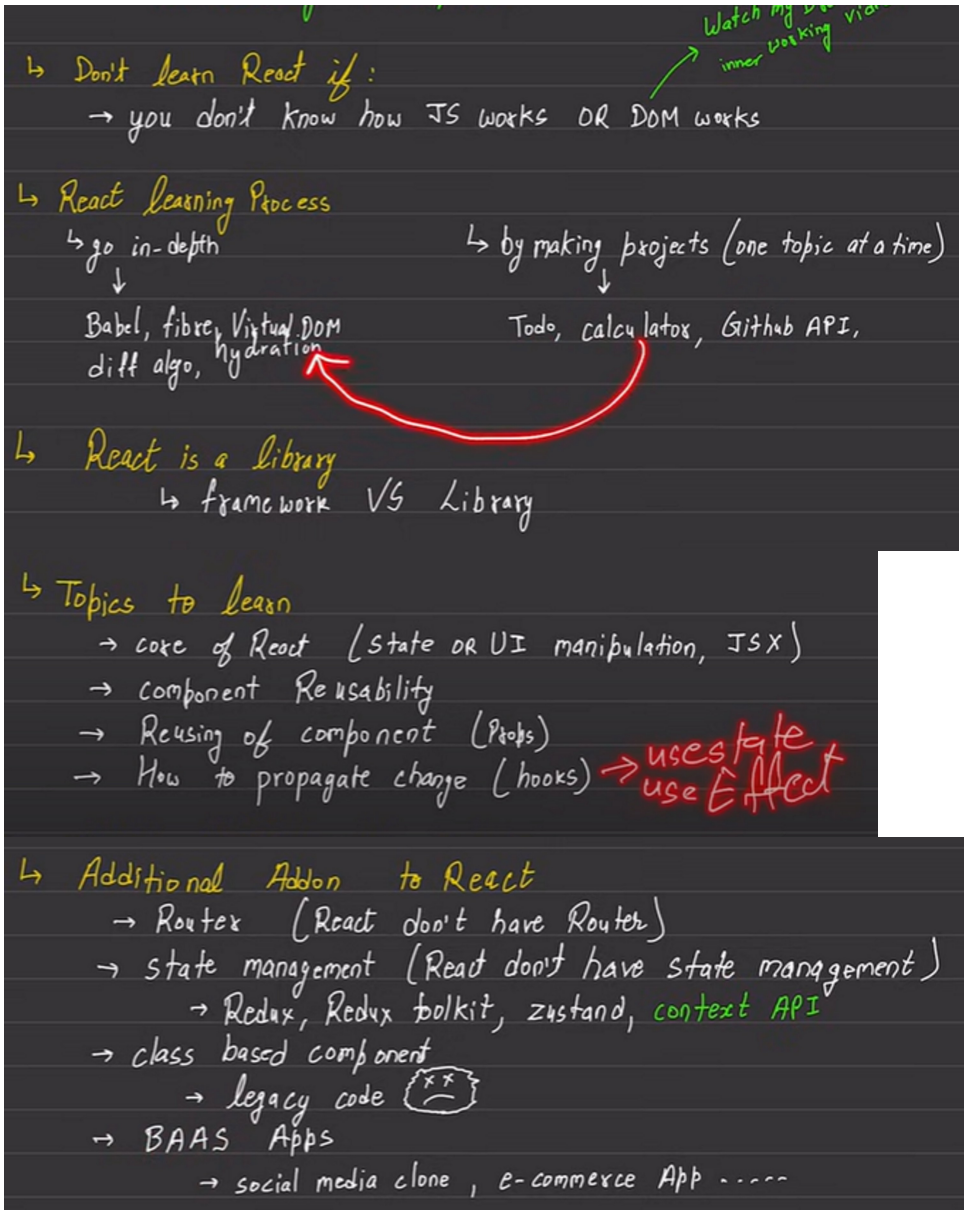
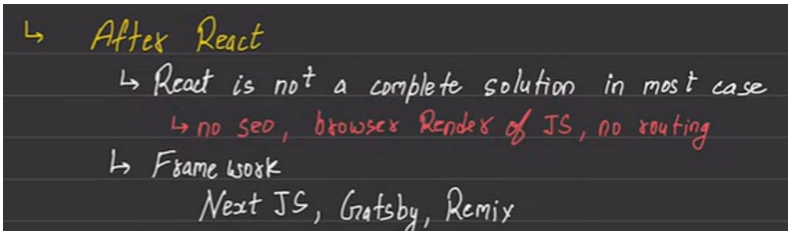
Chai aur React

18 June 2024 09:53 AM

Playlist-> Chai aur react | with projects	Code-> https://github.com/hiteshchoudhary/chai-aur-react

