#Learnings:

1:

```
<!-- these cdn links import all react methods and components and features -->
<!-- first is core react where all main react funtionalities are included -->
<script
  crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"
></script>
<!-- next is react dom which is used to build web version application (there are
others as well like mobile. 3d-react etc.) -->
<script
  crossorigin
  src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
></script>



React.createElement does not create HTML element it creates object

  const parent = React.createElement("div", { id: "parent" }, [
      React.createElement("div", { id: "child1" }, [
        React.createElement("h1", {}, "I am H1 Tag"),
        React.createElement("h2", {}, "I am H2 Tag"),
      ]),
      React.createElement("div", { id: "child2" }, [
        React.createElement("h3", {}, "I am H3 Tag"),
        React.createElement("h4", {}, "I am H4 Tag"),
      ]),
    ]);

const root = ReactDOM.createRoot(document.getElementById("root"));

// render is creating element by taking the object and converting it into heading
tag and modify dom tree
root.render(parent);

// now here render will replace current dom tree inside root, so current structure
in root will be replaced by parent
// althogh it can only modify root and does not affect outside root.
```

2:

NPM and Folder Files:

```
  npm: package manager
  npm init: start project

  package.json: configuration for npm

  bundler: bundles / packs your app to shift to production (eg. webpack, parcel)

  devDependancies: for development purpose (npm i -D package-name)

  normal dependancies: can be used for prod

  ^: automatically upgrades minor versions 2.7.4 -> (recommanded)

  ~: automatically upgrades major versions 2.7.4 -> 2.8

  package-lock.json:
  - locks record keeps track of exact version of package that is being installed
to ensure consistancy across all environments for all dependancies and
subdependancies.
  - It contains integrity hash to verify the current version machine should match
to the deployed version of prod.

  node_modules:
  - When all modules that are installed through npm, it fetched all codes and
dependancies of each library into our system.
  - So this collection is all dependancies of libraries (Transitive Dependancies)
  - Every Package in node_module has its own package.json

  npx parcel index.html : hosts on localhost:1234
  - npm : install the package
  - npx : execute the package
  - so basically parcel goes to source index, builds the development build, and
hosts.

  Install React:
  - npm install react
  - npm install react-dom
  - add type="module" to script tag and import react and react-dom/client in
app.js

  Parcel:
  - uses file-watching algorithm (in c++)
  - caches files and gives faster builds (in parcel- cache folder)
  - image optimization
  - for prod build minify files, bundle them, compress them.
  - uses consistant hasing
  - code splitting
  - uses differencial bundling (supports old browsers)
  - diagnostics and error suggestions
  - hosts on HTTPS (using --https)
  - can start in lazy mode (using --lazy)
  - Tree Shaking Algorithm to remove unused code
  - different bundles for dev and prod (npx parcel build index.html for prod)

  Dist:
  when you execute parcel, ut will bundle,minify and put in dist folder. When you change something, it will update dist and parcel-cache and show output from them.
  For production build if you have 10,20 html, css, js files, parcel will convert all of them into single html, css, js files and put it in dist.

  Support Old Browser Versions:
  - use browserlist dependancy in package.json (refer browserlist.dev)
  - This supports 80% users
  - "browserlist":["last 2 versions"]
```

3:

JSX:

```

  To redue complexity of reactelements to build HTML, JSX was created
  - JSX is not HTML in Javascript.
  - JSX is HTML-like syntax or XML-like.

  - so there is no difference between these as both return JS object
  const heading = React.createElement("h1", {},"Hello World") and
  const heading = (
    <h1 id="heading" className="head" tabIndex="2">
      Hello World
    </h1>
  );

  - JSX is not pure javascript so JS engine or browser wont understand it directly.
  - jsx => babel transpiles to react.createElement => js object => rendered to HTML element

  Babel:
  - javascript compiler and transpiler
  - takes JSX => converts to code that JS engine Understands or React understands

  - Now JSX is not HTML either as it uses className and tabIndex like camelCase.

  - Write Multiline JSX with () simple bracket.

  React Component:
  - class based component (old)
  - functional component (new)

  React Functional Component
  - just normal javascript function
  - Start with Capital Letter (must)
  - returns JSX code / React Element
  - can use arrow function or even function keyword like normal functions
  - *Cannot render it directly its component not element
  - Component Composition: using component inside another component
  - with <Componnet />
  - with <Component></Component>
  - with calling component function inside {}: {Description()}

  You can run any js code in {} inside JSX, any varibles, jsx elements, even logs.
  <h2>The value of number is {number}</h2>

  JSX even sanitizes data and avoids cross site scripting
```

