React JS - Fundamentals and Terminologies

Frontend
JS library
UI/UX
Components
JSX
Hooks
Data Binding
Babel
Webpack
Props
SPA
Containers
Data fetching - Axios
React Router - Routes
Virtual DOM
DOM
Browser APIs
Components — Containers — Router — App — Index.js
Index.html
Flexible
Optimization
JS ES6
Array Methods
JSON
Conditional Rendering - Ternary Operator
Facebook
Npm
Node
Npx
Local Storage
Redux
Create-React-App
Node Modules
Actions
Selectors
Operations
Reducers
Store
Array Destructing
Spread operator
Named Export
Default Export
Object Destructing
State Management
Events
Eventlisteners
Callback function
Closures
Regular, Arrow, Named fun?
String Interpolation
Class-based
Flow of Code
Material-UI
HTML Vs. JSX
Default Server Env. - 3000
Nested Operations
Styling React
Rendering
React Projects

#1. JavaScript Libraries and Framework

→ React JS Core Syntax - JSX.

#2. Key Pointers

→ React JS is a front-end Component-based JS library for building Reusable User Interface.

→ Easy to understand.

→ Excellent Cross-platform Support.

→ Fantastic Community & Most loved JS library.

→ Build fast and Efficient SPA.

→ Stack overflow Developer Survey and State of JS - 2021 - React.js.

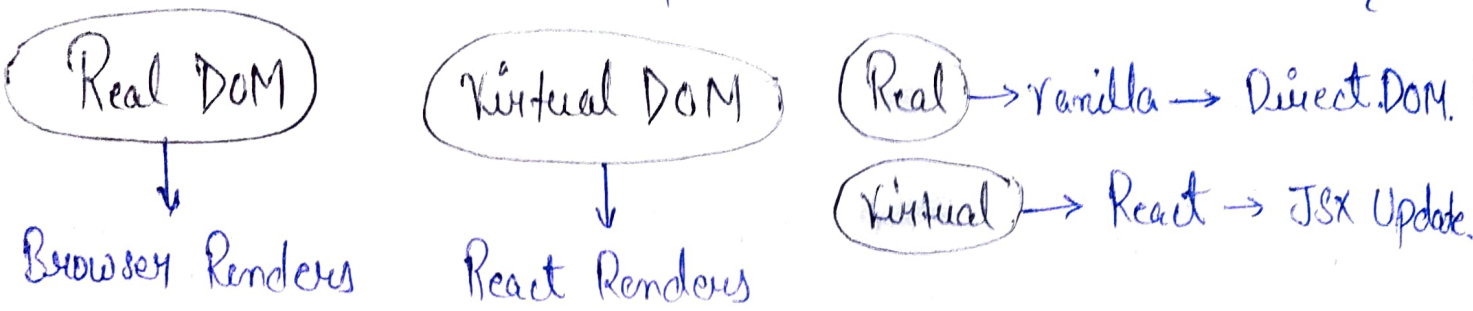
→ Popular JS Stack - MERN. - Programming Stack.

→ Website never reloads. - JS | React.