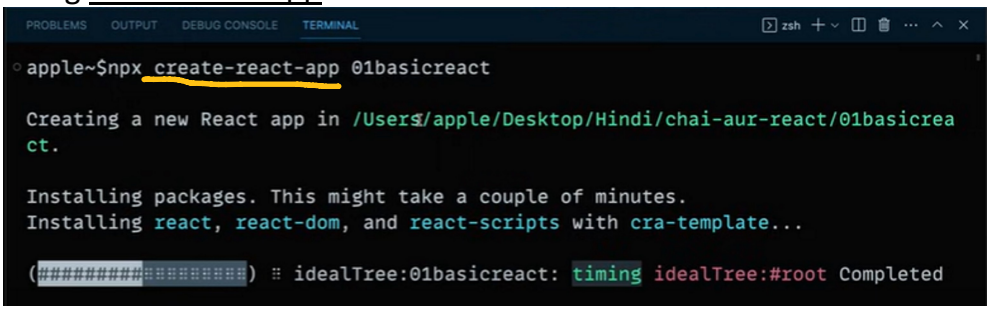
Lec-1 React JS Roadmap



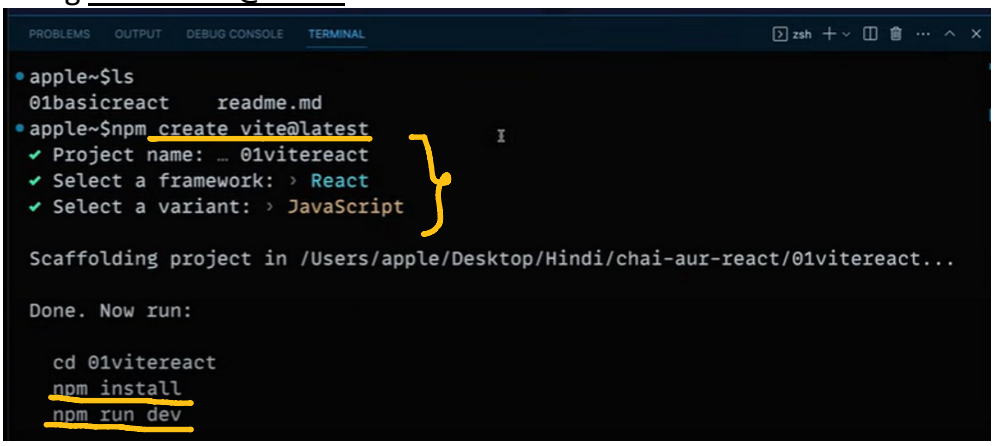


Lec-2 Creating React Projects

Using create-react-app



Using create vite@latest



Lec-3 Understanding React Flow and Structure

Using create-react-app

1. Js is automatically used as web scripts
2. Componenets names must start with capital letter
3. Componenets can be saved with extension- .jsx or .js (both okay)

Using create vite

1. Js tags are used in main.html
2. Componenets names must start with capital letter
3. Componenets must be saved with extension- .jsx

Lec-4 Creating our own React library and JSX

Rect elem is created , with parameters in format-> tagName , then all attributes as object , then the text for it, then the Js variables(props) (if any passed)

```
const reactElement = React.createElement(
    'a',
    {href: 'https://google.com',target: '_blank' },
    'click me to visit google'
)

ReactDOM.createRoot(document.getElementById('root')).
render(

    reactElement

)
```

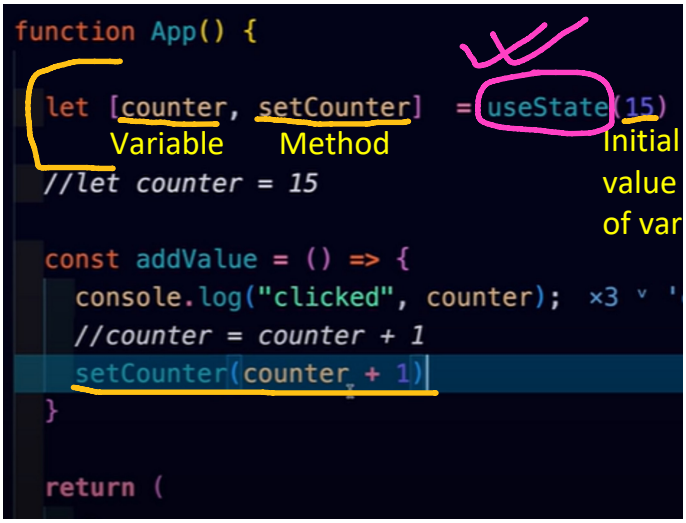
Js Variables can be injected in react using {}
But note- we use evaluated expressions in our{}, we can't write Js code in it (like if condn etc)

```
function App() {
  const username = "chai aur code"

  return (
    <>
    <Chai/>
    <h1>chai aur react {username}</h1>
    <p>test para</p>
    </>
  )
}

export default App
```

Lec-5 Why you need Hooks ? & project

UI is controlled by react, sochanging the values of any variable, doesn't gets changed on screen (though it gets changed actually for the variable)

Eg- UseState



Lec-6 Virtual DOM, Fibre and Reconcilation

A nice article-> [Must Read] https://github.com/acdlite/react-fiber-architecture
Some of it's IMP points ->

Introduction

React Fiber is an implementation of React's core algorithm. It is the culmination of over two years of research by the React team.

The goal of React Fiber is to increase its suitability for areas like animation, layout, and gestures. Its headline feature is **incremental rendering**: the ability to split rendering work into chunks and spread it out over multiple frames.

Other key features include the ability to pause, abort, or reuse work as new updates come in; the ability to assign priority to different types of updates; and new concurrency primitives.

What is reconciliation?

reconciliation
The algorithm React uses to diff one tree with another to determine which parts need to be changed.

update
A change in the data used to render a React app. Usually the result of `setState`. Eventually results in a re-render.

The central idea of React's API is to think of updates as if they cause the entire app to re-render. This allows the developer to reason declaratively, rather than worry about how to efficiently transition the app from any particular state to another (A to B, B to C, C to A, and so on).

