GraphQL is helps you with overfetching so load only the data which is required.

SEPTEMBER

	M	T	W	T	F	S	S
O			01	02	03	04	05
C	06	07	08	09	10	11	12
T	13	14	15	16	17	18	19
25	20	21	22	23	24	25	26
	27	28	29	30	31		

FRIDAY 05
Week 36 Day (248-117)

Appointments Here Link is parent jsx some we key on it.

09
Episode-08 - Let's Get Classy
10

Why should we learn class based component?

11
(1) It is asked in interviews a lot. Cause
12 a lot of companies working on older projects which uses class based component a lot.

13
(2) Understanding of react will become stronger
14 if we learn class based component cause if we learn how we write code traditionally
15 then today's concept understanding becomes stronger.

16
(3) It will exactly tells you how react cycle
17 works and it gets you ~~how~~ understand it deeply.

Q What is SPA?

SPA stands for single Page Application. It's a type of web ~~page~~ application or website that interacts with the user by dynamically rewriting the current ~~web~~ page rather than loading entire new pages from server. In other words, a single HTML page is loaded ~~and~~ initially And then the content is updated dynamically as the user interacts with the app, typically through javascript.

OCTOBER NOVEMBER DECEMBER

Appointments

Key characteristics of SPAs include :-

09

Dynamic updates :- In SPAs, content is loaded
10 and updated without requiring a full page reload. This is achieved using JS
11 and client-side routing.

Smooth User Experience :- SPAs can provide smooth
12 user experience & also more responsive because they can update parts of the page without
13 the entire page needed to be refreshed.

Faster Initial Load :- While the initial load of
14 SPA might take longer as it downloads more Javascript assets, subsequent interactions
15 with the application can be faster because only data is exchanged with the server &
16 not entire HTML pages.

Client-side Routing :- SPA often uses CSR to
17 simulate traditional page routing while staying on same HTML page.

7 SUNDAY API Centric :- SPAs are often designed to be more API centric, where the client communicates with a backend API to fetch and send data usually in JSON format. This allows decoupling the frontend and backend.

State Management :- SPAs often uses state management libraries (eg :- Redux for React or Vuex for Vue) to manage the application's
2025 state and data flow.

Appointments

CSR (Client-side Routing) VS SSR (Server-side Routing)

09

CSR In CSR, the routing & navigation are
10 handle on the client side, typically within
the web browser. JS frameworks and libraries
11 such as react router or Vue router used for
CSR.

Faster Transitions: CSR allows for faster transitions
b/w page since it doesn't require server to
send ~~load~~ a new HTML page for each route chang.
Instead, it updates the DOM and URL
14 dynamically without full page reloads.

SPA (SinglePage Application): CSR is often associated
15 with SPA where the initial page is loaded
and subsequent page ~~challenges~~ changes are
16 made by updating the content using JS.

SEO-challenges: SPAs can face challenges with
12 SEO because search-engine crawlers may not
fully index the content that relies heavily on
CSR: Special techniques like SSR or pre-rendering
can be used to address this issue.

Route Management: Routing configuration is
typically defined in code and managed on
the client side, allowing for dynamic
and flexible route handling

09 TUESDAY
Week 37 Day (252-113)

SEPTEMBER

S	M	T	W	T	F	S	S	M	T	W	T	F	S	S
E	01	02	03	04	05	06	07	08	09	10	11	12	13	14
P	15	16	17	18	19	20	21	22	23	24	25	26	27	28
25	29	30												

Appointments SSR

09 Handling on server :- SSR manages routing and navigation on the server. When a user
10 requests a different URL, the server generates and sends a new HTML for that route.
11 Slower Transitions :- SSR tends to be slower in terms of page transitions compared to
12 CSR, as it involves full page reloads.
Traditional websites :- SSR is commonly
13 used for traditional multi-page websites where each page is separate HTML docume-
14 -nt generated by the server.
SEO-friendly :- SSR is inherently more
15 SEO-friendly as each page is a separate HTML document that can be easily
16 crawled and indexed by search engines.
Route-configuration :- Routing configuration
17 in SSR is typically managed on the server and URLs directly correspond to individual
HTML files or routes.

