item(props) {
    const itemName = "Rishabh" props.name1;
    return (<p className="Name">{itemName}</p>);
}
export ____;
```

Code (App.js file)

```
import _____;
function App() {
    return (
        <div>
            <Item name1 = "Rishabh"></Item>
        </div>
    );
}
```

=> Another way (App.js)

```
function App() {
    const response = [   <- array of objects
        {
            itemName : "Kushwaha";
        }
    ]
```

```
return (
    <div>
        <Item name1 = {response[0].itemName}></Item>
        - - - - - -
    </div>
);
}
```

Index of object

⇒ But what happens when we do this

```
<Item name1 = {response[0].itemName} I am Rishabh </Item>
```

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>
```

```
return (
    <div>
        <Item name1 = {response[0].itemName}></Item>
    </div>
);
}
```

Index of object

⇒ But what happens when we do this

```
<Item name1 = {response[0].itemName} I am Rishabh </Item>
```

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

Handeling 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 function even if we ~~no~~ did'nt clicked on button.

* 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("");
```

~~Existing~~ color is our current state

It is the function that is used to update our state

HW useState is a asynchronous hook, it will wait for the component to finish its cycle, re-render, and then it will update the state.

Q. Why const is used and still the value changes?

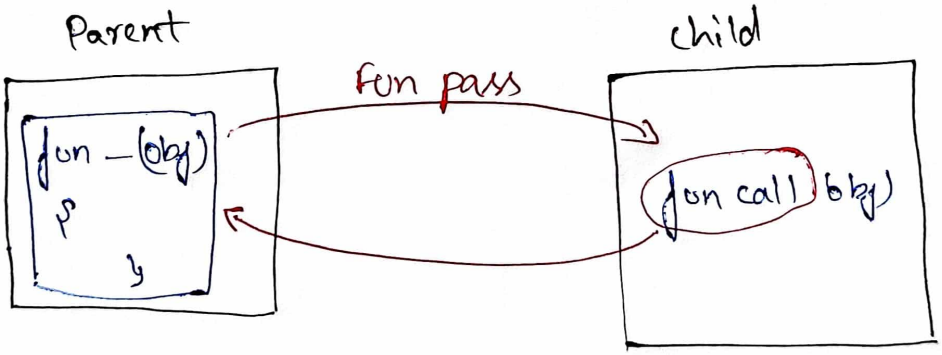
When a component is rendered, the function is executed again, creating a new 'color' variable, which has nothing to do with the previous variable.

* ~~onClick~~ 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 pbe passed from parent toe 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 effectss in your components.

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

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

Syntax-1

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

Every render

Syntax-2

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

First render

Syntax-3

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

Dependency

First render + when dependency changes

Syntax - 4

```
useEffect ( () => {
    // some code                    (A)
    return () => {                  (B)
    }
});
```

To handle unmounting of a component

Here (B) will execute before (A)

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

console.log(formData);

function changeHandler(event) {
    const {name, value, checked, type} = event.target
    setFormData( (prevFormData) => {
        return {
            ...prevFormData,
            [name]: type === "checkbox"
            ? checked : value
        };
    });
}

return (
    <div>
        <form>
```

name attribute is provided in every element

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

Here it is storing the previous state of the form with the help of ...spread operator.

```
<input
    type="text"
    placeholder="first name"
    onChange={changeHandler}
    name="firstName"
    value={formData.firstName}
/>
<br/>
<input
    type="text"
    placeholder="last name"
    onChange={changeHandler}
    name="lastName"
    value={formData.lastName}
/>

<br/>

<input
    type="email"
    placeholder="enter your email"
    onChange={changeHandler}
    name="email"
    value={formData.email}
/>
```

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

Controlled components

```
<br/>

<input
    type = "checkbox"
    onChange={ changeHandler }
    name = "isVisible"
    id = "isVisible"
    checked = {formData.isVisible}
/>
<label htmlFor="isVisible">
    Am I visible?
</label>
</form>
</div>
<br/>
```

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

// Controlled Components

// 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"}
        />
        <label htmlFor='Online-mode'>Online Mode</label>

