Igniting Our App! (Namaste-React)

 Please make sure to follow along with the whole *"Namaste React"* series, starting from Episode-1 and continuing through each subsequent episode. The notes are designed to provide detailed explanations of each concept along with examples to ensure thorough understanding. Each episode builds upon the knowledge gained from the previous ones, so starting from the beginning will give you a comprehensive understanding of React development.

 I've got a quick tip for you. To get the most out of these notes, it's a good idea to watch **Episode-2** first. Understanding what *"Akshay"* shares in the video will make these notes way easier to understand.

So far, here's what we have learned in the previous episode.

- We studied about Libraries, Frameworks, their differences.
- We have also created Hello World! using HTML, JavaScript, and React.
- We have also studies about what is Emmet, CORS (Cross Origin)

Q) To make our app production ready what should we do?

Minify our file (Remove console logs, bundle things up)

Need a server to run things

NOTE: Minify -> Optimization ->Clean console -> Bundle

* **Bundlers:**

- A bundler is a tool that bundles our app, packages our app so that it can be shipped to production. • Examples of Bundlers:
    - Webpack
    - Vite
    - Parcel

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

* **Package Manager:**

- 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

* **Configuring the Project:**

```
npm init
```

- It creates a package.json file.

Now to install parcel we will do:

```
npm install -D parcel
```

- Now we will get a package-lock.json file.

* package.json:

- Package.json file is a configuration for NPM. Whatever packages our project needs, we install those packages using `npm install <packageName>`.

- Once package installation is complete, their versions and configuration related information is stored as dependencies inside package.json file.

* package-lock.json:

- Package-lock.json locks the exact version of packages being used in the project.

Q) What is difference between package.json and package.lock.json?

- In package. json we have information about generic version of installed packages whereas in package.lock.json we have information about the specific or exact version of installed packages.

* node_modules:

- Which gets installed is like a database for the npm.
- Every dependency in node_module will have its package.json.
- Node modules are very heavy so we should always put this in git ignore.

[?] **NOTE: Never touch node_modules and package-lock.json**

*** To ignite our app:**

```
npx parcel index.html
```

- npx means 'execute using npm'
- index.html is the entry point

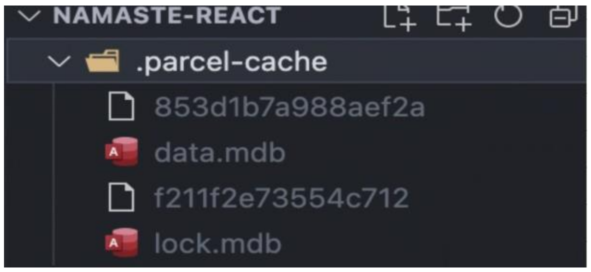
*** Hot Module Replacement (HMR):**

- It means that parcel will keep a track of all the files which you are updating.
- There is File Watcher Algorithm (written in C++). It keeps track of all the files which are changing realtime and it tells the server to reload. • These are all done by PARCEL

*** parcel-cache:**

- Parcel caches code all the time.
- When we run the application, a build is created which takes some time in ms.
- If we make any code changes and save the application, another build will be triggered which might take even less time than the previous build. This reduction of time is due to parcel cache.
- Parcel immediately loads the code from the cache every time there is a subsequent build.
-
- On the very first build parcel creates a folder .parcelcache where it stores the caches in binary codeformat.
- Parcel gives faster build, faster developer experience because of caching.

We can check deleting the file and again run the server.



*** dist:**

- It keeps the files minified for us.
- When bundler builds the app, the build goes into a folder called dist.
- The `/dist` folder contains the minimized and optimised version the source code.
- Along with the minified code, the /dist folder also comprises of all the compiled modules that may or may not be used with other systems.
- When we run command:

