Day 2 – Igniting our App

We will build our own 'create_React_app'

Code A :-

```
const heading = React.createElement (
    "h1",
    { id:"title",
    }
    "Heading 1"
);
```

(*) This code is exactly Similar to :-

Code B :- `<h1 id ="title">Heading 1 </h1>`

ie, Code A will produce Code B in our DOM

If I want to create a div having 2 children h1 and h2, I'll do it like this :-

```
const container = React.createElement (
    "div", // createElement takes first argument as the "tag"
    { id :"container", // second argument is the attributes
    }
    [heading1, heading2] // third argument is the children.
);
```

Scanned with CamScanner

The above code will be like this ↓ inside DOM

```
<div id="container">
    <h1 id="title">Heading 1 </h1>
    <h2 id="title">Heading 2 </h2>
```

Inside second argument, we can write anything.

```
{
    "id" = "container"
    hello = "world"
}
```

These are called 'Props'

"PROPS can be anything"

To make an app production ready, we should :-

(1) minify our file (remove our console logs, bundle things up)

(2) need a server to run things.

Even though we can load our "App.js", we can't get optimised version.

"Minify → Optimization → Clean console → and Bundle".

Scanned with CamScanner

In React, to get external functionalities, we use "BUNDLERS" :-

a) Webpack is a bundler
b) vite esbuild , rollup
c) parcel

These are alternatives

In create-react-app, the bundler used is "webpack".

Most bundlers do the same job.

Bundlers are packages. If we want to use a package in our code, we have to use a 'package manager'.

We use a 'package manager' known as npm or yarn.

npm :- No Problem Man

'npm' doesn't mean node package manager but everything else.

npm init (create a package.json file)

We use npm because we want a lot of packages in our project / react app.

[npm init -y] will skip lot of options.

Scanned with CamScanner

npm install -D parcel

→ dev dependency – becoz we want it in our developer machine.

→ parcel is one of the dependency

Then, we'll get package-lock.json.

[Caret & tidle sign → read.]

our project will automatically update if we use caret sign. (^)

```
"devDependencies" : {
    "parcel" : "^2.8.2"
}
```

caret

package-lock.json

will tell you which exact version of the library you are using.

Common issue

Q) "Something is working on my localhost/mymachine but not works in production". Why?

A) · package-lock file tells the exact version of the library we are using. Suppose, for dev dependency we're using 2.8.4 version & is right in package-lock.json file.

Scanned with CamScanner

But in 'package.json' file, it will be like "2.8.2"

'package-lock' file is an file that locks the version

"node_modules"

→ which gets installed is like a database for the npm.

→ This is where the Super powers comes from.

→ Our app has dependency on 'parcel'

→ 'parcel' also has dependencies on something else.

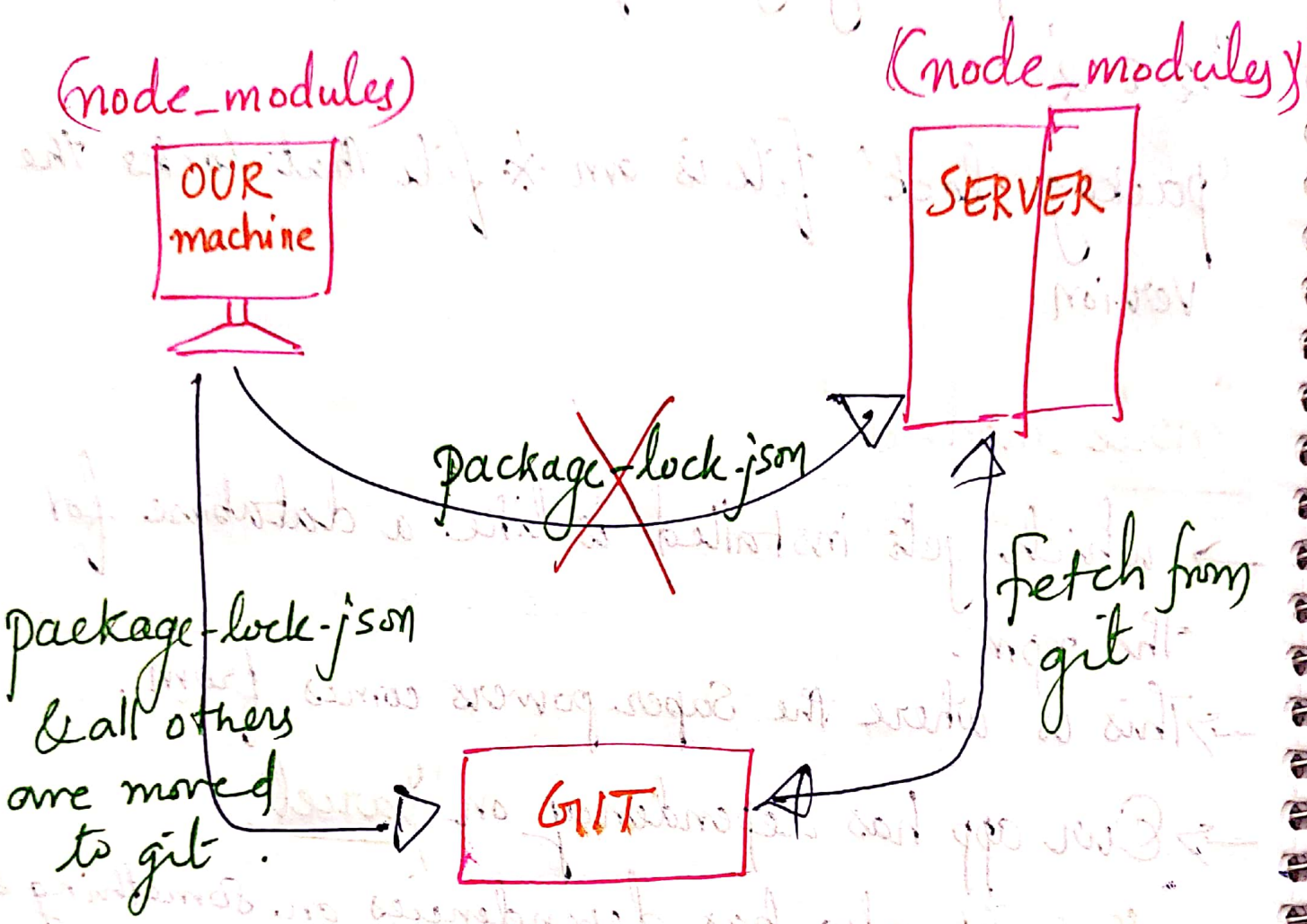
All these dependencies/superpowers are in 'node_modules'.

