Episode 02 - Igniting our app

Page No.
Date

We will build our own 'create_react-app'

Code A :-

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

This code is exactly similar to :-

Code B :-

```
<h1 id="title"> Heading 1 </h1>
```

i.e Code A will produce Code B in our DOM.

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

```
const container = React.createElement(
                    "div",
                    { id: "container" },
                    [heading1, heading2]
                );
```

createElement 1st argument as the tag

second argument is the attributes

third argument is the children.

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 the 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

↓

Bundle

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

(a) Webpack is a bundler
(b) vite
(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 package in our project / react app

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

Page No.
Date

npm install -D parcel → devdependency because we want it in our developed machine

→ parcel is one of the dependency.

Then will get package-lock.json.

[[caret & tilde sign → read]]

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

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

package-lock.json
will tell you which exact version of the library ~~Yar~~ you are using.

Common Issue

1. "Something is working on my localhost/mymachine but not works in production" why?
2. package-lock file tell 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. But in package.json file it will be like "2.8.2"

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

node_modules

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

→ This is where the superpowers come from.

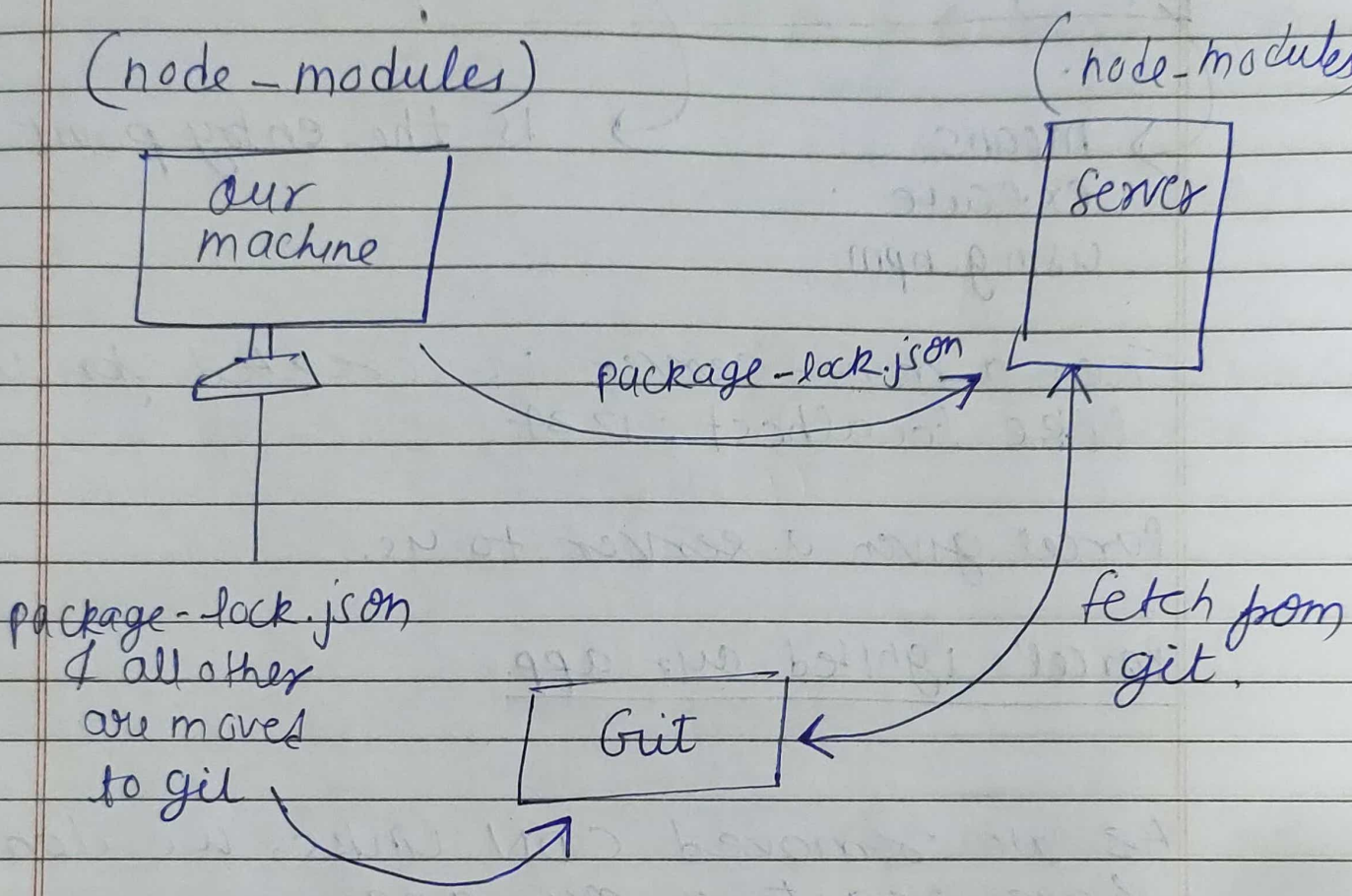
→ Our app has dependency on 'parcel'

→ parcel also has dependency on something else.
All these dependences/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.



→ node_modules in our machine can be generated in server using the package-lock.json 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.

npm install react

Page No.
Date

To ignite our app

```
npx parcel index.html
```

→ means execute using npm

→ is the entry point.

Then a mini-server 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.

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

As we get 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 script inside a tag module can import & export.

Note

Hot module Reload (HMR)

mean 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 tell the server to reload.

These all are done by "Parcel"

Video Time : 1:05:52

Page No.
Date

* There will 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 builds all the production files to the dist folder.

* What takes a lot of time to load in a website?