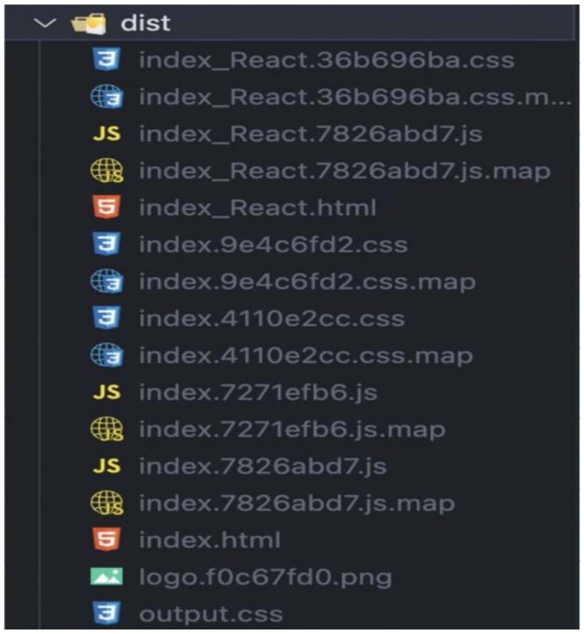
```
npx parcel index.html
```

- This will create a faster development version of our project and 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 the parcel will build all the production files to the dist folder.



```
dist
    index_React.36b696ba.css
    index_React.36b696ba.css.m...
    index_React.7826abd7.js
    index_React.7826abd7.js.map
    index_React.html
    index.9e4c6fd2.css
    index.9e4c6fd2.css.map
    index.4110e2cc.css
    index.4110e2cc.css.map
    index.7271efb6.js
    index.7271efb6.js.map
    index.7826abd7.js
    index.7826abd7.js.map
    index.html
    logo.f0c67fd0.png
    output.css
```

* Parcel features at a glance:

Hot Module Replacement (HMR) 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 versions

Https on dev

Image Optimization

Port No Consistency Hashing Algorithm

Zero Config

Tree Shaking

* Transitive Dependencies :

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

- If we've to build a production ready app which uses all optimisations (like minify, bundling, compression, etc), we need to do all these.
- But we can't do this alone, we need some dependencies on it. Those dependencies are also dependent on other dependencies.

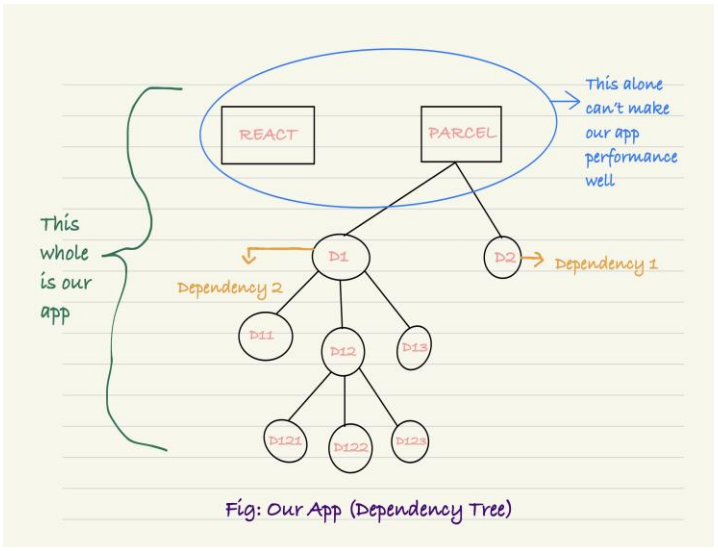
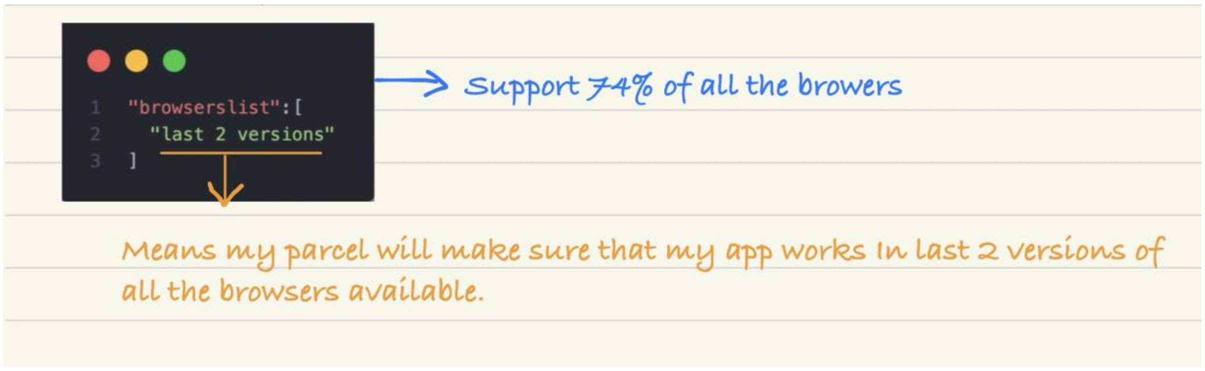


Fig: Our App (Dependency Tree)

* Browserslist:

- Browserslist is a tool that specifies which browsers should be supported/compatible in your frontend app.
- It makes our code compatible for a lot of browsers.
- In package.json file do:



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

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

* Tree Shaking:

- Tree shaking is a 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.

