Igniting Our App

Code which we have written till now is not the Production Ready.
To make it production ready we need to do many thing.
We need to do Chunking, minifieer etc, bundling etc.
We need lot of other packages, libraries etc to make our code production ready.

npm → It do not have any full form.

- npm manages packages but does not stand for node package manager. (It is package manager)
- All the package, libraries, utilities are there

Once we set up npm in our code, A new file will be added

package.json
↳ Package.json is a configuration of npm

Bundlers - parcel, Webpack, Vheet.
↳ We are using this in our App.

To Add package.json file. command to give in terminal

Command: npm init

There are two types of dependencies / App packages App can have

(1) Dev dependencies: It is generally require in development phase. even though it is not necessary to deploy the project in a server.

(2) Package dependencies. or Peer dependencies

Peer dependencies are not required to be bundled within your library. They are external modules or components that your project relies on. Peer dependencies are expected to be installed by consumer.

React and React Dom are example of it.

Installing Parcel ↙ As a dev dependency we are installing it.

npm install -D parcel

⇒ In Package.json

parcel as dev dependency will add.

"parcel" : "^ 2.8.3"

"parcel" : "~ 2.8.3"

package-lock.json → keep the record of exact version of that package
also the dependencies
→ It is most imp. file

Node Modules: It is a kind of data base or data for those dependencies which our project need.

Transitive Dependencies

Our project is have dependency on parcel, parcel as a project has its own dependency and those dependencies can have their own dependency. this is called Transitive Dependency.

Note:-
If I have package-lock.json and package.json. I can recreate node_modules. by just passing command in terminal "npm install" and node module will be recreated.
That's why there is no need to push node_modules in git.

Building Our App using Parcel.

Command :- npx parcel index.html
↳ source file

Now our App is hosted on local Server.

→ Just like we have npm similarly we have npx.
npm used to install packages but npx is used to ~~install~~ execute packages.

⇒ CDN links are not the good way to bring React and React Dom into your project

⇒ We have to make connection to get React it's better to have these dependencies in node module.

⇒ We need to update these links for the update of React as these links have react 18 what if React 19 comes.

Command to stop server: Ctrl+C

We Install React in node_modules

Command: npm install react

(short form of it is: npm i)

Install React Dom in node-modules

Command: npm i react-dom

After that we will get React and React dom dependencies in as ~~project~~ package.json.

Now we remove CDN links and import react in js file.

Command: import React from "react"; → At top.

import ReactDom from "react-dom";

But It will throw an error "Browser script can't have imports or exports".

because we have linked App.js file in HTML file using script command so it will be treated as browser scripts. So we have to assign the type = "module"

<script type = "module" src = "./App.js"></script>.

What Does Parcel Do :-

* It creates - Dev Build
  - Local Server
* It also refreshing page Automatically ⇒ HMR - Hot Module Replacement
* It makes development experience so smooth.
* It Watches all the files by File Watching Algorithm which is written in C++
* Provide faster Builds - Caching.
* Image Optimization
* Minification
* Bundling
* Compress
* Consistent Hashing
* Code Splitting
* Differential Bundling - support older Browser.
* Diagnostic
* Good Error Handling, Better Error Suggestions.
* Tree Shaking - remove unused code.
* Different dev and prod bundles.
* Parcel can also gives a way to host our app on HTTPS right now it hosting on http.

Learns more at Parceljs.org

To Create Production Build

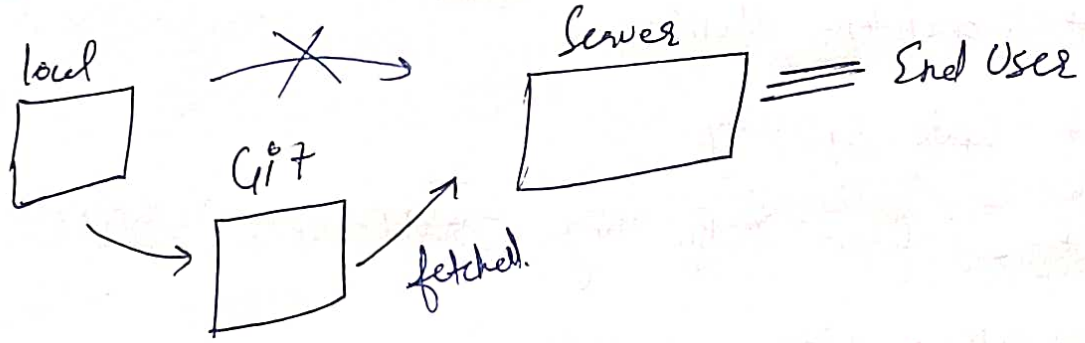
Command: npx parcel build index.html.

but it throw an error because we have also provide an entry point in package.json.
So delete from there "main" : "App.js"

It compress 3 main files .html, .css, .js
All the codes lies in these 3 files. These three file are production ready code.

