[lecture - 3]

(laying the foundation)

→ let us create scripts then we will not use npx parcel index.html, we will use npm scripts to run the app.

→ package.json –

```
"scripts" : {
  "start" : "parcel index.html",  → dev build
  "build" : "parcel build index.html"  → prod build
  "test" : "jest"
}
```

→ npm run start / npm start
npm run build / but not npm ~~but~~ build

→ React Element is not an HTML Element

– React.createElement → This basically creates an object and when we render this element to the DOM it becomes HTML element

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

↓

It will become root for our react app

- whatever will happen inside react app will happen inside root.

→ But this is not a good way to create our browser elements. It's not very developer friendly.

→ To help all the developer community, JSX was created

→ JSX is a Javascript syntax to create react Element, it is easier.

→ JSX is not the part of react, both are different

→ we can write react without JSX also but JSX makes developers life easy

→ JSX is a convention to merge HTML & JS

→ JSX is not HTML inside JS.

→ JSX is diff. than HTML

→ JSX is an HTML like syntax. or XML like Syntax

→ JSX is just a syntax, reactElement is an Object

const jsxHeading = <h1 id="heading">Namaste</h1>

Hey a react element, also an object

- JS Engine does not understand JSX code
  JS Engine understands ES6 or ECMAScript6 (pure JavaScript).
  or all the version of ES.
- Parcel is doing job behind the scenes to make understand JSX to browser
- Ever before this code goes to the JS Engine it is transpiled before it goes to JS Engine and JS engine receives the code that browser understands
- Transpile is not doing by Parcel itself, parcel is manager, it gives this responsibility to a package which is known as Babel.
- Babel is not created by Facebook

Note → **
- React.createElement ⇒ ReactElement → JS Object → HTML Element (render)

→ JSX ⇒ React.createElement ⇒ ReactElement → JS Object ⇒ HTML Element (render)

→ Babel is converting JSX code to React createElement

→ Babel's job is not just to convert this JSX to react code, it does a lot of other things.
  it is a transpiler, JS compiler

48:34

DELTA
Date / /
Pg No.

→ Some older browser don't understand ES6., Babel transpile to a code that older browser can understand

(Namaste React)

(lecture - 3) Date.....

... to be continue 48:40

- Babel is a champ in converting one code to another.
- Babel is some piece of JS code, it a node library / npm package.
- JSX is not HTML in JS. HTML is different thing. like, in HTML we want to give class name to element, so in HTML we write "class" but in JSX we write "className" keyword. This is a slight difference.

So, in transpilation phase this `className` get converted into `class`.

- If we want to give attribute to JSX we have to use camelcase, we don't use hyphen.
- In HTML → tabindex
  in JSX → tabIndex
- If we want to write JSX in multiple lines, we put that code in parenthesis ( ), we have to wrap it inside round brackets because babel needs to understand where is JSX starting and where is JSX ending
  In single line we don't need to wrap.

Spiral

Date.....

→ React Component →

- There are 2 types of component in react.
  (1) Class based components → old way of writing code
  (2) Functional components → new way of writing code

- React Functional component is just a normal JS Function.
- When we create any react component, name it starting with capital letter
- This is a normal JS arrow Function which returns some piece of JSX (element)
- JSX is a React Element
- In arrow fxn for a single line of code we can remove return keyword and curly brac

React Element

```
const heading = (
<h1 className="head" >
  Namaste React
</h1>
);
```

React Functional Component

```
const Heading = () => (
  <div id="container">
    <h1 className="heading">
    Namaste React </h1>
  </div>
);
```

Date.....

- React Element —rendering→ root.render(heading);
- React functional component —rendering→

root.render(<Heading />);

// Babel understand about component through "< />" this.

Q.7 What is component composition ?
A.7 We are composing two component, one into another.

- Apart from arrow fxn we can also write normal function for Functional component.

```
Const Title = function() {
return (
<h1 className="head" >
Namaste
</h1>
);
};
```

- Arrow fxn is a standard way

Spiral

Date.....

- If we write curly braces { } inside JSX anywhere, inside this curly braces we can run any piece of Javascript expression
- That's why it is easier said that JSX is not HTML in JS because we can write JS also, putting curly braces.
- We can put any JS variable inside this { }, any JS calculation, console.log("abcd"), anything

Q-? How we can put reactElement inside the component?

A-> ~~title is basically a React Element~~ is basically a normal JS variable, so we can put through { }.

- We can't access component before it's declaration.

```
const data = api.getData()

const HeadingComponent = () => {
  <h1>{data}</h1>
)
```

Spiral

Date.....

- whatever the api is sending we are executing it inside the component, so through this api attacker can attack.
This type of attack is known as cross side scripting

- Attacker can steal cookies through JS code even or browser and target access of storage

- But JSX takes care of these injection attacks Even if this api passes some malacious data into the code, JSX will escape it.
JSX sanitizes the data, JSX don't blindly run that, it will sanitizes the data which is coming from apis.

- It prevents cross side scripting attacks.

→ another way → <Title/> ↔ <Title></Title>

→ we can also call functional component in other functional component through curly braces.

{Title()}

→ - <Title/>

- <Title></Title>

- {Title()}

Spiral