NAMSTE REACT

EPISODE 1- INCEPTION

1.EP01.PART1

- Hello world using html
- Helloworld using js

2.EP01.PART2

- Hello world using React cdn with in html file

3.EP03.PART3

- Hello world using React CDN but with seperate js file(App.js) creating elements

4.EP04.PART04

- Hello world using React CDN but with seperate js file(App.js) creating **nested** elements

5.EP05.PART05

- Order in html code
- App.js should always be after react code otherwise it throws error.

EPISODE – 2 Igniting our App

Part-1(installations)

- **Push episode 1 code to git**
- **The code in ep1 is not a cool stuff to develop larger apps it needs dependencies we will see them all in this episode.**
- **Npm init** (pakage.json) file is created
- **Install parcel bundler**
- **(npm install –D parcel) ->package-lock.json is created**
- **then if u see package.json u will see something like this**
- `"devDependencies": {`
- `    "parcel": "^2.10.3" //here ^ represents automatic upgrade if new versions are released.`
- `  }`

- Also a new file named package-lock.json is created.(it keeps the track of exact version)
- If you search parcel you will find the exact version of parcel inside the package-lock.json file. There will be no ^symbol
- Package.json keeps the aprox version in it wherre as package.json holds exact version
- You can also see a new folder node modules is also created.
- It has multiple folders , its because we have installed parcel so those folders are dependencies and dependencies dependencies so we have that many folders

- Dependencies of dependencies is called as **transitive dependencies**
- In a project there is not only a single package.json and package-lock.json file .In npm folder every dependency has its own package.json and package-lock.json files.
- Node modules are collection of dependencies
- There is no need to keep all these files in git , by keeping these files in .gitignore file we can achieve this.
- Now if you check git symbol in vs code you will see lot many changes(5k+),so do the following
- First create a file called .gitignore and now in that file write the following
- /node_modules (then save the file and check at the git symbol you will see only 4-5 changes it is clean and simple.
- It is ok not to keep node modules in git but you should keep package.json and package-lock.json files in git it is very important.
- It is because you can recreate your node modules with package.json and package-lock.json if node modules are deleted

(TRY ONCE IF YOU DELETE NODE MODULES , JUST GIVE COMMAND **npm install**

You can recreate the node modules.so there is no need to required to push node modules into git

- Gitignore file contains the modules that are not required to push into the git.

EPISODE-2

PART-2:

HOSTING APP INTO THE SERVER

- Type the following command
- (npx parcel index.html)-->index.html is source file
- Then you see it gives you a local server where you can run your app
- Npm is used to install a package
- Npx is used to execute a package

BRINGING REACT USING NPM RATHER THAN USING CDN LINKS

- This cdn links have only particular version if version chsnges we keep on changing the number
- ```
  <script crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
  dom.development.js"></script>
  ```
- We have version 18 in this case.so we can also bring react using npm
- (npm install react)-->It brings react into our node modules
- Then install react-dom (npm install react-dom)
- Remove cdn links
- Give command npx parcel index.html
- Browse the app with in localhost server
- Now everything executes fine , the elements in html file appears in browser but the elements created using React.createElement in app.js can't displayed because in console you see error
  ```

react is not defined because we have removed cdn links and we are using React.create element in app.js file which browser could not understand so we import react and react-dom inside app.js(1:19:00)min

- Import React from “react”

- Import ReactDOM from “react-dom” in app.js
- You will see the error in browser that browser scripts does not contain a import and export statements
- Beacause browser assumes app.js as a normal js file
- So now while importing app.js to index.html give type=”module”
- Ex: `<script type="module" src="./App.js"></script>`
- Everything works fine in browser now but you will see a warning like createRoot is not a method
- Now change import ReactDOM from “react-dom” to import ReactDOM from “react-dom/client”
- In react if you made any changes in the file and saved they will automatically reflected in browser there is no need to refresh browser its all because of the parcel.

What parcel do?

- Dev builder
- Local server
- HMR –hot module replacement(auto change of browser when we save code changes)
- HMR is done because of **File Watching Algorithm** in parcel it is written in c++.
- If you do any changes in code and save , it builds app again but the time will be reduced everytime because of **Caching**
- Faster Builds By **–caching(**.parcel-cache file holds the cache)
- Image optimization
- Minification
- Bundling
- Compressing
- Consistent hashing
- Code splitting
- Differencial bundling-for supporting older browsers
- Diagnostic-beautiful error showing by exactly showing the location where error occurs
- Error handling
- Tree shaking algorithm – removes unused code in the file
- Can load as https
- Builds different dev and production bundles(because production build will take a little more time than the dev bundle)
- CREATING A PRODUCTION BUILD –run the below command.
- (npx parcel build index.html)
- You will get this error
- "description": "",
- > 5 | "main": "App.js",

- > | ^^^^^^^^ Did you mean "App.html"?
- 6 | "scripts": {
- 7 | "test": "jest"
-
- 💡 Try changing the file extension of "main" in ....package.json.
- The error is because we set main path as index.html but in the package.json main has App.js
- So now just delete that line in package.json
- When you built npx parcel build index.html it bundles all the files into the dist folder(these files are used in the production)
- The files in dist folder that is created by npx parcel index.html are used in development
- What you are seeing in browser just comes from the dist files itself

C:\Users\cvani\OneDrive\Desktop\namaste-react\Ep01.inception\Ep01.part04> npx parcel build index.html

✨ Built in 2.40s

..\..\dist\index.html 300 B 867ms

..\..\dist\index.57e55873.js 138.86 KB 654ms

- You can observe index.html and index.57e55873.js it means the entire code you wrote is compressed into these files
- Whenever u run the npx parcel index.html or npx parcel build index.html , the folders parcel-cache and dist are automatically generated even if you delete them. So keep them in gitignore
- /dist
- /.parcel-cache

CONFIGURATING OUR APP IN ALL BROWSERS

- Go to package.json and code something like this
- ```
  "browserslist":[
      "last 2 versions"
    ]
  ```

All these things makes our react apps super faster and strengthen our app

EPISODE 3. LAYING THE FOUNDATION

PART1(Adding scripts)

- To start the app we used npx parcel index.html , writing it again and again will cause difficulty
- So lets create scripsts for dev built and production built
- In package.json file in scripts add the following

Scripts in package.json