Actually re-rendering the entire app on each change only works for the most trivial apps; in a real-world app, it's prohibitively costly in terms of performance. React has optimizations which create the appearance of whole app re-rendering while maintaining great performance. The bulk of these optimizations are part of a process called **reconciliation**.

Reconciliation is the algorithm behind what is popularly understood as the "virtual DOM." A high-level description goes something like this: when you render a React application, a tree of nodes that describes the app is generated and saved in memory. This tree is then flushed to the rendering environment — for example, in the case of a browser application, it's translated to a set of DOM operations. When the app is updated (usually via `setState`), a new tree is generated. The new tree is diffed with the previous tree to compute which operations are needed to update the rendered app.

Although Fiber is a ground-up rewrite of the reconciler, the high-level algorithm described in the React docs will be largely the same. The key points are:

- Different component types are assumed to generate substantially different trees. React will not attempt to diff them, but rather replace the old tree completely.
- Diffing of lists is performed using keys. Keys should be "stable, predictable, and unique."

IMP intv. Q-> Why shold we use keys, while using array elems/objects to bring in buttons/variables etc anywhere. ==> coz of the last point in above ss, ie- coz in fiber, we need to use keys to Inc performance of lists.

Reconciliation versus rendering

The DOM is just one of the rendering environments React can render to, the other major targets being native iOS and Android views via React Native. (This is why "virtual DOM" is a bit of a misnomer.)

The reason it can support so many targets is because React is designed so that reconciliation and rendering are separate phases. The reconciler does the work of computing which parts of a tree have changed; the renderer then uses that information to actually update the rendered app.

The key points are:

- In a UI, it's not necessary for every update to be applied immediately; in fact, doing so can be wasteful, causing frames to drop and degrading the user experience.
- Different types of updates have different priorities — an animation update needs to complete more quickly than, say, an update from a data store.
- A push-based approach requires the app (you, the programmer) to decide how to schedule work. A pull-based

say, an update from a data store.

- A push-based approach requires the app (you, the programmer) to decide how to schedule work. A pull-based approach allows the framework (React) to be smart and make those decisions for you.

What is a fiber?

We're about to discuss the heart of React Fiber's architecture. Fibers are a much lower-level abstraction than application developers typically think about. If you find yourself frustrated in your attempts to understand it, don't feel discouraged. Keep trying and it will eventually make sense. (When you do finally get it, please suggest how to improve this section.)

Here we go!

We've established that a primary goal of Fiber is to enable React to take advantage of scheduling. Specifically, we need to be able to

- pause work and come back to it later.
- assign priority to different types of work.
- reuse previously completed work.
- abort work if it's no longer needed.

In order to do any of this, we first need a way to break work down into units. In one sense, that's what a fiber is. A fiber represents a **unit of work**.

Lec-7 Tailwind & props in ReactJs

Configure tailwind (simillar to Bootstrap) using official doc

Props make component reusable, for eg- we make a ard to display info , but we need to use it multiple times, so we'll use props, to use diff data / content for each card efficiently

Actual way of using props

```
function Card(props) {
    console.log(props.username);
  return (
```

Shortcut way of using props

```
function Card({username}) {
    console.log(username);
  return (
```

In our App.jsx

```
return (
  <>
    <h1 className='bg-green-400 text-black p-4
    rounded-xl mb-4'>Tailwind test</h1>
    <Card username="chaiaurcode" />
    <Card username="hitesh"/>
  </>
)
```

Giving default value to props getting passed

```
function Card({username, btnText="visit me"})
```

Lec-8 React interview Quesn on counter


```
function App() {

  const [counter, setCounter]  = useState(15)

  //let counter = 15

  const addValue = () => {
    //counter = counter + 1
    setCounter(counter + 1)
    setCounter(counter + 1)
    setCounter(counter + 1)
    setCounter(counter + 1)


  }
```

Quesn-> How much does the value gets increased on a single click on "Add value" button ?
Ans-> Only +1 on a single click, coz fiber sends data in batches for updation and as all these are same commands so they are treated as a single command only.
Rather write like this, if want to inc multiple times->

```
const addValue = () => {
  //counter = counter + 1
  setCounter(prevCounter => prevCounter + 1)
  setCounter(prevCounter => prevCounter + 1 )
  setCounter(prevCounter => prevCounter + 1)
  setCounter(prevCounter => prevCounter + 1)

}
```

Lec-9 Building a react project Background Changer

As the colors need to be changed , we must save them in a variable and coz these changes also needs to be reflected in UI, so we need to use UseState for it.

```
function App() {
  const [color, setColor] = useState("olive")

  return (
    <div className="w-full h-screen duration-200"
```

```
<div className="fixed flex flex-wrap
justify-center bottom-12 inset-x-0 px-2">
  <div className="flex flex-wrap justify-center
  gap-3 shadow-lg bg-white px-3 py-2 rounded-3xl">
    <button
    onClick={() => setColor("red")}
    className="outline-none px-4 py-1
    rounded-full text-white shadow-lg"
    style={{backgroundColor: "red"}}
    >Red</button>
```

Lec-10 useEffect, useRef and useCallback with 1 project

1. useCallback: used for optimization it calls the function inside it when the dependencies are changed and returns a memorized function

```
const memoizedCallback = useCallback(
  () => {
    // function body
  },
  [/* dependency array */]
);
```