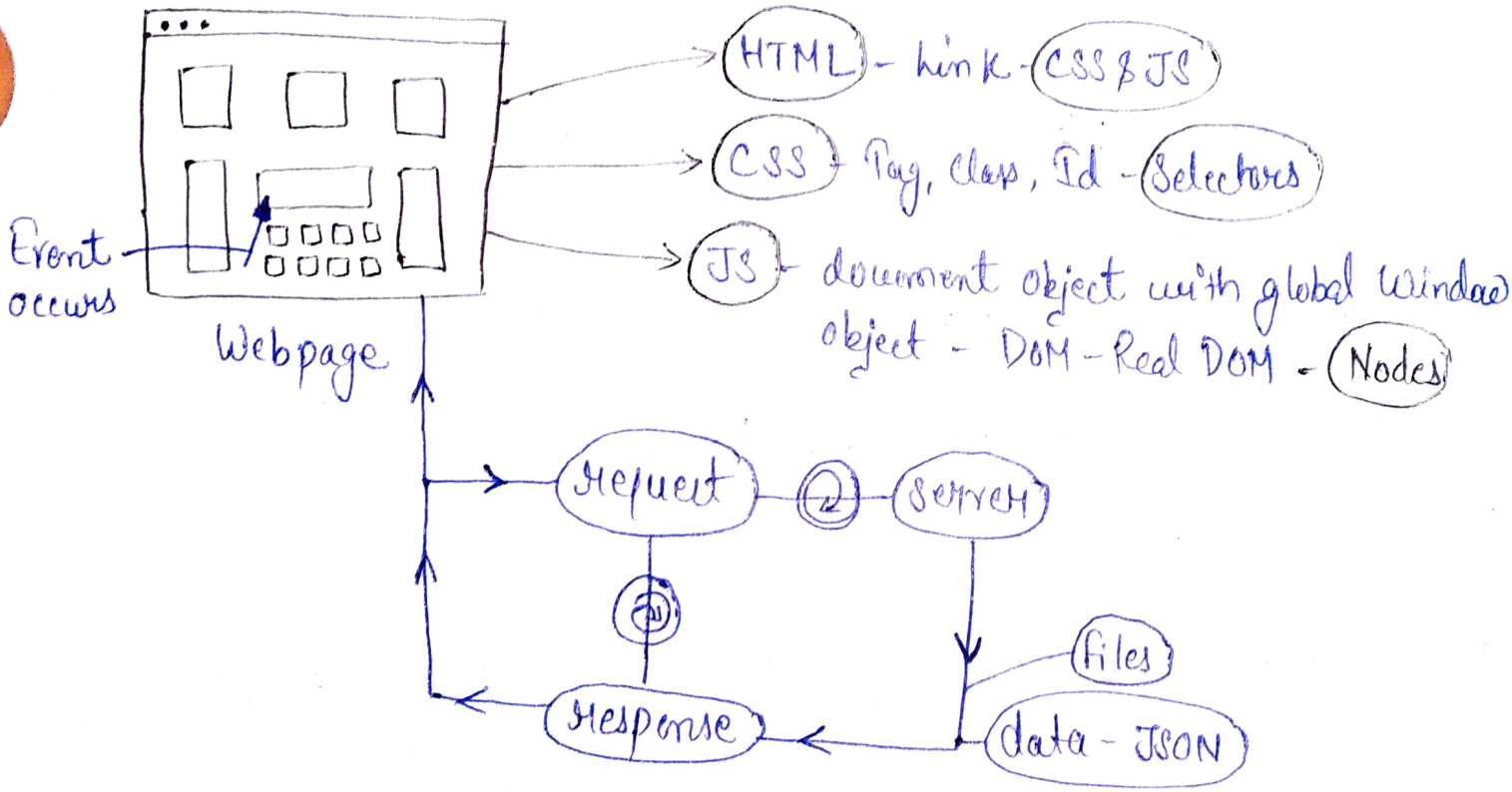
→ Independent and Reusable Components.

→ JSX produces a React Element. Core Syntax of React.

#3. Virtual DOM :- React DOM updates the Real DOM - document Object.



#4. Flow of Website :-



- ★ JS updates the Real DOM Tree Nodes using events / Event listeners.
- ★ HTTPs Request and Response Methods Cycle.
- ★ Functionality, Dynamic, Interactivity, Fetch APIs, Server.
- ★ Dynamic events triggered will respond with Server requests and Internal webpage dynamics. JavaScript ES6 will communicate with defined func. and renders the output of the page.

Event Instantiate → Mounted Component → JSX

↓

Linked Callback function → Component Rerender → Databinding/Babel

↓

Components → Containers → Router → App.js → Index.js (React DOM)

↓

Comparison b/w Real DOM / Virtual DOM using Nodes and Child Nodes.