4:

Config Driven UI:

```

  we can control UI based on Data like location, user data. ie showing different
list for different locations or no list for certain locations.
```

Props:

```

  properties that you pass to the component.
  just normal arguments to a function.
  like when you call the component <componentName arg1="akdjfn" arg2="avsdfjh" />
  arg1 and arg2 are props.
  react will take these properties and will wrap in object called props
  you can extract in component defination and use them.
  const Component = (props)=>{
    return (
      <h1>{props.arg1}</h1>
    )
  }
  we can pass any number of props
  instead of (props)=> your can also destructure them ({arg1, arg2})=>

  When you have container for cards, you need to loop over the data with inputs
and call the card component.
  Instead of array, react suggests to use map, filter, reduce, as it promotes
functional programming.

  ALWAYS GIVE UNIQUE KEY to each CARD/component template.

  Why unique key?
  - If you dont give unique keys to each component template, react will rerender
all cards/tmeplates everytime new one addded/removed, cz as there are no unique
ids it doesn not know which card has just entered/removed. So give unique ids, so
```

```
react will render only that card.

  Some put index as key, but react does not recommand that.
  - Potential issues with rendering performance, consistancy, espescially with
dynamic lists.
  - When items are added / removed generally react depends on unique keys to
identify components, now using the index as key would modify this order.
  - if list of items get reordered react might reuse wrong component as index is
same.
  - react uses unique keys to preserve states, if items get reordered, states can
get mixed up with index as key.
```

Folder Structure:

```

  - React does not recommand any specific folder structure.
  - popular approaches can be
    - according to features: common files in 1 folder and each file saperated by
feature.
    - grouping by file type: api files in 1 folder, css in one folder, components
in 1 folder.
  - avoid too much nesting
  - dont overthink, restructure later if required
  - keep file name same as ComponentName.
  - same people keep it jsx or tsx.
  - for simlicity, keep saperate files for header, body, apis, for readablity of
other developers.
  - whenere you have any mock data or hardcoded data keep it in saperate files.
For constants use constant.js or utils.js
```

Import Export:

```

  2 Types:
  1. Single File import export- Default export (for components)
    - when we export components in file.
    - const ComponentName = () ={}
    - export default ComponentName;
    - import ComponentName from 'path'

  2. Multiple Exports in one file - Named Export (for saperate constants or
elements )
    - generally used to export each constant in util.js or some elements
    - export const constantName = 'sjfks';
    - import {constantName} from 'path'
    - HERE {} IS IMPORTANT FOR NAMED EXPORTS

  - you can have both a default export and named exports in the same module in
JavaScript/React.
```

5:

HOOKS:

```
  Now suppose you want to filter data on click of button, so apply  filter function on our data and
  and call it with onclick event on that button. But this would not reflect on UI.
  This is where react comes if your data changes, DOM should change, cz react is better at DOM manipulation.

  State Variable:
  A react hook is just a normal javascript function provided by react with some special features.

  IMP HOOKS:
  1. useState()
  2. useEffect()

  80% time you will use useState() and 20% useEffect()

  You need to import them like Named Import.

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

useState() Hook:

```
  - Maintains state of variable
  - scope is inside the component

  - const [varName, setVarName] = useState(defaultState ie [], true)
  - const [hotelList, setHotelList] = useState([]);
  - varName: variableName
  - setVarName: you cannot modify varName directly, you need setVarName(value) to modify it.
  - useState([]) use state with default state
  - useState() returns and array thats why we are destructuring it with [varName, setVarName].
```

IMP:

```
  - This powerful hook keeps UI insync with that variable.
  - As soon as hotelList changes with setHotelList(newList), it will automatically refresh our component, this is called render.
  - Whenever state variable changes, react rerenders the component.
  - react will make DOM operations superfast.
```

React Reconciliation Algorithm / React Fiber:

Virtual DOM:

```
  Suppose we have container DOM which has 5 cards. Now the UI will change from 5 cards to 3 cards.
  React will create a virtual DOM of it. Its not actual DOM but representation of actual DOM, basically a react element.

  remember when you printed any compoenent, it gave an object?
  This is basically an react element.

  This React Virtual DOM is react element / JS object.
```

Diff Algorithm:

```
  - This algorithm is used by react to find out the difference between two virtual doms.
  - So it finds out changes between updated virtual dom vs previous virtual dom.
  - ** Now this difference also finds changes and updates actual DOM on each render cycle.
```

React Fiber:

```
  - A new algorithm came in React 16, the new way to update the DOM - react Fiber.
  - So similar to git diff, as it compares two files, this algorithm compares two objects.
  - If anything has changed, then it will update DOM.