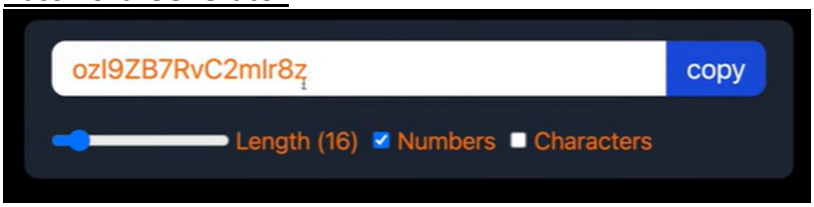
2. useEffect: runs the function inside it whenever the page renders first-time or dependencies are changed

```
useEffect(<function>, <dependency>)
```

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```

3. useRef : used to give reference of selected components in our page so that functions can be performed on referenced values

Password Generator



App.jsx

Make useStates for all parameters

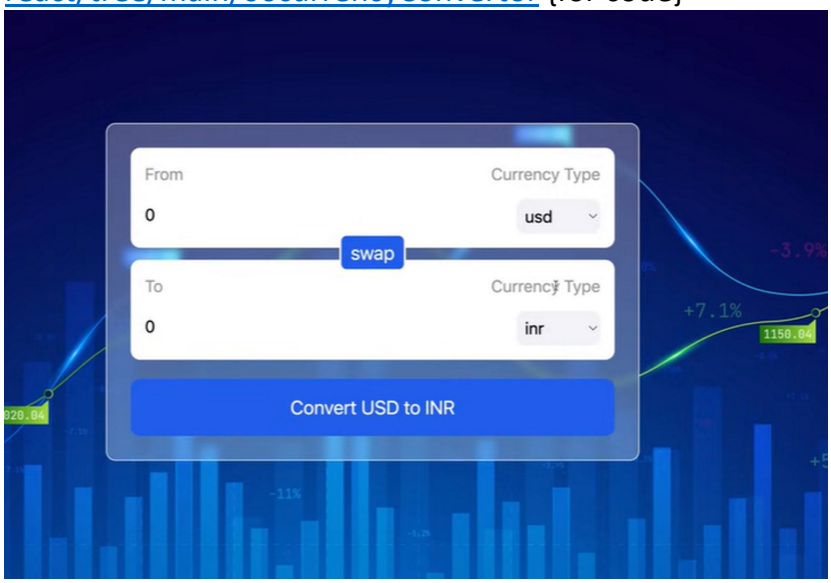
```
import { useState, useCallback, useEffect, useRef } from 'react'


function App() {
  const [length, setLength] = useState(8)
  const [numberAllowed, setNumberAllowed] = useState(false);
  const [charAllowed, setCharAllowed] = useState(false)
  const [password, setPassword] = useState("")

  //useRef hook
  const passwordRef = useRef(null)

  const passwordGenerator = useCallback(() => {
    let pass = ""
    let str = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
    if (numberAllowed) str += "0123456789"
    if (charAllowed) str += "!@#$%^&*-_+=[]{}~`"

    for (let i = 1; i <= length; i++) {
      let char = Math.floor(Math.random() * str.length + 1)
      pass += str.charAt(char)

    }

    setPassword(pass)


  }, [length, numberAllowed, charAllowed, setPassword])

  const copyPasswordToClipboard = useCallback(() => {
    passwordRef.current?.select();
    passwordRef.current?.setSelectionRange(0, 999);
    window.navigator.clipboard.writeText(password)
  }, [password])
```

```
    passwordRef.current?.setSelectionRange(0, 999);
    window.navigator.clipboard.writeText(password)
  }, [password])

  useEffect(() => {
    passwordGenerator()
  }, [length, numberAllowed, charAllowed, passwordGenerator])
  return (

    <div className="w-full max-w-md mx-auto shadow-md rounded-lg px-4 py-3 my-8 bg-gray-800 text-orange-500">
      <h1 className='text-white text-center my-3'>Password generator</h1>
    <div className="flex shadow rounded-lg overflow-hidden mb-4">
        <input
            type="text"
            value={password}
            className="outline-none w-full py-1 px-3"
            placeholder="Password"
            readOnly
            ref={passwordRef}
        />
        <button
        onClick={copyPasswordToClipboard}
        className='outline-none bg-blue-700 text-white px-3 py-0.5 shrink-0'
        >copy</button>
    </div>
    <div className='flex text-sm gap-x-2'>
      <div className='flex items-center gap-x-1'>
        <input
        type="range"
        min={6}
        max={100}
        value={length}
         className='cursor-pointer'
         onChange={(e) => {setLength(e.target.value)}}
          />
          <label>Length: {length}</label>
      </div>
      <div className="flex items-center gap-x-1">
      <input
          type="checkbox"
          defaultChecked={numberAllowed}
          id="numberInput"
          onChange={() => {
              setNumberAllowed((prev) => !prev);
          }}
      />
      <label htmlFor="numberInput">Numbers</label>
      </div>
       <div className="flex items-center gap-x-1">
           <input
               type="checkbox"
               defaultChecked={charAllowed}
               id="characterInput"
               onChange={() => {
                   setCharAllowed((prev) => !prev )
```

```
                id="characterInput"
                onChange={() => {
                    setCharAllowed((prev) => !prev )
                }}
            />
            <label htmlFor="characterInput">Characters</label>
        </div>
      </div>
    </div>

    )
  }

  export default App
```