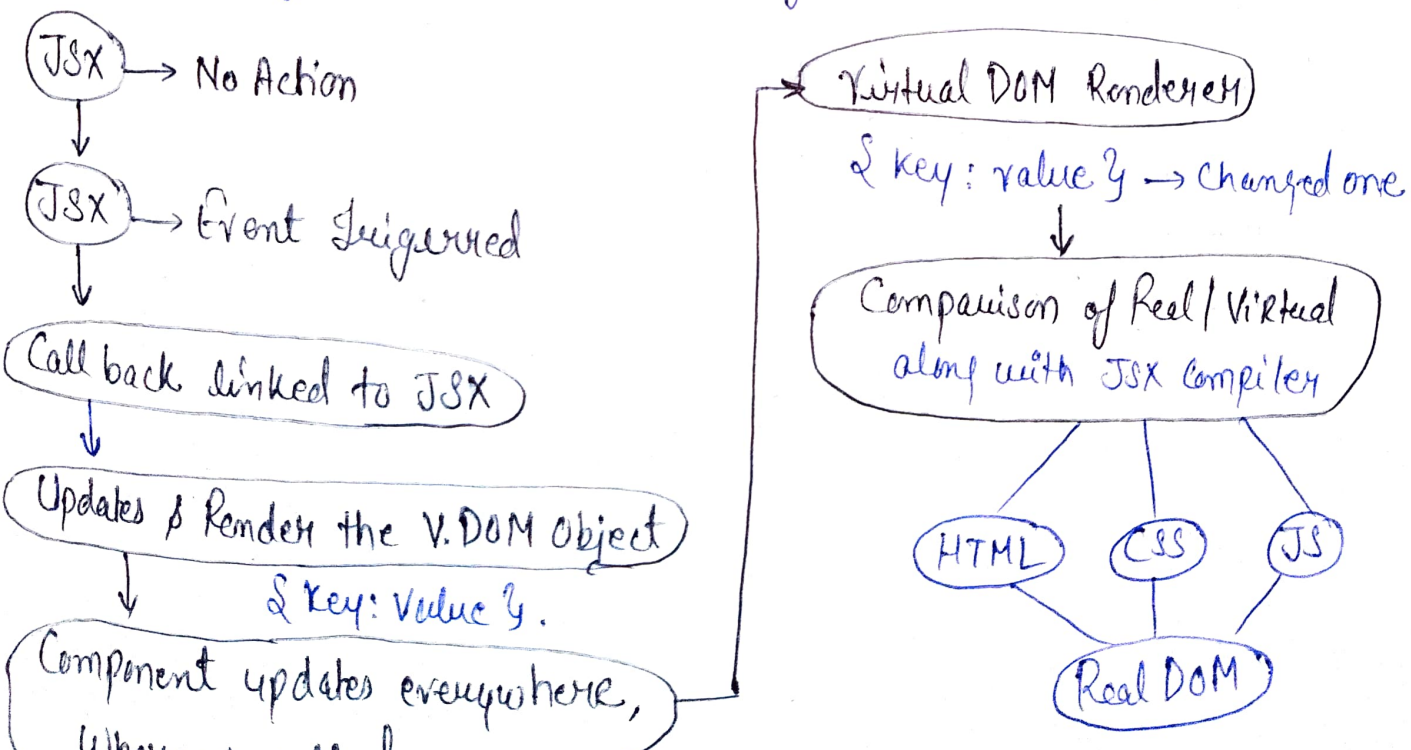
↓

Only Changed JSX/Components will be updated

↓

Babel/Webpack → Browser (Updated Webview)

→ Light Weight DOM Representation of HTML in JS (faster).



