React volume II

• Reaching out to server

Whenever react sends a request to server, server doesn't respond back with HTML, rather it sends back data usually in form of JSON. React uses this data to render elements on screen.

Sending AJAX requests :

There are two ways of sending AJAX requests

1. XHTTPRequest Object
2. Axios (3rd party library)

Axios : axios is a 3rd party javascript lib. for making AJAX requests.

STEP 1: npm i axios --save

STEP 2: import axios

Lifecycle Method for making AJAX Requests :

which lifecycle method should be used for making Requests. The lifecycle method componentDidMount should be used.

Example Application : Getting posts dynamically from server

let us consider an example in which data of posts is send by server and when a post is clicked it shows all info related to that post.

There are two lifecycle methods for reaching out to web

① componentDidMount.

② componentDidUpdate → can cause side effects ie continous network requests thus need some checking to restrict such a situation.

Handling Errors

The requests might not suceed all the time. Incase of Axios it is promise based thus we can always use .catch to handle request errors.

```
componentDidUpdate() {

axios.get("url").then((response) => ...)
.catch((error) => { if(error) console.log(error) }
```

error handlee

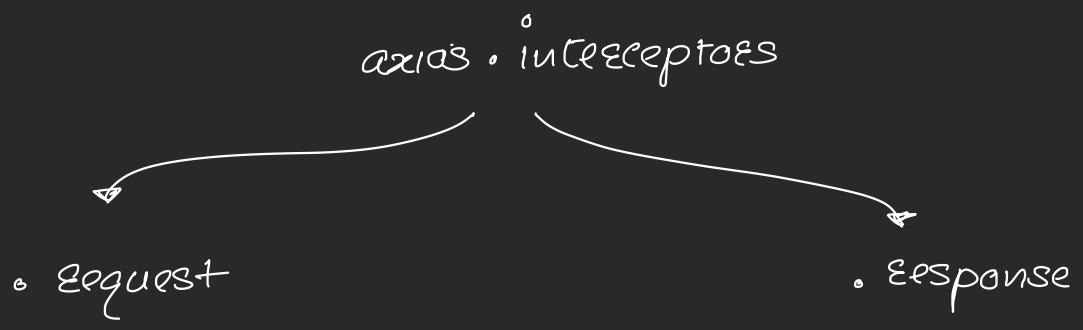
• Interceptors in Axios :

interceptors as name suggests intercepts requests or responses. It allows for globally configuring, mutating requests and handling errors irrespective of components

USUAGE :

① inside main index.js file → import axios

②



③ axios.interceptors.request.use (request =>

success block

// access to request object

// can add authorization headers etc.

→ it is important to return request otherwise

return request ; request will be blocked.

error => {

console.log (error)

return Promise.reject (error).

}

error block

• Setting Global Configuration for Axios

Sometimes we do not want to intercept a request but assign global configuration.

① baseURL : instead of mentioning entire URL in each request we can simply setup base URL and mention routes

① inside main index.js file

import axios from axios

⇒ axios.defaults.baseURL = "http://someurl.com"

② axios.get("/posts").then(....)

↳ route only

• Headers

• axios.defaults.headers.common["Authorization"] = "AuthToken" — common to all types of requests

• axios.defaults.headers.post["Content-Type"] = "application/json"

↳ specific to post requests only.

• Instances in Axios

instances allows to create templates which can be used partially. Instances can be seen as a fully customisable & configurable axios object

① create a new js file axios.js

② inside axios.js import axios

```
let instance = axios.create({
    baseURL : "some url .com"
})
```