Lec-11 Custom hooks in react | currency Project

Currency converter --> https://github.com/hiteshchoudhary/chai-aur-react/tree/main/06currencyConvertor {for code}



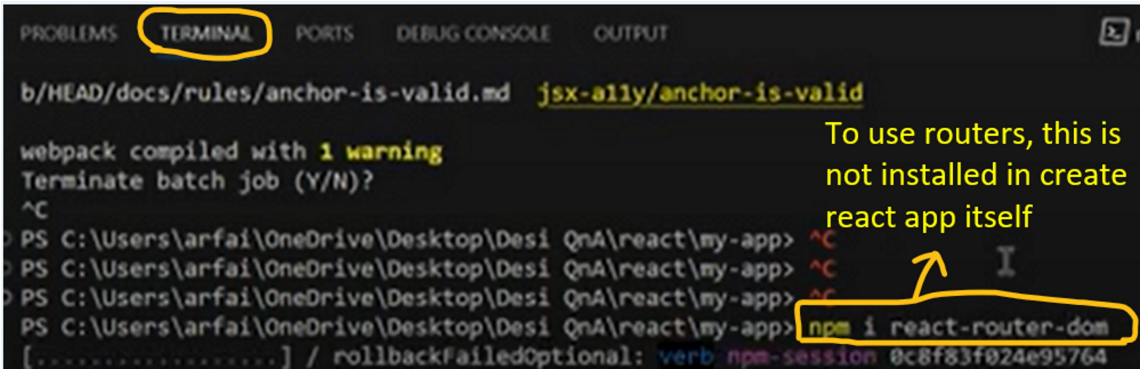
Fetching of data thru API's is done like this->

```
function useCurrencyInfo(currency){
    const [data, setData] = useState({})
    useEffect(() => {
        fetch(`https://cdn.jsdelivr.net/gh/fawazahmed0/currency-api@1/latest/currencies/${currency}.json`)
        .then((res) => res.json())
        .then((res) => setData(res[currency]))
        console.log(data);
    }, [currency])
    console.log(data);
    return data
}
```

Custom hook

Lec-12 React router crash course

If you are looking at old react github code they will have implemented routers using some <Switch/> dont get confused, the switch method is now deprecated



Anchor tag or <a> tag is not used in React as it refreshes the whole page which is not the concept of react, that's why Link tag is used in react which is imported from react-router-dom

```
<a className="nav-link active" aria-current="page" href="#">Home</a>
</li>
<li className="nav-item">
<Link className="nav-link" to='/About'>About</Link>
</li>
```

Layout.jsx

```jsx
import React from 'react'
import Header from './components/Header/Header'
import Footer from './components/Footer/Footer'
import { Outlet } from 'react-router-dom'

function Layout() {
  return (
    <>
    <Header/>
    <Outlet />
    <Footer />
    </>
  )
}

export default Layout
```

Main.jsx

```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'
import { Route, RouterProvider, createBrowserRouter, createRoutesFromElements } from 'react-router-dom'
import Layout from './Layout.jsx'
import Home from './components/Home/Home.jsx'
import About from './components/About/About.jsx'
import Contact from './components/Contact/Contact.jsx'
import User from './components/User/User.jsx'
import Github, { githubInfoLoader } from './components/Github/Github.jsx'
```

Method-i

```jsx
// const router = createBrowserRouter([
//   {
//     path: '/',
//     element: <Layout/>,
//     children: [
//       {
//         path: "",
//         element: <Home />
//       },
//       {
//         path: "about",
//         element: <About />
//       },
//       {
//         path: "contact",
//         element: <Contact />
//       }
//     ]
//   }
// ])
```

Method-ii

```
const router = createBrowserRouter(
  createRoutesFromElements(
    <Route path='/' element={<Layout />}>
      <Route path='' element={<Home />} />
      <Route path='about' element={<About />} />
      <Route path='contact' element={<Contact />} />
      <Route path='user/:userid' element={<User />} />
      <Route
        loader={githubInfoLoader}
        path='github'
        element={<Github />}
        />
    </Route>
  )
)

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>,
)
```

A loader in react-router is a function that is used to fetch data for a route before it is rendered

Header.jsx

```
import React from 'react'
import {Link, NavLink} from 'react-router-dom'

export default function Header() {
    return (
        <header className="shadow sticky z-50 top-0">
            <nav className="bg-white border-gray-200 px-4 lg:px-6 py-2.5">
                <div className="flex flex-wrap justify-between items-center mx-auto max-w-screen-xl">
                    <Link to="/" className="flex items-center">
                        <img
                            src="https://alexharkness.com/wp-content/uploads/2020/06/logo-2.png"
                            className="mr-3 h-12"
                            alt="Logo"
                        />
                    </Link>
```

.

