**##Containers** contains---these are stateful components created (class based or functional with hooks) which manages states

**##Components**

contains--- these are presentational or dumb components created (class based or functional) which are not having states only they present with given props

**##props**

**\*\*\*boolean Props\*\***

simply giving prop in markup will give true value to the boolean prop

**\*\*\*stateful component props\*\***

here props are accessed by this.props.----

**\*\*setState\*\***

if u r using a this.state in setState then arrow function for setState due to asynchronous issue

**\*\*prop-type\*\***

prop types are recommended to use only when we distribute our code to others like in developing team ,third party etc..

**\*\*react.memo,pure components\*\***

these are used to have a check for prop changes to re render without using shouldComponentUpdate

react memo is used to have a ability render only props changes

we can use pureComponent instead of Component in class based if we want to get rid of props check

**\*\*JSX elements\*\***

jsx element markup always starts with caps

**\*\*tricks\*\***

--> dont update state directly to that state object instead take a clone of it then make setState

--> if u r using a this.state in setState then arrow function for setState due to asynchronous issue

**\*\*REACT ROUTER\*\***

react application will have routing concept using two packages

1)react-router (logic)

2)react-router-dom (parsing)

**\*\*\*Steps:-\*\*\***

1) to hve in application we need to import browserrouter from 2nd package wrapped a component under which need routing so generally we use that in appjs or indexjs

2) then import Route , LINK from 2nd package in page of wrapped component to use routing functionality

3) now use

---> <Route

path="specify path here"

(exact--optional to exactly match path)

component= { direct class or function component name } // to render a component

render = {()=>{<h1></h1>} } // to render direct jSX elements

/>

----> <Link

to="path as string"

---or---

to={

{

pathname:'path as string',

hash : '#submit',

search : '?flag=true' // query parameters

}

}

(exact--optional to exactly match path)

>"link display name"</Link > // this link will dont reload page simply re render pages

here we need to use anyone between render or component

Note:- component is consider a new component each time if pass inline function to render component instead of named component. Also even render is a anonyomous function it will return same type so it wont consider as new component

so recomended approach in route is

1) use component={Componentname} ---- when there are no props to pass

2) use render ={()=><Componentname {...props}>}

----------Or--------------

we can use NAVLINK instead of link only if we want to style link based on active ness

navlink provided by react router will give active class to a tag by default so that we can style active links

Ex:- <NavLink

to="path as string"

---or---

to={

{

pathname:'path as string',

hash : '#submit',

search : '?flag=true' // query parameters

}

}

(exact--optional to exactly match path)

activeClassName='our own class name' //overriding classname

activeStyle ={

{

color :'blue',

textDecoration:'underline' // we can have any css rule here to style active link

}

}

>"link display name"</NavLink >

here there are two props are added extra to have our active class name instead of active & also for styling inline

using React router , Route will send some props by default to component(if component specified) rendered .these props are very useful managing some stuff

props are 1) history 2) location 3) match

these are called as "ROUTE PROPS"

These Route props are not passed down the component tree , we cant acess them in components which we simply embedded them as jsx code

so inorder to do pass these props underneath to components in the Routed components there are 2 ways

1st way ----> directly passing route props to underneath component props

2nd way ----> using "withRouter" HOC provided by react router , we need to import it from react router package in underneath components & wrap it with that

**\*\*Absoulte path vs relative path\*\***

You learned about <Link> , you learned about the to property it uses.

The path you can use in to can be either absolute or relative.  **##Absolute Paths**

By default, if you just enter to="/some-path" or to="some-path" , that's an absolute path.

Absolute path means that it's always appended right after your domain. Therefore, both syntaxes (with and without leading slash) lead to example.com/some-path .

**##Relative Paths**

Sometimes, you might want to create a relative path instead. This is especially useful, if your component is already loaded given a specific path (e.g. posts ) and you then want to append something to that existing path (so that you, for example, get /posts/new ).

If you're on a component loaded via /posts , to="new" would lead to example.com/new , NOT example.com/posts/new .

To change this behavior, you have to find out which path you're on and add the new fragment to that existing path. You can do that with the url property of props.match :

<Link to={props.match.url + '/new'}> will lead to example.com/posts/new when placing this link in a component loaded on /posts . If you'd use the same <Link> in a component loaded via /all-posts , the link would point to /all-posts/new .

There's no better or worse way of creating Link paths - choose the one you need. Sometimes, you want to ensure that you always load the same path, no matter on which path you already are => Use absolute paths in this scenario.

Use relative paths if you want to navigate relative to your existing path.