Self Notes-

Chapter 02 - Assignment - Igniting our App

Igniting our App- Make the production ready app.

Here we will learn how we can create your own production ready app instead of already existing one .

Q. README.md – markup file

README.md file typically serves as a documentation file that provides information about the project like Project Description, Installation Instructions, Usage, Folder Structure, Dependencies, Contributing, License

==

In the last episode we are simply created the html , js file and use the react there but that code is ready to go to the project – answer is no. so now we are creating the App which is ready to push the production / how we can create production ready applications.

Your code ready to go to the production you have to do lot of processing in your local, lot of processing doing to push the code in production, you need to do code bundling, code splitting, you need to do image optimization, need to remove the comments, need to minified the code, compressed it, need lot of things before move the code in production.

- When we run the npx create-react-app my-app and it is created the already ignited react app.(the code which is ready to push to the production), in this episode we will learn about how how to create own react app.

- Peoples says react make our app fast.
  React is making our app fast – No there are lots of libraries, lots of other packages, javascript code that make the code fast, scalable and production ready app not just react.

Q . To make our app Production ready what should we do?

1. Minify our file (Remove console logs, bundle things up)
2. Need a server to run things

Minify —> Optimization —>Clean console —> Bundle

Q. Package Manager:

We use a package manager known as npm or yarn *

Npm – it is not the node package manager (does not have the full form).
Npm manages packages, but it does not stand for node package manager.

It is the standard repository for all the packages, it is biggest packages manager, any packages if you need to add in you project you need npm.

All the packages hosted there, all the library, utilities comes from npm

Configuration –

First– create the package.json

When we run npm init command package.json file will create inside the folder.

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

npm init (Creates a package.json file),

Q. package.json – It is basically the configuration for npm, whatever description, name we gave after running the npm init command showing here, it is created the JSON file.

<u>Why do we need package.json file –</u>

Our project dependent on other packages, those projects depend on dependencies and npm manages at, npm manages of what is the version of the package, if I am using some other library, some other packages, what is the version of the package. Npm take care of version of that package and where it will manage, it will manage/ take care in package.json.

Second- Now Installing are dependencies.

Now we are installing the most import package in aur project is bundler.

When we install any of the dependency, we are using npm

npm install -D parcel –> D is for Dev dependency (details is in pg. below): because we want it in our development machine

Parcel is one of the dependency Then, we'll get a package-lock.json file
we are just fetching the parcel in the npm

Q. Bundlers: In React, to get external functionalities, we use Bundlers.

Job of the bundler – it will bundle your app; it packages your app properly so that it can be shipped to production.

Or we can say that a bundler is a tool that bundles our app, packages our app so that it can be shipped to production
Examples: 1. Webpack 2. vite 3. Parcel

In short, bundlers are tools used to combine multiple files (such as JavaScript, CSS, and images) from a project into a single file or multiple files in a format optimized for the web.

or we can say that - The whole code needs to be minified. The whole code needs to be cached. The whole code needs to be compressed, cleaned before it can be sent to production They help manage dependencies, improve performance, and simplify the deployment process by reducing the number of HTTP requests needed to load a webpage. Popular bundlers for React projects include Webpack, Parcel, VEET and Rollup.

```
In create-react-app, the bundler used is webpack
```

Create react app uses webpack bhind the scenes it uses webpack bundler and babel behind the scenes to build things up.
In our project we are going to use parcel.
When you install any dependency in your app you would have the same command –

Npm install -D (dependency name*) parcel.

There are two type of dependency we install

1. Dev dependencies – it means it is generally required for in our development phase. When we are developing our app then we required that dev dependencies
2. Normal dependencies – are use in production also.

Now the parcel is a bundler chunking minification and we will not done production, we will do this in our development phase so we will install parcel as a dev dependency.

