# ⊛ EPISODE - 01 ⊛ INCEPTION

What is CDN?
↳ Content Delivery Network.
↳ Where react library is hosted

② What is crossorigen? → Sets the mode of the request to an HTTP CORS Request.

③ ~~render~~ → is Responsible for making object into a named HTML element created an object.

→ React Element

→ ④ JSX → Make Developer life is easy.

③ root. Render → is Responsible for putting code inside in root component which includes in body tag with div.

→ Replaced the whole things which previously inside in ~~div~~ body tag. existed code

→ React is only working inside of div, id root. i.e

```
<div id='root'>
</div>
```

→ React is just piece of javascript code.
→ It not full fledged Framework.
→ is a library

# EPISODE-02 IGNITING THE APP

npm → npm does not have full form.
→ is work as node package manager behind the scene..
→ standard Reposatory for all the packages.

Webpack → is like a bundler.
it basically bundles the App and shift to the production.
→ create-react-app uses webpack behind the scene.

Parcel → npm install -D DevDep. parcel → is like a bundler.
2 types Dependency
DevDep. [in Development phase]
normal [used in production also]

"parcel": "^2.8.3"
↳ carrate to manage the ^2.8.3 Major update or minor update
keeps

package.json → is basically approx version.
package-lock.json → ~~[illegible]~~.
keep tracking the exact version..

NODE MODULE → contains all the code that we fetched from npm.

npx → when execute the package with the help of npx.

bpm → is use to install a package.

→ ✱ CDN link is not a good way to do to bring React and React Dom project. rather than
→ we use **npm install react.**
as it a prefered way.

read more
parcel documentation

work

# Parcel → ① Dev Build created
② local server created
③ Refreshing page.

XXX HMR → Hot module Replacement
with the help of HMR, page it autometically refresh after update or change anything in code.

→ parcel is also caching thing and enhancing the performance of web.
→ do image optimization

Read myself]

- Support older browsers.

ode

EPISODE-03 Laying the foundation

① React.createElement ⇒ object ⇒ When we render this element onto the DOM then it becomes the HTML Element

② npm run start ⇒ npm start (for deupt)
npm run build ⇒ npm run build (for prod)
React → using of this, we can create element.
React Dom → using of this, we can display onto the Browser.

import React from 'react';
import ReactDom from 'react/client';

③ React.createElement ⇒ object ⇒ HTML Element (render)

```
const heading = React.createElement(
'h1', { id: "heading" }, "Namaste React" );
```

using JSX → above code written using JSX.

```
const JSXheading = <h1 id="heading">
Namaste React </h1>
```

```
console.log(heading);
const root = ReactDom.createRoot(document.getElementById("root"));
root.render(heading);
```

JSX → React is diff^nt & JSX is diff^t.
JS syntax which is easier to create element.

⊗ JSX is not a HTML in JS

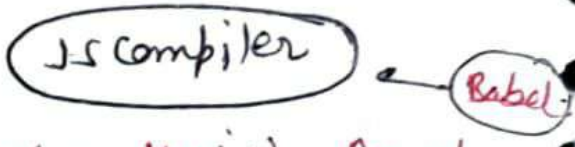
JSX is a like HTML like

---

⊗ JSX is not a HTML in JS

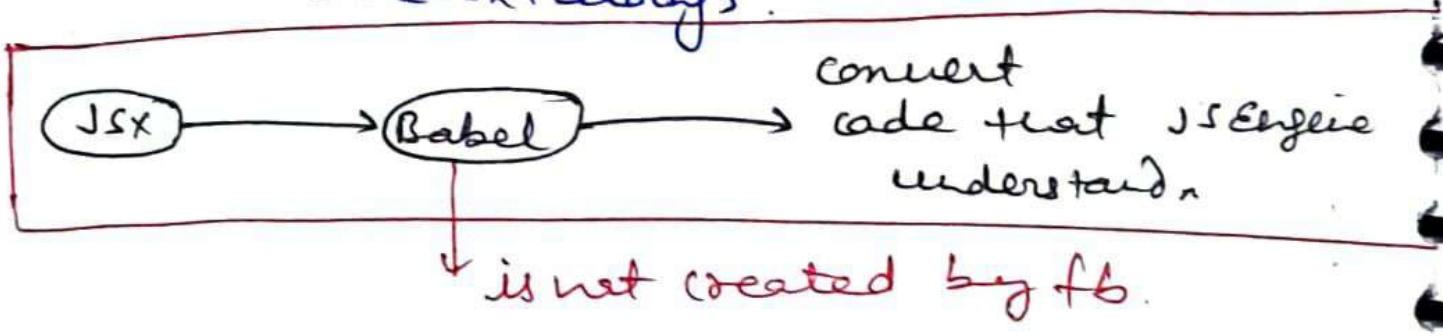
JSX is looks like HTML syntax.