③ export this instance

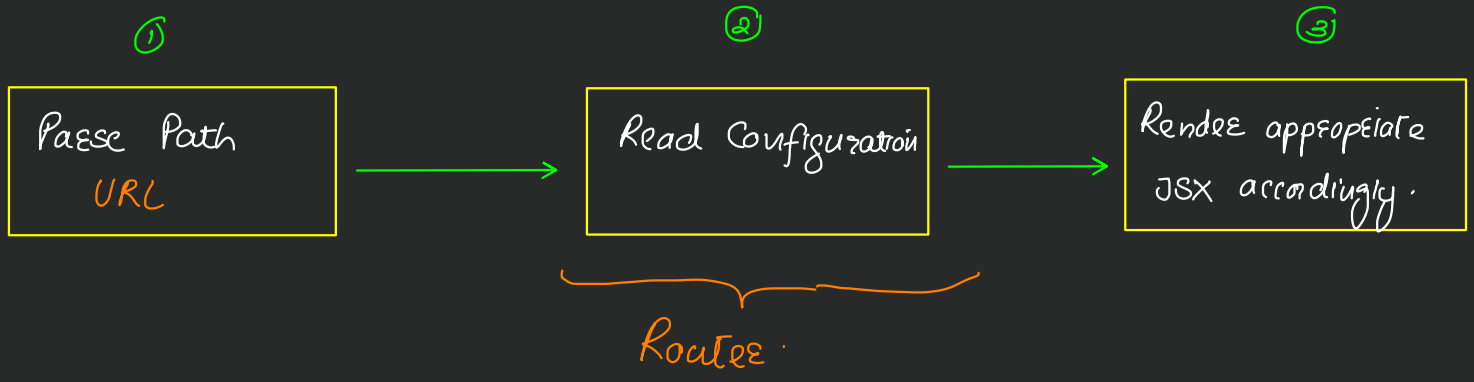
To use this instance anywhere we will need to import this in our application

Routing : Multipage feeling in SPA

- Routing is not build into core of React.
- For purpose of Routing we use a 3rd party package which is now the defacto standard.

- Incase of single page application there exists a single HTML file. We use javascript to render different components based on different routes

How does Routing work incase of SPA :



Installation

- npm install --save react-router react-router-dom
- import { BrowserRouter } from 'react-router-dom'
- We need to wrap things in BrowserRouter Object to make routing accessible.

```
class App extends Component{
    render(){
        return(
            <BrowserRouter>
                <div className='App'>
                    ....
                </div>
            </BrowserRouter>
        )}
}
```

Routing accessible within these tags

React Router vs React Router DOM:

React ReactDOM is required for web development. technically, it uses ReactRouter as a dependency.

- Setting up Rendering paths
  - import { Route } from 'react-router-dom'

```
<Route path="/" → if path starts with '/'.
       render={() => { ... JSX ... }} />
```

- render contains a function which outputs the JSX when this Route becomes active

Note : we can use multiple Routes for same path eg.

```
<Route path='/' render={() => <h1> Home 1 </h1>}
<Route path='/' render={() => <h2> Home 2 </h2>}
```

OUTPUT:

Route '/'

Home 1

Home 2.

By mentioning path='/', a Route only checks if a path starts with '/' however it doesnot require it to be exact for ensuring exact behaviour we need to pass exact as a prop to Route component.

Rendering Components for Routes:

- Instead of using render prop, we use another property of Route namely component as following

```
<Route path="/" component={component_reference}/>
```

Disabling Reloading

normal anchor tags or <a> ... tags leads to entire page reloads. These page reloads cause loss of session / user data.

Thus we donot want to reload page and still route are react SPA.

To prevent reloading problem caused in SPA via use of anchor tags, in such case anchor tag is replaced by

```
<Link to="/">Home</Link>
```

to property in its simplest form can be a string otherwise a javascript object as following

Complex Link to property.

```
<Link to={{
    pathname: '/new-post',
    hash: '#submit'
    search: '?search=something'
}}> HOME </Link>
```

→ for anchor tags (to autoscroll in page)

→ query parameters for inserting query strings

- Routes automatically pushes some properties as props to the rendered component. These mainly include
  - match.
  - history.
  - location

Note: Route props are not passed down the component tree

To pass route props down into tree we use a Higher order component withRouter.