**\*\*ROUTE PARAMETERS\*\***

we can pass route parameters to route by putting ":" in path attribute

we have to place that route where we want that routed component

Ex:- <Route

path="/:id" // here given id as parameter u can give anything

or

path={'/post/:'+ prop.id} // this is to have dynamic path

(exact--optional to exactly match path)

component= { direct class or function component name } // to render a component

render = {()=>{<h1></h1>} } // to render direct jSX elements

/>

we can extract that parameter using props.match.params.'here parameter name' to acess parameter passed

**\*\*Query prameters & fragments in Route\*\***

how do you extract search (also referred to as "query") parameters (=> ?something=somevalue at the end of the URL)? How do you extract the fragment (=> #something at the end of the URL)?

Query Params:

You can pass them easily like this:

<Link to="/my-path?start=5">Go to Start</Link>

or

<Link

to={{

pathname: '/my-path',

search: '?start=5'

}}

>Go to Start</Link>

React router makes it easy to get access to the search string: props.location.search .

But that will only give you something like ?start=5

You probably want to get the key-value pair, without the ? and the = . Here's a snippet which allows you to easily extract that information:

componentDidMount() {

const query = new URLSearchParams(this.props.location.search);

for (let param of query.entries()) {

console.log(param); // yields ['start', '5']

}

}

URLSearchParams is a built-in object, shipping with vanilla JavaScript. It returns an object, which exposes the entries() method. entries() returns an Iterator - basically a construct which can be used in a for...of... loop (as shown above).

When looping through query.entries() , you get arrays where the first element is the key name (e.g. start ) and the second element is the assigned value (e.g. 5 ).

Fragment:

You can pass it easily like this:

<Link to="/my-path#start-position">Go to Start</Link>

or

<Link

to={{

pathname: '/my-path',

hash: 'start-position'

}}

>Go to Start</Link>

React router makes it easy to extract the fragment. You can simply access props.location.hash .

**\*\*Switch in Route\*\***

if we use Route directly , the components will be rendered for all routes whose paths are matched

Instead if we want to render only one component at a time .

we have to put the routes out of which we need only components to be rendered in a "switch"

Switch can be imported from react-router-dom & wrap routes in switch to have a single route navigation

Ex:- <Switch>

<Route 1>

<Route 2> // so here out of 3 routes only single route whose path is first matched

<Route 3>

</Switch>

parsing of routes will be done from top to bottom

We can have of mix of routes like some routes in switch , some resides outside

**\*\*Navigating programatically\*\***

By using <Link> or <NavLink> we can route to a certain page or component

but to render it programatically we have to use history prop send by route by default

Ex:- this.props.history.push('path here to navigate')

or

this.props.history.push({pathname:'path here to navigate'})

here what they are doing is pushing path page to stack of pages in browser

stack of pages will have pages which navigated from start of web application

Note:- this method will be used whenever we want to render something after a particular task finished not directly like http request

**\*\*Redirect\*\***

sometimes we need to redirect to a certain page in web application by default or by clicking a button or by something conditionaly

we can use "Redirect" of react-router-dom for redirection

Ex:-

1) <Redirect to="path u want to redirect"> ---- used outside of switch

2) <Redirect from ="from path from where u want to redirect" to="path u want to redirect"> ---- used inside of switch

we can also redirect by using history props send by route

Ex:- this.props.history.push('path here to navigate')

or

this.props.history.push({pathname:'path here to navigate'})

Note:-here push will push page to browser but using redirect we replace current page with redirect page in stack of pages of browser

using histroy prop only u want to replace page use replace instead of push

this.props.history.replace('path here to navigate')

**\*\*handling unknown routes\*\***

simply putting route in jsx without path & also we need to specify it last of all routes

Ex:- <Route

component={'component of 404'}

or

render={()=>{

<h1>Not found </h1> //any jsx code u want to show for unknown post , u can have here

}}

/>

**\*\*LAZY LOADING\*\***

this concept is used whenever we have a big application with lot of components which are having components unnecessarilly loaded in browser in bundle instead we can have upfront chunk

To prevent that unnecesarry loading we will use this concept

there are two ways of implementing this concept

1) for those react apps which are less than 16.6 version , we implement lazy loading by HOC asynComponent function--(tutorial)

2) for higher than & equal 16.6 , we have a new way by in bulit method on react object called "React.lazy()"

ex:- Lazycmpnt = React.lazy(()=>import('path to component'))

when we use this, we should use render method to use this component asynchronously .

for that we use suspense object of react

<Route path='/...' render=(()=>{

<Suspense fallback={<div>loading....</div>}

><Lazycmpnt/></Suspense>

})>

we can directly use suspense method for lazy loading component if we want to render component conditionally

server side wont work for this only client side

--- note:- both cases import should contain default export rather than named export -----

**\*\*encodeURIComponent\*\***

this is a helper method which provided by javascript which simply encodes my elements such that they can be used in the URL this is relevant for whitespace & so on===============================================

**\*\*\*\*Redux\*\*\*\***

Redux is a 3rd party library which works totally independent of react

WHY -----> redux is used to make state management among different components very easy in complex react projects.