This production build is highly optimized builds which we can push to the production and can serve to the user.
It will be fast, performant, optimized.

* dist, .parcel-cache and node-modules are not needed they can be auto generated



How to Make App Browser Specific

Add
```
"browserlist": [
    "last 2 versions"
]
```
→ It took array of browserlist

in package.json.

To check usercoverage go browserlist.dev

Laying the foundation

Till now to run the project we are using command: npx parcel index.html
It is lengthy and not the industry standard.

Scripts

We will now use the scripts.. In package.json file We can write script command as

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

You can give any name to this's

To Run the Scripts

Command:

1. npm run start or npm start → shortcut only for start.
2. npm run build

Note: In industry if you don't know how to Run project check its package.json file and search for start and build.

React.createElement ⇒ React Element - JS object ⇒ HTML Element (render)

JSX

* Javascript Syntax to write HTML Tags
* JSX is not the part of React, We can write React using JSX and without JSX as well, but JSX makes developers life easy..
* Creating elements using React.createElement is difficult.

[Const jsxHeading = <h1> Namaste React using JSX </h1>;]

JSX is not HTML in Javascript,

JSX HTML like or XML-like Syntax.

// React Element

Const jsxHeading = <h1 id="heading"> Namaste React using JSX </h1>;

It is not a proper JS and does not understable for JS engine.
JS engine understand "echma sciipt" language.

This code is transpiled by parcel using "babel package" before going to JS engine and makes it understable for JS engine.

* JSX ⇒ React.createElement ⇒ React Element - JS object ⇒ HTML element (render)

babel transpile this

⇒ Babel is a Javascript Compiler

Learn more at babeljs.

In JSX we give class using className not by only class.

Const jsxHeading = <h1 className="head" tabIndex="1">Namaste React using JSX</h1>;

* In JSX we use CamelCase for writing attribute name it is case of HTML.

If we have to write this JSX in one line then it is not mandatory to have paranthesis but if we need to write in multiple lines then it is mandatory to have () for babel to understand where JSX starts and ends.

React Component

There are two ways of creating React Component.

1. Class Based Component - Old way
2. Functional Component - New way

Functional Component is a normal JS function which returns some JSX code.

Note!- Always Name of functional component should start with Capital letter

```
const HeadingComponent = () => {
  return <h1> Namaste React functional Component </h1>
};
```

Acc. to JS we can also write

```
const HeadingComponent = () => <h1> Namaste React functional Component </h1>;
```

If we write class name or any attribute ← Parenthesis

```
Const HeadingComponent = () => (
    <div> <h1 className="heading"> Namaste React functional
                                   Component </h1>.
    </div>
);
```

How to Render React functional Component.

```
root.render(<HeadingComponent />);
```

```
root.render( Heading);  ← method of returning React Element.
```

Component Composition

Composing one component to other

e.g.

```
Const Title = () => (
    <h1 className="head">
        Namaste React using JSX
    </h1>
);
Const HeadingComponent = () => (
    <div id="container">
        <Title />
        <h2 className="heading">
            Namaste React functional Component
        </h2>
    </div>
);
```

All the code will come here

We can also write as `<Title></Title>`

`{Title()}`

As React Component is Normal JS fn

In Component creating we can also use normal functions instead of arrow functions. But Arrow functions are industry standard.

In JSX by using { } → curly Brackets we can put any JS code and it will come as HTML.

React element is ~~normal~~ JS object but const. is normal JS variable so by using { } we can inject React Element Inside React Component.

```
const elem = <span> React Element </span>

const HeadingComponent = () => (
    <div id = "container">
    {elem}
    <h1 className = "heading"> Namaste React Functional Component </h1>
    </div>
);
```

So, We can compose anything inside anything.

JSX take Care of Cross Sight scripting Attack

Suppose if elem is taking data from API and API send malicious data to any piece of code is run in the browser and can be attacked by attackers.

```
const elem = api.getData()
```

JSX will take care of it You can run any thing inside { }

We can also call functions in React Component.

As React Component is also a Normal JS function so we can also do component composition by calling it as a function.

⇒ { Title() }

<Title/>

<Title></Title>

} These 3 are same thing.

Code is Readable because of JSX.