  - Now comparing HTML nodes / elements is tough, but comparing objects is easy as javascript is fast with objects
  - So it keeps track of all HTML code as virtual DOM like object representation.

  Now when we click filter button, a new object is formed.
  It compares current and previous objects then it actually updates DOM.
```

Increamental Rendering:

```
  - Ability to split rendering work into chunks and spread out into multiple
frames.
```

WHY REACT is FAST?

```
  Its doing efficient DOM Manipulation cz it has virtual DOM.
  Now this concept was not new, but react took this and built core algo on top of
it, and made it snappy
  by comparing 2 virtual DOMs and updating the actual DOM.

  So as soon as you call setHotelList, it starts its reconcilliation algorithm,
and starts rerendering your page.
  Thats why you need saperate call function to update state, so when you call it
react will find the div and update UI.
```

6:

Fetching data from API:

```
  2 ways to show:

  1. As soon as our Page loads --> make API call --> wait for page around 500ms --
> render whole page

  2. As soon as our Page loads --> we will render UI with skalaton (Shimmer UI)-->
make API call --> rerender app with new Data

  In react we will always be using 2nd approach, why? react has one of the fastest
render cycle speed.
```

useEffect() Hook:

```
  Takes 2 arguments,
  - call back function
  - dependancy array

  the call back function will be called after your component is rendered ie after
finishing render cycle
  If you want to do something after rendering the component, use it with
useEffect.

  useEffect(()=> {console.log("useeffect call")}, [])
  - Empty Dependency Array ([]): The effect will run only once after the initial render.

  - No Dependency Array: The effect will run after every render (including on every state or prop change).

  - Dependencies in the Array: The effect will run only when the specified dependencies change.
      - eg
      const [count, setCount] = useState(0);
      useEffect(() => {
          console.log(`Effect runs when count changes: count is ${count}`);
      }, [count]); // Effect runs only when 'count' changes

  - the rest of the code first runs then after rendering is completed, callback inside useeffect will run
```

Conditional Rendering:

```
  - rendering based on certain condition
  - if (loading) {return <ShimmerUI />}
```

Why State Variable is used in first place?

```
  When you change any normal varible, react wont know the variable value has been changed.
  because React doesn't track regular variables.  and it wont upate UI accrodingly.
  But with state variable, when you modify it with setVaribleName, react will rerender that component and refresh specific component that will renrender DOM.
  YES IT WILL RE-RENDER WHOLE COMPONENT THAT STATE VARIABLE IS IN, ie REACT TRIGGERS RECONSCILLIATION CYCLE.
```

but isnt it expensive to reload everything in component?

```
  It may sound expensive to re-render the whole component, but React uses an efficient reconciliation algorithm that minimizes performance costs.

    - Virtual DOM: React creates a virtual representation of the DOM (Virtual DOM). When the state changes, React compares the new virtual DOM with the previous version using a process called diffing.

    - Efficient Updates: React doesn't replace the entire DOM. Instead, it updates only the parts of the actual DOM that have changed. This makes updates faster and
```

```
more efficient because React avoids unnecessary modifications to the DOM.

  So, while React triggers a re-render for the entire component where the state
variable is used, React only updates the actual DOM where necessary. The virtual
DOM comparison process ensures that only the specific parts that have changed are
modified in the real DOM.
```

How const changes state varibles?

```
  Now, suppose, we have const [btnName, setBtnName] = useState("login")
  and onclick of button, we want to change btnName to logout. with
setBtnName("logout")

  But how are we modifying const variable?

  actually react is inserting new value in useState and rerendering component, it
means its all new varible is being created,
  but as whole compoenent is recreated const doesnt give error, so whole new
variable is created with updated value.
```

Function Reference Issue:

```
  suppose you are toggling this button with onclick on toggleBtn function
  <button className='login-btn' onClick={toggleBtn()} >{loginBtn}</button>

  But this will give error: Too many re-renders. React Limits number of renders to
prevent infinite loop.
  Why?

  cz inside toggleBtn we are using setBtnName(value), and if we call toggleBtn()
immediately during the render phase,
  it will cause state to update striggering rerender, which will again call
toggleBtn() thus entering in infinite loop

  The solution is to pass function reference instead of calling the function.
  so onClick={toggleBtn}
```

7:

Hooks Tips:

```
  - Try to call hooks on top of components inside them.
  - and it doesnt make any sense using hooks outside compoenent, cz they are part
of compoenet.
```

```
  - Dont User State Variables inside if else conditions, or for loops, or even
functions.
```

Router Pages:

```

  - use react-router-dom
  - import { createBrowserRouter} from "react-router-dom";
  - correct way to handle routings
  - const appRouter = createBrowserRouter([
    {
    path:"/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
        {
            path:"/",
            element: <Body />
        },
        {
            path:"/restaurant/:resId",
            element: <RestaurantPage />
        }
      ]
    }])
  - render app router instead of applayout
  - root.render(<RouterProvider router={appRouter} />);

  - Error element is used to handle errors and show customized page
  - It uses error Hook, useRouteError.