#5. Fundamentals

→ Browser DOM API → Direct DOM Manipulation.
→ Components → Class and Functional.
→ JSX → How the UI look like in React.
→ Start React App - Babel / Webpack / Create-react-app tool.
→ Node.js, npm, npx. (Node - JS runtime env. to execute JS on Server).
→ package.json - react, react-dom, react-scripts.
→ Html vs. Jsx - className, embed JavaScript, .jsx, {} valid JS Expr.
→ Dynamically renders the data with no events with virtual DOM.

#6. React App Concepts

→ Files and folder structure.
→ Hello, React App.
→ var declared and initialized and called in JSX. {name}
→ Ternary expression → var defined → True. ⇒ {exp. ? ( ) : ( )}
→ Dynamic behavior using ES6 JS.
→ React Fragment / Adjacent JSX element wrapped - Syntatic Sugar.
→ if userLoggedIn = true / false ? ( ) : ( ).
→ Multiple components can be called / created in same file.
→ null means not defined Value. → defined as nothing.
→

#7. Props (Parent to Child) - Data transfers - Plain object - Args/Obj. used as parameters / Input to hold the Components passed args. attributes. {props.name} <Heat name="No" />

{Property : Value}.

→ Every Component has inbuilt prop object. props as default name.

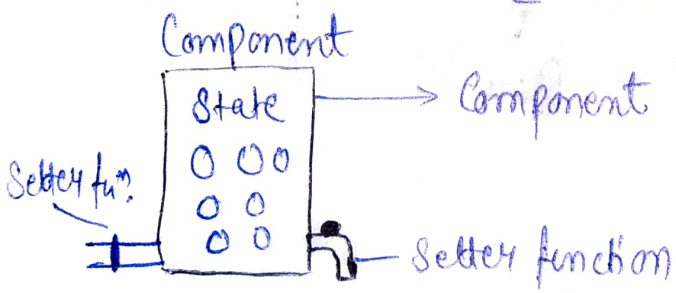
#8. State :- State in React is a plain javascript objects is used to represent the piece of information about the current state Component. It is managed by Component itself.

→ Import {useState} from 'React' → Named Import.

→ Syntax :-

```
Const [ Count, setCount ] = useState(0)
```

State var. — Setter fun? / update fun. — Initial state — Hook Invoking



→ Change in state variables will lead to rerendering of Comp. without loading page. (Browser Sessions).

→ Build Counter App and Usage of usestate Hook.

→ Events — Actions → user generated events / user action.

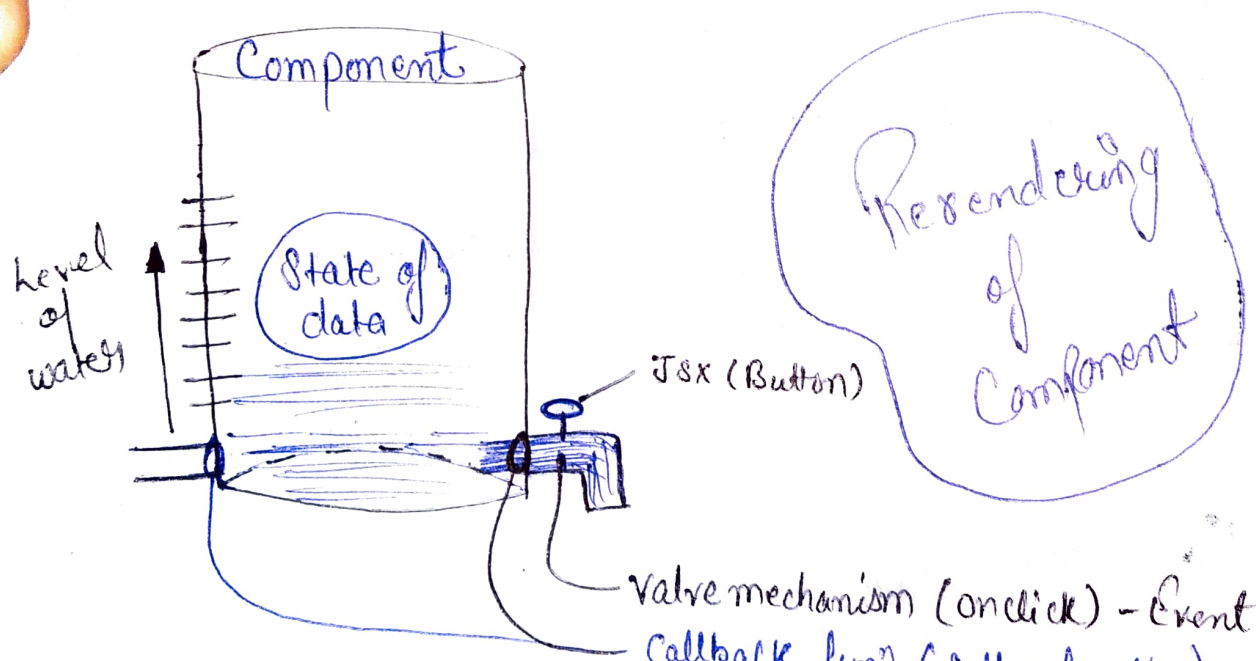
onClick, onHover, onmouseover.