```
"scripts": {
```
```

```
    "start": "parcel index.html", (we added this line)
    "build" :"parcel build index.html",(we added this line)
    "test": "jest"
  },
```

- Now u can use **npm run start** instead of using npx parcel index.html.(for dev built)
- And **npm run build** for production built.
- **C:\Users\cvani\OneDrive\Desktop\namaste-react\Ep03.laying the foundation> npm run start**
-
- > ep03.laying-the-foundation@1.0.0 start
- > parcel index.html
-
- Server running at http://localhost:1234
- ✨ Built in 1.19s

You will see the result in the browser

For production built

C:\Users\cvani\OneDrive\Desktop\namaste-react\Ep03.laying the foundation> npm run build

> ep03.laying-the-foundation@1.0.0 build

> parcel build index.html

✨ Built in 2.68s

dist\index.html 300 B 901ms

dist\index.10f72f82.js 138.86 KB 667ms

PS C:\Users\cvani\OneDrive\Desktop\namaste-react\Ep03.laying the foundation>

- Simply you can also call npm start instead of npm run start ->it only works for dev built not for prod built

EP3:PART2

Normally we used React.createElement to create react element, this React.createElement is a object when it rendered it become html element.

EP3:PART3(introduction to jsx)

- Using react element is not a good idea as the code becomes clumpsy.
- So JSX is introduced
- JSX is not a html

Ex: using

older reactElement version

```
import React from "react";

import ReactDOM from"react-dom/client";

Const heading =React.createElement(

            "h1",

            {id: “heading”},

            "I'm h1 tag without cdn!!!"

        );

console.log(heading);//object

const root = ReactDOM.createRoot(document.getElementById("root"));

        root.render(heading);
```

USING JSX:

```
import React from "react";

import ReactDOM from"react-dom/client";

const jsxheading =<h1 id= “heading”>I’m h1 tag with jsx</h1>

const root = ReactDOM.createRoot(document.getElementById("root"));

        root.render(jsxheading);
```

Both codes mean the same but jsx has simpler code (38:31)

- Every broswer renders after compilation of js compiler ,Javascript compiler doesnot understand the jsx
- So browser doesnot understand jsx code

- As this jsx code is not understand by the js engine and the compiler the jsx code is transpiled before reaching the javascriptengine
- That transpilation is done by parcel.parcel gives this responsibility of transpiling to its package called **“Babel”**

How the reactelement works vs jsx work?

- React.createElement-creates react element which is a javascript object
- When it renders it renders as a html element.
- Simply React.createElement->reactElement(js object->html element(render)

Jsx working

- Jsx-> React.createElement->reactElement(js object->html element(render)

Babel is converting jsx code to react element

Babel is a javascript compiler

Difference b/w jsx and html:

In html we use keyword **class** to create the class.

But in jsx **className** is the keyword to to create class of the element.

- When you write jsx code in a single line it is ok but if you have a multiple jsx lines you have to wrap them inside the parenthesis
- Ex:single line jsx

```
Const jsxheading =<h1 id=”heading”>I’m h1 tag with jsx</h1>
```

- Ex:multiline jsx

```
Const jsxheading =( <h1 id=”heading”>

      I’m h1 tag with jsx

</h1>)
```

EP3-PART4: COMPONENTS

- In react everything is a component that means a button, card , nav bar everything is a component
- Two types of components
  - Class based components(oldway)
  - Functional components(new way)

React Functional component is just a normal javascript function which returns a jsx element.