  - now if you want to keep header in all routes so you need to put others in
childeren
  - and use Outlet from react-router-dom
  - const AppLayout = ()=>{
        return (
            <div className="app">
            <Header />
            <Outlet />
            </div>
        )
    }
  - Here Outlet will replace other compoeenents according to path.
```

Link:

```
  When you are in React and you want to route to some other page route, Never use anker tag, <a>, cz it will relode entire page.

  Use Link Componenet from react-router-dom.
  its similar to anker tag, but it wont reload page. Cz react keeps track of those links and keeps pages on single page, without reloading.
  Thats why its single page application, cz its not reloading to saperate page.

  - so instead of <a href="/about" >About Us</a>
  - use <Link to="/about" >About Us</Link>
```

Routing in Web Apps:

```
  1. Server Sie Routing:
      When you have /about, in <a> it reloads whole page, sends network call to /about html page, fethes that html, and renders it on UI.

  2. Client Side Routing:
      We are not making any network calls cz all components are already loaded, we are not fetching page.
```

Dynamic Route:

```
  Suppose when we click on any card, we want ot get info about that specific hotel card.
  and get detailed hotel info and menu, offers from it.
  so for that we need to implement, dynamic route for generic card template based on hotelId.
  like /restaurant/:hotelId

  To extract this id in component another hook is used.
  - useParams Hook
  - import { useParams } from "react-router-dom";
  - const { resId } = useParams();
```

8:

Class Based Components:

```
  As functional components are just javascript functions, class components are javascript classes.

  class About extends React.Component{
    render(){return (<div></div>)}
  }

  How to Recieve props in class based componenets?
  with Constructor !!

  constructor (props){
    super(props) // this is must
  }
```

why super(props)?

```
  Basically it allows accessing this.props in a constructor function, infact
super() calls the constructor of the parent class ie React.Component, Without this
the component cannot inherit the behaviour methods of React.Component.

  if super() is not called, component cannot access the lifecycle methods and
importantly "this", as without super(), this will not be defined and trying to
access this.props will result in error.

  Now by passing the current props to super(props) it ensures that the
React.Component constructor recieve current props
  and initialize them withint that parent class, so this.props will be available
throughout the lifecycle.

  Now to access properties from other component passed with props in render,
  you can use this.props.name to get value directly.
```

State Variables in Class Based Components:

```
  - class based components have 1 big object to maintain state.
  - So instead of writing 2-3 state variable objects, all variables of state will
be under 1 object
  - Use inside constructor.

    this.state = {
        count: 0,
        count2: 10
    }
    Even in functional compnent react uses object to maintain state but the logic
is not same with class based component exactly.

    Why Inside the Constructor?
      - States were created when instance of class was created.
      - ie when you load component, you are making instance of classand giving
props.
      - So its best to recieve props and create states here.
```

Update State Variable:

```
  - Never update state variable directly, ie this.state.count + = 1
  - React gives access to
  - this.setState({count: this.state.count+1})
  - This setState takes an object and this object will contain updated value of
your variable.
  - After this react wil re-render the component and update the value.
  - if you need to update multiple values, update in same object of setState.

  Behind the Scenes:

    Suppose you are updating values onClick button.
    button was clicked -> this.setState called -> react takes object (only updates
state variables that were in object others not touched)
    -> retrigger concilliation event -> get difference of objects -> update state
variable -> re-render component
```

Lifecycle Methods and Execution Process :

```
  - When parent component is mounted on web page, first constructor is loaded, and
then render is called.
  - now it starts rendering JSX and it came across child class component.
  - takes child class from import
  - starts to load child class now
  - new instance of class created
  - first when class loads constructor is called
  - then render is called


  Mounting Cycle
    |
  Constructor(dummy Data)
    |
  Render (dummy Data)
    - render happens with default values of state variables
    |
  Updates Actual DOM
    |
  ComponentDidMount() - will load after rendering compoenent in mounting cycle
    - API Call
    - this.setState - updates state variable
    |
  Mounting Cycle Finished
  Update Cycle
    |
  Component Re-Renders due to setState
    |
  Updates Actual DOM
    |
  ComponentDidUpdate() - called after compnent re-renders and get updated set
state
    |
  Update Cycle Continues until Component is removed / unmounted



  UnMounting Pahse
    |
  ComponentWillUnmount() - called just before unmounting ie component removing
from DOM
    |
  Component removed from DOM
```

ComponentDidMount():

```
  this will run after component has been rendered.