→ Never mutate the State variable manually.

useState Hook → function to manage the current state of Component
Calling setter function leads to change in state variable. It means calling useState Hook.
Rendering of Component without Page Reloading.

→ prevCount → Parameter of setter fun? → refer to State var Value

Analogy of useState

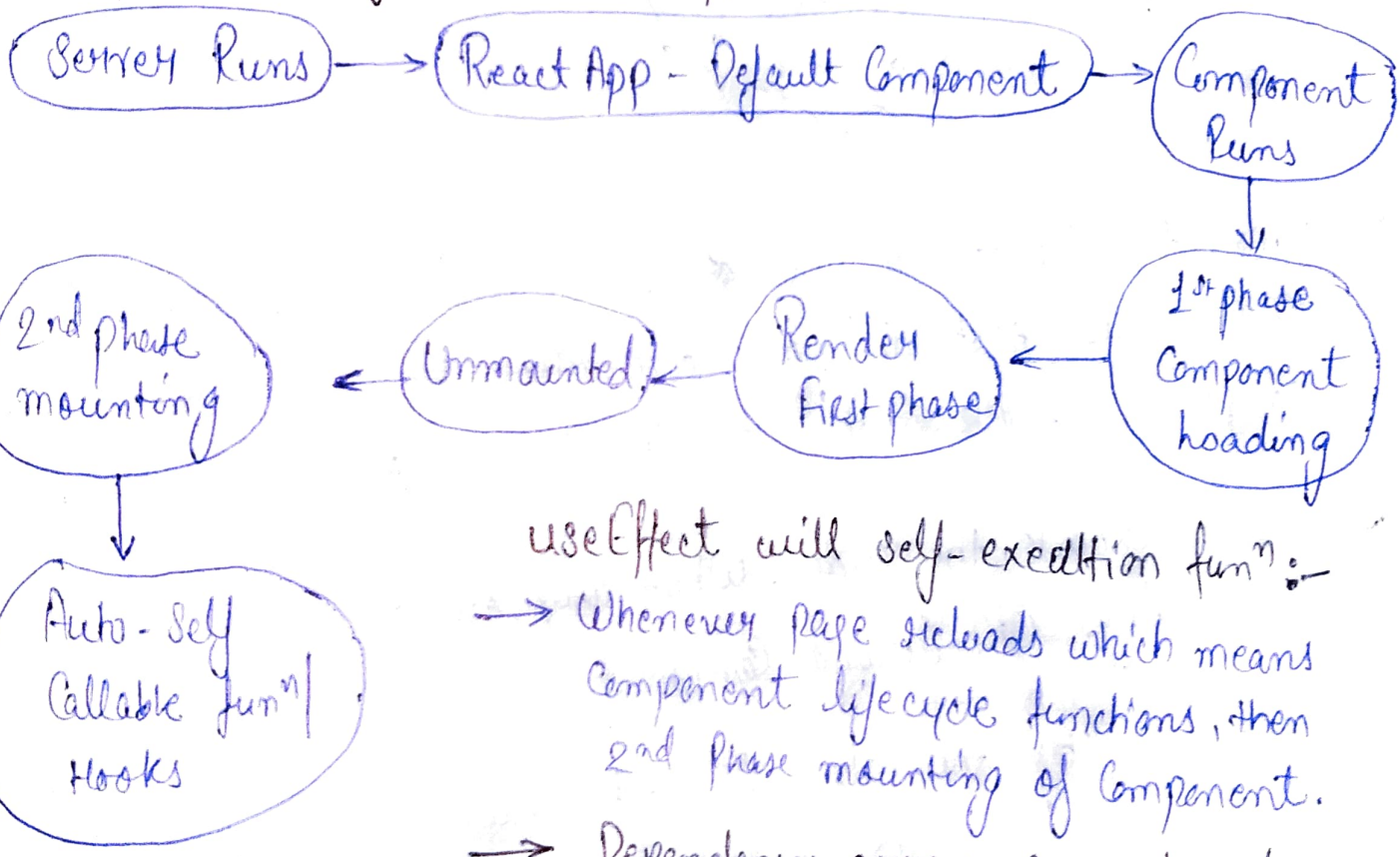


Basic Hooks — useState — useEffect — useContext

Additional Hooks — useParams — useReducer — useCallBack
— useRef — useImperativeHandle — useMemo
— useLayoutEffect — useDebugValue

#9. UseEffect ⇒ Some of the Hooks | fun? runs twice due to StrictMode in ReactDOM render method.
React 18 version works as Mounting, Unmounting, Mounting again. Remove Strictmode.