Npm this repository which contains all the packages we are just installing we are fetching parcel from there.

Parcel is the beast it will do lot of things to our app.

****

package.json – where you will see all the dev dependencies, it is configuration for npm it basically keeps a track all the dependencies all the packages our code and here we can also see the version of the parcel (bundler)

```
"parcel": "^2.12.0"
```

*** Caret sign(^):**

Now the version is 2.12.0, suppose new version of parcel released what will happen if you have this carrot into you app parcel will automatically be upgraded minor version automatically.

*** Tilde sign(~):** it will upgraded the major version. When we do the major upgrade lot of things will break, So better to use caret.

When we run npm install -D parcel – one more file also added in folder i.e package-lock-json

Q. Diff packagelock.json and package.json

`package.json:` keeps the approx. version and it is editable.

`packagelock.json:` keeps the exact version, whatever version is there in my dev machine is the same version which being developed on to my production, it keeps track of all the exact versions and it is not editable.

When we run npm install -D parcel – one more file also added in folder i.e node module

`node modules` contains all the code that we fetched from npm, when we did npm install parcel what happened was it just went to the production, it took some time and it was fetching all the code of parcel and putting it inside node modules. If you will go to the node module you will see there will be parcel over there. That is the actual parcel code which npm has fetched.

Node module is kind of like a database it contains the actual data of that dependencies of those packages that our project needs that is why there is no modules.

Node module is very heavy basically fetches all the code all the dependencies into our system. It is like database where all our packages exist.

Node modules have transitive dependencies – parcel have some dependency and dependencies have some other dependencies.

Node module is the collections of dependencies.

Parcel needs lot of other dependencies that is the reason so many folders inside the node module, basically it contains transitive dependencies.

Q. In the whole project how many package.json do we have?

Every dependency inside the node module have its own package.json.

Q. Should I put my package.json and package.lock json into git ?

Ans – is absolutely yes because it maintains note of what all dependencies our project needs. It is maintaining the version, integrity(hash) so whatever is there in my local, should be the same in production.

Q. Should I put node module in production, git?

No, Size of the node module is huge**,** so the best practice is we will put the node module inside **git ignore.(**If you want some file not go into the production or github, just put inside the gitignore) and If I have my package.json and package.lock json we can regenerate the node module. Even if I delete the node module from my local we can recreate by (npm install) command, Whatever we can regenerate we can not push into the github.

Q. Common Issue: It is working on my local, how it break on production?

To avoid this, package-lock.json keeps an hash (in integrity field) to verify that whatever is in my dev machine, is the same version deployed on the production.

Now we are Igniting Our App- (Building the app using parcel)

Run npx parcel index.html

1. It will create the server for us (http://localhost:1234) and the app available on the server,port.

Just like npm, we have npx, when we do npm that means we are installing the package , npx means executing the package using npm. Here that means we are executing parcel now / we are executing the bundler .

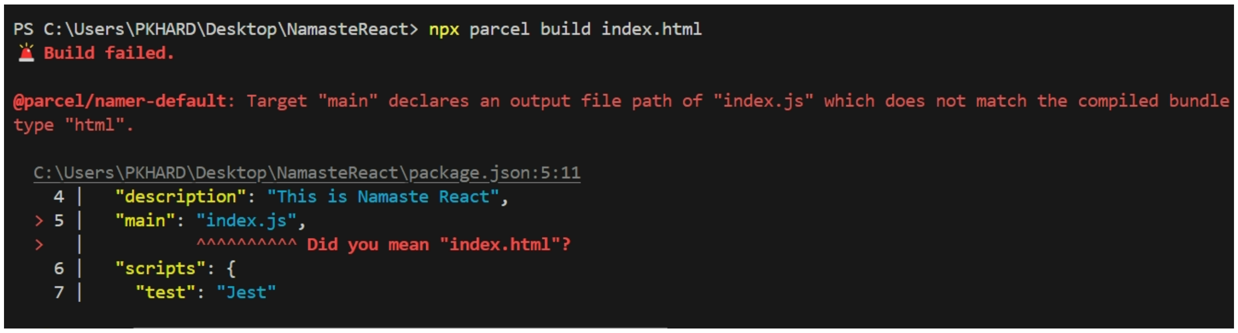
There is two ways to get react into App- 1.CDN 2. Npm
React is hosted in cdn and npm

This cdn link is not a preferred way to bring react and react dom in our project because fetching from cdn takes costly operation, it will get react from unpkg.com.
If react already in my node modules how easy would it be using react. We don't need to another network call to get react , will already have it in our node module.
According to the react version the cdn link get changes . so it will support all the versions. So it is better to just have it inside the package.json, it easer to manage all the dependency and it is easy to manage react also, is one of the dependency inside npm package.json.