  Why its used? To Make API calls.
  ie react first renders the UI skalaton, then calls the API, and then re-renders
with updated data.
  thats why after dummy UI is rendered, api calls takes place in
ComponentDidMount() and rerenders component.

  so always contructor -> then render - > component did mount.
  parent constructor
  parent render       // now parent will load child component but its not finished
rendering
    First  user class constructor
    First  user class render
    First  user class component did mount // child will render fully
  parent compoenent did mount

  After everything in child component has loaded then parent compnent will be
called.
```

Optimization by React for Multiple Childs:

```
  What about Multiple children in parent

    parent constructor
    parent render
      First  user class constructor
      First  user class render
      Second user class constructor
      Second user class render
      Third user class constructor
      Third user class render
      First  user class component did mount
      Second user class component did mount
      Third user class component did mount
    parent compoenent did mount

  - But incase of multiple child classes, REACT optimises the cycle
  - completes Constructor and renders process of all children ie Render Phase
  - then REACT updates the actual DOM and quickly dummy UI is rendred
  - then component did mount is called.
  - Now as react wants to quickly render the UI, it will batch render phase of child components
  - Why? because updating actual DOM is very expensive process,
  - and when we want to Render UI from virtual DOM to actual DOM, for every component DOM manipulation needs to be considered
  - so DOM manipulation for each component saperatly is huge toll, so thats why it batches them in virtaul dom only
  - then whole changed UI in virtual dom is then directly updated in actual DOM together of all component changes.
  - So it batches all child changes in render phase and then update actual DOM
  - then all ComponentDidMount executes for all childs
  - then ComponentDidMount for parent as all childs are rendered and parent is also fully rendered now.
```

How to use ComponentDidMount():

```
  make it async !

    async componentDidMount(){
      const data  = await fetch("https://api.github.com/users/varun21vaidya");
      const json = await data.json()
      console.log("got user data");
      this.setState({userInfo: json})
    }
```

Remeber Never Compare Life Cycle to Functional Components

How to use ComponentDidUpdate():

```
  componentDidUpdate(prevProps, prevState) {
    if (this.state.count1 !== prevState.count1) {
      // Perform some action based on the change
    }
  }
  now if you need to change multiple state variables with different results,
  for each variable you need to write different condition
```

How to use ComponentWillUnmount():

Issue of Single Page Application:

```
  - this is called when leaving the component
  - this is used for cleanup like setTimeout, setInterval timers
  - cz if you have used them in componentDidMount(),
    everytime it will create new instance when you come to the component
  - EVEN if you leave the component it will create another instance of the timer
and still run in background
  - This happens even if you switched other components, still it will create new
instance of timers from old component
  - WHY? cz its single page application, yes one the disadvantage of it.
  - and old compnent never
  - thus its important to clear those timers in ComponentWillUnmount()

      componentDidMount(){
          this.timer = setInterval(()=> {console.log("parent Interval calls")},
1000);
          console.log("parent compoenent did mount");
      }
        Suppose parent component has child component as well which fetches data
and shows on parent component

        - parent constructor
        - parent render
           - First  child class constructor
           - First  child class render
           - First  child class component did mount
        - parent compoenent did mount
          - got user data (from child component from mount())
          - First  user class render
          - child - component Did Update
        - parent Interval calls 125 (continued till we left the component)

        --component switched--
        - parent compoenent will unmount
          - child - component will unmount
        - parent Interval calls 96 (continuing even after leaving the component)


      - if you switch to other components it will stop previous instances create
new instances everytime As its single Page Application.
      - Observe here first parent component will unmount then child component will
unmount

      componentWillUnmount(){
          clearInterval(this.timer);
          console.log("parent compoenent will unmount");
      }

      after this it will not call last parent Interval calls 96
```

Same Issue happens with useEffect():

```
  if we use setInterval or such timers in functional component it will behave
same.
  Solution? : Use Return Statement

  useEffect( ()=>{
    const timer = setInterval(()=>{
        console.log("parent interval calls in useEffect");
    }, 1000);
    console.log("parent use effect used")

    return ()=>{
        clearInterval(timer);
        console.log("parent use effect return")
    }
  }, [])


  Use Effect Return will only Be called when you leave the component

      child class child - component constructor
      child class user class render
      child class child - component did mount
    parent use effect used
      got user data
      child class user class render
      child - component Did Update
    parent interval calls in useEffect 2

      --component switched--
      child - component will unmount
    parent use effect return

  - Here Child component will unmount then parent component will be returned
```

Why we cant use async with useEffect callback:

```
  you can use async componentDidMount() but if you use useEffect(async()=>{
  you will get warning, Why?

