**Namaste React**

**Lesson 1: Inception**

1. Browsers do not understand what React JS is.
2. For browsers to understand react we will have to add react to our project.
3. One way of adding React JS to the code is by **Using the CDN (Content Delivery Network) link.**
   1. When we added the CDN links in our code we imported React JS into our project.
   2. So, whatever code we write in React JS our browser will be able to understand React.
   3. <https://legacy.reactjs.org/docs/cdn-links.html>
4. When we look at the script tags and open the src link we see that all the code that is written is plain JavaScript code.
5. At the end of the day, React JS is just JavaScript code.
6. This code is written by some Facebook developers.
7. When we now go to the developer console and type React, we will have access to all the superpowers of React.
8. We have imported two CDN links, the first one is React, which is the core of React, and the second one is React DOM, this is the library that we need to modify the DOM (Document Object Model).
9. Why do we have two different files instead of one? This is because react works with different devices such as Laptops, Mobile phones (React Native), iPad, etc. There are different functions that are used in mobiles and laptops so there are different files.
10. React DOM is like a bridge between the code and the browser.
11. In JavaScript, we use something called **document.createElementById** but in React we use something called React.createElement
12. This createElement takes three arguments.
    1. The tag that we want to create. For example, h1, h2, h3, div, p etc.
    2. Object – These are attributes to a tag. For example: id, class, etc.
    3. Content to put inside the tag.
13. Things are a bit different in React JS. To inject the heading in our application we will have to have a root element. All the DOM manipulation is done here.
14. Creating an element is the responsibility of react.development.js. Creating a root and rendering some data in the element is the job of the react DOM library. We will use reactDOM.createRoot(). It will take the root element id as the parameter.
15. Const heading = React.createElement("h1", {}, "Hello from react JS");
16. const root = ReactDOM.createRoot(document.getElementById("root"));
17. root.render(heading);
18. React element is nothing but an object.
19. We will have complex or nested structure in our application. To create this structure, we will use the createElement and do it for all the parent child required.
20. Writing such complex structures is difficult and we have something called JSX.
21. Order of the script files matters in ReactJS. So, if we place app.js above the React JS CDN files then our application will break saying that react is not defined.
22. When we do the root.render and if there is something that already exists in the root tag then the data in the root tag will be replaced by the latest code that we want to render.
23. React is a library and it can be used in a particular section of the whole application.
24. We can use it only for header, footer, or main body.