→ JSX is different syntax.




→ JSX transpiled before it reaches the JS - Parcel -

```
const JSXHeading = <h1 id="heading">Namaste </h1>
console.log(JSXHeading);
const root = ReactDOM.createRoot.getElementId("root"))
root.render(JSXHeading)
```




JSX ⇒ React.createElement ⇒ ReactElement -


JS object ⇒ HTML Element (render)

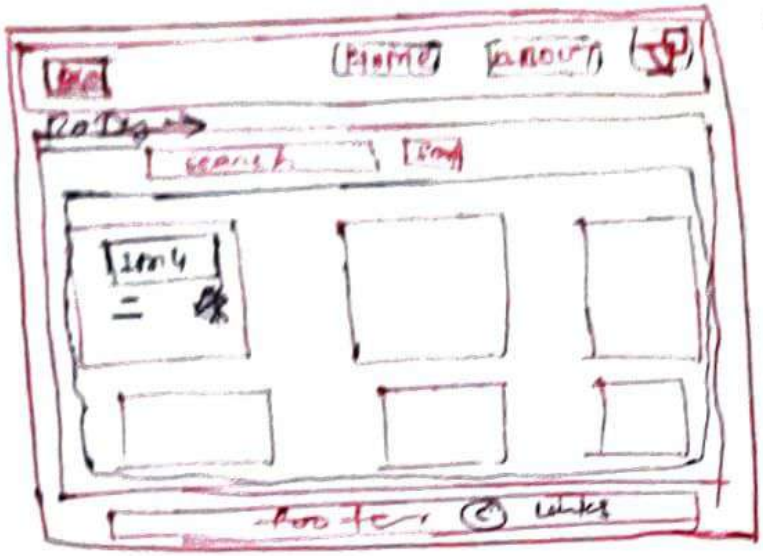
JSX takes camel Case attbutes

Ex → <h1 className="head" /h1>

# EPISODE 04 {TALK IS CHEAP, SHOW me the code!}

chair

## ⓐ build food ordering App.



Components
- HEADER → Logo, Nav
- BODY
- FOOTER → copyright some → links

BODY → Search, Container, Card

HEADER →
- ⓐ Logo
- ⓑ Nav

BODY →
- ⓐ ~~search~~ Search
- ⓑ container
- ⓒ card. →
  - ① image
  - ② Name of Rest
  - ③ * Rating
  - ④ ~~cuisine~~ cuisine
  - ⑤ time for delivery

FOOTER → copyright some links etc.

#probs → just normal argument of an function

↦ A website run by config driven vs # config driven UI.

→ ~~All UI is driven by~~ All UI is driven by config → config comes from backend

# Episode-05 Let's get Hooks

types of export & import

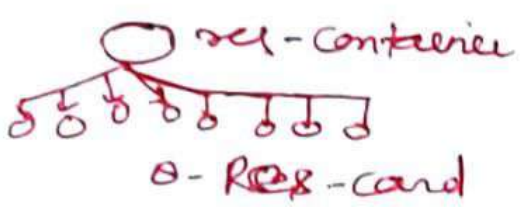
- Default: Export/import
  - export default component
  - import component from 'path'.
- name: export/import
  - export const component;
  - import {component} from "Path"

Hooks → normal JS function
- ① useState() → maintaining the state variable
- ② useEffect()

React is fast because of it can easily manipulating the DOM.