  - Because React expects its callback function to either return undefined or
    a cleanup function, not a promise which async returns.
  - async function always returns a promise and useEffect is not designed to handle promise directly.
```

9:

How to Optimize App:

```
  Use Single Responsiblility Principle:
  each component should have only one responsiblity.
  break down the code into reusable, maintainable, testable components.
```

Custom Hooks:

```
  Creating custom hook is not mandatory but it makes your code, modular, reusable, maintainable.
  take out functionality from your component which is not its sole purpose.
  eg. fetching data is not sole purpose of body or restaurant page, so we took it out
  and created custom hooks for fetching data and sending to respective compoenents.

  - create it in utils
  - name the hook starting with "use..."
  - keep same name as hook
  - export and use it anywhere
```

Lazy Loading / Code Splitting / Dynamic Bundling / Dynamic Import / On Demand Loading / Chunking

```
  when we develop our app it has so many components, and react makes 1 js file out of it.
  suppose there are hundreds of components and it will create huge js file, which is unnecessarily big
  and if user is not going to specific routes its unused code being pulled
  now with increased size of js, reduces efficiency and performance.
  so split the js file into bundles according to logical saperations such that
  each bundle can have multiple childs and according to need each bundle will be fetched.

  so smaller modular bundles can be created ie smaller js files with limited components inside them
  how to saperate file from main file.

  inside app.js where you define the children of roots, import your file like this.

  const About = lazy (() => import("./components/About"));

  and when calling children and their component with path,
    {
        path:"/about",
        element:
            <About />
    },

    but the moment you go to the route, react will throw error, because its so fast, that
    even before file will be fetched, react tries to get the file but could not find it.
    So use Suspense method from react itself.

    This suspense also has fallback ie default JSX to show until file is fetched and rendered.

    import React, {lazy, Suspense} from "react";
    {
        path:"/about",
        element:
            <Suspense fallback={<h1>Loading Screen...</h1>}>
            <About />
            </Suspense>

    },

    Now this will saperate about content from main js file, then when you go to /about
    it will fetch another js file and render the about component
    so the main size of js is reduced and efficiency will also be increased when you this for all bundles
```

10:

Styling Your App:

```
  - SaSS, SCSS - superpowers to css
  - Styled Components - used in some big orgs
  - Frameworks - Has prebuilt components -  Material UI, Bootstrap, Chakra UI, ant.design

  PostCSS - Tailwind Uses behind the scenes, to transform css inside the javascript
  as you need to use .postsrc to read tailwind
```

Cons of Tailwind:

```
  - Makes JS look ugly
  - too many inline classes makes code unreadable
  - sometimes you need to repeat classes much more times  unnecessarily
```

Advantages of Tailwind CSS:

```
  - Tailwind is lightweight
  - Makes css development easy and good to go while writing HTML itself.
  - When parcel make bundle of css, suppose our application is using 100 classes,
    but tailwind library has thousands of classes, still it will only include those that we have used
    on our webpage when compiling process.
  - If in one file m-4 is used 100 times, still it would include it only once.
  - In regular css, working with various component, by various developers, they create duplicate
    and redundant classes, adopting to tailwind, reduces that, so develpers adopt certain way to give styles.
```

11:

Higher Order Components:

```
  Higher Order Components is a function that takes existing component and enhances it and returns back a new component.