Third – Remove the CDN link and now we will install react as a package to our code.
How do we install it -npm install react
Now install react dom- npm install react-dom
Start the server – npx parcel index.html

Forth – Create the prod build
npx parcel build index.html

But we will get the error

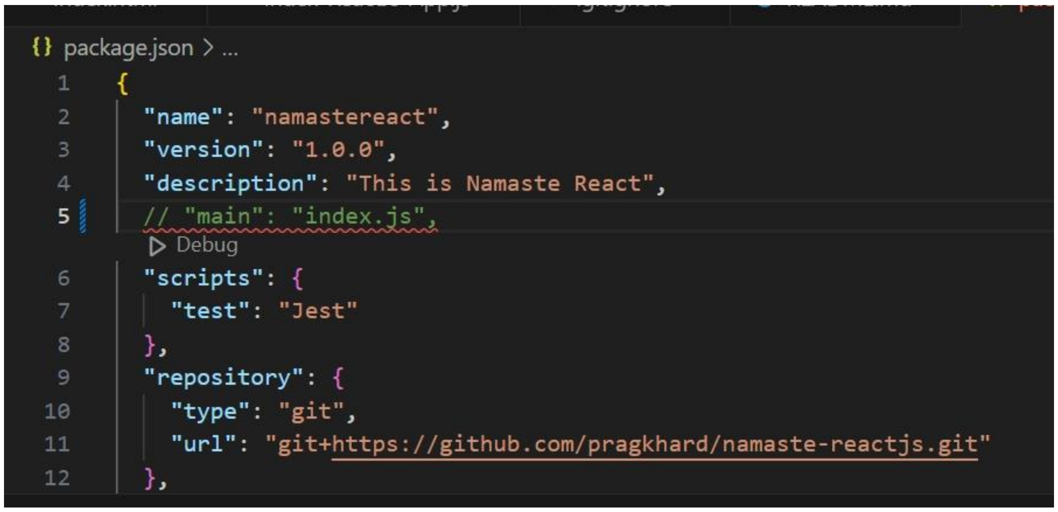


```
PS C:\Users\PKHARD\Desktop\NamasteReact> npx parcel build index.html
🚨 Build failed.

@parcel/namer-default: Target "main" declares an output file path of "index.js" which does not match the compiled bundle
type "html".

  C:\Users\PKHARD\Desktop\NamasteReact\package.json:5:11
    4 |   "description": "This is Namaste React",
  > 5 |   "main": "index.js",
  >   |           ^^^^^^^^^^ Did you mean "index.html"?
    6 |   "scripts": {
    7 |     "test": "Jest"
```

To remove the error –



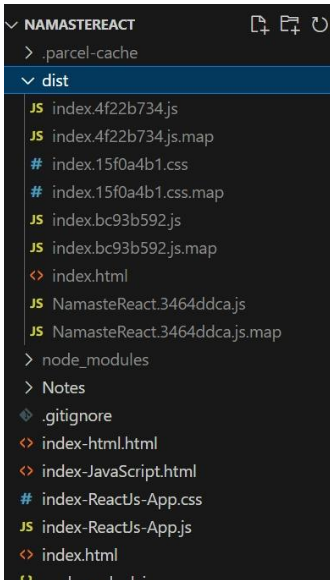
```
{} package.json > ...
{
  "name": "namastereact",
  "version": "1.0.0",
  "description": "This is Namaste React",
  // "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "Jest"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/pragkhard/namaste-reactjs.git"
  },
```

This main is not required when we are using parcel , this is the way to tell npm this is the entry point of the app. When you are using parcel we gave entry point index.html, so parcel conflict with this main.html, so will get the error so remove this.

When we do npx parcel build index.html it will bundle the app, minified the app and it will put all those file inside the dist folder and those file will use in the development so basically this dist contain development file(for production).