→ Whenever a State variable is changes, React is Re-rendering of components. (whenever something change in UI screen)

React uses "Reconciliation Algorithm, also known as "React Fiber" (in React 16)




→ React create Virtual Dom → Virtual Dom is not Actual Dom, it only Representation of Actual Dom.

Diff Algoryth → ~~React it finds diffrt b/w Actual Dom & Virtual Dom.~~


OLD Virtual Dom  New Virtual Dom

→ Diff Algoryth → it finds the diff^c b/w previous Virtual Dom to new virtual Dom then it ~~actual update~~ update the Actual Dom.

→ Virtual Dom is kind of like a object Representaion. that will ~~represent~~ represent the HTML.

↳ known about new React Fibre from 'React fibre Architecture'.

→ React ~~does~~ does efficient Dom manipulation because it has virtual Dom.

↳ learn from github link

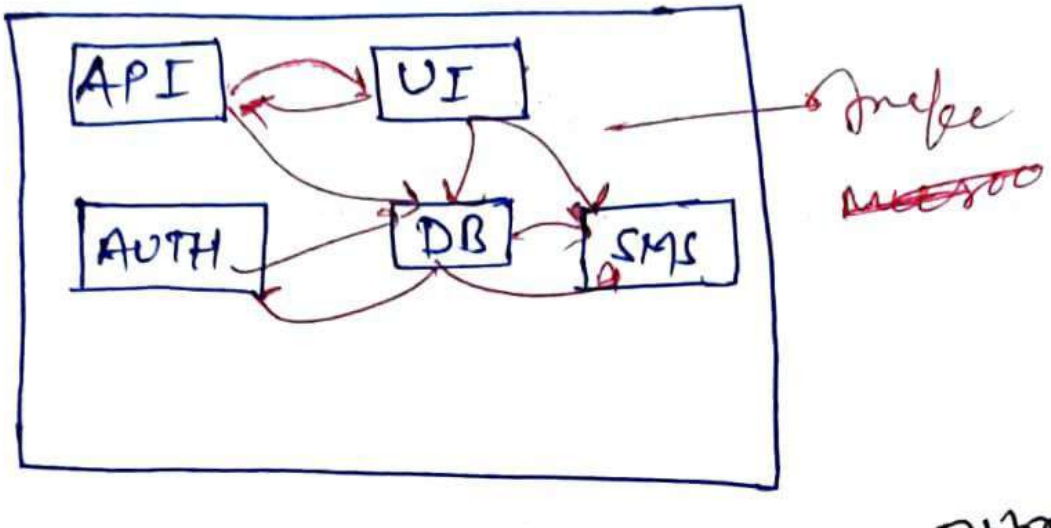
react fibre
(React 16)

↓

it new way of finding the diff & updating the Dom

# EPISODE - 06 EXPLORING THE WORLD.

## → Monolith Architecture →




→ Microservices Architecture → max Approached used.

BE | UI | Auth | DB | SMS | Email

→ based on single Responsible possible

→ API Call → Approach

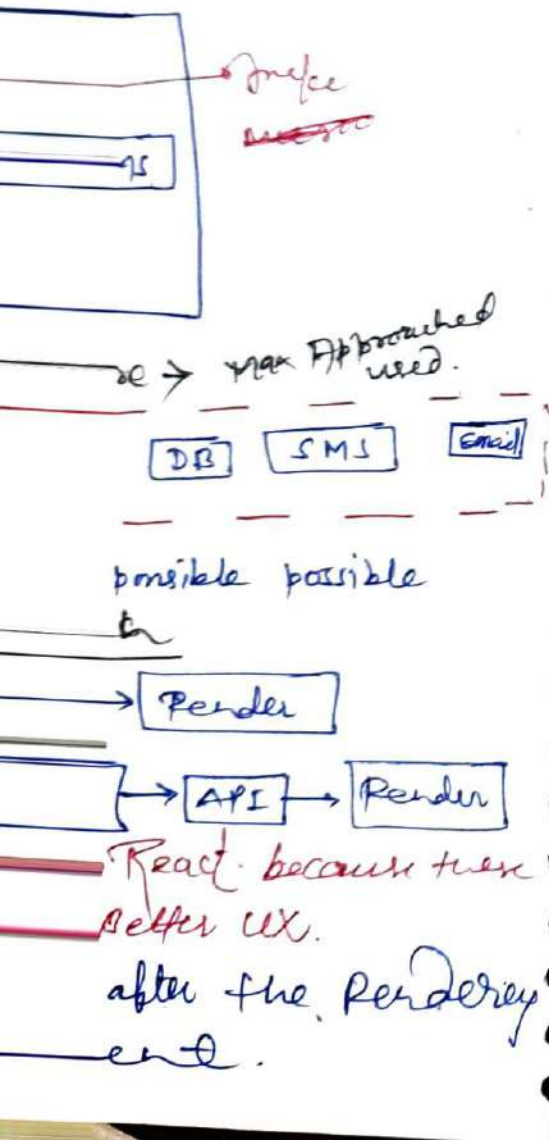
① LOADS → API → Render
(500ms)