**Lesson 2: Igniting Our App**

* 1. The code that we have written here is not production ready.
  2. We do not want to push comments, log statements to our production code.
  3. We want to minify and bundle our code. If we have 1000 files we will want to minify the code and if we have 1000 images then we want to optimize our images. To make our app ready to go to production we will have to do a lot of processing before we push the code to production.
  4. We will have to do:
     1. Bundling
     2. Code Splitting
     3. Chunking
     4. Image Optimization
     5. Remove comments.
     6. Remove log statements.
     7. Minify the code.
  5. What do we need to make our app ready to go to production?
  6. React alone cannot do a fast production ready app. Lot of other libraries are used.
  7. NPM is everything but not Node Package Manager. NPM does not have a full form.
  8. NPM manages packages but does not stand for Node package Manager.
  9. NPM is a standard repository for all the packages.
  10. When we do npm init we get package.json. This file contains the configuration for our npm.
  11. Packages that are application is dependent on is called dependency and our package.json will take care of the version of this dependency
  12. The most important package in our application is the bundler. Our code has to be cleaned, cached, bundled before we send it to production. Bundler helps us do this. Webpack, Parcel, Vite are some examples of bundlers.
  13. Bundlers helps package our app so that I can be shipped to production.
  14. Create-react-app behind the scenes uses the webpack bundler behind the scenes.
  15. To install a package, we use npm install packagename
  16. There are two types of dependencies.
      1. Dev – Generally required for development phase.
      2. The caret sign before the version number is that is does minor upgrades. Ex: ^2.8.3 -> 2.8.4
      3. If we use ~ it will install major version. Ex: 2.8.3 -> 3.0.0
      4. It is safe to use ^ because there might be huge changes in major version, and it might break our app.
      5. Normal – Used in production as well.
  17. When we installed npm init parcel we also got a file called package-lock.json.
  18. Package-lock.json will have the exact version of that dependency that is installed.
  19. Package.json – has approximate version and the lock version has the exact version.
  20. Integrity is basically a hash. When we face the issue, the code works on my machine but not on server. The hash/integrity value here makes sure that whatever is the code on local the exact same code is being deployed on production.
  21. node\_modules contain all the code fetched from npm. All the code of parcel is fetched and put in this folder. It is a database that has all the packages.
  22. Our app has dependency as parcel and parcel has other dependencies this is called transitive dependency.
  23. Every dependency has its own package.json.
  24. If we have package and package-lock we can re-generate our node\_modules.
  25. We will use parcel to ignite our app – npx parcel index.html
      1. Just like we have npm we have something called npx. When we want to install a package, we do npm. When we do execute something, we do npx.
      2. We are executing parcel.
      3. When we run this command parcel will create a development build and host it to the local server 1234
  26. The second way to get have React in our app is by using npm. React is a JavaScript package that is hosted on npm.
  27. Using CDN links is not a good way because Fetching from CDM is a costly operation as we will make a network call.
  28. If we have react in our node\_modules, if there is an update in the future then it will be easy rather than changing/updating the links to the latest version.
  29. Installing react as a package. npm install react. When we install react we also install react-dom as they are interrelated.
  30. When we do a npx install react and remove the cdn links and start our application on the local server it will not work because we just installed React but have not used it anywhere.
  31. import React from “react” this means that we are getting React from the react that is in node\_modules.
  32. After we import React and ReactDOM in our App.js in that case we will not be able to run our application. Browser will say “Browsers scripts cannot have imports or exports”
  33. When the App.js file is loaded the browser treats it as a normal JavaScript file and JS does not have import in it.
  34. We will have to tell the browser that this App.js file is not a normal file and we will have to tell the browser that it is a module. To do that we use the attribute type=”module”.
  35. Previously ReactDOM was present in react-dom but now it is available in react-dom/client
  36. Whenever we make some changes to the code the changes are automatically loaded in the application, and this is done by parcel.
  37. Parcel creates:
      1. dev build
      2. local server
      3. HMR – Hot Module Replacement
      4. Parcel uses File Watching Algorithm written in C++ for HMR.
      5. Faster builds because of caching. It is caching things in .parcel-cache.
      6. Image optimization – Loading images is the most expensive thing so parcel will do that.
      7. Minifies files.
      8. Bundling
      9. Compress files.
      10. Consistent Hashing
      11. Code Splitting
      12. Differential bundling – App can be opened in different browsers and parcel builds different versions of the bundles for different browsers. Basically, supports older browsers.
      13. Diagnostic
      14. Error handling
      15. Feature to hoist on HTTPs
      16. Tree shaking – remove unused code
      17. Different dev and prod bundles. More optimizations in prod build.
  38. To create a prod build we do npx parcel build index.html
  39. But the above command will give an error because in package.json we have something called as main keyword which will conflict with index.html so we remove that.
  40. The dist folder will contain the files of the build be it dev or prod.
  41. To make our app compatible for older browsers we will use something called browsersList.
  42. Browserslist is a npm package. We will configure it in package.json
  43. Ex: "browsersList": [ "last 2 Chrome version", "last 2 Firefox version"] This means the app may or may not work in other versions but the app will definitely work in the latest 2 versions of Chrome and Firefox.

**Lesson 03 – Laying the foundation.**

1. Instead of using npx parcel file name we will be using scripts.
2. In package.json under the scripts object we add start key and value will be parcel index.html
3. Now to start our application we will do npm run start OR npm start.
4. Creating headings, paragraphs using createElement is difficult and it might get confusing and messy as we write huge code.
5. So Facebook community created something called JSX. JSX is not part of React both are different. We can build react without JSX as well. To make our life easier we use JSX.
6. JSX is a convention where we merge HTML and JS together.
7. JSX is not HTML inside JavaScript.
8. JSX is HTML like syntax.
9. Browsers do not understand JSX it will give an error but when we run our code with JSX there is no error that we see because parcel takes care of it behind the scenes. The code is transpiled before it goes to JS engine.
10. Parcel alone is not doing the transpilation. Babel does it.
11. Babel is not developed by Facebook.
12. Some browsers do not understand ES6 and babel transpiles the code for the older browsers to understand the code.
13. In JSX, if we wish to give class then we use className instead of class.
14. Attributes are written using camelCase.
15. Multiple lines should be wrapped in parentheses.
16. React Components
    1. Everything in react is a component. Header, Footer, Body etc.
    2. Two types of components – Classed based(Old way) and Functional(New way)
17. Functional component is just a JavaScript function. This basically returns a JSX
18. We cannot directly render our component using the render function. We will have to use <HeadingComponent />
19. Component composition – Placing a component inside a component.
20. If we would like to execute any JS expressions inside a function then we use {}. Example {title} or {100+100}
21. JSX takes care of cross-site scripting attacks.
22. Code is readable because we write JSX and JSX is not react.