Q What is config.js & why it is important?

Config.js is a central place to store applica-
-tion configuration values that
- change b/w environments (dev / staging / prod).
- are reused across multiple files.
- should not be hardcoded everywhere.

2025

Appointments

Importance

- Avoid hard coding values

09 fetch ("https://api.example.com/users"); X wrong

10 import { API_BASE_URL } from "@/config";

11 fetch (`${API_BASE_URL}/users`); ← RIGHT

12

- Environment specific configuration

13 Environment — Example.
Development — localhost API
14 Staging — test server
Production — live server

15

export const API_BASE_URL =
16 process.env.NODE_ENV === "production"
? "https://api.prod.com"
17 : "https://localhost:3000";

- Improves maintainability
  - Easier to read, update & for new developers to understand.

- Better security & best practices
  - Public config
  - Private secrets (via .env)

export const PUBLIC_API_URL = process.env.NEXT_PUBLIC_API_URL

Never hardcode Tokens, APIkeys, secrets.

2025

Appointments Helps with scaling the app

09

As your app grows

10 ✗ Without config file :

- Magic strings everywhere

11 - Risky changes

- Bugs during deployment

12

✓ with config file :

13 - Clean architecture

- Easy refactoring

14 - Safer releases

15 - Easier testing & mocking

You can swap config values for tests.

16

```
jest.mock("@/config", () => ({
17  API_BASE_URL: "http://mock-api"
}));
```

- Improves consistency

Same values everywhere → no mismatch

```
export const DATE_FORMAT = "DD-MM-YYYY";
```

Config.js Vs .env file

Aspect	config.js	.env
(1) Purpose	App-level config & constants	Environment specific secrets & variables
(2) Stored in repo	Yes	No (usually)
(3) Contains secret	Never	Yes

Appointments

	config.js	.env
Read by	Your JS/TS code	Build/runtime via process.env
Environment Specific	Sometime	Always
Editable without rebuild (production environment)	X	dependson platform

How to make class based component?

Syntax :-

```
import React from "react";
class class_Name extends React.Component {
  render() {
    return (
      <div className="user-card">
        <h2> Name : Thoon </h2>
      </div>
    );
  }
}

export default class_name
```

Appointments

09 A class based component is a javascript class
that extends react.component and uses
10 render() method to return a piece of jsx.

11 And now to import in another
component & same like functional component.
12
import class_name from 'file path'
13 <class_name />

14 How to pass props in class based component?
<~~class_name~~ />
15 Suppose my class name. userClass.

16 <UserClass name = {"John(Class)"} />

17 Now in our class component.

14 SUNDAY

```
class userClass extends React.Component {

  constructor(props){
    super(props);
    console.log(props);
  }
  render(){
    return(
      <div className="user-card" />
      <h2> Name: {this.props.name}</h2>
    </div>);}}
```

Appointments

To pass the props we do the same like functional component but to use it we have to use constructor & props) & super(props) & then to use it in jsx we will use this.props.name

In javascript, a constructor is used to initialize the class.

Props are passed from the parent component. React passes them into your components constructor. To use them inside the constructor we must receive them as a parameter.

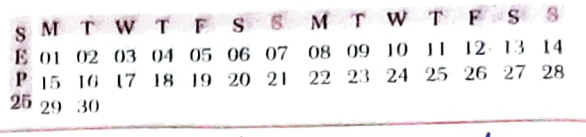
Why we call super(props)
If a class extends another class we must use super before using this

<UserClass name={"John(class)"}/>

React internally convert it to
new UserClass({name: "John(class)"});
so the props object is
props = {name: "John(class)"}

Why constructor(props) exist?
- props is received from React
- The constructor runs when the component instance is created.