#10. Component lifecycle with useEffect ⇒

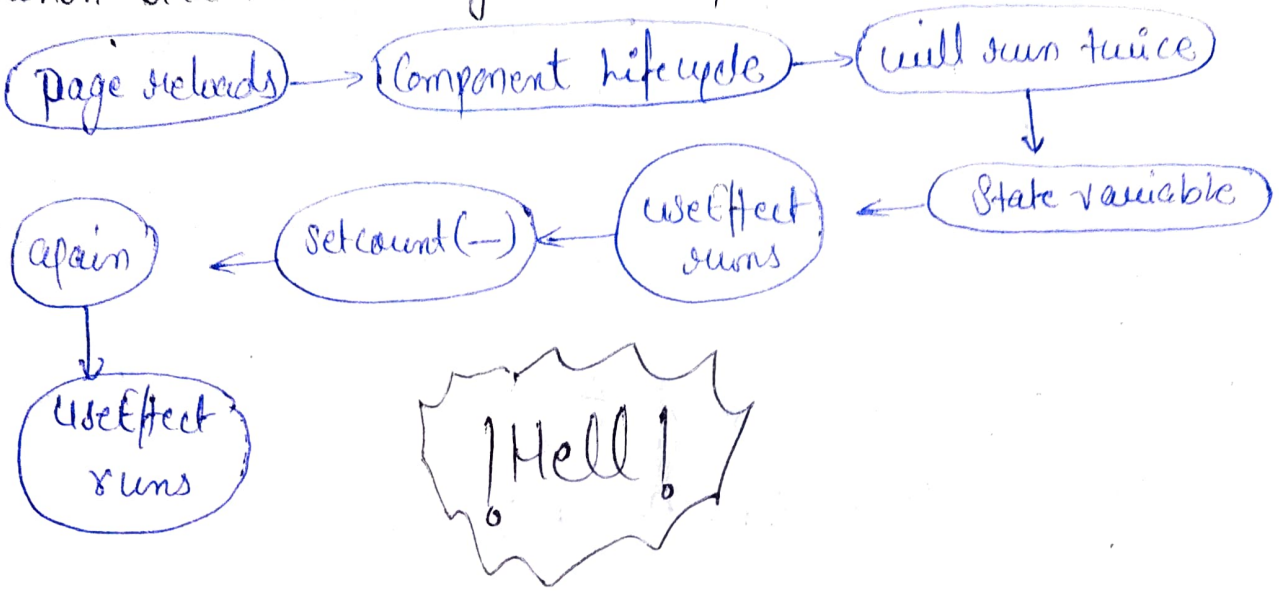


useEffect will self-execution fun^n :-
→ Whenever page reloads which means Component lifecycle functions, then 2nd Phase mounting of Component.
→ Dependency array, parameter of a useEffect. Change in parameter will run the useEffect.

Three Conditions in useEffect ⇒
- when dependency array is not mentioned.
- mentioned and Empty
- mentioned and var is mentioned.

- When Component loads, unloads, loads again → Everything runs for once

When state variable is given in dependency array.



★ When the dependency array is not mentioned, mentioned as empty, useEffect will runs only when Server → App → Page loads → Component Lifecycle method → It will runs only for once.

★ Used to execute the code, When any event occurs like Page reloading. (Rendering Component).

★ UseState – Store the values in state variables and flow the value within the Components or external Component.
normal var vs. State var.

Syntax :-

```
useEffect(() => { })
```

React App Steps

- → Basic files and folder structure.
- → Create Index.js
- → Create App.js
- → Fetch API using asynchronous JS. (Omdb API - Fake APIs).
- → Store API Url with key in variable.
- → use the useEffect to fetch the data as the page reloads or component loads.
- → Create a fun^n inside Component and then call in the useEffect.
- → Async - Await Method
- → Render & Map the data in JSX. using dot notation.
- → Images and Css file. (search.svg) ('To be Read') — Import them.
- → Start building your JSX and Card Component.
- → Static object and render the data in JSX.

★. State Management.
★ Props.
★ Components - Parent / Child.
★. Hooks
★. State
★. Fake APIs.

- → app
- → title
- → Search — Input, img
- → Container
  - → movie
- → movie
  - → Year
  - → img
  - → Span
  - → — —

{movie} → object destructuring.

→ fetch the movies and store them in movies state variable.

Const [movies, setmovies] = useState([]).

Setmovies (data.Search)

→ Apply condition, if movies array is empty - Notfound.

→ Array method → {movies.map((movie) => {return (<> <> movie

→ Search functionality works?

→ Multiples Hooks can be used.

⇒ Why we need another state for Search field - SearchTerm.

To store our typed / filled field.

onChange → value

while typing / filling → value must be assigned to state var.

onClick → searchMovies (searchTerm).