② LOADS → Render → API → Render

→ Mostly widely used on React. because these Approach gives us better UX.

Useeffect → CB func called after the Rendering of component.

EXPLORING THE WORLD.




CORS fundamental!!!

→ optional chaining → Home work 6-1

→ optional chaining → it introduce in ES2020.

# primitive value don't have properties.

→ The optional chaining is a safe way to access nested object properties, even if an intermediate property doesn't exist. it is represented by ?.

→ The optional chaining stop the evaluating if the value before ? is undefined or null and returns undefined.

⊛ We usually prefer optional chaining rather use of if & else statement.

⊛ Whenever state variable update, react trigger a reconsilation cycle (re-render component).

# CHAPTER-07 = FINDING THE PATH.

## ⇒ use effect →

→ if no dependency array ⇒ useeffect is called in every Render

→ if dependency array = [ ] ⇒ use effect on initial render (first once)

→ if dependency array is ['btnNameReact'] ⇒ ~~called~~ use effect is called when 'btnNameReact' is updated.

## ⇒ useState →

→ it is use for creating local state variables inside function component on the higher level.

→ never use ~~and~~ inside if else case / conditional ) / for loop / func.

→

# ROUTING SECTION

→ created configuration from 'React-Router-Dom'.

→ create Boowser Router & Router Provider from 'React-Router-Dom'

→ ⊙ there are many way to create Router but A/c to the official docum^t they also recomend to use of create boowser Router.

→ use Router Error hook gives us better msg ⊙ regards Error.

3 create children Route

→ there is outlet comp^nt which provided by React-Router-Dom,

→ outlet are using for → children Route.

# EPISODE-08 (A) GET HETE CLASSY

→ older way to use of react.

→ oldder way to create class based components.

Q Why do we write super(props).

A

Never update State variable. directly. instead of using this.setState({}) I updated if contain the variable.

lyf cycle →

constructor → Render → componentdid mount.

→ ComponentDidMount are used to make Apis call.

→ Parent constructor → Parent render → child constructor → child render → Parent childDidMount → child comp didMount

How does work component life cycle →

Parent constructor → Parent render → first Child Constructor → first Child render → second Child constructor → second Child render → First Child Component did mount → second Child Component Did Mount → Parent Component did Mount

→ Referred → www.w3jtekmai.pl

→ React is fast because of it has two phases → ① Render phase ② Commit phase.

Render phase → Pure and has no side effects. May be paused, aborted or restarted by React.

commit phase → Can work with DOM, run side effect, schedule updates.

→ Mounting cycle happened after that updating cycle happened

↓

→ A Component mounts when it's added to the screen/UI.

→ A Component updates when it receives new props or state, usually is response to an interactions.

→ A component unmounts when it's removed from the screen.

EPISODE-10, JO DIKHATA HAI WO BIKATA HAI.

## SASS & SCSS

- ↳ not recomended way in production ready app^n.
- ↳ only way to make beautiful UI.

→ styled-components.com
↳ recomended way.

→ Material-UI

→ Tailwind

→ ~~chakar-UI~~.
chakra UI.

→ boot strap

→ ant design

ther are many way to design the web page and make awesome user experience.

→ We mostly used tailwind, Bootstrap, scass, material-UI.

# EPISODE-11 #

## # DATA IS THE NEW OIL.

→ HOC → High order component

→ controlled & uncontrolled component.