2025



Appointments

This is the only place where props comes as a parameter.

Why super(props) is required?

Javascript rule

Our class extends another class

Class UserClass extends React.Component

React.Component is the parent & userClass is child then super(props) does 2 critical things here.

① Create 'this'

Until super() runs this doesn't exist

② Assign props to the instance

** Inside 'React.Component' Class

```
constructor(props) {
    this.props = props
}
```

super(props); → gives access to props received in parent class

this.props = {name: "John(class)"};

Why this.props (not props)

render() does not receive props as an argument. So <h3>{props.name}</h3> fails

Appointments

Why this.props works?
- this = component instance
- React stored props on the instance
- So we access them via this.props.name

Why this.props exist?
Class components are OBJECT
const instance = new UserClass(props);
This object stores
instance.props
instance.state
instance.render
So inside the class
this === instance
That is why this.props.name

One line answer

- constructor(props) — receives props during component creation
- super(props) — initialize parent class & set this.props
- this.props — props are stored on the component instance

→ Before UserClass can use 'this' JS say constructor of React.Component must run. That's where super(props) comes in

When you rendering a function component or loading a function component that means you are mounting or invoking that function into that webpage.

Appointments

09 When React does
new UserClass(props)
JS executes constructor(props){ super(props)
10 This calls React.Component.constructor(props)
which runs this.props = props.

11

Why is it not available directly to UserClass
12 because JS forbid using this before super().

13

We can do this without using
14 constructor(props){
super(props)
15 } as JS does it automatically.

16 so we can do
render(){
17 <h3>{this.props.name}</h3>
}

Q How we can create state variables in class based component? (One webpage)

When loading class based component that means we are creating an instance of the class. When we create instance the constructor is called & which is the best place for : props & creating state component.

When loading a class based component on a webpage that means the creating instance of class.

SEPTEMBER

M	T	W	T	F	S	S
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

FRIDAY 19
Week 38 • Day (262-103)

Appointments

```
constructor (props) {
    this.state = {
        count : 0,
        count2: 2
    }
}

render () {
const { count, count2 } = this.state
return (
    <h1> count: {count} </h1>
    < h1> count2: {count2} </h2>
    <button onClick={ () => {
        this.setState ( {
            count : this.state.count +1 })
    }}> count Increase </button>
);
}
```

We can pass as many state variables or update state in this.state or this.setState respectively. The this.setState will only update state variable passed on it & the rest will not be updated.

2025

OCTOBER NOVEMBER DECEMBER

Appointments

09 this.props Vs this.state

10 React.Component is a base blueprint, it provides the props but not state
11 variables
this.props = props
12 this.state = null

13 It defines the concept of state & setState
but the value & variable are defined
14 by child class

15 The state is stored on userClass instance,
not inside react component.

16

What happens step by step
17 React create instance

21 SUNDAY const instance = new userClass(props);

At this moment instance = { }
super(props) runs (React.Component)
React.Component.constructor(props)
which does this.props = props.
instance = {
props : { name: "John" }
}

loading and mounting is same.

SEPTEMBER

O	M	T	W	T	F	S	S	M	T	W	T	F	S	S
C			01	02	03	04	05	06	07	08	09	10	11	12
T	13	14	15	16	17	18	19	20	21	22	23	24	25	26
25	27	28	29	30	31									

MONDAY 22
Week 39 Day (265-100)

Appointments

09 ~~Base~~ constructor runs (userClass)
this.state = {count: 0}

10 instance = {
props: {name: "John"}
11 state: {count: 0}
}

12 this === instance
so, we use this.props or this.state
13 for same.

14

Aspect	props	State
who provides	React/Parent	Component itself
who initializes	React.component	userClass
Mutability	Read Only	via setState
16 Purpose	Input data	Internal data
Ownership	External	Internal

17

Imp. Never update state variables directly it will only update with setState function

LifeCycle of React Class based component.

