2 - Igniting our App

* For the production ready

  - you need server

  - optimize code

  - remove console

  - bundler

* Bundler (Optimize code)

  → webpack config
  → parcel (we use this)
  → vite

  } It's a package

* npm it manage package.

Configuring the project

- npm init

  we get package.json file here

  ↳ it's a configuration for npm

- The package is dependencies.

- It take care of the versions.

- Important package in our project is BUNDLER.

- Job of our bundler is to pack all our code so that we can send them to production.

- create react app - it use webpack config.

We will use "parcel".

Parcel:

To install → npm install -D parcel
↳ used for development.

Two types of dependencies:

- devDependencies - it generally requires for development phase, those package are parcel, webpack, vite, These package are neccessary only while you are developing your project.

To save a dependency as a devDependency

npm install --save-dev / npm install --save

- Dependencies - It's library and framework in which app is built on, need to function effectively. Eg React, Angular, Vue.

```
PS D:\Namaste React\02-Igniting our App> npm install -D parcel
[#########.........] - idealTree:02-Igniting our App: timing idealTree:#root Completed in 4272ms
```

index.html U App.js U package.json U ×

package.json > ...

```
16    "author": "Rizon Kumar Rahi",
17    "license": "ISC",
18    "bugs": {
19      "url": "https://github.com/rizor
20    },
21    "homepage": "https://github.com/ri
22    "devDependencies": {
23      "parcel": "^2.8.3"
24    }
25  }
26
```

— ^ and ~ ?

~ → It's a tilde

^ → It's a caret

1) ~version → "Approximately equivalent to version", it will update the version, without incrementing the minior version.

~1.2.3 will we release from 1.2.3 to <1.3.0

2) ^version → "Compatible with version", it will update the version, without incrementing the major version.

^2.3.4 will use release from 2.3.4 to 3.0.0

3) 2.3.4 ⇒ only work for this version (2.3.4) No update to the package.

When we installed parcel, new package-lock is created.

Package.json

— It keep track of the version is installed like parcel ^2.3.0 new update came it will update

like parcel ^2.3.0 now update some it will update in package.json.

Package-lock.json

- In this if the update of the package comes, it will not upgrade in package-lock, because to avoid uncertainty we lock our package-lock.json.

- It allow future dev to download the same dependencies as the project.

- It has the integrity which keep the hash.

```
"node_modules/@parcel/bundler-default": {
  "version": "2.8.3",
  "resolved": "https://registry.npmjs.org/@parcel/bundler-default/-/bundler-default-2.8.3.tgz",
  "integrity": "sha512-yJvRsNWWu5fVydsWk3O2L4yIy3UZiKWO2cPDukGOIWMgp/Vbpp+2Ct5IygVRtE22bnseW/E/oe0PV3d2IkEJGg==",
  "dev": true,
```

Q Why should I not modify package-lock.json?

- The file contain the information about the dependencies and their versions used in the project. Deleting it would cause dependencies issues in the production enivorment.

Node-Modules

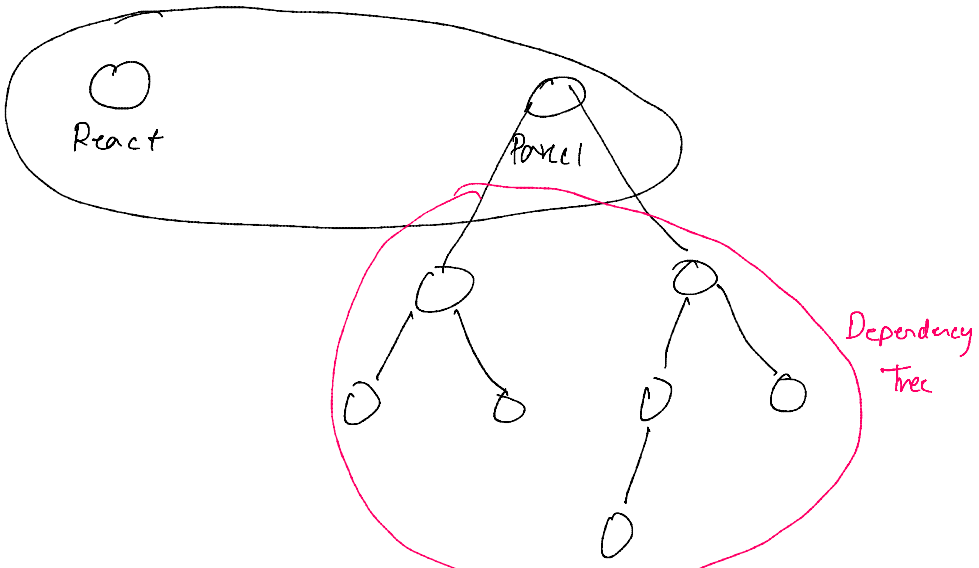
- It contain all the code from npm.

- Don't push node-modules in git

- Add it to git ignore.

- It's like database

Transitive Dependencies



Parcel itself has it own package.json it is dependent to others have it's known as Transitive Dependency.

- Package and Package-lock.json should be pushed to the git.
  - with help of them we can re-create the node_modules.