→ libling the flatup. ~~XXX~~

↳ Read it from React Documentry.

→ Prop drilling.

→ Context

two way

by using a Hook (very common way) with fn comp^nt.

by using user context consumer [with class comp^nt)

EPISODE-12

# LET'S BUILD OUR STORE.

→ Redux is not mandatory.

→

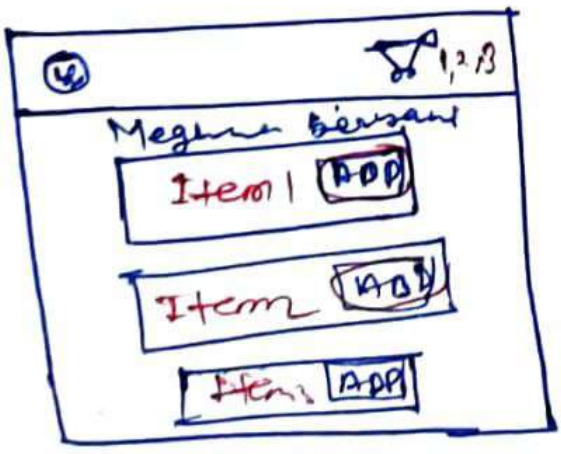
→ Zustand is another option for Storing.

→ Using Redux
  (a) easily debugging
  (b) managing State in a very apropriate manner.

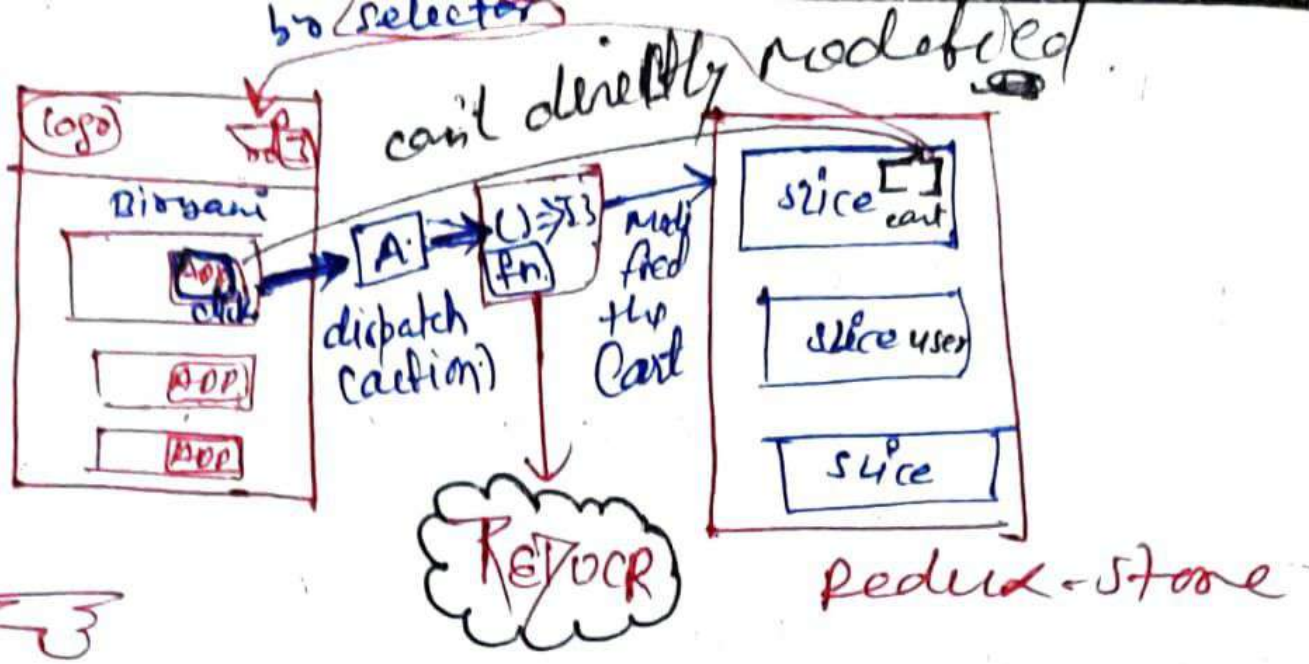
→ it works in many library.