  Higher order functions are pure functions.

  const export EnhancedComponent = (card)=>{

    // props given to component and it returns new component
    return (props)=>{
      return (
        <div>
          some new enhancement
          // use props to send data to og card component
          <card {...props}></card>
        </div>
      )
    }
  }

  now where you will be calling this new card component
  create new card component by calling it as this function
  // Higher Order Component calls Higher order component function
  // which returns a component with closest label by taking hotel card
  const ClosestHotelCard = closestLabel(HotelCard)

  you can then use this card,
  {condition ?  <ClosestHotelCard hotelData={hotel?.info}/> : <HotelCard
hotelData={hotel?.info} />}

  Here you are not modifying og component, you are only adding new enhancement to
component.
  Higher Order components add new feature to exisiting functionality and return
new component.
```

Controlled Component:

```

  - UI layer - JSX
  - Data Layer - JS code in {}, states, Props, data

  Suppose there are multiple components with accordian
  when we click on component -> onclick it chnages state (!showItems) -> It
enables / disables list of items
  But we need to modify so that when we click on component, it should expand but
close other components

  so we cant change this internally from child component. we should control
expanding fom parent component
  so when we click on child, state of other componenets should change. to
showItems = false

  SO we are passing the control to parents -> child components become Controlled
Components

  - If child component is controlling itself, it is uncontrolled component.
  - If child component is controlled by parent, it is controlled component.

  How to do? we can send setChangeState which changes showItems state from parents
to child in arrow function.

```

```
  <child changeStateFn={()=> setChangeState(index of compoent)}>
  now inside child when you click on div, call the setChangeState which will
change state from inside child and
  affect parent of other childs and this child

  handleClick(){
    changeStateFn()
  }
```

THIS IS LIFTING STATE UP

Props Drilling:

```

  When react project grows big, passing data between components is big challange,
  React has 1 way data flow, parents to children.

  Suppose you want to pass data inside hotel menu from top to last children in
hierarchy
  hotelmenu -> child -> grand child - > supergrandchild

  but should you really need to pass that data to each child which will pass to
its own child
  and at the end it will reach last child.
  Its not good way, because middle components are not even using that data

  THIS IS PROPS DRILLING.

  there are many ways to resolve props drilling, context is one of the way.
```

Context:

```

  React provides - Context - global kind of object outside components (not exactly
global object)

  eg. logged In , dark theme toggle.

  createContext: to create context outside components in file

    import { createContext } from "react";

    const UserContext = createContext({
        loggedInUser : "Default User"
    });

    export default UserContext;
```

useContext: Accessing Component in functional Components

```
    const {loggedInUser} = useContext(UserContext);

    you can have as many contexts.

    Only Data needs to be accessed in multiple components should be used as
context.
```

UserContext.Consumer: Accessing Component in Class Components

```
    // inside UserContext.Consumer you need to have js {} inside which have
callback function {()=>}
    // this takes required context ({loggedInUser}) and returns jsx => (<h1></h1>)
    <UserContext.Consumer>
      // takes callback function
      {({loggedInUser})=>
          (<h1>The user is {loggedInUser}</h1>)
      }
    </UserContext.Consumer>
```

UserContext.Provider: Modify Context

```
    Wrap usercontext on app so we can use loggedInUser in entire app anywhere and
    it will change default value given in create context.

    If you wrap it onllly on header, it will cange only for header and at
remaining places
    in app, it will be modified userName only.

      <UserContext.Provider value={{loggedInUser: userName, setUserName}} >
        <div className="app">
            <UserContext.Provider value={{loggedInUser: "Kavtya Mahakal"}} >
                <Header />
            </UserContext.Provider>

            <Outlet />
        </div>
      </UserContext.Provider>

    you can even send setUserName to userContext so you can update it from any
component
```

you can use context for small and mid sized applications easily, for more large size projects you can use precise state management libraries like redux.

12:

Redux

```
  first of all its not mandatory.
  It is predictable state container for javascript apps

  Two Libraries:
  - React-Redux - It is bridge between react and redux.
  - Redux toolkit (RDK) - In older day we used to have different way to write
redux (Vanilla Redux).
    This toolkit is new way to write now and more standerdized redux.
```

There were 3 problems with old redux

```
  1. Configuration was too complicated (Huge learning curve).
  2. needed to add lot of packages, now just redux toolkit and react-redux is
enough.
  3. Redux required too much boilerplate code.

  Redux is big object kept in global central place, any react component can read
write from this.
  Is it good to keep very big data in big object ? yes its fine.
  But to not make our redux store clumsy we have slices in redux store.
  slices are logical partition, like one for cart, one for loggedInInfo, another
for dark time.
```

Writing Data in Redux:

```
  Redux says you cant change your data directly.
  When you click add button -->
    It dispatches an action -->
      It calls a reducer function -->
        reducer function modifies cart -->
          updates slice of redux store.
```

Reading Data from Redux:

```
  We use a selector to read data from store.
```

```
    You fetch data from slice through selector -->
      ie selector subscribe to slice and get data -->
        selector then updates the cart

  So basically you subscribe to cart slice using selector and
  it automatically update cart directly.