**Lesson 04 – Talk is cheap, show me the code.**

1. Style is given in react as an object. Ex: style= {backgroundColor: ‘red’}
2. Functional components are normal functions in JavaScript.
3. Props are just normal arguments to the function
4. <RestaurantCard resName="Millet Express" cuisine="Healthy food, South Indian, North Indian"/> - Passing props to a component and at the end of the day it is passing arguments to a function.
5. Const ResCard = (props) => {}
6. Const ResCard = ({resName, rating, cuisine, eta}) => {} – de-structuring on the fly.
7. Config Driven UI – Controlling how the UI looks like using the data from backend.
8. Cloudnary is a CDN. All images are hosted there
9. When we loop anything we will have to pass a key attribute with unique id.
10. React optimizes the render cycle. Lets say we have four restaurants in the UI and a new restaurant came in at the first place so the Dom will have to insert the new element at the first place. If we did not have id for the RestaurantCard react will re-render all the data because react does not know which res card is the new one and it will treat each res card the same but if we give an id react will only render the new res card that came in.
11. We should never use index(the property of map function) as the key because when the new item is added to the list as the order of the list is changed and the whole data is re-rendered.
12. Not using keys(not acceptable) <<<<<<<<< index as key <<<<< unique id (best practice)

**Lesson 05 – Let’s get Hooked**

1. Default export – export default Header; import Header from path
2. Named export – export const logoUrl; import {logoUrl} from path
3. We can have both named and default exports in a single file
4. Hooks are normal JS utility functions developed by Facebook. These are written inside react.
5. Two very important react hooks – useState, useEffect
6. useState – maintains the state of the component. Scope is inside the component.
7. When we use normal JS variable and filter the data, the list is updated but the UI is not. For the UI and data layer to work together we use the useState method.
8. Whenever state variable updates react re-renders the component.
9. Behind the scenes whenever the data is updated react updates the component super quickly.
10. React uses something called Re-conciliation algorithm also called as React Fiber.
11. Lets say I have 7 cards on the UI and after filtering it is only 3 cards on the UI. Initially create creates a virtual DOM. Original has tags, images etc but in virtual DOM it is just a representation of the original DOM.
12. Virtual DOM is just an object of react elements.
13. Diff algorithm finds out the difference between previous virtual DOM and the updated virtual DOM and then updates the original DOM.
14. React does not make much DOM manipulation that is the reason it is faster.
15. React does efficient DOM manipulation.

**Lesson 06 – Exploring the world**

1. Monolith vs MicroService architecture
2. Monolith architecture – We have UI, API, Auth, Database, SMS in the same single project. We will have to deploy the whole thing at a go.
3. Micro Service architecture – Here we have different services for different jobs. Example UI service, Backend service, Auth service etc.
4. This is known as Separation of concerns and Single responsibility principle.
5. These services interact with each other.
6. Two ways to fetch the data.
   1. As soon as the page loads, we can make the API call to fetch data and render it on the UI. In this case if the API takes 500 ms then the page loads for 500 ms then renders the data.
   2. As soon as the page loads we will render the UI and then we will make the API call and re-render our application with the data.
   3. We will always use the second approach because it gives better user experience.
7. useEffect hook – This has two arguments the first one is callback function and the second is dependency array.
8. The callback function in the useEffect is called after the component is rendered.
9. CORS error – Our browsers block us from calling API’s from one origin to the other origin.
10. Optional Chaining – data?.data?.info?.name
11. Why do we need state variables? When we use the state variables react updates the variable value and updates the UI.
12. When a state variable is changed the whole component is re-rendered and not just the specified part where there is data changed.
13. Since the variable that we use in useState is a constant variable. How are we able to update the const variable? When we update the variable, react understands that we will have to update the data.