?) We don't put 'node_modules' into git. Why?

→ Because, our package-lock.json file have sufficient information to recreate node_modules.

→ package-lock.json file keep & maintain the version of everything in node_modules.

Scanned with CamScanner



→ 'node_modules' in our machine can be regenerated in server using the <u>package-lock.json</u> file.

Previously, we used CDN links to get react into our app. This is not a good way.
We need to keep react in our node-modules.

$\boxed{\text{npm install react}}$

Scanned with CamScanner

To ignite our app.

npx parcel index.html

▷ means 'execute using npm'

▷ is the entry point

Click 'Enter'

Then, a miniserver is created for us:
like localhost:1234
Parcel given a server to us.
"Parcel" ignited our app.

As we removed CDN links, we don't have react in our app.
So, we want to import it into our app.
For that we use the keyword "import"

Never touch 'node_modules' & 'package-lock.json'

Normal js browser don't know "import".
So, it shows error.

Scanned with CamScanner

```
import React from "react";
import ReactDOM from "react-dom/client";
```

As we got an error, we have to specify to the browser that 'we are not using a normal script tag, but a module'.

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

We cannot import & export scripts inside a tag. Modules can import and export.

Note :

* Hot Module Reload (HMR)

  → means that parcel will keep a track of all the files which you're updating.

* How HMR works?

  → There is File Watcher Algorithms (written in C++). It keeps track of all the files which are changing realtime & it tells the server to reload.

  → These all are done by "PARCEL".

Scanned with CamScanner

* There'll be a folder called .parcel-cache which will be there automatically.
In our project, parcel needs some space. So, it creates .parcel-cache.

* 'dist' folder keeps the files minified for us.
When we run command:

```
npx parcel index.html
```

This will creates a faster development version of our project & serves it on the server.

When I tell parcel to make a production build:

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

It creates a lot of things, minify your file.
And "parcel will build all the production files to the dist folder"

* What takes a lot of time to load in a website?
A):- Media - Images
Parcel does image optimization also.

* Parcel also does 'Caching while development'.

Scanned with CamScanner

* Parcel also takes care of your older version of browser

Compactible with older version of browsers'.

Sometimes we need to test our app on <u>https</u> becoz something only works on https.

Parcel gives us a functionality that we can just build our app on <u>https</u>. on dev machine.

npx parcel index.html --https

* Should put the <u>.parcel-cache</u> in gitignore

↳ because, anything which can be auto-generated should be put inside gitignore.

* Parcel ~~does~~ uses:

Consistent Hashing Algorithms

Scanned with CamScanner

* Parcel is 'Zero Config'.

Parcel features in a glance:

→ HMR - Hot Module Reloading
→ File Watcher algorithm - C++
→ Bundling
→ minify code
→ Cleaning our code
→ Dev and production build
→ Super fast build algorithm
→ Image Optimization
→ Caching while development
→ Compression
→ Compactible with older browser version
→ HTTPS on dev
→ port Number
→ Consistent Hashing Algorithm
→ Zero Config
→ Tree Shaking

Scanned with CamScanner

※ TRANSITIVE DEPENDECIES

We have our package manager which handles and takes care of our transitive dependencies of our code.