  Add --dispatches--> Action --calls--> reducer function --> | Slice |

        Cart <--Subscribes to store using--> <--Selector --> | Slice |
```

Steps:

```
  1. Install redux toolkit -> npm i @reduxjs/toolkit
  2. Install react-redux -> npm i react-redux
  3. Build appStore (redux store where you will configure and store reducers)
  4. Create Slice (create as many slices like cartSlice, LoginSlice) and create
reducers functions
  5. for modifying data use Dispatchers (useDispatcher hook)
  6. for reading data use Selectors (useSelector hook)
```

Configure redux store:

```
- Configure in new file and add reducer which will be use slice reducers.

  import {configureStore, createReducer} from "@reduxjs/toolkit"
  import cartReducer from '../utils/cartSlice';

  const appStore = configureStore({
      reducer:{
          cart: cartReducer
      }
  })
  export default appStore;
```

Wrap Provider over entire app or specific part:

```
- Provider for redux store, if you want redux over only specific part wrap it only
on that part
- Can be wrapped over Context.Provider

  import {Provider} from 'react-redux';

  <Provider store={appStore}>
```

```
      <div className="app">
      </div>
  </Provider>
```

Create Slice:

```
- name is used during subscribing with selector
- notice initial state as default state
- can have as many reducer functions
- each reducer function take state and optional actions.payload
- when you mutate state, the payload that you give comes directly as
actions.payload
- there will be 2 exports one for each reducer function to use with action
dispatcher
- and other for configure reducer

- When you createSlice it will return object in cartSlice
- This object will have actions and reducer


  import { createSlice } from "@reduxjs/toolkit";

  const cartSlice = createSlice({
      name: 'cart',
      initialState:{
          items: []
      },
      reducers: {
          addItems: (state,action)=> {
              state.items.push(action.payload)
          },
          removeItem: (state)=>{
              state.items.pop()
          },
          clearCart: (state)=>{
              state.items.length = 0
          }
      }
  });

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

  export default cartSlice.reducer;
```

Dispatch Action to mutate state

```
  - you need to import useDispatch hook
  - now on click event you can call function which will take call back as item
    and dispatch action event with reducer function
  - what ever you pass as item in addItems(...) it will go as action.payload which
will go inside reducer function
    whihc will update the state items.

    import { useDispatch } from "react-redux";

    const dispatch = useDispatch()
    handleAddItems = (item)=>{
        dispatch(addItems(item));
    }
```

Subscribe store with Selector to fetch data

```
- to fetch data by subscribing to store using useSelector hook
- it takes callback function with store, then use slice name with exact location
of store
- do not use generic store location as it is constantly subscribed to store, so
use specific location.

- useSelector gives access to store but callback function gives specific portion
of store.
- now as we subscribed to items, whenever items in that appstore changes we will
get update value in cartItems.

  import { useDispatch, useSelector } from "react-redux";

  const cartItems = useSelector((store) => store?.cart?.items);


Import differences:

import {configureStore, createReducer,createSlice} from "@reduxjs/toolkit"
import { Provider, useDispatch, useSelector } from "react-redux";
```

Important Things to notice for interviews:

```
- For selectors use exact location of your items, as it has huge performance
benifit.

- Reducer - one big reducer in app store which contains reducer functions of slice
- reducers - multiple reducer functions in any slice, but when exporting it will
be
```

```
            export default cartSlice.reducer;

- In vanila version it didnot allow to mutate state and we used to return the new
state.
  Return was must
  const newState = [...state] //shallow copy
  newState.items.push(action.payload)
  return newState;

- Now with redux toolkit we have to mutate state and return is not mandatory.
  state.items.push(action.payload)

- Redux still maintains its immutable state nature behind the scenes using immer
library.
  so immer finds difference between immutable state and mutable state and gives
you new copy of immutable state.
  Immer is a tiny package which allows you to work with immutable state in easy
way

- thats why we have to mutate state thats why it doesnot allow, in clearCart

  state.items=[]
  this does not work as it replaces items array with new array ie adds new
reference
  This breaks the immers tracking as its saperate from old reference and doesnot
change anything.

  but with state.items.length = 0 ;
  It mutates existing array.

- When to Use Redux: when there are thousands of componennts and many are mutating
state, and you need to manage them all.
  To track these changes use redux dev tools

- In older vanilla redux there were middlewares and thungs for asynchronous
operations.
  basically you want to make api call and want to store data in redux store.
  there was data pattern which used these middleware and thungs.

  Now it uses RTK (React tool kit) query ie RTKquery, its way of fetching data.
```