Fundamentals of React JS - Netflix Clone

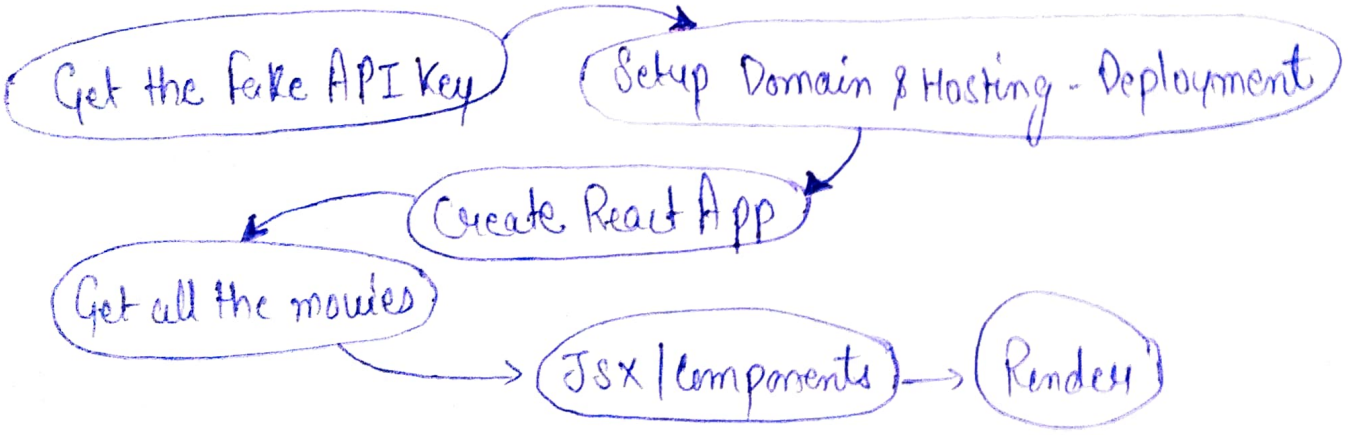
Clever Programmer

Terminologies & Steps required to build a React App

(#1). Basic Introduction – Axios, Fetch API, Play Trailer, VS Code Live Share Extension,

(#2). Procedure to build a project :-

1 → Demo App
2 → Get TMDB API Key. → The movie Database
3 → Start React App. - Create React App.
4 → Setup Firebase Hosting.
5 → Get all the movies.
6 → Build the Rows.
7 → Build the Banner
8 → Build the Navbar
9 → Add Trailer Popups



* Access the Services from Server via API Key. (Request).
* Scalable, flexible, powerful built apps.

#3. Register and get the TMDB API Key.

Key

Request Link - https://api.themoviedatabase.org/3/movie/550.

#4. Setup your deployment - firebase hosting platform.

→ Suite of tools by google
→ Services - DB, Host, Auth, Domain.

1. Create a Project with google Analytics.
2. Type of Project - (Web). - Install firebase-tools in Project → Console.
3. Setup your firebase Hosting.

#5. Setup your new React App. - Run the App. - 3000. (Cleanup Process)

#6. Fetch all the movies with Api fetch Mechanism

(Axios) (Fetch Api).

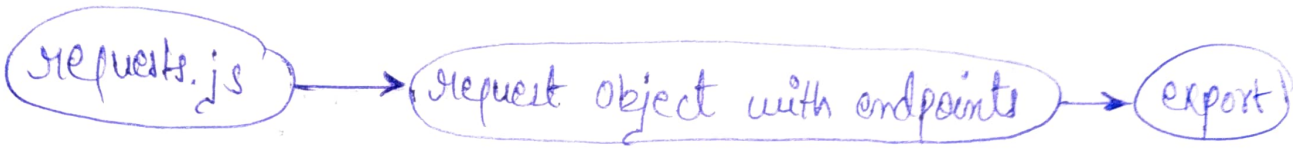
→ We have an api key.
→ create default request to Server for Specific Categories/movies.
→ Install and Configure Axios. (Postman).
→ Setup our axios - request sender.

~~Create a requests.js file~~

1. Create a JS functional module as requests.js to add all the url requests. List of all the request endpoints for various movie categories. (lowercase - filename).

→ Store an API_Key in Const.

→ Store all endpoints inside an object as requests. (Key: Value)



→ To make the request process | url - ~~easier~~ easier. (shorter). (ends of the url). and inserting an Api key. ${" "}.

2. Axios.js - file Setup. (Postman)

→ import axios.

→ Create Instance Store Instance

Axios.create ({ baseURL: }) → export

→ (with weird params) ?API_key = ${ } & with

(with)

instance.get ("/foo")

Components - Props

#6. #7. Fetch the movies & Build the Row's

→ Create a Row Component.

Components → Containers → App → index.js

↓

index.html

→ ES7+ React Snippets.

→ Row → Headings

→ Posters Block.