.

```
                    <div
                        className="hidden justify-between items-center w-full lg:flex lg:w-auto lg:order-1"
                        id="mobile-menu-2"
                    >
                        <ul className="flex flex-col mt-4 font-medium lg:flex-row lg:space-x-8 lg:mt-0">
                            <li>
                                <NavLink
                                to="/"
                                    className={({isActive}) =>
                                        `block py-2 pr-4 pl-3 duration-200 ${isActive ? "text-orange-700" : "te
                                    }
```

To change the text colour

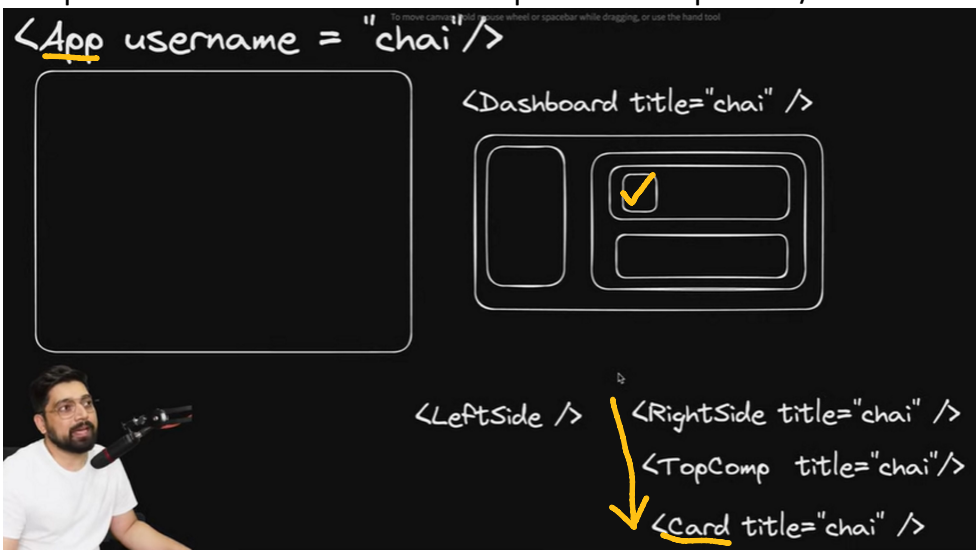
```
38                          className={({isActive}) =>
39                              `block py-2 pr-4 pl-3 duration-200 ${isActive ? "text-orange-700" : "te
40                              }
41                      >
42                          Home
43                      </NavLink>
44                  </li>
45                  <li>
46                      <NavLink
47                      to="/about"
48                          className={({isActive}) =>
49                              `block py-2 pr-4 pl-3 duration-200 ${isActive ? "text-orange-700" : "te
50                          }
51                      >
52                          About
53                      </NavLink>
54                  </li>
```

To change the text colour when section is active

Simillarly, footer, home, about, contact sections.....

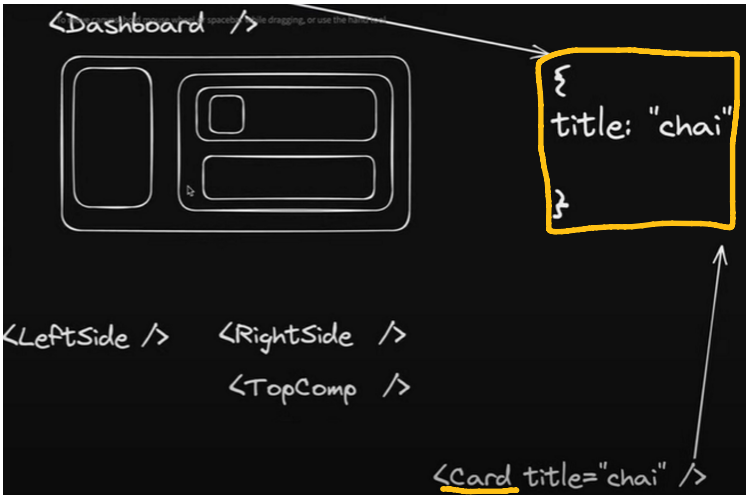
Lec-13 Context API crash course with 2 projects

Props in props are passed, to pass data in components inside components, by the grandparent, parent components for the use of child component. This process/method is called Props Drilling



In this method, all the components which do not require the data/props also need to handle data, just for the sake of passing the data to inner component. This is very inefficient way.

So it is better if we make a global file of the data, and our innermost component which requires it directly accesses it from there, reducing the load of other components.



To solve this issue we use concept of State Management ->
State represents the value of a dynamic properties of a React component at a given instance.
State management refers to the management of these states of user interface controls, where the state of one UI control depends on the state of other UI controls

Context API(only for react) , Redux, react-redux, redux toolkit (RTK) are some of the solutions for sate management.

Using Context API ->

Main.jsx

```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Method-i

App.jsx

```jsx
import './App.css'
import Login from './components/Login'
import Profile from './components/Profile'
import UserContextProvider from './context/UserContextProvider'

function App() {


  return (
    <UserContextProvider>
      <h1>React with Chai and share is important</h1>
```

```
  return (
    <UserContextProvider>
      <h1>React with Chai and share is important</h1>
      <Login />
      <Profile />
    </UserContextProvider>
  )
}

export default App
```

Context/UserContext.js

```
import React from 'react'

const UserContext = React.createContext()

export default UserContext;
```

Context/UserContextProvider.jsx

```
import React from "react";
import UserContext from "./UserContext";

const UserContextProvider = ({children}) => {
    const [user, setUser] = React.useState(null)
    return(
        <UserContext.Provider value={{user, setUser}}>
        {children}
        </UserContext.Provider>
    )
}

export default UserContextProvider
```

Key Diff

Components/Login.jsx

```
1   import React, {useState, useContext} from 'react'
2   import UserContext from '../context/UserContext'
3
4   function Login() {
5       const [username, setUsername] = useState('')
6       const [password, setPassword] = useState('')
7
8       const {setUser} = useContext(UserContext)
9
10      const handleSubmit = (e) => {
11          e.preventDefault()
12          setUser({username, password})
13      }
14      return (
15        <div>
16            <h2>Login</h2>
17            <input type='text'
18            value={username}
19            onChange={(e) => setUsername(e.target.value) }
```