**Lesson 07 – Finding the Path**

1. useEffect is called after every render of the component. Everytime a component is re-rendered the useEffect will be called.
2. But the dependency array changes the behavior of the render.
3. Dependency array is not mandatory for useEffect only callback function is.
4. If we do not pass a dependency array the useEffect will be called every time the component is rendered.
5. If dependency array is empty, then useEffect is called only on initial render and just once when the component is rendered for the first time.
6. If we have some thing in the dependency array then the useEffect will be called only when the value of that variable is changed.
7. **useState** – Always call useState inside the component and it has a purpose i.e to create local state variables.
8. Try to call hooks at the top since JS is an asynchronous language
9. Never use useState or useEffect inside if else condition, for loops as it will create inconsistencies.
10. React-router-dom is used for routing in React JS
11. To create the configuration we use something called createBrowserRouter from react-router-dom.
12. createBrowserRouter takes list of objects and each path defines what happens when you navigate to a particular link.
13. RouterProvider will provide our routing config to the app. root.render(<RouterProvider router={appRouter}/>);
14. There are multiple routers available such as HashRouter, MemoryRouter, Static Router etc. React documentation recommends using createBrowserRouter.
15. When some random path is attached at the end of the url then we see 404 error but we can also create our own error page using errorElement.
16. React gives us access to an important hook for error called the useRouteError by react-router-dom.
17. If we want our header to be intact over the whole application and just the content below the header has to change we will create something called as children routes.
18. The children key takes a list of objects in the same way we specify the config path.
19. To do the above react-router-dom gives us something called as Outlet
20. We can use anchor tags but never use that in react because the whole page gets refreshed rather than just that particular component render.
21. To give links to text to navigate to go to a page we use something called Link from react-router-dom.
22. By using Link we will not reload the whole page and this is called Single Page Application.
23. There are two types of routing.
    1. Client Side – Here all components are present at our end so when contact is called the page is shown or rendered.
    2. Server Side – Here on each link of about, contact a request is sent the server and the whole page is reloaded.

**Lesson 08 – Let’s get Classy.**

1. Functional components are nothing but function in JavaScript whereas Class Based components are nothing but normal classes in JavaScript.
2. To write a class based component we write class User extends React.Component to make the User class a component.
3. To return some JSX we write the render method and return the JSX that we want to.
4. React.Component is a class present in the React package.
5. We send props to a class in the same way we did for functional components.
6. To receive props we use the constructor and also specify the super keyword which will receive the props.
7. When we use super in the constructor we inherit the props from parent component as well as any new props that the component has.
8. To access props inside our class we will use this.props
9. Invoking a functional component means that we are mounting the component.
10. Loading a class-based component means that we are creating an instance of the class. Whenever the instance is created the constructor is called and the best place to create props and state.
11. The state variable is a big object that contains all the state variables.
12. Updating the state variable
    1. this.state.count = this.state.count + 1 – This is the wrong way to do it. Never update state variables directly.
    2. This.setState({count: this.state.count + 1}) – This is the right way of doing it
13. Lifecycle of React Class based component.
    1. Constructor
    2. Render
    3. componentDidMount
14. We use componentDidMount to make API calls because we want to render the component, make API call and then render the data.
15. When we use two children in the parent component then the order goes as follows
    1. Parent constructor
    2. Parent render
       1. Child1 constructor
       2. Child1 render
       3. Child2 constructor
       4. Child2 render
       5. Child1 componentDidMount
       6. Child2 componentDidMount
    3. Parent componentDidMount
16. Component mounting is done in two phases
    1. Render – constructor, render
    2. Commit – React updates DOM, componentDidMount
17. When we call the same class multiple times react will batch the render phase and then call the commit phase for optimization purposes.
18. Manipulating the DOM is expensive, and it takes time so React tries to batch things up.
19. To make API calls in componentDidMount we can make it async.
20. When setState is called the update cycle begins and React triggers the render again. Once render is done componentDidUpdate is called.
21. When the render happens it is called with the latest API data fetched.
22. componentWillUnmount is called when we leave the page/component.
23. Never compare reacts lifecycle methods to the functional component.
24. Never say useEffect is equivalent to componentDidMount.
25. For unmounting in functional component we can write return statement in useEffect and that will act as unmount.
26. We cannot make the callback inside useEffect as async. React will throw an error. This is because async functions return a Promise and React’s event handling may not always work as expected with Promises.