Note:- by using global object variable which has all states we can make state management is easy (using context) but problem with that is changes in global objects doesnt do update or re rendering stuff

Redux gives us a certain flow of data, a certain way of managing data that we can then nicely integrate with another package into react app

so that react does react to changes of data globally

Its all about a centeral store which will have entire application state and having a clearly defined process of how your state may change

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**PROCESS:--**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) component should dispatch an "action" with type & payload

this action is like predefined information package

type is like addIngredients removeIngredients etc..,

payload is like ingredient to add or remove

2) this action reaches to "Reducers" which will detect type of action & create new object with recieved new update of state & updates central store

Reducers are "pure sync functions" which recieves action & old state as inputs ( done in a immutable way )3) with that central store is updated , now central store has "subscription trigger mechanism" which will trigger everytime a central store is updated so that all components which subscribed central store will get notified & recieves updated state as props

-----------------------------------------------------

programmatic process for connecting redux to react

1) first we need to create action const file & reducer with intial state & function for handling dispatch actions in a folder named store ( recommended not mandatory)

2) now to connect this 3rd party redux with react import provider package from react-redux module wrap that with app we are using in index js

Ex:- <Provider store={store}>

<BrowserRouter>

<App />

</BrowserRouter >

</Provider>

3) above store provided will be created using redux module by passing reducer we created in 1st step

Ex:- const store = createStore(reducer)

here createStore function is imported from redux

4) Now for components to use redux state through we have to setup 3 things

--> import connect from react-redux

--> create mapStateToProps function which takes redux state as default & then map ur req prop to state item u want to acess through as props

--> create mapDispatchToProps function which takes dispatch as default & then map ur dipatch actions to dipatch events through props

Ex:- const mapStatetoProps =state=>{

return {

ings:state.ingredients

};

}

const mapDispatchtoProps =dispatch=>{

return {

addIngredient: (ingKey)=> dispatch({type:actionTypes.ADD\_INGREDIENT,ingredientKey:ingKey}),

removeIngredient: (ingKey)=> dispatch({type:actionTypes.REMOVE\_INGREDIENT,ingredientKey:ingKey})

};

}

export default connect(mapStatetoProps,mapDispatchtoProps)(withErrorHandler(BurgerBuilder, axios));

note :- context api is alternative which is used when we want to use a varible across components with less upadte to it because of its less optimzation like authentication & theme change

for frequent updates use redux kind central store otherwise we can use custom hooks to have this type of functionality

**\*\*\*\*Advance Redux\*\*\*\***

we can make use of synchrounous code using advanced Redux concepts

**Middleware** :---- a middleware is used to execute something between action dispatch & its way to reach to reducers

To give a middleware to redux react app , first we need to import applyMiddlware from redux then pass a middleware as wrapper to that applyMiddleware Hoc function & then pass Hoc function as a enhancer to createStore method

Ex:- const store = createStore(reducer, applyMiddleware(logger)) // here logger is middlware function

//logger middleware function

const logger = store =>{

return next =>{

return action =>{

console.log('middleware dispatching',action)

const result = next(action);

console.log('middleware next state',store.getState())

return result;

}

}

}

**redux dev tools** :---- we can add redux dev tool which will help us enormously in debugging & analysing our react application . It make sense in bigger application with lot of state & dispatch actions

prerequitsies are we need react dev extension in our chrome

then we need to import composer hoc from redux if we have both middleware & redux dev tool to add otherwise we can directly add that composer Enhancer

( middleware also 1 type of enhancer)

const composerEnhancer = 'u get will get in offical github page'

const store = createStore(reducer, composerEnhancer(applyMiddleware(logger)))

or

const store = createStore(reducer, composerEnhancer)

ActionCreators are special concept which will return a dipatch object for reaching out to reducers

it will be more useful when running async code

"redux thunk" is package which added as middlware to redux which allows us to run async code which allows us to acess dispatch & getState parameters in action creators