- Name the react components with capital letter.

Syntax of functional component is:

```
  Const HeadingComponent = () => {
return <h1> hello jsx </h1>;
 };
```

Shortway of using arrow functions:

```
Const HeadingComponent2 = () =>(
       <h1 className = “heading”> hello jsx </h1>
) ;
```

FUNCTIONAL COMPONENT AND ITS WAY OF RENDERING:

```
const HeadingComponent = () => {
    return(
        <h1>functional COMPONENT</h1>
    );
}
const root =ReactDOM.createRoot(document.getElementById('root'));
root.render(<HeadingComponent/>)//RENDERING
```

Renderning component within other component(component composition)

```
const TitleComponent = () => {
    return(
        <h1>Title COMPONENT</h1>
    );
}
const HeadingComponent = () => {
    return(
        <div>
```

```
            <TitleComponent/>//rendering title component within functional
component
            <h1>functional COMPONENT</h1>
        </div>

    );
}
const root =ReactDOM.createRoot(document.getElementById('root'));
root.render(<HeadingComponent/>)
```

Ep3-part5(using js code inside jsx)

You can write js code inside the jsx but within flower braces {}

Ex:const number = 1000;

```
const TitleComponent = () => {
    return(
        <h2>{number}</h2>
        <h1>Title COMPONENT</h1>
    );
}
```

Composing react element inside other react element:

```
const element1 = <h1>Element1</h1>
const element2 = (
    <div>
        {element1}
        <h2>Element2</h2>;
    </div>
);
const root =ReactDOM.createRoot(document.getElementById('root'))
 root.render(element2)
```

Jsx sanitizing:

```
Const data = api.fetch();//data contains info from some api

Const Component1 = () =>{

   <div>

        {data}

        <h1>Component1</h1>

  </div>

}
```

Suppose if the api sends a malicious data , the data variable holds that content and we are executing {data} in jsx , but before executing the code inside curlybraces{} jsx sanitizes the code whether its good or not.

Ways of calling a component:

- <Component/>
- <Component></Component>
- {Component()}

All the three things ate same

EPISODE-4(TALK IS CHEAP SHOW ME THE CODE)

Part1-(Foodapp):

Created header

Body containing restaurant cards

Part2:

Introduced props

Config driven ui-it is the process of controlling your UI based on the data(ex:data from api)

Lets assume in swiggy in banglore location the banner will be different.

In north India banner will be different.

Here config is nothing but data.

Ways of passing props:

Way-1:

```
Const Component2 = (props) => {

    <h3>{props.name}</h3>
```

```
    <h4>{props.cuisine}</h4>
}
<Component2 name= “KFC”  cuisine= “burger ,FastFood” /> //passing the props
```

Way-2:

```
Const Component2 = ({name,cuisine}) => {
    <h3>{name}</h3>
    <h4>{cuisine}</h4>
}
<Component2 name= “KFC”  cuisine= “burger ,FastFood” /> //passing the props
```

Way-3:

```
Const Component2 = (props) => {
Const {name , cuisine } = props;
    <h3>{name}</h3>
    <h4>{cuisine}</h4>
}
<Component2 name= “KFC”  cuisine= “burger ,FastFood” /> //passing the props
```

Using key in map function:

- It is always a good idea to use keys while using map function.
- Ex:Lets take an example ,we have a large data of array which contains list of objects in it.the data in that object include name of item,unique key,delivery time which will be used in cards of food item.
- You can execute all this array items data in broswer by map() . If new object is added the all the initial cards will rerender in browser because react fall in the ambiguity to where to keep that card.so all the cards will be reredered .it is not a good practice.
- So if you use key the react will identify the new card and it only renders it all the previous cards will not re-rendered which improves performance.
- So,using keys is must and should while working with map() function.
- Ex: objectArray.map((items)=>{

```
  <component key={items.id} data={items}
})
```

INDICES AS KEYS IS A BAD PRACTICE:

- Some people use indices rather than using unique keys that is availble in api data.It is not a recommended way to use indices instead of unique keys.
- Incorrect Component State: If you use indices as keys and remove an item from the list, React may not properly update the component's state. This can cause components to render incorrect data, leading to inconsistencies and bugs in your application.
- Why not to use index as key in React?
- Array index can change

  If you use the array index as a key, the keys would be [0, 1, 2]. Now, if you add a new item D to the beginning of the list, the array index of A, B, and C would change to 1, 2, and 3 respectively. This would cause React to re-render all the elements instead of just updating the newly added item.

NOTE:

- Map() without unique key is not acceptable.using unique index is most acceptable way
- Incase if you dont have data unique key then only use index.