Whenever the parent component (here About) is loaded & jsx is rendered then when it see UserClass, It starts loading the userclass component and the first thing is constructor is called. to create the new instance of class and render() function of class is called.

OCTOBER NOVEMBER DECEMBER 2025

You can also import { component } from "react"

class UserClass extends Component { }

Appointments

09 Suppose our parent is also class component

About.js

```
class About extends Component {
  constructor (props) {
11    super (props)
    console.log ("Parent Constructor");
12  }

13  componentDidMount () {
    console.log ("Parent Did Mount");
14  }

15  render () {
    console.log ("render Parent");
16    return (
      <div>
17      <h1> About </h1>
        <UserClass />
      </div> );
    }
  }
```

UserClass.js

```
class UserClass extends Component {
  constructor (props) {
    super (props)
    console.log ("Child constructor");
  }
```

Appointments

```
09  componentDidMount () {
      console.log ("child did mount");
    }
10

11  render () {
      return console.log ("child render");
12    return (
        <div>
13        <p>hello</p>
          </div>
        );
14    }
15  }
```

16 Console.

Parent Constructor
17 ~~Parent~~ render ~~Parent~~ Parent
Child constructor
Child render
Child did mount
Parent Did mount

Because the componentDidMount() method called when the component mount / rendered fully succesfully and the Parent will mount succesfully once the child is mounted 'cause child is included in parent div.

Appointments

Use Case of componentDidMount()

- To make api calls cause it should call after the component is mounted.

as in functional component in useEffect, we make api calls and useEffect is called after the rendering of component.

So,

as the user will skeleton until api returns actual data instead of showing blank page.

So, componentDidMount call after the component is rendered then we will re-render the component after the API call.

What will happen when we have two children in our parent

In About.js

```
render () {
  <div>
  return (
    <div>
      <h1> About </h1>
      <UserClass name = "First" />
      <UserClass name = "second" />
    </div> ); }
```

In console.log (this.props.name + " child constructor");
" " " Render
" " " did Mount

SEPTEMBER

O C	M	T	W	T	F	S	S	M	T	W	T	F	S	S
T			01	02	03	04	05	06	07	08	09	10	11	12
25	13	14	15	16	17	18	19	20	21	22	23	24	25	26
	27	28	29	30	31									

FRIDAY 26
Week 39 • Day (269-096)

Appointments

09 Console.

Parent Constructor
10 Parent render
First Child Constructor
11 First child render
Second child Constructor
12 Second Child render → DOM Manipulation begin
First child Did Mount
13 Second Child Did Mount
Parent Did mount

Render Phase
Commit Phase

14

React updates the lifecycle in 2 phases
15 ① Render Phase.
where react find the difference b/w
16 virtual & actual DOM run the reconciliation
algo to get to the correct DOM by
17 converting them into object and finding
the difference which is a fast process
that's why it batches the render phase together.
and that's why our console have
render first.
② Commit Phase where it actually updates
the DOM as it is a time taking process
so react does it later and updates the DOM
and also componentDid Mount after
That's why react is fast and our console
was like this
So react batch render phase to make & optimize faster rendering.

2025

OCTOBER NOVEMBER DECEMBER

27

Week 39

SATURDAY

Day (270-095)

SEPTEMBER

S	M	T	W	T	F	S	S	M	T	W	T	F	S
E	1	02	03	04	05	06	07	08	09	10	11	12	13
P	15	16	17	18	19	20	21	22	23	24	25	26	27
25	29	30											

Appointments

09

10

11

12

13

14

15

16

17

28 SUNDAY

2025

unmounting

componentWillUnmount

updating

newProps setState() forceUpdate()

shouldComponentUpdate

render

React updates DOM and refs

componentDidUpdate

Mounting

constructor

render

componentDidMount

Render Phase
Pure no side effects may be paused, aborted or restarted by react

commit Phase
can work with DOM, schedule update & run side effects

Appointments API call in class based component.