When we execute parcel, when we write npx parcel build index.html, it generate development build for your project and it will hosted on http://localhost:1234, so when it will generate development build it will put up in dist folder so, after that the code coming from the dist folder and when you reference the page it will use parcel-caches updating using HMR, so when you build production build it will build inside dist.
So like this way we will get the production ready code, it will give the minified code for us. The production build is the highly optimize build which will push to the production and serve the app to the users, It will be fast , it will be performant and it will be optimized.

We can regenerate dist and parcel-cache file using npx parcel index.html so no need to push into the github, so will put both the files inside the gitignore

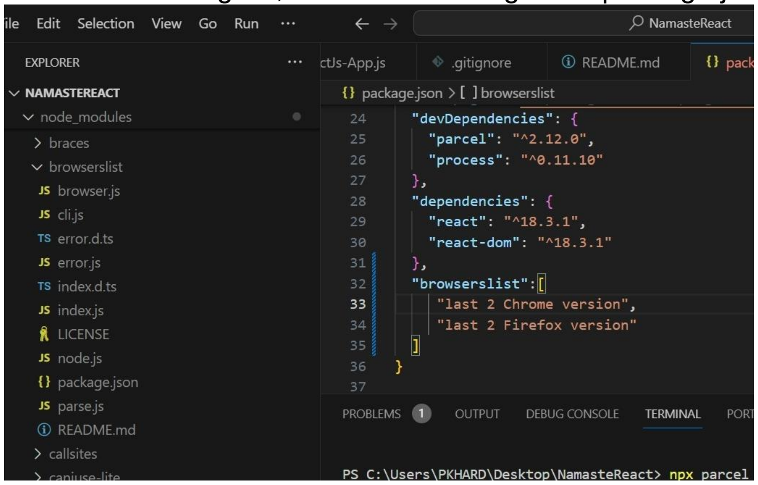


Let me the app compatible to the older version of the browser.
So for that we will use browser list .
If we will go to the node module we can see the browser list , but we have to tell the browser list what are the browser we have to support my app in, so for that we will do some configuration.

Browser list is basically the npm package.

Website - https://browserslist.dev/?q=bGFzdCAyIHZlcnNpb25z
Git repo - https://github.com/browserslist/browserslist#query-composition

Now we have to tell your project what all those browser should your app be supported in.
we have to configure, we have to configure in package.json



```
  "devDependencies": {
    "parcel": "^2.12.0",
    "process": "^0.11.10"
  },
  "dependencies": {
    "react": "^18.3.1",
    "react-dom": "^18.3.1"
  },
  "browserslist":[
     "last 2 Chrome version",
     "last 2 Firefox version"
  ]
}
```

```
PS C:\Users\PKHARD\Desktop\NamasteReact> npx parcel
```

Suppose if I want my app support to the last 2 version of the chrome so we will write –
last 2 chrome version.
It might or might not will work on the other browser version, but it will definitely work on the

chrome last 2 versions.

`Please Note:` *Write the answers and code on your own while finishing your assignments. Try toput down your thoughts into words by yourself in your own words. (This will help you develop muscle memory and you will remember all the concepts properly)* ☐

# **Theory Assignment:**

- - What is `NPM`?
- - What is `Parcel/Webpack`? Why do we need it?
- - What is `.parcel-cache`
- - What is `npx` ?
- - What is difference between `dependencies` vs `devDependencies`
- - What is Tree Shaking?
- - What is Hot Module Replacement?
- - List down your favourite 5 superpowers of Parcel and describe any 3 of them in yourown words.
- - What is `.gitignore`? What should we add and not add into it?
- - What is the difference between `package.json` and `package-lock.json`
- - Why should I not modify `package-lock.json`?
- - What is `node_modules` ? Is it a good idea to push that on git?
- - What is the `dist` folder?
- - What is `browserlists`
  Read about dif bundlers: vite, webpack, parcel
- Read about: ^ - caret and ~ - tilda
- Read about Script types in html (MDN Docs)

# **Project Assignment:**

- In your existing project

- - intialize `npm` into your repo
- - install `react` and `react-dom`
- - remove CDN links of react
- - install parcel
- - ignite your app with parcel
- - add scripts for "start" and "build" with parcel commands
- - add `.gitignore` file
- - add `browserlists`
- - build a production version of your code using `parcel build`

# **References**

- Creating your own create-react-app
- Parcel Documentation
- Parcel on Production
- BrowsersList: https://browserslist.dev/