**\*\*DEPLOYMENT STEPS\*\***

1)check and adjust base path

<BrowserRouter basename="/my-app/" >

2)Build & optimize project

npm run build in create-react-app project

3)Server must always serve index.html (also for 404 cases)

to ensure that routing works correctly because routes are configured at react level not server level. therefore even for 404 cases it should return index.html

then after react will handle unknown routes

4)upload build arttifacts to (static) server

in /build folder when using create-react-app

**\*\*WEB PACK\*\***

webpack is a bundler but it actually is more than that , a bundler alone would just concatenate files,

but it also allows you to optimize your files and you hook in various plugins and so called loaders to transform files( ex transpiling next generation js to current generation js )

In short , idea behind web pack is to have multiple js ,css, image, what ever files and bundle them together

so webpack is powerfull tool which is to build efficient workflow in our projects by bundling them together

**---> HOW IT WORKS**

webpack has four important features

1) It always need at least one entry point (multiple possible)

ex:- app.js in react

it analyses the dependencies of this files & since it is entry it contains all necessary files for project underneath it so webpack analyses all other dependenices in project by going through entry point & build up dependency graph & bundles them together into one single file

By ths we can assure that all these dependenices are correctly ordered & concatenated into one output file

This is core functionality of webpack

2)It needs loaders

Ex:- babel-loaders & css loaders.

These are very important to transpile all js scripts (babel-loaders) & css files(css loaders) to be understand by webpage while rendering into browser

converting next gen js features to current gen features for old browsers

These are applied on each independent files & transforms that

3)It needs plugins

Ex:- uglify

These are used to optimize the bundles before it bundling & given as output to bundle.js used in deployment

These are applied globally after all loaders & dependicies work done ie jst before writing into output it applies to bundled file to get optimized bundle

4) finally bundling

---> basic workflow requirement ( which are satisified by above webpack)

1)compile next gen js features

2) Handle JSX

3) CSS autoprefixing

4)support image imports

5)optimize code

---> we need to install webpack & webpack dev server to build this workflow

----> make "configuration file" so that webpage can do webpack functionalities all above 4 features & bundles them & render in SPA

Ex:- webpack.config.js

\*\*\*---> we need aleast one root components which should have all routes & link to navigate

Ex:- app.js

we need 1 js file to mount this root component to react app where we import react-dom which will render our react app to dom

Ex:- index.js

**\*\*NEXT JS\*\***

It is a library building upon react which enforces a specific folder structure you have to use and gives you some things like server side rendering out of box

since it uses folder structure it can manage a lot of things for you so that you dont have to configure them manually

In short word , we can say nextjs is minimilistic framework for a server rendered react applications

In next js , the file system is our main API , we dont use react router with next js to create routes instead we create folders & files to reflect our URLS in the file system in "PAGES" folder

for further info, go to next js documentation

**\*\*REACT ANIMATIONS\*\***

For animations we can use tranisition & animations concept of css but limitation of that animation if disappear or fadeout kind transitions are there . even if it is not visible but it still it will be in dom with opacity 0 .Therefore it will become problem in case of big appliction with huge dom tree which slows down performance due to this limitations

this is not react ish behaviour

To solve this , a third party library called react transition group will help you .(Recommended)

also ALternatives are react move , react motion ,react router tranistion

for further info, go to respective documentation

**\*\*REACT SAGA\*\***

react saga is a package which is alternative to redux thunk

It is mainly used to dipatch your actions after performing asynchrounous code not directly dispatching actions.

for more info , go to respective documentation.

**\*\*CONTEXT API\*\***

this is a mechanism provided by react which is used for communication between components instead of passing props through all components

note :- this is used when we want to use a varible across components with less upadte to it because of its less optimzation like authentication & theme change

for frequent updates use redux kind central store otherwise we can use custom hooks to have this type of functionality

**\*\*REACT HOOKS\*\***

---> These are the functions which can be only be used from inside functional components or other hooks

---> Main is to expose certain (possible stateful) functionalities to functional components

---> naming convention is it always start with use ex: useXYZ();

---> Hooks are highly re-usable and independent for each component.

---> Hooks allow youu to add state to functional components and to share (possibly stateful) logic across components

---> introduced with 16.8

---> Allow you to use functional components only.

---> Hooks for managing state, side effects (http request etc..)

---> Build custom hooks to share stateful or stateless logic across multiple components.