        <input
            type="radio"
            onChange={changeHandler}
            name="mode"
            value = "Offline-mode"
            id = "Offline-mode"
            checked={formData.mode === "Offline-Mode"}
        />
        <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="scarpio">Scarpio</option>
    <option value="tharr">Tharr</option>
    </select>
    <button>Submit</button>
</form>
</div>
```

REACT ROUTER

It is a standard liberary 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 ~~an~~ About.

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

<NavLink to="/"> Home <NavLink>

By using above tag we can autometically add active class in the link in which we have clicked.

We can also use nesting in <Route> tag but by default it will not show the childs of the parent <Route> element.

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

Ex
```
<Routes>
    <Route path="/" element={<MainHeader/>}>
        <Route index element={<Home>}/>
        <Route path="/About" element={<Suppor/>}/>
    </Route>
</Routes>
```

Parent Route

It makes it default route

child route

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.

Parent Route

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

child Route

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

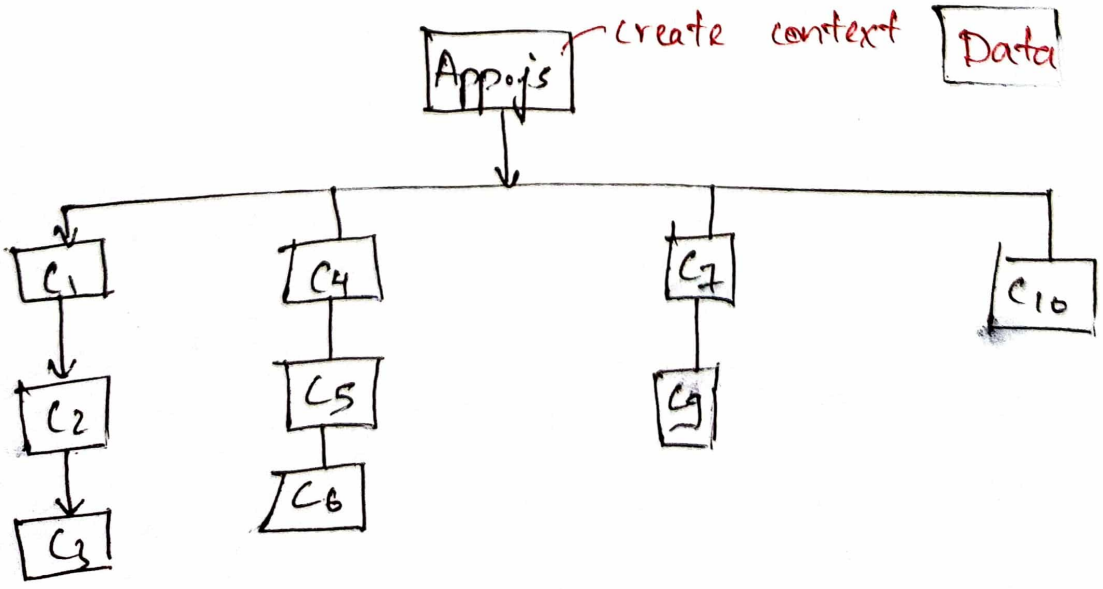
The React Context API is a way for a React app to effictively produce global variables that can be passed around.

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 Data context is created in App.js file and it can be accessed to any of its childs.

Rules to follow while using Context API

(i) Create context

(ii) Provide context

(iii) 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 returns 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 liberary.

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

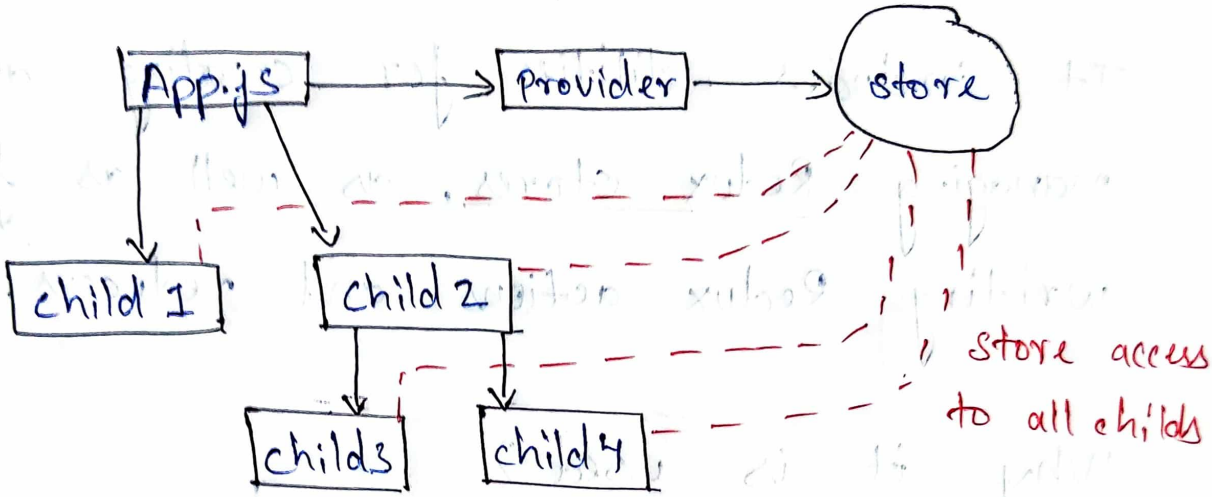
Why it is used ?

Redux Toolkit makes it easier to write good Redux applications and speed up development, by baking in our 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 sfate.

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.