```
            <input type='text'
            value={username}
            onChange={(e) => setUsername(e.target.value) }
            placeholder='username'  />
            {" "}
            <input type='text'
            value={password}
            onChange={(e) => setPassword(e.target.value) }
            placeholder='password'  />
            <button onClick={handleSubmit}>Submit</button>
        </div>
      )
    }

    export default Login
```

Components/Profile.jsx

```
import React, {useContext} from 'react'
import UserContext from '../context/UserContext'

function Profile() {
    const {user} = useContext(UserContext)

    if (!user) return <div>please login</div>

    return <div>Welcome {user.username}</div>
}

export default Profile
```

Method-ii (example used is Theme Switcher)

App.jsx

```
import { useEffect, useState } from 'react'
import './App.css'
import { ThemeProvider } from './contexts/theme'
import ThemeBtn from './components/ThemeBtn'
import Card from './components/Card'

function App() {
  const [themeMode, setThemeMode] = useState("light")

  const lightTheme = () => {
    setThemeMode("light")
  }

  const darkTheme = () => {
    setThemeMode("dark")
  }

  // actual change in theme

  useEffect(() => {
    document.querySelector('html').classList.remove("light", "dark")
    document.querySelector('html').classList.add(themeMode)
```

Declare where needed

```
21    useEffect(() => {
22      document.querySelector('html').classList.remove("light", "dark")
23      document.querySelector('html').classList.add(themeMode)
24    }, [themeMode])
27    return (
28      <ThemeProvider value={{themeMode, lightTheme, darkTheme}}>
29        <div className="flex flex-wrap min-h-screen items-center">
30            <div className="w-full">
31                <div className="w-full max-w-sm mx-auto flex justify-end mb-4">
32                    <ThemeBtn />
33                </div>
34
35                <div className="w-full max-w-sm mx-auto">
36                    <Card />
37                </div>
38            </div>
39        </div>
40      </ThemeProvider>
41    )
42  }
43
44  export default App
```

Contexts/Theme.js

```
1   import { createContext, useContext } from "react";
2
3   export const ThemeContext = createContext({
4       themeMode: "light",
5       darkTheme: () => {},
6       lightTheme: () => {},
7   })
8
9   export const ThemeProvider = ThemeContext.Provider
10  Custom hook
11  export default function useTheme(){
12      return useContext(ThemeContext)
13  }
```

Key Diff

Components/ThemeBtn.jsx

```
1   import React from 'react'
2   import useTheme from '../contexts/theme';
3
4   export default function ThemeBtn() {
5
6       const {themeMode, lightTheme, darkTheme} = useTheme()
7       const onChangeBtn = (e) => {
8           const darkModeStatus = e.currentTarget.checked
9           if (darkModeStatus) {
10              darkTheme()
11          } else {
12              lightTheme()
13          }
14      }
15      return (
16          <label className="relative inline-flex items-center cursor-pointer">
17              <input
18                  type="checkbox"
19                  value=""
20                  className="sr-only peer"
21                  onChange={onChangeBtn}
```