→ Redux Toolkit.

## Architecture of Redux Toolkit →



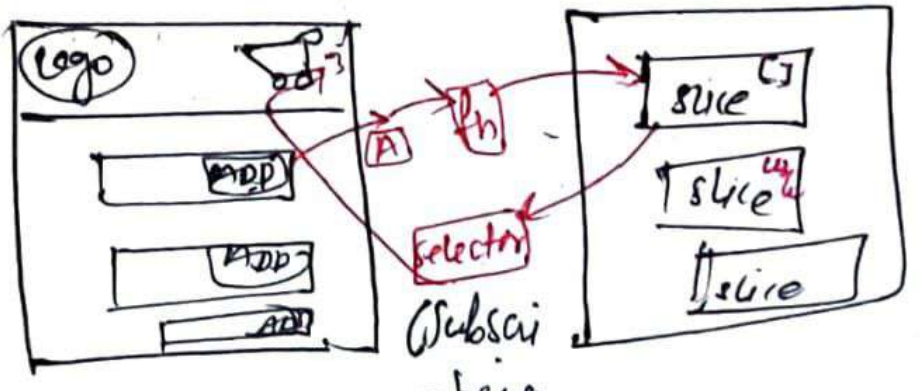

redux is like very big object and it keeping central global place.

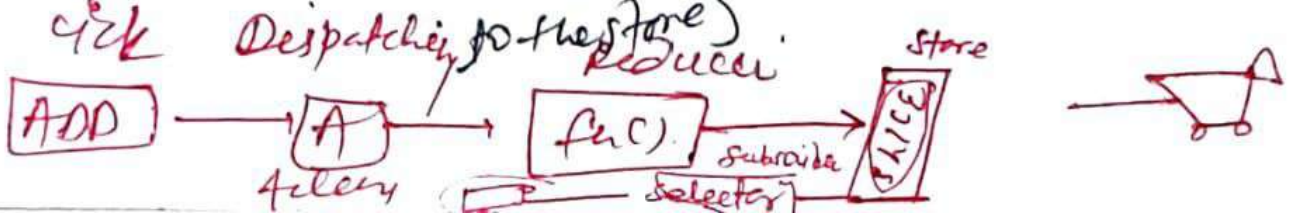



When click on ADD button, it Dispatches an action which calles the Reducer function which update the slice of a Redux Store.

→ SELECTOR → it give the data in cart.

& this phenamon is known as subscribing to the store.



# Redux toolkit Steps →

(a) install library → ~~@reduxjs/toolkit & react-redux~~

@reduxjs/toolkit & react-redux.

(b) build our store.

(c) connect our store to our App.

~~→ add slices to store~~

(d) create slice (cartslice) → Reducer fn^n with {state, action}

→ add naked slice to our store ↳ export cartslice.reducer
↳ export cartslice.actions

(e) dispatch(action)

(f) Selector ⇐ read the data using (selector)

→ configurestore is the redux job.

→ Providing it to react app^n is the react-redux job.

→ use selector gives us access to the store

→ Reducer → Reducer is a function which handle the actions.

useselector((s) => s.cart.items)

s = store

→ Redux uses ~~inside~~ Inner lib. behind the scene. ~~IDM~~

# basic concept of Redux →

→ Redux-toolkit.

→ independent state library.

→ ⟶

Store

① import configureStore from 'redux/toolkit';

export const store = configurestore({})

② import {createSlice} from '@redux/toolkit';

const initialstate = [ ] ;

Reducers

→ UseDispatch → it change the value of store with using of Reducer fn.

→ dispatch (cReducerfn))

→ Redux is core library & React-Redux is nothing but it is implementation for ~~making~~ bridge with react

→ Never mutate State.

Store - Single source of truth.

→ there is no data flow

→ reducer fn get two value i.e (State, action)

State → updated state value in the store.

Action → Action.Payload.

⇒ Redux toolkit uses | Immer | RTs.

Behind the scene.

RTK →

it say that either mutate the existing state or return a new state.

in useselector → carefull about subscribe the specific portion of store.