A) Media — Images
Parcel does images optimization also.

* Parcel also does caching while development

* Parcel also takes care of your older - version of browser

compatible with older version of browsers

Sometimes we need to test our app on https because something only works on https.
parcel gives us a functionality that we can first build our app on https on dev machine

```
npx parcel index.html --https
```

* should put the parcel-cache in gitignore

→ because anything which can be auto-generated should be put inside gitignore

* Parcel uses:

Consistent Hashing Algorithms

* parcel is zero config

Page No.
Date

Parcel feature 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
- Compatible with Older browser Version
- HTTPs on dev
- port Number
- Consistent Hashing Algorithm
- Zero Config
- Tree shaking

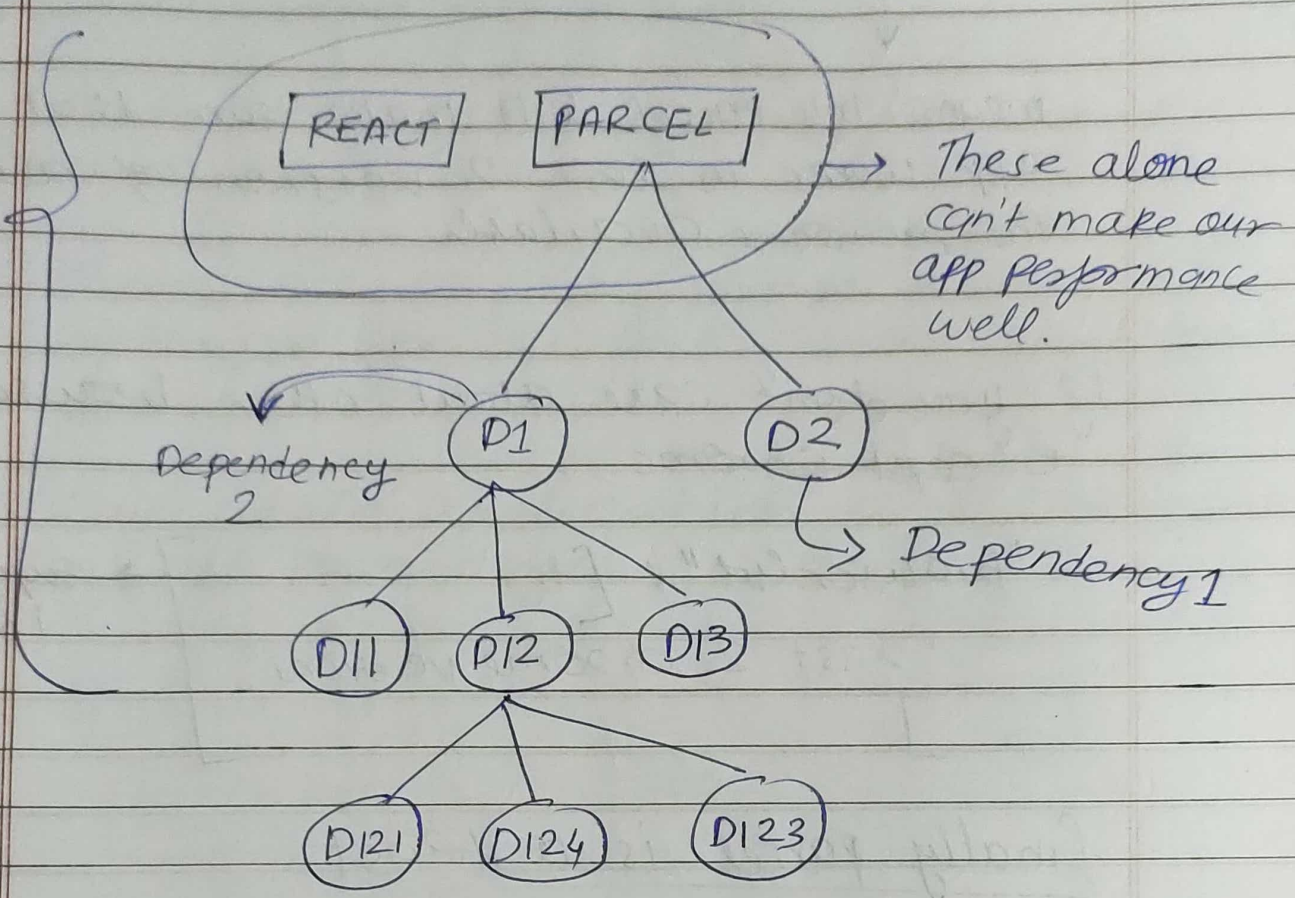
Transitive Dependencies

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

If you have to build a production ready app which use all optimization (like minify, cleaning, bundling, compression, consistent hashing) we need to do all these but we can't do this alone. We need some dependencies on it. Those dependencies are also dependend on many other dependencies.

This whole is our App.

Fig: Our App (Dependency tree)



How do i make our app compatible with older browser.

There is a package called "browserlist" & parcel automatically gives it to us.

Browserlist makes our code compatible for a lot of browsers.

go to browserlist.dev

In package.json do :-

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

→ Support 74%

→ feeded with some configuration

last 2 version

↓

means my parcel will make sure that my app works in last 2 version of all the browsers available.

If you don't care about other browser except chrome

```
"browserlist": [
    "last 2 chrome versions"
]
```

→ support 16%

finally parcel is a beast

Tree shaking

- parcel has this superpower
- mean removing unwanted code.

eg: Suppose your App is importing a library which has a lot of functions (say, 20 helper function). 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

How can you build a performant - web scalable app?

→ there are so many things that react optimize for us and parcel (bundler) gives us.

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