Rules:- 1) Hooks should be inside a functional component or inside a custom hooks

2) Hooks should be under root level not inside any functions or block level

**\*\*useState\*\***

---> this is a crucial probably a core hook provided by react. It allows manage state and functional components

But it works a bit different than state in class based components

useState will intialize with intial state which can be anything means not only object but string,number,boolean (class based ,it should be object)

It will return a array with two elements :- first element is snapshot of state

second element is the function to update that state // it can have function to acess current snapshot of state

unlike in setState (merges state), second array element of useState requires all state properties to update otherwise the old properties will get lost. (since it will not update , it will overwrites that state)

Solution to above is hooks can able to manage multiple states by using useState for multiple state parameter

whenever u update state by its second element function , there will be no reintialization of state as react will save that state configuration

**\*\*\*ArrayDestructing in useState\*\*\***

ex:- const [inputState,setInputState]=useState({...inputstateObject})

**\*\*useEffect\*\***

This is used when we want the component life cycle kind of functionality of class based components in functional components.

name itself , it is used in cases of any side effect or async code.

this will run after and for every render cycle.

It accepts second argument which decide when to run the function of useEffect to behave like life cycle method.

this argument is an array with the dependencies of your function and only when such a dependency changed only then the function will rerun

It will acts as componentDidUpdate when useEffect is used simply without any second argument==> i,e Runs after every component update

It will acts as componentDidMount when useEffect is used with []--empty array as second argument==> i,e Runs only once after the first render.

useEffect can be used multiple times with different second arguments

Note:- to use componentWillmount we cant use useEffect , but we can directly run that inside functional component before returning so that it will make same of componentWillMount

**\*\*useCallback\*\***

(------ Note:- functions are objects and behave like any other value---- can be used whenever we use any prop property in useEffect because even other prop changes this will run ( prevention---so need to destruct that prop property from prop & pass to useEffect) ----- )

this is used to allow you to wrap one of your functions which is used whenever we dont need to re create that function

**\*\*useRef\*\***

this is used to get reference element value by adding a connection in react. Initiate a variable with useRef & add that variable to dom element.

Ex:- const inputRef = useRef() inputRef.current.value

<input

ref={inputRef}

----

/>

**\*\*\*Cleanup of useEffect\*\*\***

cleanup function is a function which is used to run after evry second useEffect runs to cleanup first useEffect values or conditions

to use this , we need to return this function in useEffect , also rule is it should return a function

here this is also used when we want same functionality of componentWillUnmount in functional component

Note:- React batches state updates

-->setState works in batches in react where if different setState are written one after other it wont cause different render cycles for each setState instead it will take all setState written one after each and react runs in a batch executes only 1 render cycle

udemy note:-

That simply means that calling

setName('Max');

setAge(30);

in the same synchronous (!) execution cycle (e.g. in the same function) will NOT trigger two component re-render cycles.

Instead, the component will only re-render once and both state updates will be applied simultaneously.

Not directly related, but also sometimes misunderstood, is when the new state value is available.

Consider this code:

console.log(name); // prints name state, e.g. 'Manu'

setName('Max');

console.log(name); // ??? what gets printed? 'Max'?

You could think that accessing the name state after setName('Max'); should yield the new value (e.g. 'Max') but this is NOT the case. Keep in mind, that the new state value is only available in the next component render cycle (which gets scheduled by calling setName()).

Both concepts (batching and when new state is available) behave in the same way for both functional components with hooks as well as class-based components with this.setState()!

**\*\*\*useReducer\*\*\***

useReducer is a hook which is used manage the state & its actions together as one when it mainly depends on prev state

useReducer concept is similar kind of redux reducer but not same.

Here we need to define useReducer function outside the component (recommended) so that it wont reinitiaze the function each time re render occurs & to decouple it , the function will take two arguments which are current state & action used for setting that state.

Now after defining it we need to pass above useReducer function to useReducer hook inside component along with intial value of state. This will return an array of 2 elements

1) state to access

2) dispatch function call outside reducer function for a specific action on state . It takes 2 arguments action type & action payload and returns a new state.

Each time a dispatch function returns a new state , it will cause a re render

**\*\*useContext\*\***

This is a hook which is used to make use of context in functional component.

**\*\*useMemo\*\***

useMemo is a hook similar to useCallback where here its used not to create a value again when rerender occurs instead of function

**\*\*useSelector & useDispatch\*\***

these 2 hooks are used when we dont want to use connect method of react redux .

Using this we can create normal variables instead of props for using state method & values

useDispatch will return dispatch method which is used in mapDispatchToProps,

useSelector is used to get a snapshot of current redux state & to get desired property value

**\*\*Custom hooks\*\***

The idea behind custom hooks is to share logic across components but not data.Logic which also influences the state of the components. They are normal function but we use must prefix it with use before function name. Then react will make it as hook behind scenes.

custom hook should return something so that after logic implementation we can use data returned by that implementation

All components who uses custom hook will have a each indiviual snapshot of it but not shares same thing across .Also each snapshot of hook will re renders for its respective component render cycle