```
                value=""
                className="sr-only peer"
                onChange={onChangeBtn}
                checked={themeMode=== "dark"}
            />
            <div className="w-11 h-6 bg-gray-200 peer-focus:outline-none peer-focus:ring-4 peer-focus:ring-blue-300 dark:peer-focus:ring-blue-800 ro
            <span className="ml-3 text-sm font-medium text-gray-900">Toggle Theme</span>
        </label>
    );
}
```

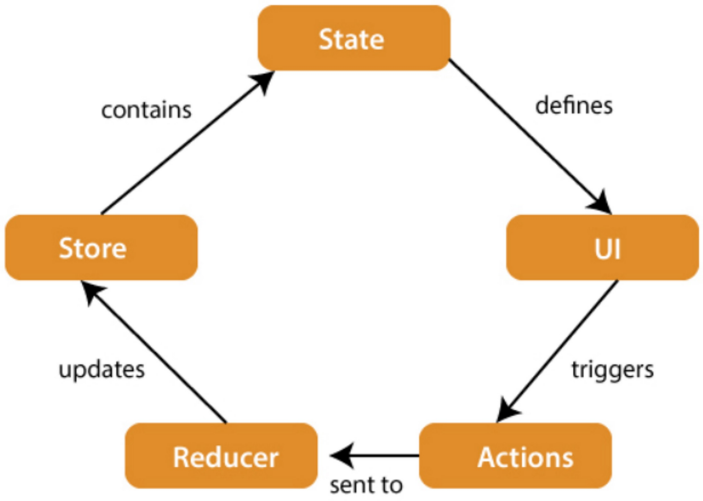
Lec-14 Context api with local storage | project-> ToDo

Not done till now

Lec-15 Redux toolkit crash course

Redux is an open-source JavaScript library for managing and centralizing application state.
React-Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch Actions to the Store to update data.
Redux was inspired by Flux. Redux studied the Flux architecture and omitted unnecessary complexity.

Redux Architecture (Data flow)



The components of Redux architecture are explained below.

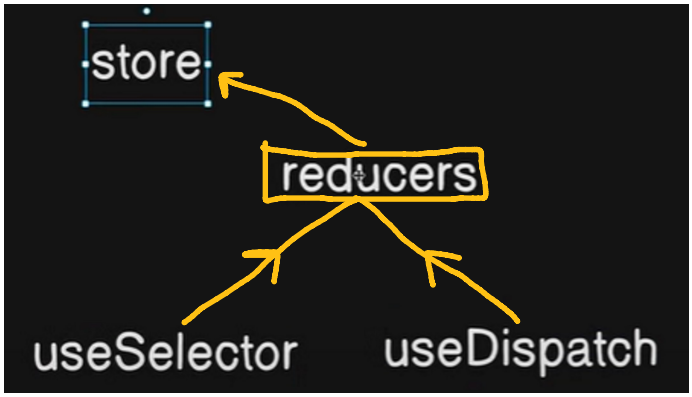
STORE: A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.

ACTION: Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.

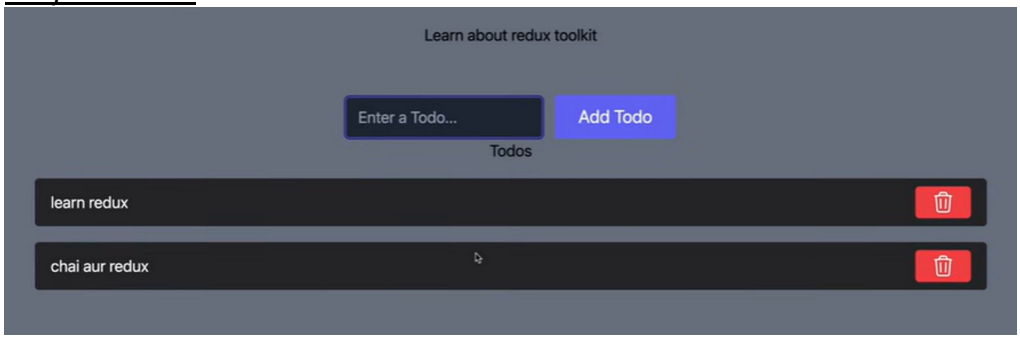
REDUCER: Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.
You use the action.type to determine what kind of action is being performed, and you can also access the

REDUCER: Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.

You use the action.type to determine what kind of action is being performed, and you can also access the payload or other properties of the action object to make decisions on how to update the state.



Project- ToDo



src/store.js (Always a single store is made)

```
1  import {configureStore} from '@reduxjs/toolkit';
2  import todoReducer from '../features/todo/todoSlice';
3
4  export const store = configureStore({
5      reducer: todoReducer
6  })
```

src/features/todo/todoSlice.js

```
1   import {createSlice, nanoid } from '@reduxjs/toolkit';
2
3   const initialState = {
4       todos: [{id: 1, text: "Hello world"}]
5   }
6
7
8
9   export const todoSlice = createSlice({
10      name: 'todo',
11      initialState,
12      reducers: {
13          addTodo: (state, action) => {
14              const todo = {
15                  id: nanoid(),
16                  text: action.payload
```

```
            const todo = {
                id: nanoid(),
                text: action.payload
            }
            state.todos.push(todo)
        },
        removeTodo: (state, action) => {
            state.todos = state.todos.filter((todo) => todo.id !== action.payload )
        },
    }
})

export const {addTodo, removeTodo} = todoSlice.actions

export default todoSlice.reducer
```

src/components/AddTodo.jsx

```
import React, {useState} from 'react'
import {useDispatch} from 'react-redux'
import {addTodo} from '../features/todo/todoSlice'

function AddTodo() {

    const [input, setInput] = useState('')
    const dispatch = useDispatch()

    const addTodoHandler = (e) => {
        e.preventDefault()
        dispatch(addTodo(input))
        setInput('')
    }

  return (
    <form onSubmit={addTodoHandler} className="space-x-3 mt-12">
      <input
        type="text"
        className="bg-gray-800 rounded border border-gray-700 focus:border-indigo-500 focus:ring-2 focus:ring-indigo-900 tex
        placeholder="Enter a Todo..."
        value={input}
        onChange={(e) => setInput(e.target.value)}
      />
      <button
        type="submit"
        className="text-white bg-indigo-500 border-0 py-2 px-6 focus:outline-none hover:bg-indigo-600 rounded text-lg"
      >
        Add Todo
      </button>
    </form>
```

src/components/Todos.jsx

```jsx
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import {removeTodo} from '../features/todo/todoSlice'

function Todos() {
    const todos = useSelector(state => state.todos)
    const dispatch = useDispatch()

  return (
    <>
    <div>Todos</div>
    <ul className="list-none">
        {todos.map((todo) => (
          <li
            className="mt-4 flex justify-between items-center bg-zinc-800 px-4 py-2 rounded"
            key={todo.id}
          >
            <div className='text-white'>{todo.text}</div>
            <button
             onClick={() => dispatch(removeTodo(todo.id))}
              className="text-white bg-red-500 border-0 py-1 px-4 focus:outline-none hover:bg-red-600 rounded text-md"
            >
              <svg
                xmlns="http://www.w3.org/2000/svg"
```

Lec-16

- Lec-16 -> till end : pending… (it's advance material, till here is enough to start with projects)