**Lesson 09 – Optimizing our App.**

1. Single Responsibility Principle – Each component should have a single responsibility or task to do.
2. If we have Restaurant card it should be only display the card.
3. Like we have hooks from library we can also create our own custom hooks.
4. Hook at the end of the day is a utility function and a normal JavaScript function.
5. Using custom hooks makes our code more readable, modular, and re-usable.
6. Parcel is a bundler and all the code that we have written until now is bundled into a single JS file.
7. As our application grows the size of the file keeps increasing.
8. The process of creating smaller bundles is called chunking, code splitting, dynamic bundling, lazy loading, on demand loading, and so on.
9. We do lazy loading using the lazy function from react.
10. Const componentName = lazy(() => import(componentPath))
11. When we do this and directly import the path we will get an error because react is superfast and the component will take time to load/render and will throw an error.
12. To fix this issue react gives us something called suspense.
13. This step is very important as we must make our application perform better.

**Lesson 10 – Jo Dikhta hai, vo bikhta hai**

1. Ways of writing CSS
   1. Internal styes
   2. External styles
   3. Inline
   4. SASS
   5. SCSS
   6. Styled components
   7. Material UI
   8. Ant
   9. Bootstrap
   10. Chakra UI
   11. Tailwind
2. When we install tailwind CSS, we install both tailwind and post CSS. Post CSS
3. PostCSS helps transform CSS from JavaScript

**Lesson 11 – Data is the new oil.**

1. Higher Order Component is a function that takes a component and returns a component.
2. Takes an existing component and returns an enhanced component.
3. Higher order functions are pure components meaning nothing will be changed in the original component.
4. All react applications have two layers: a UI layer and data layer. The UI layer is powered by data layer.
5. Controlled component – This means that the parent component will control the state of the child component.
6. Lifting the state up is where the parent has the state of the child.
7. Props Drilling – Data flows in one-direction or in one way stream. Dat flows from Parent to child to grandchild
8. If we have to pass data from parent to grandchild, we can’t do it directly we have to pass it through child.
9. Doing this will re-render all the pages through which props are passed this is a bad way of doing as many re-renders happen even though child doesn’t use the props that are being passed to grandchild. This gets difficult for large applications.
10. To avoid this react gives us something called as ReactContent. With this we can avoid props drilling.
11. Creating a user context using createContext from react library
12. Context should be used when the data will be used in multiple places
13. Class based components cannot access hooks.
14. We will import the created context and use it using the Consumer. We write it as follows – UserContext.Consumer
15. In the application let’s say after authentication we want to change the name of the user from Default user to the credentials fetched from the API then we will do so by using the Provider.
16. <User.Provider value = {{loggedInUser: “Preethi”}}></User.Provider>. Now whenever there is a change in the name and wherever the loggedInUser is used the name will be updated accordingly.
17. Context is something that is global for the app. We can use it for the whole application or just for specific components of the app.
18. We can create multiple contexts in the application.
19. We can also have nested context.
20. We can change the value of the context by passing the setUserName through the value in Provider.
21. If we are building a small application Context should be enough.

**Extra Points**

1. The most important operation that happens in a browser is the DOM manipulation.
2. What is CDN?
   1. CDN stands for Content Delivery Network. It is a distributed network of servers located in different geographic locations that work together to deliver internet content, such as web pages, images, videos, and other resources, to users in a faster and more efficient manner.
3. Why do we use CDN?
   1. Improved website performance
   2. Global Reach
   3. Traffic Offloading
   4. Load Balancing
   5. Caching
   6. Content Optimization
   7. Security and DDoS protection
   8. Scalability
   9. Content Availability
4. How does CDN work?
   1. Content replication - When a website or online service uses a CDN, it duplicates its content (like HTML files, images, videos, etc.) and stores copies on multiple servers in various locations around the world.
   2. Geographic Distribution - These servers, often referred to as "edge servers" or "nodes," are strategically placed in data centers across different cities or countries. This geographical distribution helps reduce the physical distance between a user and the server, which can significantly improve load times.
   3. Request Routing - When a user tries to access a piece of content, the request is automatically routed to the nearest CDN server rather than the original server where the website is hosted.
   4. Content Delivery - The CDN server then fulfills the request by serving the content directly to the user. This leads to faster loading times and better performance because the data doesn't have to travel as far.
   5. Load balancing & Scalability - CDNs are designed to handle many simultaneous requests. They use techniques like load balancing to distribute traffic evenly among the servers.
   6. Caching - CDNs employ caching techniques. When a CDN server receives a request for a specific piece of content, it checks if it already has a copy. If it does, it serves the cached copy, which is faster than fetching it from the original server.
5. What is the crossorigin keyword in script tag?
   1. The crossorigin attribute sets the mode of the request to an HTTP CORS request.
   2. Web pages often make requests to load resources on other servers. Here is where CORS comes into picture.
   3. A cross-origin request is a request for a resource from another domain.