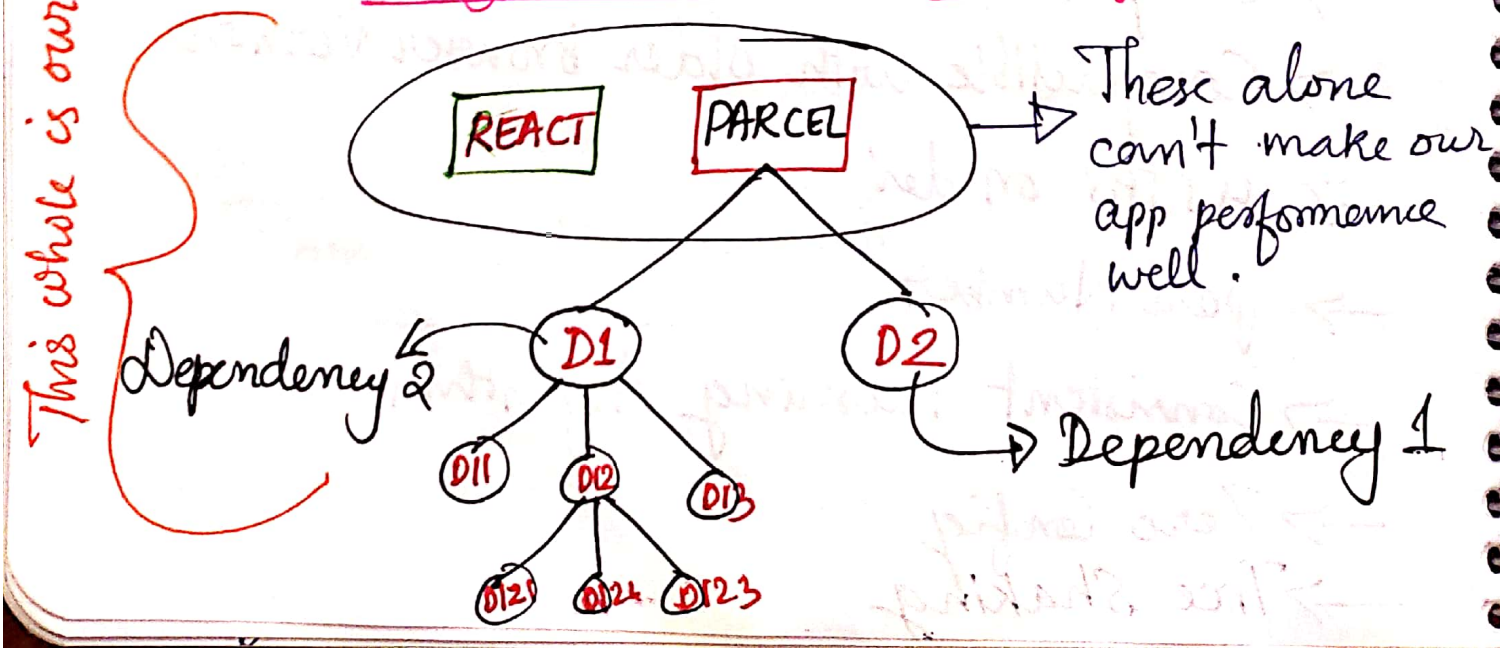
If you've to ~~use~~ build a production ready ~~build~~ app. which uses all optimizations (like minify, cleaning bundling, compression, consistent hashing etc) ~~then~~, We need to do all these.

But we can't do this ~~do~~ alone.

We need some dependencies on it.

Those dependencies are also dependent on many other dependencies.

Fig :- Our App (Dependency tree)



Scanned with CamScanner

Q) How do I make our app compactible with older browsers?

A): There is a package called 'browserslist' & parcel automatically gives it to us.

Browserslist makes our code compactible for a lot of browsers

go to 'browserslist.dev'

In package.json file, do :-

```
"browserslist": [
    "last 2 versions"
]
```

Support 74%

feeded with some configurations

means my ~~app~~ Parcel will make sure that my app works in last 2 versions of ~~to~~ all the browsers available.

If you don't care about other browsers, except chrome

```
"browserslist": [
    "last 2 Chrome versions"
]
```

Support 16%.

FINALLY,

"Parcel is a beast"..

Scanned with CamScanner

→ Tree Shaking

→ parcel has this superpower

→ means removing unwanted code.

Eg:- Suppose your App is importing a library which has a lot of functions (say, 20 helper funths). Then, all those 20 fns will come into your code. But in my App, I may want to use only 1 or 2 out of it.

Here, PARCEL will ignore all the unused code

Create-react-app : uses 'webpack' along with 'babel'.

Q) How can you build a performant-web-scalable app?

→ There are so many things that react optimises for us and parcel (bundlers) gives us.

→ Our whole application is a combination of all these things.

Scanned with CamScanner