09

10 Class UserClass extends Component {
constructor (props) {
11 super (props)
this.state = {
12 userInfo : {
name : "Dummy"
13 location : "Default"
}
}
14 }
}

async componentDidMount () {
const data = await fetch ('')
16 const json = await data.json
this.setState ({
17 userInfo : json
}) ; }
componentDidUpdate ( ) { console.log ("component updated") }
render ( ) {
~~rel~~
const { name, location, avatar_url } = this.state.
userInfo;

return (

<h2> name : {name} </h2>
<h2> location : {location} </h2>

) ; } }

30 TUESDAY Step by Step lifecycle.

Week 40 · Day (273-092)

SEPTEMBER

S	M	T	W	T	F	S	S	M	T	W	T	F	S
E	01	02	03	04	05	06	07	08	09	10	11	12	13	14
P	15	16	17	18	19	20	21	22	23	24	25	26	27	28
25	29	30												

Appointments

--- MOUNTING ---.

09 * constructor (with dummy data)

* Render (" " ")

10 <HTML Dummy>

* Component Did Mount

11 <API Call>

<this.setState> → State variable is updated

12

--- Updating

13 * render (with API data)

<HTML (new API data)>

14 * component Did Update

15 # NEVER COMPARE REACT Class lifecycle

16 with Functional Component

useEffect ≠ componentDid Mount.

17

Traditional writing code.

after running componentDid Mount then we run componentDid update.

```
componentDidupdate (prevProps, prevState) {
    if ( this.state.count === prevState.count||
         this.props.name === prevProps.name)
    {
    }
}
```

2025

and we need to do this if, else if condition for every state variable.

Action Plan

Schedule your plan .....For Better future.

October 2025

That's why we have dependency array in useEffect so that we can have multiple state variable in array. [count, count2]

useEffect(() => { }, [count, count2]);

=> CONS of single Page Application

Suppose we have setInterval in About.js then we to get print Hello in every seconds and we put this in componentDidMount and then suppose the user will go to Home page then it will still print Hello by restarting interval even though we are not in About page. So, here component will unmount comes in place we need to clear Interval so it will only call when we are in About page only

componentDidMount() {
this.timer = setInterval(() => {
console.log("Hello");
}, 1000);
}

componentWillUnmount() {
clearInterval(this.timer);
}

Respect
Obey
Love

Respect Acquaintance
Obey Parents
Love Children
are basic qualities for being Human

OCTOBER
NOVEMBER
DECEMBER

It happen because it store in the browser memory as the callback is in browser Web API (JS runtime) so it will run until we explicitly clear it.

Appointments

09 Similar thing, we need to in useEffect()

10 useEffect(() => {

11 const timer = setInterval(() => {
console.log("Hello");

12 }, 1000);

13 return () => clearInterval(timer);
}

14 , unmounting the component this will run.

15

16 Why useEffect does not have async callback?

17 The useEffect hook is designed to handle side effects in functional components. It's a powerful and flexible tool for managing asynchronous operations, such as data fetching, API calls, and more. However, useEffect itself cannot accept an async callback function directly. This is because useEffect expects its callback function to return either nothing (i.e., undefined) or a cleanup function and it doesn't work well with promises returned from async functions. There are few reasons for this:

2 OCTOBER: GANDHI JYANTI

Appointments

Return value Expectation - The primary purpose of the useEffect callback function is to handle side effects and perform cleanup. React expects us to either return nothing (i.e., undefined) from the callback or return a cleanup function. An async function returns a Promise, and it doesn't fit well with this expected behaviour. If a cleanup function were a Promise, React would have to wait for it, unwrap it and guess whether it resolves to a function. That would:

- complicate React's internal
- make effects unpredictable
- introduce race conditions

Execution Order and Timing — With async func, we might not have fine-grained control over the execution order of the async code & the cleanup code. React relies on the returned cleanup func to handle cleanup when the component is unmounted or when the dependencies specified in the useEffect dependency array change. If you return a promise, React doesn't know when or how to handle cleanup.