Chap 03 - Laying the Foundation

Pollyfill

→ to make older browsers understand our new code, the new code is converted into a older code which browser can understand called pollyfill.

→ babel do this conversion automatically.

Eg :- ES6 is the newer version of javascript.
If I'm working on 1999 browser, my browser will not understand what is this const, new Promise etc.
So, there is a replacement ~~for~~ code for these functionalities which is compatible with older version of browsers.

→ So, this is what happen when we write "browserslist" → our code is converted to older one.

Babel

→ is a javascript package / library used to convert code written in newer versions of JS (ECMAScript 2015, 2016, 2017 etc) into code that can be run in older JS engines.

To run our app, command is :-

```
npx parcel index.html
```

We always don't have to write this command.

Generally, we build a script inside package.json which runs this command in an easy way.

package.json

```
"scripts": {
    "start" : "parcel index.html",
    "test" : "jest"
}
```

So, to run the project, I've to use :-

```
npm run start
```

Shortcut :-

```
npm start
```

"start" script execute this command.

Build command

```
npx parcel build index.html
```

```
"scripts": {
    "build" : "parcel build index.html"
}
```

Note :-

`npx = npm run`

So, `npm run build` will build a project.

Console logs are not removed automatically by parcel. You have to configure your projects to remove it.

→ There is a package which helps to remove console logs :-

`babel-plugin-transform-remove-console`

→ Before ~~After~~ installing this package, create a folder called `.babelrc` } for configuration

→ And include :-

```
{
  "plugins": [["transform-remove-console",
    { "exclude": ["error", "warn"] }]]
}
```

→ Then, build : `npm run build`
to see that all console.logs are removed.

Render – means updating something in the DOM.

Reconciliation

→ helps to make React applications fast and efficient by minimizing the amount of work that needs to be done to update the changes.

→ So, you don't have to worry about what changes on every update.

Eg:-

```
<ul>
    <li> first </li>
    <li> second </li>
</ul>
```

} siblings.

When adding an element at the end of the children: The tree works well

```
<ul>
    <li> first </li>
    <li> second </li>
    <li> third </li>
</ul>
```

- render() function as creating a tree of React elements.
- On the next state or props update, render() fn will return a different tree of React elements.

Whenever react is updating the DOM, for eg :-

```
<ul>
    <li> Duke </li>
    <li> Villanova </li>
</ul>
```

Now, I introduced one child over the top, then react will have to do lot of efforts; react will have to re-render everything. That means, [react will have to change the whole DOM tree.]

```
<ul>
    <li> Connecticut </li>
    <li> Duke </li>
    <li> Villanova </li>
</ul>
```

As react has to re-render everything, it will not give you good performance.

In large-scale application, it is far too expensive.

SOLUTION – Introduction of Keys

→ React supports 'key' attribute.

→ When children have keys, React uses the key to match # children in the original tree with children in subsequent tree. Thus, making tree-conversion efficient

```
<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

Thus, react has to do very less work.
So, always use keys whenever you have multiple children.

CreateElement

→ React.createElement() is creating an object.

→ This object is converted into HTML code and puts it upon DOM.

If you want to build a big HTML structure, then using 'createElement()' is not a good solution.
So, there comes introduction of JSX.

JSX

When facebook created React, the major concept behind bringing react was "that we want to write a lot of HTML using Javascript" because JS is very performant.

```
import {createElement as ce} from "react";
Const heading = React.ce(
  "h1",
    { id : "title",
      key : "h1"
    }
  " Hello World "
);
```

Instead of writting all these :-

```
const heading =
  <h1> Hello World </h1>
```

This is JSX

※ JSX is not 'HTML inside javascript'

⟹ JSX has 'HTML-like' syntax

This is a valid javascript code :-

```
const heading = (
    <h1 id = "title" key = "h1" >
        Helloworld
    </h1>
);
```

JSX expression

React keeps track of 'key'.

H.W :- 1) What is diff b/w HTML & JSX?

Our browser cannot understand JSX.

'Babel' understands this code.

HW

2) What are different usage of JSX ?

3) How to create image tags inside JSX ?

~~JSX uses creat~~

→ JSX uses React.createElement behind the Scenes.

⇒ JSX ⇒ React.createElement ⇒ Object ⇒ HTML(DOM)

→ Babel converts JSX to React.createElement()

(Read Babel's Documentation)

→ JSX is created to empower React.

Advantages of JSX

- → Developer experience
- → Syntactical Sugar
- → Readability
- → Less code
- → maintainability
- → No Repeatition.

Babel comes along with parcel.

COMPONENT

'Everything is a component in React'.

React Components

→ 2 types :-

a) Functional component - NEW WAY of writing code.

b) Class Based component - OLD WAY

→ Functional component

- is nothing but a javascript function.
- is a normal JS fn which returns some piece of react elements (here, JSX).

Eg :-

```
Const HeaderComponent = () => {
    return <h1> HelloWorld </h1>;
};
```

- For any component,

  Name starts with capital letter.

(It is not mandatory, but it's a convention)

to render functional component, write :-

<Header Component />

React element

```
const heading = (
  <h1 id="title"
      key="h1">
    Hello World
  </h1>
);
```

⇓

React element is finally an object

converting it into arrow fn will make functional component.

Functional Component

```
const heading = () => {
  return (
    <h1 id="title"
        key="h1">
      Hello World
    </h1> );
};
```

⇓

Functional component is finally a function.

The Next Amazing thing ⇓

(1)

* `const Title = () => {`
  `<h1>Hello World</h1>`
  `}`

} is a functional component

* const HeaderComponent = () => {

return (

<Title />

<h2> Nomaste React </h2>

<h2> Hai all </h2>

);

};

instead of this you can write { Title() }

This is 'component composition'

This is a normal javascript function !!

---

② * const title = (

<h1> Helloworld </h1>

);

} is a normal variable

* const HeaderComponent = () => {

return (

{title}

<h2> Nomaste React </h2>

<h2> Hai all </h2>

);

};

NB:

* Whenever you write JSX, you can write any piece of javascript code between parantheis {}. It will work.

* JSX is very secure.
  JSX makes sure your app is safe.
  It does sanitization.

```
const data = api.getData();

const HeaderComponent = () => {
  return (
    <div>
      {data}
      <h2> Hello World </h2>
    </div>
  );
};
```

Write anything inside {}, JSX will sanitize the code.

Component Composition

If I have to use a component inside a component. Then, it is called component composition / Composing components.

babel is a transpiler