**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.

**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.