To build our app:

npx parcel index.html → entry point

- It build the app.
- It's executing the package.
- It created a folder called .parcel-cache and dist

- CDN is not a good way of importing React, so we

```
PS D:\Namaste React\02-Igniting our App> npm install react
```

= npm i react

we need it as normal dependencies, so we are not using -D.

```
PS D:\Namaste React\02-Igniting our App> npm install react-dom
```

```
"dependencies": {
  "react": "^18.2.0",
  "react-dom": "^18.2.0"
}
```

→ normal dependencies.

```
import React from "react";
```

→ this is coming from node-modules

App.js > parent

```
import React from "react";
import { ReactDOM } from "react";
```

```
@parcel/transformer-js: Browser scripts cannot have imports or exports.

D:\Namaste React\02-Igniting our App\App.js:1:1
> 1 | import React from "react";
>   | ^^^^^^^^^^^^^^^^^^^^^^^^^
  2 | import { ReactDOM } from "react";
  3 | D:\Namaste React\02 Igniting our App\index.html:18:5
 17 |
> 18 |     <script src="./App.js"></script>
>    |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ The environment was originally created here
 19 |   </body>
 20 | </html>
```

H treat it as normal JS file.

So we tell the script file as module

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

I am H1 tag

I am H2 tag

I am H1 tag

I am H2 tag

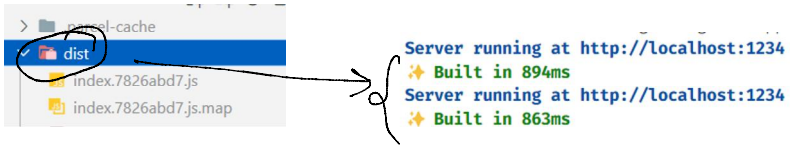
▸Warning: You are importing createRoot from "react-dom" which is not supported. You should instead import it from "react-dom/client". index.js:1

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

const parent = React.createElement("div", { id: "parent" }, [
  React.createElement("div", { id: "child" }, [
    React.createElement("h1", {}, "I am H1 tag"),
    React.createElement("h2", {}, "I am H2 tag"),
  ]),
  React.createElement("div", { id: "child2" }, [
    React.createElement("h1", {}, "I am H1 tag"),
    React.createElement("h2", {}, "I am H2 tag"),
  ]),
]);

console.log(parent); //object

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

```
# Parcel

- Dev Build
- Local Server
- HMR (Hot Module Replacement)
  - exchanges, adds, or removes modules while an application is running, without a full reload. This can significantly speed up development in a few ways: Retain application state which is lost during a full reload.
- File Watching Algorithm this is written in C++.
- Caching - Faster Builds (parcel-cache)
```

parcel- cache

- Parcel need space so parcel-cache is created.
- Should be added in git.ignore



- The build will be created in dist folder
- Should be in gitignore

for the production code :

→ npx parcel build index.html.

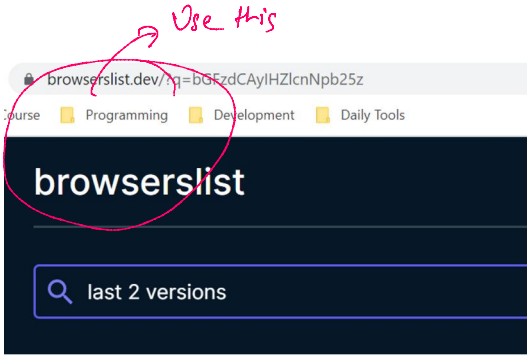
it will give error, so go to package.json and remove the main.

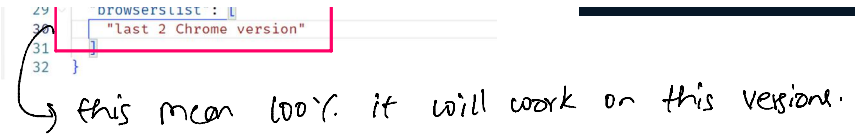
```
{
  "name": "02-igniting-our-app",
  "version": "1.0.0",
  "description": "This is namaste react  - 2",
  "main": "App.js",
```

```
PS D:\Namaste React\02-Igniting our App> npx parcel build index.html
/ Optimizing index.html...
```

- Now suppose this project should only support latest 2 version of browser we use.

```
"license": "ISC",
"bugs": {
  "url": "https://github.com/rizonkumar/namaste
},
"homepage": "https://github.com/rizonkumar/nama
"devDependencies": {
  "parcel": "^2.8.3",
  "process": "^0.11.10"
},
"dependencies": {
  "react": "^18.2.0",
  "react-dom": "^18.2.0"
},
"browserslist": [
  "last 2 Chrome version"
]
}
```





this mean 100%. it will work on this versions.

```
# Parcel

- Dev Build
- Local Server
- HMR (Hot Module Replacement)
  - exchanges, adds, or removes modules while an application is running, without a full reload.
  This can significantly speed up development in a few ways: Retain application state which is
  lost during a full reload.
- File Watching Algorithm this is written in C++.
- Caching - Faster Builds (parcel-cache)
- Image Optimization
- Minification
- Bundling
- Compress
- Consistent Hashing
- Code Splitting
- Differential Bundling - support older browsers
- Tree Shaking - remove unwanted/unused codes
  Tree shaking is process of removing the unwanted code that we do not use while developing the
  application. In computing, tree shaking is a dead code elimination technique that is applied
  when optimizing code.
- Different dev and production bundles
```