• Absolute vs Relative paths:

By default paths are absolute in nature to build relative paths we need to dynamically generate them

```
<Link to={{
    pathname={this.props.match.url + '/new-post'}
}}> .... </Link>
```

dynamic in nature

• Parameter Passing in Routes

Route parameters allows us to load content dynamically

eg `<Route path='/post/:id' exact component={Post} />`

Route parameter.

Now this Route can be activated by wrapping <Post /> component inside a <Link> component as follows

• Retrieving Route parameters

this parameter is accessible in this.props.match.param object

all this is passed by Router to the component

• React Router : parameters

To load component data dynamically such as a particular post. The id of post can be mentioned on link and retrieved in a component. This allows to load information related to that post dynamically.

STEP 1 : Mentioning the Routes inside BrowserRouter

```
import React, { Component } from 'react'
import {Route} from "react-router-dom"
import Home from "../Home/Home"

import Post from "../Post/Post"
import PostInfo from "../PostInfo/Post"

export default class App extends Component {
    render() {
        return (
        <React.Fragment>
            <Route path="/" exact component={Home}/>
            <Route path="/post" exact component={Post}/>
            <Route path="/post-info/:id" exact component={PostInfo}/>
        </React.Fragment>
        )
    }
}
```

→ Route path having id as parameter.

STEP 2 : Generating such dynamic parameterized links

```
import React from 'react'
import styles from "./Post.module.css"
import {Link} from "react-router-dom"

export default function Post(props) {
    return (
        <Link to={"post-info/"+props.id}>
            <div className={styles["main_wrapper"]}>
                <h3>
                {props.title}
                </h3>
                <p>
                {props.body}
                </p>
            </div>
        </Link>
    )
}
```

- using <Link> as a wrapper

to = { "post-info/" + props.id }.

static Route

dynamic prop

STEP 3 : Rendering dynamic content inside components by retrieving parameters

that parameter is accessed in props.match object

```
componentDidMount(){
    let id=this.props.match.params.id
    console.log(this.props)
    axios.get(`https://jsonplaceholder.typicode.com/posts/${id}`).then((response)=>{
        console.log(response.data)
        this.setState({post:response.data})
    })
}
```

- Router also puts in some props / properties into rendered components. Using match object of one such prop we retrieve dynamic parameter to load post information dynamically.

• React Router : query

unlike parameter which can pass a single field dynamically. query allows to send multiple key value pairs

• Route for query based component is same as any other component.

• STEP: Generating Links containig queries

```
export default function Post(props) {
    return (
        <Link to={{
            pathname:"/post",
            search:"id="+props.id
        }}>
            <div className={styles["main_wrapper"]}>
                <h3>
                {props.title}
                </h3>
                <p>
                {props.body}
                </p>
            </div>
        </Link>
    )
}
```

query parameters is mentioned in search key of to prop of Link component.

• STEP : Retreiving data inside component :

```
componentDidMount(){
    let param = new URLSearchParams(window.location.search)
    let id = (param.get("id"))
    axios.get(`https://jsonplaceholder.typicode.com/posts/${id}`).then((response)=>{
        console.log(response.data)
        this.setState({post:response.data})
    })
}
```

inbuild JS object to retreive query from URL.

extracts value of particular query.

- Switch : to load a single Route

By default Router will load all Routes that matches the URL. Sometimes this is the desired phenomenon but othertimes it can lead to unexpected results.

- To load single first matching Route we use Switch

```
<Switch>
  <Route ...../>
  <Route ...../>
    :
</Switch>
```

- Navigating Programatically :-

instead of using <Links to move, we can use history object passed via props. It contains various methods such as push, goback etc. Mostly used to progrimatically change UI once some operation is completed.

Understanding Nested Routes :-

App.js

```
<Route path = '/' exact component = {Home} />.   (1)
<Route path = "/posts" component = {something} />.
```

(2) Inside Home Component :

```
Render ( ) {
  return ( <React.Fragment>
    <div> Home Component </div>
    <Route path = '/new' component = {Posts} />
    </React.Fragment>
  )
}
```

- (1) let us consider an incoming url /new . Initially only only one Route is rendered labelled as (1).

As Home component is rendered it contains a nested Route. /new since url contains /new . This Route will also be rendered.

- url. /new

(match).
<Route path = '/' exact component = {Home} ✓ (match).
{ Home .
<Route path = '/new' component = {Post} /> ✗ (Fail).
(match).

- url. /old.

- To create nested Routes such as /posts
  /posts/new

we need to use.

{this.props.match.url}

- Redirects

to redirect from one Route to another Route. We use a component called Redirect provided by React Router.

<Redirect from='/' to='/posts' />

- Conditional Redirects :-

To conditionally redirect we can use (this.props.history.push) and push a Route

- push → pushes this to stack so that if we use back button it moves back.
- replace → replaces current url/page but back option will not work since it does replaces current entry on STACK.

• Navigational Guards :-

These are some routes which are required to be accessed only via authenticated users. Thus such routes need to be guarded.

① creating a guard

- inside App.js

```
class App extends React.Component {
    state = { auth: false }
    render() {
        return (
            { this.state.auth ? <Route path="secure"
                component={Dashboard} /> : null }
```

if auth is not true Route will not be rendered and thus its related component will not also render.

• Rendering Unknown Routes (404)

unknown route request can be handled using <Route component={...} /> thus without using path property Note: This rule should be mentioned at the last of all Routes.

• Loading Routes Lazily

- Lazy loading can be done using HOC but this depends upon webpack very closely. Hence is not very convinient to follow.

• Lazy Loading with React Suspense:

React Lazy is a method added with React 16.6 that allows to load components asynchronously. It allows to defer loading of code until that component is required.

It is useful for Routing and conditional rendering.

• Using Lazy Loading.

① → For component that needs to be loaded Lazily instead of Normal import we need to use dynamic imports

```
const Posts = React.lazy(() => import('./container/Posts'));
```

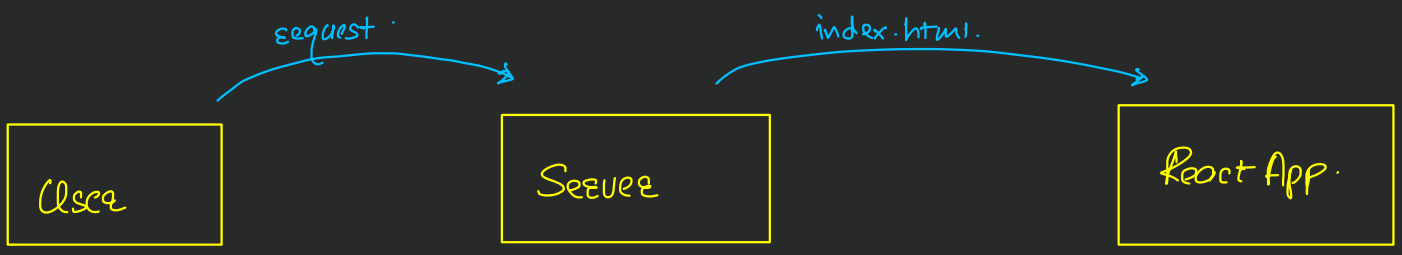
② Using Lazy Loaded component through Suspense.

React.Suspense

```
<Route path='/' render={() => {
<Suspense fallback={<div>Loading</div>}><Posts/></Suspense>
}} />
```

- Lazy loading can drastically improve performance of an app

• Routing and Server Deployment :-



- user sends request to server ; server handles all requests first. But it is the React App that knows about the Routes

- The development server of React is preconfigured to handle all such requests

- We need to configure our development server in such a way that for every request it returns the index.html page. The Route is handled by React Router.

• Hosting Issues

if React app is hosted somewhere other than the root directory then there needs to be certain modifications incorporated within React App configuration.

```
<BrowserRouter basename='/my-app'>
    .... <Route> .... </Route>
    .....
</BrowserRouter>
```

use property basename.
incase our React App is being served from sub. directory of ..../app.

- Forms and Form Validation